Appendix A

EDEN Language Guide

A.1 Introduction 1

A.2 Lexical Conventions 1

A.3 Syntax notation 1

A.4 Comments 1

A.5 Identifiers (Names) 2

A.6 Keywords 2

A.7 Objects and Lvalues 2
A.7.1 Read/Write Variables 2
A.7.2 Function Definition 3
A.7.3 Formula Variable 3
A.7.4 Action Specification 3

A.8 Data Structures 3
A.8.1 @ (undefined) 3
A.8.2 Integer 4
A.8.3 Character 4
A.8.4 Floating point 5
A.8.5 String 5
A.8.6 Pointer 6
A.8.7 List 6
A.8.8 Function 6

A.9 Expressions 8
A.9.1 Lvalue 8
A.9.2 Primary expression 9

A.10 Operators 9
A.10.1 Arithmetic operators 9
A.10.2 String operators 9
A.10.3 List formation operator 10
A.10.4 Unary operators 10
A.10.5 Relational operators 10
A.10.6 Equality operators 11
A.10.7 Logical AND operator 11
A.10.8 Logical OR operator 11
A.10.9 Conditional operator 11
A.10.10 Assignment operators 11
A.10.11 Precedence and order of evaluation 12

A.11 Procedural Statements 12
A.11.1 Expression Statements 12
A.11.2 Insert Statement 12
A.11.3 Append Statement 13
A.11.4 Delete Statement 13
A.11.5 Shift Statement 13
A.11.6 Compound Statement 13
A.11.7 Condition Statement 13
A.11.8 While Statement 14
A.11.9 Do Statement 14
A.11.10 For Statement 14
A.11.11 Switch Statement 14
A.11.12 Break Statement 15
A.11.13 Continue Statement 15
A.11.14 Return Statement 15
A.11.15 Null Statement 15

A.12 User-defined Functions 15

A.13 Definitive Statements 16
A.13.1 Formula Definition 16
A.13.2 Action Specification 17

A.14 Miscellaneous Commands 17
A.14.1 Dependency Link 17
A.14.2 Query 18

A.15 Syntax Summary 19
A.15.1 Expressions 19
A.15.2 Statements 20
A.15.3 Function Definition 20
A.15.4 Dependency Link 20
A.15.5 Query Command 21

A.16 Pre-defined Variables 21

A.17 Pre-defined Functions 21

A.1 Introduction

The EDEN language has a number of C-like statements and operators. C programmers may find it familiar but they are still advised to go through this chapter since there are new data types and operators. Not all C operators have been implemented.

An EDEN program is a list of formula definitions, action specifications, function (procedure) definitions and C-like procedural statements. These definitions, specifications and procedural statements can appear in any order (though the order may affect the result of the program).

program:
procedural-statement program
function-definition program
formula-definition program
action-specification program
dependency-link program
query-command program

The procedural statements are the only executable statements. When a procedural statement is encountered, this statement will be executed. The effect of execution is to evaluate an expression, assign values to variables or call other functions/procedures (these must be defined earlier). The order of statements reflects the order of execution, and thus affects the results of computation. The user is responsible for arranging the statements in proper order to get the correct results.

In addition, the EDEN interpreter will do the (re-)calculation of the formula definitions and/or invoke the procedures defined by the action specifications automatically. The order of calculation of formula definitions and execution of actions should not be of concern to the user, and is fully controlled by the interpreter.

The dependency-link command is an alternative way of specifying actions. The query command inspects the current definitions of the objects.

A.2 Lexical Conventions

There are five classes of tokens: identifiers, keywords, constants, operators, and other separators. Blanks, tabs, newlines, and comments (collectively, "white space") as described below are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

A.3 Syntax notation

In the syntax notation used in this chapter, syntactic categories are indicate by *italic times roman* type, literal words and characters in `courier` type, and keywords in **`bold courier`** type. Alternative categories are listed on separate lines. An optional terminal or non-terminal symbol is indicated by the subscript "opt", so that

$$[\ \textit{expression-list}_{opt}\ ]$$

indicates an optional expression list enclosed in square brackets.

A.4 Comments

Comments are arbitrary strings of symbols placed between the delimiters `/*` and `*/`. The whole sequence is equivalent to a white space. Note that `/* */` can be nested. It is useful to comment a block of program with comments in it.

```
/* This is a comment */
/* --- Begin of comment ---
      /* This is a nested comment */
--- End of comment --- */
```

Comments are not parts of the executable program, but are used by the programmer as a documentation aid.

A.5 Identifiers (Names)

An *identifier* (*name*) consists of a sequence of letters and digits. The first character must be a letter. The underscore character _ is considered a letter. EDEN imposes no limit on the number of characters in a name, but the implementation of the EDEN interpreter does (about 255 characters). An EDEN keyword cannot be used as an identifier.

Examples of identifiers:

```
hello       this_is_a_most_unusually_long_name
IF          foO         bAr         HorseSense
var1        var2        _auto_      _
```

Upper- and lowercase letters are distinct, so `Count` and `count` are different identifiers. An identifier is a symbolic name of a variable, function or other objects.

A.6 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
append      default     func        proc
auto        delete      if          return
break       do          insert      shift
case        else        is          switch
continue    for         para        while
```

A.7 Objects[1] and Lvalues

An *object* is a manipulatable region of storage; an *lvalue* is an expression referring to an object. An obvious example of an lvalue expression is an identifier. There are operators which yield lvalues: for examples, if `E` is an expression of pointer type, then `*E` is an lvalue expression referring to the object to which `E` points. The name "lvalue" comes from the assignment expression `E1 = E2` in which the left operand `E1` must be an lvalue expression.

Objects can be a read/write variable, a function (procedure), a formula variable or an action specification.

A.7.1 Read/Write Variables

A *read/write variable* (RWV) is a named storage that holds a data. The data can be any one of the data structures described in §A.8.

There is no need to declare a variable before it is used. Memory storage is allocated when the variable name first appears in a program.

Data can be assigned to RWV's using assignment statements (see §A.10.10). For example,

---

1 The vocabulary is based upon the C reference manual. The term "object" is NOT used in the OOP sense.

```
V = 1;
```

The identifier (name) V denotes a RWV whose value is assigned to integer 1 by the operator =. The semi-colon terminates the statement.

The value of a RWV is referenced if its name appears in an expression. If the RWV hasn't been assigned a value before its value is referenced, the value @ (means "undefined"; see §A.8.1) is assigned to it automatically.

Different typed data can be assigned to the same variable at different time. The EDEN interpreter checks for data type clash at run-time.

The interpreter assumes all objects are RWV's until they are defined to be functions, formula definitions or action specifications.

A.7.2 Function Definition

A function definition is an object that stores an entry point (address) of a sequence of instructions (the function) which computes a result from the parameters. There are some functions pre-defined by the intepreter. The user can defined his/her own functions using some procedural statements.

The value of a function definition is equivalent to the function (address of the instruction sequence). (See §A.8.8)

Note that the interpreter allows an identifier designating a RWV to be re-used as a function name but not vice versa, because the intepreter assumes all objects are RWV's initially.

A.7.3 Formula Variable

A formula variable is an object that stores a formula expression (unlike a RWV which stores a value). See §A.13.1 for the syntax of a formula definition. The value of a formula variable is equivalent to the current value of the formula expression that the variable stores. To optimize the computation speed, the formula variable also stores the up-to-date value of the formula expression. So, whenever, the value of the formula variable is referenced, this value is used. The interpreter is responsible for updating this value.

An identifier designating a RWV can be re-used to designate a formula variable and vice versa.

A.7.4 Action Specification

An action specification is an object that stores an entry point (address) of an instruction sequence — the action procedure. See §A.13.2 for the syntax of an action specification. It also stores a list of objects on which the action depends. Whenever the values of these objects (usually are RWV's or formula variables) are changed the action procedure will be invoked by the system.

Note that the value of an action specification is equivalent to the address of the action procedure.

An identifier designating a RWV can be re-used as an action name but not vice versa.

A.8 Data Structures

There are 8 different data types: @, integer, character, floating point, string, pointer, list, and function. Integer and character types are sometimes collectively called integral type.

A.8.1 @ (undefined)

The constant `@` (undefined)[2] is a special value. It has a unique (un-named) data type. Some operators, such as arithmetic operators, accept `@` as their operands, and usually `@` will be returned. Other operators which do not take `@` operands will generate an error. See §A.10 for the discussion on operators.

A.8.2 Integer

Decimal integer constants

A decimal integer constant is a stream of digits with non-leading zero. E.g.

`123`	*valid*
`A123`	*invalid (begins with non-digit)*
`0123`	*invalid (begins with zero but is a valid octal)*

Octal integer constants

An octal integer constant is a stream of digits with a leading zero. The digits `8` and `9` have octal values 10 and 11 respectively. E.g.

`0456`	*valid (= 302 decimal)*
`018`	*valid (= 020 octal = 16 decimal)*
`456`	*invalid (but is a valid decimal)*
`A456`	*invalid (begins with non-digit)*

Hexadecimal integer constants

A hexadecimal integer constant is a stream of hexadecimal digits starting with `0x` (zero followed by the character `x`). Letters `A` to `F` have the hexadecimal values 10 to 15 respectively. Lowercase letters `a` to `f` are equivalent to their corresponding uppercase letters. E.g.

`0xAB`	*valid (= 171 decimal)*
`0x1f`	*valid (= 31 decimal)*
`AB`	*invalid (begins with an alphabet letter)*
`01f`	*invalid (missing `x`)*

A.8.3 Character

A character constant is a character enclosed in single quotes, as in `'x'`. The value of a character constant is the numerical value of the character in the machine's character set. E.g.

`'A'`	*valid (value is 65 for the ASCII character set)*
`'AB'`	*invalid (more than one character)*

Certain special characters, such as the single quote `'` and the backslash `\`, may be represented according to the following table of escape sequences:

---

2 The behaviour of `@` is similar to the ⊥ (bottom) of some algebras.

newline	NL(LF)	'\n'
horizontal tab	HT	'\t'
backspace	BS	'\b'
carriage return	CR	'\r'
form feed	FF	'\f'
backslash	\	'\\'
single quote	'	'\''
double quote	"	'\"'
null	NUL	'\0'
bit pattern	*ddd*	'*ddd*'

The escape *ddd* consists of the backslash followed by a stream of octal digits which are taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash is ignored.

A.8.4 Floating point[3]

A floating point constant consists of an integer part, a decimal point, a fraction part, and an exponent part; where an exponent part is an `e` or `E`, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the `e` (`E`) and the exponent (not both) may be missing.

Note that a floating point constant may not contain any embedded blanks or special characters.

Some floating point constants are:

```
1.23   .23    0.23   1.     1.0    1.2e10       1.23e-15
```

but not

`1,000.00`	*comma not allowed*
`1 000.00`	*embedded space not allowed*
`1000`	*decimal point or exponential part needed*
`.e0`	*integer part or fractional part needed*
`-3.14159`	*this is a floating point expression, not a constant*

A.8.5 String

A string constant is a character sequence enclosed in double quotes:

```
"this is a string"
```

The backslash convention for representing special characters can also be used within a string. This makes it possible to represent the double quote and the escape character backslash \ within a string. It is **not** possible for a string to contain the `\0` (NUL) character since it is used to represent the end of string internally. Therefore,

```
"this is the end\0here we passed the end of string"
```

is equivalent to

```
"this is the end"
```

If *s* denotes a string expression, and *i* an index (an integer expression) then *s*`[`*i*`]` denotes the ith character in the string. (see also §A.9.1)

---

3 Floating point type was not implemented in the earliest version of the EDEN interpreter.

An individual character of a string can be accessed randomly by giving the index — an integer expression enclosed in `[]` — after the string expression. The first character is numbered 1 (not zero). For example, if

```
s = "abcdef";
```

then

`s[1]`	is the first character of `s`, i.e. `'a'`.
`s[`*i*`]`	is the ith character of `s`.

It is illegal to index beyond the current number of characters in a string. So `s[7]` is an error since `s` has only six characters.

The suffix operator # returns the current number of characters in the string. In the last example, `s#` is 6. For the empty string, `""#` is 0.

A.8.6 Pointer

Pointers are the addresses of objects in the memory space. The prefix operator & gets the address of an object. For example,

```
iptr = & int_variable;
cptr = & s[5];      /* if s is a string, cptr points to the 5th char of s */
```

The prefix operator * refers to the object which the pointer is pointing to. For example,

```
i = * iptr ;              /* i = int_variable; */
c = * cptr ;              /* c = s[5]; */
```

There are no pointer constants, and no pointer operations except the pointer equality checks.

A.8.7 List

The *list* is the only structured data type in EDEN. Data items of different types can be grouped to form a list using the list formation operator `[]` (see §A.10.3). Commas are used to separate the data. The whole list is considered as a single data item. For example, the list

```
[ 100, 'a', "string", [1,2,3] ]
```

holds four items: an integer (`100`), a character (`'a'`), a string (`"string"`) and a list (`[1,2,3]`).

There are no list constants in the language. The characteristics of the list data type are:

- Different typed data can be stored in the same list. For instance,

  ```
  L = [ 100, 'a', "string", [1,2,3] ];
  ```

  `L` holds four items: an integer (`100`), a character (`'a'`), a string (`"string"`) and a list (`[1,2,3]`).

- The individual items of a list can be accessed randomly by giving the index (an integer expression quoted by `[]`) after the list. The first item is numbered 1 (not zero). (See also §A.9.1.) For example,

`L[1]`	is the first item of `L`, i.e. `100` (in the previous example).
`L[4][2]`	is 2.

- It is illegal to index beyond the current number of items of a list. So `L[5]` is an error since `L` has only four items.

- The suffix operator # returns the current number of items in the list. In the last example, `L#` is 4, and `L[4]#` is 3. For the empty list, `[]#` is 0.
- List-related statements are: **`append`**, **`insert`**, **`delete`** and **`shift`** statements.
- There are no list constants.

A.8.8 Function

EDEN allows the user to define functions (or procedures). The syntax of defining a function is: (see §A.12 for the formal description)

`func` *identifier* { *para-alias*$_{opt}$ *local-var-declaration*$_{opt}$ *statement-list*$_{opt}$ }

- Notice that there is no parameter list after the identifier. The parameters form a list named `$` (dollar sign). Hence

  `$[1]` represents the first argument
  `$[2]` represents the second argument
  etc.

  For convenience, `$[`*n*`]` can be replaced by `$`*n*, where *n* is a decimal integer constant. The optional *para-alias* gives nicknames to the arguments. The syntax is:

  `para` *identifier-list* ;

- `$#` is the current number of arguments. List operations are applicable to `$`.
- All parameters are passed by value.
- Local variables must be declared at the beginning of the function body and are preceded by the keyword **`auto`** (means dynamic allocated). All local variables are RWV's. There is no need to declare the type of local variables because EDEN does the run-time type checking. The syntax is:

  `auto` *identifier-list* ;

  where *identifier-list* is a list of identifiers separated by commas.

- The **`return`** statement returns the value of the expression. The syntax is:

  `return` *expression*$_{opt}$;

  If the expression is omitted, `@` will be returned. Flowing off the end of a function is equivalent to return `@`.

- The value returned can be ignored by the caller. In this case, the function acts as a procedure. The keyword **`func`** can be replaced by **`proc`**. There are no differences between these two keywords. Thus they can be interchanged. The purpose of having another keyword is to self-comment the program. It is intended (not restricted) that **`func`** should be used for defining side-effect-free operators.

Examples:

```
func max    /* returns the max. value of its arguments */
{
   para m;                  /* m is the first argument $1 */
   auto i;                  /* declare local variables */

   for (i = 2; i <= $#; i=i+1) {
                            /* for the other arguments */
      if ($[i] > m)         /* the i^th argument */
         m = $[i];
   }
   return m;                /* returns max to the caller */
}
```

The above example defines a function named `max` which returns the maximum value of its arguments.

- To call a function, the arguments must be put in parentheses preceded by the function name. For example,

  ```
  MaxNumber = max(0,i,j,k);
  ```

  evaluates the maximum value of `0`, `i`, `j` and `k`, and stores the results in the RWV `MaxNumber`. The parentheses cannot be omitted even if no arguments are passed to the function.

- The function name, itself, denotes an entry point (a pointer) of the function code. It is a valid value (a pointer to a function) that can be used in an expression. Thus, if

  ```
  F = max;
  ```

  then the statement

  ```
  MaxNumber = F(1,3,2);
  ```

  assigns 3 to `MaxNumber`.

  Also a function can be put into a list. For example, if

  ```
  G = [ min, max ];
  ```

  where `min` is supposed to be a function that returns the minimum value of its arguments, then

  ```
  Num = G[n](1,3,2);
  ```

  `Num` will have the value `1` if `n` is 1 (`G[1]=min`), 3 if `n` is 2 (`G[2]=max`).

- Once an identifier is used as a name of a function/procedure/action, it cannot be re-used to designate a RWV/formula variable. This restriction is posed by the interpreter (not by the language) to prevent destroying a function definition accidently.

- There are some pre-defined functions, such as `write`, `writeln`, and `exit`. (see §A.17)

A.9 Expressions

The precedence of expression operators is the same as the order of the major sub-sections of §A.10, highest precedence first.

All integer overflows are ignored in the current implementation. Division by 0 causes a run-time error.

A.9.1 Lvalue

lvalue:
identifier
`$`
`$`*number*
lvalue `[` *expression* `]`
`*` *expression*
`` ` `` *expression* `` ` ``
`(` *lvalue* `)`

Identifier were discussed in §A.5.

The dollar sign `$` is the actual argument list of a function; `$`*number* is the *number*-th argument where number is a decimal integer constant.

A lvalue followed by an expression in square brackets is a lvalue. The intuitive meaning is that of a subscript. The lvalue must have type *string* or *list*. The expression must be of integer type.

The unary `*` operator means *indirection*: the expression must be a pointer, the result is an lvalue.

An expression enclosed in a pair `` ` `` (open quotes) is a lvalue where the expression must be of string type. The object, referred to by it, is the object having the name identical to the string. For instance, `` `"A"` `` is equivalent to the object `A`.

A parenthesized lvalue is a lvalue which is identical to the unadorned lvalue.

A.9.2 Primary expression

primary-expression:
lvalue
`(` *expression* `)`
primary-expression `[` *expression* `]`
primary-expression `(` *expression-list*$_{opt}$ `)`

A lvalue is an simple expression. The value of the object is returned.

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression.

A primary expression followed by an expression in square brackets is a primary expression. The intuitive meaning is that of subscript. The primary expression must has type *string* or *list*. See 3.8.1.

A function call is a primary expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the actual arguments to the function. The primary expression must be of type *function*. See 3.7.8.

In preparing for the call to a function, a copy is made of each actual parameter (even it is a string or a list); thus all argument-passing in EDEN is strictly by value. A function may change the values of its formal parameters, but these changes cannot affect the values of the actual parameters. On the other hand, it is possible to pass a pointer on the understanding that the function may change the value of the object to which the pointer points. The order of evaluation of arguments is undefined by the language. (Although the current implementation evaluates parameters from left to right, the user should not assume this fact.)

A.10 Operators

A.10.1 Arithmetic operators

Expressions with binary operators group left-to-right. The usual arithmetic conversions are performed.

arithmetic-expression:
expression + *expression*
expression − *expression*
expression * *expression*
expression / *expression*
expression % *expression*

The result of the + operator is the sum of the operands.
The result of the − operator is the difference of the operands.
The binary * operator indicates multiplication.
The binary / operator indicates division.
The binary % operator yields the remainder from the division of the first expression by the second.
The two operands must be of integral type (i.e. integers or characters).
All arithmetic operators are *strict*, i.e. they return @ if either operand is @.
If both operands of an arithmetic expression are of integral type, the result is always an integer. If any operand is a floating point, the result is a floating point (except for the % operator which takes integral operands only).

A.10.2 String operators

string-expression:
expression / / *expression*

The result of / / returned is a string which is the concatenation of the two operands which are of string or character type. If either of operand is @, the result will be @. (see pre-defined function `substr`)

A.10.3 List formation operator

list-formation-expression:
[*expression-list*$_{opt}$]

expression-list:
expression
expression , *expression-list*

The [] operator groups the values of the expressions into a list. The expression list is a list of expressions which separated by commas. If the expression list is omitted the list returned is the null list.

A.10.4 Unary operators

unary-expression:
− *expression*
! *expression*
& *lvalue*
expression #

The result of the unary − operator is the negative of its operand which is of integral type or floating point type. If the operand is of integral type, the result is of integer type. If the operand is @ the result is @. There is no unary + operator.

The result of the logical negation operator ! is 1 if the value of its operand is zero, 0 if the value of its operand is non-zero. The type of the result is an integer. It is applicable to any integral operand.

The result of the unary & operator is a pointer to the object referred to by the lvalue.

If the expression is of string type, the suffix # operator returns the number of characters in it. If the expression is of list type, # returns the length of the list. The result is always an integer.

A.10.5 Relational operators

The relational operators group left-to-right, but this fact is not very useful; `a<b<c` (`(a<b)<c`) does not mean what it seems to (`(a<b)` and `(b<c)`) in a normal mathematical expression.

relational-expression:
expression < *expression*
expression > *expression*
expression <= *expression*
expression >= *expression*

The operators < (less than), > (greater than), <= (less than or equal to) and >= (greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true. The type of the result is integer. The usual arithmetic conversions are performed.

Two strings may be compared. Single characters are compared from the left to the right according to their codes in the machine's character set. Note that a string is always terminated by `\0`, so the shortest string is considered the smaller. Strings are equal only if their lengths as well as their contents are identical.

It is an error if the two operands are of different types, but if either argument in an relational expression is `@` then the value is `@`. Lists and pointers cannot be compared using the relational operators.

A.10.6 Equality operators

equality-expression:
expression == *expression*
expression != *expression*

The == (equal to) and != (not equal to) operators are exactly analogous to the relational operators except for their lower precedence. (Thus `a<b == c<d` is 1 whenever `a<d` and `c<d` have the same truth-value).

Note that two lists can be compared for equality. Lists are equal only if their lengths as well as their contents are identical. Two pointers can be compared for equality. Pointers are equal only if they points to the same object.

A.10.7 Logical AND operator

logical-and-expression:
expression && *expression*

The && operator 1 if both its operands are non-zero, 0 otherwise. && guarantees left-to-right evaluation; moreover the second operand is not evaluated if the first operand is 0.

A.10.8 Logical OR operator

logical-or-expression:
expression || *expression*

The || operator returns 1 if either of its operands is non-zero, 0 otherwise. || guarantees left-to-right evaluation; moreover the second operand is not evaluated if the first operand is non-zero.

A.10.9 Conditional operator

conditional-expression:
expression ? *expression* : *expression*

Conditional expressions group right-to-left. The first expression is evaluated and if it is non-zero, the result is the value of the second expression, otherwise that of third expression. The second and third expressions need not have same type; moreover only one of them is evaluated.

A.10.10 Assignment operators

assignment-expression:
lvalue = *expression*
lvalue += *expression*
lvalue -= *expression*
++ *lvalue*
lvalue ++
-- *lvalue*
lvalue --

There are three binary assignment operators, =, +=, and -=, all of which group right-to-left. All require an lvalue as their left operand. The right operand is evaluated and the value stored in the left operand after the assignment has taken place.

In the simple assignment with =, the value of the expression replaces that of the object referred to by the lvalue.

The behaviour of an expression of the form `E1` *op*= `E2` may be inferred by taking it as equivalent to `E1 = E1` *op* `(E2)`; however, `E1` is evaluated only once. Both of the operands must be of integral or floating point types.

The object referred to by the lvalue operand of prefix ++ is incremented. The value is the new value of the operand, but is not an lvalue. The expression `++x` is equivalent to `x+=1`.

When postfix ++ is applied to an lvalue the result is the value of the object referred to by the lvalue. After the result is noted, the object is incremented in the same manner as for the prefix ++ operator. The type of the result is the same as the type of the lvalue expression.

The lvalue operands of the prefix and postfix -- is decremented analogously to the prefix and postfix ++ operators respectively.

All the objects referred to by the lvalues of these assignment operators must be read/write variables, not formula variables.

A.10.11 Precedence and order of evaluation

The table below summarizes the rules for precedence and associativity of all operators. Operators on the same line have the same precedence; rows are in order of decreasing precedence, so, for example, `*`, `/`, and `%` all have the same precedence which is higher than that of + and -.

Operator	*Associativity*
`()` `[]` `` ` ` ``	left to right
`!` `++` `--` `-` `*` `&` `#`	right to left
`*` `/` `%`	left to right

+ - //	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
?:	right to left
= += -=	right to left

A.11 Procedural Statements

Procedural Statements are executable statements. The statements are executed in sequence except as indicated.

A.11.1 Expression Statements

Most statements are expression statements, which have the form:

expression ;

Usually expression statements are assignments or function/procedure calls. For instances:

```
I = 1;                      /* assignment statement*/
J++;                        /* another form of assignment */
DoSomething(I, J);          /* procedure call */
```

A.11.2 Insert Statement

The statement

insert *lvalue , expression-1 , expression-2 ;*

inserts a value evaluated from *expression-2* into the list object referred to by *lvalue* at the position *expression-1*; *expression-1* must be of integral type and cannot be smaller than 1 or be greater than the current number of items of the list plus one. The list object referred to by *lvalue* must be a read/write variable.

A.11.3 Append Statement

The statement

append *lvalue , expression ;*

appends a value evaluated from *expression* to the end of the list referred to by *lvalue*. It is equivalent to do

insert *lvalue , lvalue* # + 1 *, expression ;*

but the lvalue expression is evaluated only once.

A.11.4 Delete Statement

The statement

delete *lvalue* , *expression* ;

deletes a value from the list object referred to by lvalue at the position evaluated from *expression*; *expression* must be of integral type and cannot be smaller than 1 or be greater than the current number of items of the list; whence the list cannot be a null list. The list object referred to by *lvalue* must be a read/write variable.

A.11.5 Shift Statement

The syntax of the shift statement is:

shift $lvalue_{opt}$;

The shift statement deletes the first value from the list referred to by *lvalue*; whence the list cannot be a null list. If *lvalue* is omitted, the argument list `$` is assumed; hence it can only be used within a function body. The equivalent delete statements of the shift statements are:

```
delete $ , 1 ;          /* shift; */
delete lvalue , 1 ;     /* shift lvalue; */
```

A.11.6 Compound Statement

So that several statements can be used where one is expected, the compound statement is provided:

compound-statement:
{ $statement\text{-}list_{opt}$ }

statement-list:
statement
statement statement-list

A.11.7 Condition Statement

The two forms of the conditional statement are

if (*expression*) *statement*
if (*expression*) *statement* **else** *statement*

In both cases the expression is evaluated and if it is non-zero, the first sub-statement is executed. In the second case the second sub-statement is executed if the expression is 0. As usual the "else" ambiguity is resolved by connecting an `else` with the last encountered `else`-less if.

A.11.8 While Statement

The while statement has the form

while (*expression*) *statement*

The sub-statement is executed repeatedly so long as the value of the expression remains non-zero. The test takes place before each execution of the statement.

A.11.9 Do Statement

The do statement has the form

do *statement* **while** (*expression*) ;

The sub-statement is executed repeatedly until the value of the expression becomes zero. The test takes place after each execution of the statement.

A.11.10 For Statement

The for statement has the form

for (*expression-1*$_{opt}$; *expression-2*$_{opt}$; *expression-3*$_{opt}$) *statement*

This statement is equivalent to

```
expression-1 ;
while ( expression-2 ) {
        statement
        expression-3 ;
}
```

Thus the first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression becomes 0; the third expression often specifies an incrementation which is performed after each iteration.

Any or all of the expressions may be dropped. A missing *expression-2* makes the implied `while` clause equivalent to `while(1);` other missing expressions are simply dropped from the expansion above.

A.11.11 Switch Statement

The switch statement causes control to be transferred to one of several statements depending on the value of an expression. It has the form

switch (*expression*) *statement*

The statement is typically compound. Any statement within the statement may be labelled with one or more case prefixes as follows:

case *constant* : *statement*

There may also be prefix of the form:

default : *statement*

When the switch statement is executed, its expression is evaluated and compared with each case constant. If one of the case constants is equal to the value of the expression, control is passed to the statement following the matched **case** prefix. If no case constant matches the expression, and there is a **default** prefix, control passes to the prefixed statement. If no case matches and if there is no **default** then none of the statements in the **switch** is executed.

A.11.12 Break Statement

The statement

break ;

causes termination of the smallest enclosing **while**, **do**, **for**, or **switch** statement; control passes to the statement following the terminated statement.

A.11.13 Continue Statement

The statement

continue ;

causes control to pass to the loop-continuation portion of the smallest enclosing **while**, **do**, or **for** statement; that is to the end of the loop.

A.11.14 Return Statement

A function returns to its caller by means of the **return** statement, which has one of the forms:

return *expression*$_{opt}$;

In the optional expression is omitted the value returned is `@`. Otherwise, the value of the expression is returned to the caller of the function. Flowing off the end of a function is equivalent to return `@`.

A.11.15 Null Statement

The null statement has the form:

;

A null statement is useful to carry a label just before the } of a compound statement, for example:

```
switch (...) { ... case 1: ; }
```

or to supply a null body to a looping statement such as **while**, for example:

```
while (1) ;       /* an infinite waiting loop */
```

A.12 User-defined Functions

Function definitions have the form:

function-definition:
function-declarator function-body

function-declarator:
func *identifier*
proc *identifier*

function-body:
{ *para-alias*$_{opt}$ *local-var-decl*$_{opt}$ *statement-list*$_{opt}$ }

para-alias:
para *identifier-list*$_{opt}$;

local-var-decl:
auto *identifier-list*$_{opt}$;

identifier-list:
identifier
identifier , *identifier-list*

The identifier is declared to be a function using the keywords **func** or **proc**. There is no difference between **func** and **proc**. The word **proc** is more meaningful when the function serves as a procedure, i.e. a function which does not return a value.

All parameters are passed by value. The actual parameters forms a list called `$` (dollar sign). The first arguments is `$[1]`, the second argument is `$[2]`, and so forth. For convenience, `$[1]`, `$[2]`, ... can be replaced by `$1`, `$2`, ... **para** gives nicknames to the parameters. The first identifier in *para-alias* matches `$1` and so forth. The number of identifiers does not required to

match the number of actual parameters. $n can be referenced by either $n, $[n] or the nickname given.

See also §A.7.2 and §A.8.8.

A.13 Definitive Statements

A.13.1 Formula Definition

A formula definition has the form:

formula-definition:
identifier **is** *expression* ;

It is normally expected that the operators appearing in the *expression* are "pure" functions, i.e. are without side-effect. The interpreter does not allow the use of assignment operators in the *expression*, but does not otherwise detect the use of "impure" functions. It is not advisable to use such functions.

In addition, a formula variable cannot be (directly or indirectly) cyclically defined. For example,

```
f is f + 1;
```

is an error because f is defined in terms of itself, and not logically meaningful. The definitions:

```
i is j;
j is i;
```

are legal definitions, but conflict with the restriction. The EDEN interpreter rejects the second definition, and produces the error message:

```
j : CYCLIC DEF : ABORTED near line ...
```

The examples below are valid definitions of formula variables (assuming they are not cyclically defined):

```
f is a + b;
M is max(a,b,c);
```

The EDEN interpreter controls all the evaluations of formulae. The order of evaluation is not specified by the language, i.e. can happen in any order. One version of EDEN interpreter may evaluate the formula of f before that of M, whenever the value of a, for instance, is changed, but some other versions may do it in the reverse order. A concurrent EDEN system (if it existed) might do both evaluations in parallel. Therefore, the user should not make any assumption on the order of evaluation.

A.13.2 Action Specification

An action specification has the form:

action-specification:
function-declarator dependency-list function-body

dependency-list:
: *identifier-list*

The *function declarator* and *function body* are the same as those of function definition (see §A.12). An identifier is declared to be an action using the keywords **proc** or **func**. There is no difference between **proc** and **func**; but the user is recommended to use **proc** because an action usually serves as a procedure.

The dependency list is a list of comma-separated identifiers preceded by a colon.

An action is normally a procedure invoked automatically by the system when any object specified in the dependency list is changed. No parameters will be passed to the procedure; so the argument list $ is always empty when called by the system. All values returned by the actions are ignored by the system.

The order of invoking actions is controlled by the EDEN interpreter, but the user can invoke an action explicitly by giving a pair of parenthesis (a null argument list) after the action name (exactly like a procedure/function call). This is useful for debugging.

Notice that if the identifier list is omitted in the dependency list the action is just an ordinary procedure.

The use of actions (and the use of "impure" functions as operators in formulae) leads to procedural activity outside the user's direct control when expressions are evaluated. The user should make no assumptions about the order in which such procedural actions are performed. For instance, no two actions that can be invoked in the same time should write to the same RWV. EDEN does not provide any facility to detect or prevent such "interference" between actions.

A.14 Miscellaneous Commands

A.14.1 Dependency Link

An alternative way of defining an action specification is to declare the action as a procedure/function. Then the dependency link can be added using the command:

dependency-link:
identifier ~> [*identifier-list*] ;

where each identifier in the *identifier list* refers to an action (procedure) that depends on the value of the object referred to by *identifier*. So the action:

```
proc p : a, b, c { /* action body */ }
```

is equivalent to

```
a ~> [ p ] ;
b ~> [ p ] ;
c ~> [ p ] ;
```

Note that the *dependency link* command adds dependency linkage to actions. If the user wants to remove a particular linkage, the whole action must be re-specified. For example, to remove c from the specification of action p above, p must be re-defined as:

```
proc p : a, b { /* action body */ }
```

A.14.2 Query

The query command has the form:

? *lvalue* ;

The query command prints the definition of the object referred to by *lvalue* on the *stdout* of the UNIX environment.

The format of printing a RWV is:

value
RWV-name ~> [*identifier-list*] ;

The format of printing a formula variable is:

formula-variable-name **is** *expression* ;
formula-variable-name ~> [*identifier-list*] ;

The format of printing a function definition is:

func *function-name*
{ *function-body* }
function-name ~> [*identifier-list*] ;

where **proc** may be printed instead of **func** depends on which word was used by the user.

The format of printing an action specification is:

proc *action-name* : *identifier-list*
{ *action-body* }
action-name ~> [*identifier-list*] ;

where **func** may be printed instead of **proc** depends on which word was used by the user.

The query command is an ad hoc feature in the language and contains bugs in the current implementation of EDEN interpreters; for instance,

```
? L[1];
```

prints the definition of `L` instead of `L[1]`.

However it is the only way of inspecting the current definition and data dependency relations associated with an object.

A.15 Syntax Summary

A.15.1 Expressions

expression:
primary-expression
- expression
! expression
& lvalue
expression #
++ lvalue
lvalue ++
-- lvalue
lvalue --
[*expression-list*$_{opt}$]
expression binop expression
expression ? expression : expression
lvalue asgnop expression

primary-expression:
lvalue
(*expression*)
primary-expression [*expression*]
primary-expression (*expression-list*$_{opt}$)

lvalue:
identifier
$
$*number*
lvalue [*expression*]
* *expression*
` *expression* `
(*lvalue*)

expression-list:
expression
expression , *expression-list*

The primary-expression operators

```
()      []      ``
```

have highest priority and group left-to-right. The unary operators

```
*       &       -       !       #       ++      --
```

have priority below the primary operators but higher than any binary operator, and group right-to-left.

Binary priority decreasing as indicated below.

binop:

```
*       /       %
+       -       //
>       <       >=      <=
==      !=
&&
||
```

The conditional operator

```
?:
```

has priority lower than the binary operators, and groups right-to-left.

Assignment operators all have the same priority, and all group right-to-left.

asgnop:

```
=       +=      -=
```

A.15.2 Statements

statement:
expression ;
function-definition
formula-definition
action-specification
dependency-link
query-command
compound-statement
insert *lvalue , expression-1 , expression-2 ;*
append *lvalue , expression ;*
delete *lvalue , expression ;*
shift $lvalue_{opt}$;
if (*expression*) *statement*
if (*expression*) *statement* **else** *statement*
while (*expression*) *statement*
do *statement* **while** (*expression*) ;
for ($expression_{opt}$; $expression_{opt}$; $expression_{opt}$) *statement*
switch (*expression*) *statement*
case *constant* : *statement*
default : *statement*
break ;
continue ;
return $expression_{opt}$;
;

compound-statement:
{ $statement\text{-}list_{opt}$ }

statement-list:
statement
statement statement-list

A.15.3 Function Definition

function-definition:
function-declarator function-body

function-declarator:
func *identifier*
proc *identifier*

function-body:
{ $local\text{-}var\text{-}decl_{opt}$ $statement\text{-}list_{opt}$ }

local-var-decl:
auto $identifier\text{-}list_{opt}$;

identifier-list:
identifier
identifier , identifier-list

A.15.4 Dependency Link

dependency-link:
identifier ~> [*identifier*] ;

A.15.5 Query Command

query-command:
? *lvalue* ;

A.16 Pre-defined Variables

The following is a list of pre-defined read/write variables. The user should not alter the value of these variables, except `autocalc`.

`stdin`
`stdout`
`stderr`

These three variables are pre-defined file pointers equivalent to those file pointers in C. All of them are of integer type.

`autocalc`

If the value of `autocalc` is set to 0, the mechanism of auto-recalculate of formula definitions will be switched off. In such a situation, formulae and actions will not be recalculated or invoked when the values of those variables on which they are dependant are changed. However, when `autocalc` is set to a non-zero value, the auto-recalculation mechanism will be back in action. By default, `autocalc` is set to 1.

`calc_list`

In the earliest version of EDEN interpreter, the pointers pointing to the formula variable and action specifications are queued in the list variable `calc_list`. The interpreter would not update or invoke those queued formula variables and actions when `autocalc` is switched "off". The following action specification can update these queued formulae and actions automatically when `autocalc` is on again:

```
proc _autocalc : autocalc
{
  while(calc_list#) {
    *calc_list[1];    /* update formula var's */
    shift calc_list; /* remove from list */
  }
}
```

Note that this action will only invoked when the auto-calculate mechanism is on (i.e. `autocalc` is non-zero). This action forces the formula variables to be updated by simply evaluating them (the interpreter always updates the formula variables if they are evaluated). Although this action does not invoke queued actions, the updating of the formula variables will cause the related actions to be invoked.

In the new version of EDEN interpreter, all queued formulae and actions would be updated and invoked as soon as the auto-recalculate mechanism is switched on. The references to the formula variables and action specifications are stored in an internal format and hence the variable `calc_list` is not needed. Thus the user cannot alter the list. To inspect the formula variable and action specification queues while `autocalc` is "off", the user can call the pre-defined functions, `formula_list()` and `action_list()`, which transform the internal information into EDEN's list type. See the next section for the description of these two functions.

A.17 Pre-defined Functions

A number of functions are pre-defined by the EDEN interpreter. These pre-defined functions cannot be re-defined by the users to prevent ruining their definition accidently. The following is a list of pre-defined functions. The output of the examples are in *`italic courier`* typeface.

`debug(`*status*`)`
int *status*

If *status* is a non-zero integer, `debug` turns on the debugging mode. When the debugging mode is on the interpreter prints out some information showing the status of the machine. Default: off (`0`). For example,

```
debug(1);    /* turn on debugging mode */
```

It is a subroutine for developing the interpreter. It is available to the user only if the interpreter was compiled with the flag `-DDEBUG`.

`write(...)`
`writeln(...)`

`write` and `writeln` print the arguments on the *stdout* (standard output file). `writeln` appends a newline `\n` at the end. For example,

```
for (i = 1; i <= 10; i++) write(i, ' ');
1 2 3 4 5 6 7 8 9 10

writeln("sum of ",1,'+',2," = ",1+2);
sum of 1+2 = 3
```

`substr(`*string, from, to*`)`
string *string*
int *from, to*

`substr` returns a substring of *string*. *from* and *to* are the starting and ending positions respectively; they are integer values. If *to* is greater than *from*, the null string will be returned. *from* must not be small than 1. For example,

```
substr("1234567890", 4, 8)
```

will evaluated to *"45678"*.

`strcat(`*string,...*`)`
string *string,...*

`strcat` returns the string of the concatenation of its arguments. All arguments must be of string or character types. Characters are considered as a string of length 1. If it is called with no arguments, the null string will be returned. For example,

```
s = strcat("Garden", " of ", "Eden", '.');
```

`s` has the value *"Garden of Eden."*.

`execute(`*string*`)`
string *string*

`execute` executes a string as a program. It returns `0` if no errors occurred in the execution of the string, non-zero otherwise. The execution terminates as soon as it encounter a syntax error or run-time error. Note that it is not a macro; the string must be some valid and complete EDEN statements. Also this function can be executed within a function body. For example,

```
proc reset_F {execute("F is A+B;");}
```

Every time `reset_F` is called, `F` will be defined to be a formula variable contains the expression `"A+B"` unless there are run-time errors, such as the recursive definition conflict.

`include` (*filename*)
string *filename*

`include` works in the same way as `execute` except it takes a file as a program; the file is specified by the first argument which is a string expression. If the execution is successful, include returns `0`. The execution terminates as soon as it encounter a syntax error or run-time error. The file name and line number where the error occurs will be reported by the interpreter. For example,

```
include("utility");
```

executes the file called `utility`.

`apply` (*function, list*)
func *function*
list *list*

`apply` calls the function specified in the first argument with the second argument as its actual argument (i.e. `$`). Thus the first and second arguments must be of type function and list respectively. For example,

```
apply(writeln, [1,2,3]);
```

is equivalent to call

```
writeln(1,2,3);
```

`exit` (*status*)
int *status*

`exit` terminates the program and returns *status*, an integer, to the system. If status is omitted, `0` is assumed.

`nameof` (*pointer*)
pointer *pointer*

`nameof` returns the name of object to which the first argument (of pointer type) points. The function returns a string. This function is an ad hoc function. Bugs: If the pointer points to a character of a string object or an item of a list object, the string or the list object name is returned. For examples,

Example	*Value Returned*
`nameof(&Object)`	*`"Object"`*
`nameof(&A_String[1])`	*`"A_String"`*
`nameof(&A_List[5])`	*`"A_List"`*

`formula_list()`

`formula_list` returns the current list of addresses of formula variables queued. This function is useful to inspect the internal queue of formulae when the `autocalc` is "off". See §A.16 for the description of `autocalc`. This function is implemented in the later versions of the EDEN interpreter.

`action_list()`

`action_list` returns the current list of addresses of action specifications queued. This function is useful to inspect the internal queue of actions when the `autocalc` is "off". See §A.16 for the description of `autocalc`. This function is implemented in the later versions of the EDEN interpreter.

`symboltable()`

`symboltable` returns the current internal symbol table of the interpreter as a list. Each symbol entry is of the form:

[*name, type, text, by-list, to-list*]

where

name	the object name (a string)
type	the type code (an integer) of the object. The code denotes whether the object is a RWV, function definition, formula variable, or action specification.
text	if the object is a function definition, formula variable, or action specification, then *text* stores the definitions in textual form.
by-list	a list of addresses of objects by which this object is referenced.
to-list	a list of addresses of objects to which this object refers.

For example, if the following definitions are defined

```
...
a is b + c;
b = 3;
f is b * g;
...
```

and,

```
S = symboltable());
```

then, `S` is:

```
[ ... [ "a", 263, "b + c;", [], ["b","c"] ],
    [ "b", 275, "", ["a"], [] ], ... ]
```

This function is useful to inspect the internal queue of actions when the `autocalc` is "off". See §A.16 for the description of `autocalc`. This function is implemented in the new versions of the EDEN interpreter.

`eager()`

eagerly evaluate/execute all queued definitions/actions despite of the status of `autocalc`.

`pack(...)`

allocate a continuous space on the heap to store the packed data. The values are packed in a machine-dependent fashion, so it is not portable. Data is packed as C type:-

integer	is considered as	int	(4 bytes)
real		float	(4)
character		char	(1)
string/pointer/function/etc		char *	(4)

`pack` returns the address of the beginning of memory block as an integer. Example:

```
x = [0.0, 100.0, 100.0, 0.0];     /* x-coordinates */
y = [0.0, 0.0, 100.0, 100.0];     /* y-coordinates */
polygon_abs_2(pack(x), pack(y), 4);
                    /* will draw a filled square */
/* in fact, you could do:
      polygon_abs_2(pack([0.0, 100.0, 100.0, 0.0]),
                    pack([0.0, 0.0, 100.0, 100.0]),
                   4);
*/

move_abs_2(0.0, 0.0);
polyline_abs_2(pack(x), pack(y), 4);
                    /* will draw a hollow square */

/* c.f. SunCore Reference Manual */
```

Bugs: It is very ad hoc, and not universal, i.e. cannot handle any struct. It converts real numbers into "float"s instead of "double"s (so we can call Suncore polyline & polygon primitives). There should be an "unpack" function.

`getenv(env)`
string *env*
returns the string of the environment variable *env*. See getenv(3).

`putenv(env)`
string *env*
sets the environment variable. *env* should have the form: `"name=value"`. See putenv(3).

`error(err_msg)`
string *err_msg*
generates an EDEN error & print the error message *err_msg*.

`error_no()`
returns the last system (not EDEN) error number (an integer).

`touch(vp1, vp2, vp3, ...)`
pointer *vp1, vp2, vp3, ...*
vp1, ... are pointers to variables. `touch` puts the targets of variables, pointed to by *vp1, vp2, vp3, ...*, to the evaluation queue.

The followings are interprocessing communication routines. Note that Sun3 workstation has problems to use message queue, so these functions can only work on Sun4 (marked by *).

`backgnd(path, cmd, arg1, arg2, ...)`
string *path, cmd, arg1, arg2, ...*
executes a process at background named by *path*. `backgnd` returns the process id (–1 if fail). Bug: `backgnd` may returns a +ve id even if it can't execute the command.

`pipe(path, cmd, arg1, arg2, ...)`
string *path, cmd, arg1, arg2, ...*
pipes output (stdout) to process named by *path*. `pipe` returns the process id (–1 if fail). Bug: `pipe` may returns a +ve id even if it can't execute the command.

`get_msgq(key, flag)` *
int *key, flag*
gets a message queue using *key*. flag denotes the permission and options (see msgget(2)). It returns the message queue id (integer), –1 if fail.

`remove_msgq(msqid)` *
int *msqid*
removes a message queue whose id is *msgid*. It returns –1 if fail. (see msgctl(2))

`send_msg(msqid, [ msg_type, msg_text ], flag)` *
int *msqid, msg_type, flag*
string *msg_text*
sends a message (text string) to message queue *msqid*. *msg_type* denotes the message type. *msg_text* is a string (terminated by \0, i.e. at least having length 1).
flag = 0 for wait
04000 (octal) for no wait
(see msgsnd(2)).
It returns -1 if fail.

`receive_msg(msqid, msg_type, flag)` *
int *msqid, msg_type, flag*
receives message of *msg_type* from message queue *msqid*. flag: c.f. send_msg & msgsnd(2). It returns @ if fail, else `[m_type, m_text]` where *m_type* is the actual message type received and *m_text* is the text string received.

The following functions, `type`, `int`, `char`, and `str`, are type conversion functions.

`type (`*data*`)`

It returns the type of data as a string:

Data Type	*String Returned*
@ "undefined"	`"@"`
integer	`"int"`
character	`"char"`
string	`"string"`
floating point	`"float"`
list	`"list"`
function	`"func"`
procedure	`"proc"`
pre-defined function	`"builtin"`
pre-binded C function	`"C-func"`

For example, `type(1)` gives *`"int"`*.

`int (`*data*`)`

If *data* is an integer, the same value will be returned. If it is a character, the code of the character will be returned. If it is a string, the string will be assumed as a stream of digits, and these digits are converted into an integer which will be returned. If it is a floating point, it will be truncated into an integer before it is returned. If it is a pointer, the address is casted to an integer which will be returned. The function returns `@` otherwise.

`char (`*data*`)`

If *data* is a character, the same character will be returned. If it is an integer, the character having the code equals to the integer will be returned. If it is a string, its first character will be returned. If it is a floating point, it will be truncated into an integer and then cast to a character before it is returned. The function returns `@` otherwise.

`str (`*data*`)`

If *data* is a string, the same string will be returned. If it is `@`, the string `"@"` will be returned. If it is a character, the character will be converted into a string and returned. If it is an integer or a floating point, the string of digits corresponding to the value will be returned. The function returns `@` otherwise.

`float (`*data*`)`

If *data* is a floating point, the same value will be returned. If it is an integer, the floating point having the same value will be returned. If it is a character, the floating point having the same value as the code of the character will be returned. If it is a string, the string will be assumed as a stream of digits, and these digits are converted into a floating point which will be returned. If it is a pointer, it will be casted into an integer and then converted to a floating point before it is returned. The function returns `@` otherwise.

The following list of functions are pre-binded C functions. Users should refer to their own reference manual.

Function	*Description*
`fclose`	close a file
`fgetc`	get a character from an opened file
`fgets`	get a string from a file
`fopen`	open a file
`fprintf`	print to a file
`fscanf`	formated read data from a file
`gets`	get a string from *stdin*
`pclose`	close a pipe
`popen`	open a pipe
`putw`	put a machine word to a file

`setbuf`	set the buffer size of a file
`sprintf`	print to a string
`sscanf`	formated read data from a string
`system`	execute command in a sub-shell
`ungetc`	unget a character

Math Lib:

`sin`	`cos`	`tan`	`asin`	`acos`	`atan`
`atan2`	`sqrt`	`pow`	`log`	`log2`	`log10`
`exp`	`exp2`	`exp10`			

Different releases of the EDEN interpreter support different C function packages, such as CURSES (standard UNIX windowing package) and **SunCore** (standard SUN workstation graphics package). The user should consult to the release notes and the packages' reference manual for the descriptions of the functions.

Appendix B
Celsius-Fahrenheit Convertor

This appendix lists the entire program of a Celsius-Fahrenheit convertor described in §4.2.3.

Celsius-Fahrenheit Convertor Main Program	B-1
macro.e	B-2
definition.e	B-3
Macro Expanded Listing	B-5

Celsius-Fahrenheit Convertor Main Program

```
include("macro.e");
include("definition.e");

new_op ("ADD", [ r_is_a_plus_b,          /* fa */
                 r_is_a_plus_b,          /* fb */
                 b_is_r_minus_a          /* fr */
               ]
       );

new_op ("MULT",[ r_is_a_times_b,         /* fa */
                 r_is_a_times_b,         /* fb */
                 b_is_r_div_a            /* fr */
               ]
       );

set(ADD[a],  32);
set(MULT[a], 1.8);

connect("P", MULT[b], "C"    );
connect("Q", ADD[b],  MULT[r]);
connect("R", "F",     ADD[r] );
```

macro.e

macro

```
/*
**    macro       -- an EDEN utility function
**
**    Usage:      string = macro( macro_string , real_strings ... )
**
**    Description:
**          macro() will take the 1st argument to be a macro string
**          and expand '?n', where n is a digit 1-9, by the n-th
**          arguments followed.
**
**          '??' produces a single '?'.
**
**    Example:
**                macro("?1 ?2 ?3 ?2 ?1","A","B","C");
**          gives:
**                "A B C B A"
*/

func macro                      /* $1=macro string; $2..n = real string */
{
      auto i, j, l, m, n, c, s;

      s = "";
      l = (m = $1)#;      /* m = macro string; l = length of string  */
      shift;              /* remove macros from arguments, so ?i = $i */
      i = 1;
      while (i <= l) {
            for (j = i; j <= l && m[j] != '?'; j++); /* skip to '?' */
            if (i != j) s = s // substr(m, i, j - 1);
            if (j <= l) {             /* i.e. '?' */
                  j++;
                  n = (c = (j > l) ? '?' : m[j]) - '0';
                  s = s // ((1 <= n && n <= $#) ? $[n] : c);
            }
            i = j + 1;
      }
      return s;                     /* s is the expanded string */
}
```

definition.e

```
a = 1;      /* indices of operator variables */
b = 2;
r = 3;
```

set

```
/*************************************************************************
**    set("variable_name", value);                                        **
**      set the variable, whose name (a string ) is given by 1st arg,     **
**      to value given in 2nd arg.                                        **
*************************************************************************/
proc set { `$1` = $2; }

/*************************************************************************
**    action functions                                                    **
*************************************************************************/
```

r_is_a_plus_b

```
proc r_is_a_plus_b  { `$1[r]` = `$1[a]` + `$1[b]`; }
```

b_is_r_minus_a

```
proc b_is_r_minus_a { `$1[b]` = `$1[r]` - `$1[a]`; }
```

r_is_a_times_b

```
proc r_is_a_times_b { `$1[r]` = `$1[a]` * `$1[b]`; }
```

b_is_r_div_a

```
proc b_is_r_div_a   { `$1[b]` = `$1[r]` / `$1[a]`; }
```

define_action

```
/*************************************************************************
**      define_action(action_name, Vi, SV, fV, V)                       **
**          Create an action with the following parameters              **
**                                                                      **
**          action_name = name of action                                **
**          Vi = name of variable on which action depends               **
**          SV = control flag name                                      **
**          fV = action function                                        **
**          V  = variable name holds a list of constraint vars          **
**                                                                      **
**          This procedure is called by new_op() and connect()          **
*************************************************************************/
proc define_action      /* action_name, Vi, SV, fV, V */
{
  execute(
    macro(
      "proc ?1:?2{if(?3==1)?3=0;else{?3=1;?4(?5);}}",
      $1,   /* action_name */
      $2,   /* variable on which action depends */
      $3,   /* control flag name */
      $4,   /* function name */
      $5    /* variable name which holds a list of constraint vars */
    )
  );
}
```

new_op

```
/*************************************************************************
**      new_op("op_name", [fa, fb, fr])                                  **
**          Creates a new constraint operator, named by 1st arg          **
**          e.g. new_op("A", [Fa,Fb,Fr]) gives                           **
**              A = ["A_a", "A_b", "A_r"];                               **
**              f_A = [Fa, Fb, Fr];                                      **
**          where Fa,Fb,Fr must be procedures                            **
**          and 3 actions, namely                                        **
**              "action_A_a", "action_A_b", "action_A_r"                 **
**          with Fa, Fb, Fr as their action functions respectively       **
*************************************************************************/
proc new_op                          /* name, [fa, fb, fr] */
{
  auto    V, SV, fV, Va, Vb, Vr;

  V = $1;                            /* V  = op name                */
  set((fV = "f_"//V), $2);           /* fV = constraint functions */
  set((SV = "S_"//V), 0);            /* SV = control flag name    */
  set(V, [ (Va = V//"_a"), (Vb = V//"_b"), (Vr = V//"_r") ]);
  define_action("action_"//Va, Va, SV, fV//"[a]", V);
  define_action("action_"//Vb, Vb, SV, fV//"[b]", V);
  define_action("action_"//Vr, Vr, SV, fV//"[r]", V);
}
```

connect

```
/*************************************************************************
**      connect(connector_name, Vx, Vy)                                  **
**      Creates a new connector operator named by 1st arg                **
**      e.g. connect("P","Vx","Vy") gives 2 actions namely               **
**           "action_P_x:Vx" and "action_P_y:Vy"                         **
**           where P is the connector name                               **
*************************************************************************/
proc connect                  /* connector_name, Vx, Vy */
{
  execute(
    macro(
      "proc ?1_x {}           /* remove any previous connection */
       proc ?1_y {}
       proc ?1_x: ?2 { if (?2!=?3) ?3=?2; }    /* new connection */
       proc ?1_y: ?3 { if (?2!=?3) ?2=?3; }",
      "action_"//$1, $2, $3
    )
  );
}
```

Macro Expanded Listing

```
a = 1;       /* indices of operator variables */
b = 2;
r = 3;

proc r_is_a_plus_b  { `$1[r]` = `$1[a]` + `$1[b]`; }
proc b_is_r_minus_a { `$1[b]` = `$1[r]` - `$1[a]`; }
proc r_is_a_times_b { `$1[r]` = `$1[a]` * `$1[b]`; }
proc b_is_r_div_a   { `$1[b]` = `$1[r]` / `$1[a]`; }
```

ADD

```
f_ADD = [ r_is_a_plus_b, r_is_a_plus_b, b_is_r_minus_a ];
S_ADD = 0;
ADD   = [ "ADD_a", "ADD_b", "ADD_r" ];
proc action_ADD_a: ADD_a
{
      if (S_ADD == 1) S_ADD = 1;
      else { S_ADD = 0; f_ADD[a](ADD); }
}
proc action_ADD_b: ADD_b
{
      if (S_ADD == 1) S_ADD = 1;
      else { S_ADD = 0; f_ADD[b](ADD); }
}
proc action_ADD_r: ADD_r
{
      if (S_ADD == 1) S_ADD = 1;
      else { S_ADD = 0; f_ADD[r](ADD); }
}
```

MULT

```
f_MULT = [ r_is_a_times_b, r_is_a_times_b, b_is_r_div_a ];
S_MULT = 0;
MULT   = [ "MULT_a", "MULT_b", "MULT_r" ];
proc action_MULT_a: MULT_a
{
      if (S_MULT == 1) S_MULT = 1;
      else { S_MULT = 0; f_MULT[a](MULT); }
}
proc action_MULT_b: MULT_b
{
      if (S_MULT == 1) S_MULT = 1;
      else { S_MULT = 0; f_MULT[b](MULT); }
}
proc action_MULT_r: MULT_r
{
      if (S_MULT == 1) S_MULT = 1;
      else { S_MULT = 0; f_MULT[r](MULT); }
}

`ADD[a]` = 32;                /* set(ADD[a], 32)
                                 i.e.   ADD_a = 32;      */
`MULT[a]` = 1.8;              /* set(MULT[a], 1.8)
                                 i.e.   MULT_a = 1.8     */
```

P

```
proc action_P_x {}              /* remove any previous connection */
proc action_P_y {}
proc action_P_x: MULT_b
      { if (MULT_b != C) C = MULT_b; }          /* new connection */
proc action_P_y: C
      { if (MULT_b != C) MULT_b = C; }
```

Q

```
proc action_Q_x {}              /* remove any previous connection */
proc action_Q_y {}
proc action_Q_x: ADD_b
      { if (ADD_b != MULT_r) MULT_r = ADD_b; }  /* new connection */
proc action_Q_y: MULT_r
      { if (ADD_b != MULT_r) ADD_b = MULT_r; }
```

R

```
proc action_R_x {}              /* remove any previous connection */
proc action_R_y {}
proc action_R_x: F
      { if (F != ADD_r) ADD_r = F; }            /* new connection */
proc action_R_y: ADD_r
      { if (F != ADD_r) F = ADD_r; }
```

Appendix C

Simulation of Data-Flow Machine

This appendix lists the entire program of a simulation of parallel data flow machine described in §4.3.

Main Program	C-1
Macro Expanded Listing	C-2
Testing	C-3
Test Output	C-4

```
include ("macro.e");                    /* see Appendix B */

                                    /*** ALU FUNCTIONS ($1=P_a, $2=P_b) ***/
func ADD    { return $1 + $2; }         /* R = A + B */
func MULT   { return $1 * $2; }         /* R = A * B */
func DELAY  { return $1; }              /* R = A     */

proc alloc_processor                    /* $1 = processor id */
{
   execute(macro(
     "proc ?1_L1: clock { if (clock==0) ?1_a = ?1_A; }
      proc ?1_L2: clock { if (clock==0) ?1_b = ?1_B; }
      proc ?1_L3: clock { if (clock==1) ?1_R = ?1_r; }
      ?1_r is ?1_F(?1_a, ?1_b);", $1));
}

                                                /*** ALLOCATE 11 PROCESSORS ***/
for (i = 1; i <= 11; i++)
   alloc_processor("P"//str(i));

                                                /*** PROGRAM THE PROCESSORS ***/
P1_F = P2_F = P6_F = ADD;
P3_F = P8_F = P11_F = DELAY;
P4_F = P5_F = P7_F = P9_F = P10_F = MULT;

                                                 /*** INTER-PROCESSOR WIRES ***/
fx    is P1_R;
P1_A  is P2_R;          P1_B  is P3_R;
P2_A  is P4_R;          P2_B  is P5_R;
P3_A  is P6_R;
P4_A  is P7_R;          P4_B  is P9_R;
P5_A  is P8_R;          P5_B  is P9_R;
P6_A  is P10_R;         P6_B  is P11_R;
P7_A  is a;             P7_B  is x;
P8_A  is b;
P9_A  is x;             P9_B  is x;
P10_A is x;             P10_B is c;
P11_A is d;
                /****** END OF SPECIFICATION ******/
```

```
                                        /*** ALU FUNCTIONS ($1=P_a, $2=P_b) ***/
func ADD    { return $1 + $2; }      /* R = A + B */
func MULT   { return $1 * $2; }      /* R = A * B */
func DELAY  { return $1; }           /* R = A     */

                                                              /* processor P1 */
proc P1_L1: clock { if (clock==0) P1_a = P1_A; }
proc P1_L2: clock { if (clock==0) P1_b = P1_B; }
proc P1_L3: clock { if (clock==1) P1_R = P1_r; }
P1_r is P1_F(P1_a, P1_b);

                                                              /* processor P2 */
proc P2_L1: clock { if (clock==0) P2_a = P2_A; }
proc P2_L2: clock { if (clock==0) P2_b = P2_B; }
proc P2_L3: clock { if (clock==1) P2_R = P2_r; }
P2_r is P2_F(P2_a, P2_b);

                                                              /* processor P3 */
proc P3_L1: clock { if (clock==0) P3_a = P3_A; }
proc P3_L2: clock { if (clock==0) P3_b = P3_B; }
proc P3_L3: clock { if (clock==1) P3_R = P3_r; }
P3_r is P3_F(P3_a, P3_b);

                                                              /* processor P4 */
proc P4_L1: clock { if (clock==0) P4_a = P4_A; }
proc P4_L2: clock { if (clock==0) P4_b = P4_B; }
proc P4_L3: clock { if (clock==1) P4_R = P4_r; }
P4_r is P4_F(P4_a, P4_b);

                                                              /* processor P5 */
proc P5_L1: clock { if (clock==0) P5_a = P5_A; }
proc P5_L2: clock { if (clock==0) P5_b = P5_B; }
proc P5_L3: clock { if (clock==1) P5_R = P5_r; }
P5_r is P5_F(P5_a, P5_b);

                                                              /* processor P6 */
proc P6_L1: clock { if (clock==0) P6_a = P6_A; }
proc P6_L2: clock { if (clock==0) P6_b = P6_B; }
proc P6_L3: clock { if (clock==1) P6_R = P6_r; }
P6_r is P6_F(P6_a, P6_b);

                                                              /* processor P7 */
proc P7_L1: clock { if (clock==0) P7_a = P7_A; }
proc P7_L2: clock { if (clock==0) P7_b = P7_B; }
proc P7_L3: clock { if (clock==1) P7_R = P7_r; }
P7_r is P7_F(P7_a, P7_b);

                                                              /* processor P8 */
proc P8_L1: clock { if (clock==0) P8_a = P8_A; }
proc P8_L2: clock { if (clock==0) P8_b = P8_B; }
proc P8_L3: clock { if (clock==1) P8_R = P8_r; }
P8_r is P8_F(P8_a, P8_b);

                                                              /* processor P9 */
proc P9_L1: clock { if (clock==0) P9_a = P9_A; }
proc P9_L2: clock { if (clock==0) P9_b = P9_B; }
proc P9_L3: clock { if (clock==1) P9_R = P9_r; }
P9_r is P9_F(P9_a, P9_b);
```

```
                                                        /* processor P10 */
proc P10_L1: clock { if (clock==0) P10_a = P10_A; }
proc P10_L2: clock { if (clock==0) P10_b = P10_B; }
proc P10_L3: clock { if (clock==1) P10_R = P10_r; }
P10_r is P10_F(P10_a, P10_b);

                                                        /* processor P11 */
proc P11_L1: clock { if (clock==0) P11_a = P11_A; }
proc P11_L2: clock { if (clock==0) P11_b = P11_B; }
proc P11_L3: clock { if (clock==1) P11_R = P11_r; }
P11_r is P11_F(P11_a, P11_b);

                                              /*** PROGRAM THE PROCESSORS ***/
P1_F = P2_F = P6_F = ADD;
P3_F = P8_F = P11_F = DELAY;
P4_F = P5_F = P7_F = P9_F = P10_F = MULT;

                                              /*** INTER-PROCESSOR WIRES ***/
fx     is P1_R;
P1_A   is P2_R;          P1_B   is P3_R;
P2_A   is P4_R;          P2_B   is P5_R;
P3_A   is P6_R;
P4_A   is P7_R;          P4_B   is P9_R;
P5_A   is P8_R;          P5_B   is P9_R;
P6_A   is P10_R;         P6_B   is P11_R;
P7_A   is a;             P7_B   is x;
P8_A   is b;
P9_A   is x;             P9_B   is x;
P10_A  is x;             P10_B  is c;
P11_A  is d;
                  /****** END OF SPECIFICATION ******/
```

Testing

```
x_stream = [ 2, 3, 4, 5, @, @, @, @ ];     /* input screams */
a_stream = [ 1, 1,-4, 2, @, @, @, @ ];
b_stream = [ 3,-3, 5, 6, @, @, @, @ ];
c_stream = [ 3,-3, 5,-2, @, @, @, @ ];
d_stream = [ 1, 1, 2, 7, @, @, @, @ ];

for (time = 1; time <= x_stream#; time++) {
   write("time = ", time);
   x = x_stream[time];              /* load inputs */
   a = a_stream[time];              /*   before clock = 0 */
   b = b_stream[time];
   c = c_stream[time];
   d = d_stream[time];
   write("\t(x,a,b,c,d) = (",x,",",a,",",b,",",c,",",d,")");
   clock=0;                         /* trigger clock */
   clock=1;
   writeln("\tfx = ", fx);          /* print fx */
}
```

```
time = 1     (x,a,b,c,d) = (2,1,3,3,1)      fx = @
time = 2     (x,a,b,c,d) = (3,1,-3,-3,1)    fx = @
time = 3     (x,a,b,c,d) = (4,-4,5,5,2)     fx = @
time = 4     (x,a,b,c,d) = (5,2,6,-2,7)     fx = 27
time = 5     (x,a,b,c,d) = (@,@,@,@,@)      fx = -8
time = 6     (x,a,b,c,d) = (@,@,@,@,@)      fx = -154
time = 7     (x,a,b,c,d) = (@,@,@,@,@)      fx = 397
time = 8     (x,a,b,c,d) = (@,@,@,@,@)      fx = @
```

Appendix D

Listing of the DoNaLD-to-EDEN Translator

File Function	Page
Makefile	**1**
yacc.y	**2**
lex.l	**6**
error.h	**8**
oper.h	**9**
symbol.h	**11**
tree.h	**12**
Ivalue	12
Rvalue	12
Lexpr	13
Mexpr	13
Rexpr	13
Ltype	13
Mtype	13
Rtype	13
Etype	13
check_left	13
check_right	13
check_mid	13
check.c	**14**
TypeClash	14
count_id	14
check_children_type	14
semantic_1	15
semantic_2	15
semantic_3	15
count_expr	15
check_id	17
is_shape	18
is_openshape	18
error.c	**19**
don_err	19
main.c	**20**
main	20
print_prompt	20
print_prompt2	20
yyerror	21
symbol.c	**22**
change_context	22
resume_context	23
reset_context	23
change_scope	23
resume_scope	23
symbol_search	24
context_search	24
new_symbol	24
look_up	24
declare	25
declare_segment	25
declare_attribute	25
declare_action	26
eden_declare	26
declare_openshape	26
print_all_symbols	27
print_symbol_table	27
donald_full_name	27
eden_full_name	28
expr_to_eden_name	28
tree.c	**29**
tree3	29
tree2	29
tree1	29
tree0	30
match_operator	30
nary	30
freetree	30
lastchar	31
print_comma	31
print_rparen	31
print_tree	31
define	32
utility.c	**34**
strsave	34
getmem	34
drawgfx.e	**35**
objtype	35
new_object	35
set_attributes	35
reset_attributes	35
plot_point	35
plot_line	35
plot_shape	35
draw_shape	36
get_xy_coord	36
moveto	37
lineto	37
init.e	**38**
cart	38
polar	38
line	38
open2shape	38
rot	39
scale	39
line_reverse	39
dotx	39
doty	40
dot1	40
dot2	40
vector_add	40
vector_sub	40
scalar_mult	40
scalar_div	40
scalar_mod	40
start	40
quit	41
suncore.e.h	**42**
Sample DoNaLD Specification	**44**
Sample DoNaLD Translator Output	**48**

Table of Functions (Sorted)

Function	Page
cart	38
change_context	22
change_scope	23
check_children_type	14
check_id	17
check_left	13
check_mid	13
check_right	13
context_search	24
count_expr	15
count_id	14
declare	25
declare_action	26
declare_attribute	25
declare_openshape	26
declare_segment	25
define	32
don_err	19
donald_full_name	27
dot1	40
dot2	40
dotx	39
doty	40
draw_shape	36
eden_declare	26
eden_full_name	28
Etype	13
expr_to_eden_name	28
freetree	30
get_xy_coord	36
getmem	34
is_openshape	18
is_shape	18
Ivalue	12
lastchar	31
Lexpr	13
line	38
line_reverse	39
lineto	37
look_up	24
Ltype	13
main	20
match_operator	30
Mexpr	13
moveto	37
Mtype	13
nary	30
new_object	35
new_symbol	24
objtype	35
open2shape	38
plot_line	35
plot_point	35
plot_shape	35
polar	38
print_all_symbols	27
print_comma	31
print_prompt	20
print_prompt2	20
print_rparen	31
print_symbol_table	27
print_tree	31
quit	41
reset_attributes	35
reset_context	23
resume_context	23
resume_scope	23
Rexpr	13
rot	39
Rtype	13
Rvalue	12
scalar_div	40
scalar_mod	40
scalar_mult	40
scale	39
semantic_1	15
semantic_2	15
semantic_3	15
set_attributes	35
start	40
strsave	34
symbol_search	24
tree0	30
tree1	29
tree2	29
tree3	29
TypeClash	14
vector_add	40
vector_sub	40
yyerror	21

Makefile

```
CFLAGS    = -g
YFLAGS    = -d
CFILES    = check.c error.c main.c symbol.c tree.c utility.c
HFILES    = error.h oper.h symbol.h tree.h
YFILE     = yacc.y
LFILE     = lex.l
ALL       = $(YFILE) $(LFILE) $(CFILES) $(HFILES)
EFILES    = init.e drawgfx.e suncore.e.h
OBJS      = yacc.o lex.o \
            check.o error.o main.o symbol.o tree.o utility.o

donald.trans: $(OBJS)
      cc $(CFLAGS) $(OBJS) -ll -o donald.trans

donald.a: $(ALL) Makefile $(EFILES)
      @echo SAVING $? TO donald.a
      @-chmod +w donald.a
      @ar r donald.a $?
      @chmod -w donald.a

main.o:   error.h
check.o:  error.h oper.h tree.h symbol.h x.tab.h
error.o:  error.h
symbol.o: error.h symbol.h oper.h tree.h x.tab.h
tree.o:   symbol.h error.h oper.h tree.h
utility.o:error.h

x.tab.h: y.tab.h
      @-cmp -s x.tab.h y.tab.h || cp y.tab.h x.tab.h

lex.o: x.tab.h error.h oper.h tree.h $(LFILE)

yacc.o y.tab.h: symbol.h error.h oper.h tree.h $(YFILE)
```

yacc.y

```
%{
#include <stdio.h>
#include "symbol.h"
#include "error.h"
#include "oper.h"
#include "tree.h"
#ifdef DEBUG
#define    display(a)        fprintf(stderr, a)
#define    display2(a,i)     fprintf(stderr, a, i)
#endif
#ifndef DEBUG
#define    display(a)        /* nothing */
#define    display2(a,i)     /* nothing */
#endif
#define ERROR -1
extern char *prompt;
%}

%union {
     char       *s;   /* identifier, etc */
     int        i;    /* integer value   */
     double     r;    /* floating point value */
     tree       t;    /* expression tree */
}

%token     <i>   INF
%token     <i>   DIV MOD
%token     <i>   IF THEN ELSE
%token     <i>   INT REAL
%token     <i>   POINT
%token     <i>   LINE SHAPE OPENSHAPE
%token     <i>   WITHIN
%token     <s>   ID
%token     <i>   INUMBER
%token     <r>   RNUMBER
%token     <i>   ROT SCALE

%type      <i>   type_name
%type      <t>   definition expr expr_list identifier id local_id
%type      <t>   within_id

%right     EQUALS
%left      COMMA
%token     <s>   QUERY       /* non-standard DoNaLD command */
%left      ELSE
%nonassoc  ATSIGN
%left      PLUS MINUS
%left      STAR PERCENT DIV MOD
%left      TILDE
%left      DOTX DOTY DOTARG DOTRAD
%left      DOT1 DOT2
%nonassoc  UMINUS
%left      SLASH
%nonassoc  LID
```

```
%left       COLON
%nonassoc   NEWLINE
%nonassoc   LPAREN RPAREN LBRACK RBRACK LCURLY RCURLY

%%          /*----- grammar -----*/

program_list:
        /* nothing */
      | program_list stmt       { print_prompt(); }
      | program_list error      { yyerrok; }
      ;

stmt:
        declaration             {
#ifdef DEBUG
                                  print_all_symbols();
#endif
                                }
      | definition
      | evaluation
      | within_clause
      | NEWLINE                 /* null statement */
      ;

within_id:
        WITHIN identifier       { change_scope($$ = $2); }
      ;

within_clause:
        within_id stmt          { resume_scope(); freetree($1); }
      | within_id LCURLY stmt_list RCURLY
                                { resume_scope(); freetree($1); }
      ;

stmt_list:
        stmt
      | stmt_list stmt
      ;

declaration:
        type_name expr_list NEWLINE
                                { declare($1, $2); }
      ;

type_name:
        INT                     { $$ = INT; }
      | REAL                    { $$ = REAL; }
      | POINT                   { $$ = POINT; }
      | LINE                    { $$ = LINE; }
      | SHAPE                   { $$ = SHAPE; }
      | OPENSHAPE               { $$ = OPENSHAPE; }
      ;

definition:
        expr_list EQUALS expr_list NEWLINE
                                {
                                  int       no_id;

                                  if ((no_id = count_id($1)) == ERROR)
                                      don_err(IdListExpect, 0);
                                  if (no_id != count_expr($3))
                                      don_err(IdExprUnmatch, 0);
                                  if (no_id > 1)
```

```
                                    printf("autocalc = OFF;\n");
                                define($1, $3);
                                freetree($1);
                                freetree($3);
                                if (no_id > 1)
                                    printf("autocalc = ON;\n");
                            }
        ;

evaluation:
          QUERY NEWLINE         { printf("%s\n", $1); free($1); }
        ;


expr_list:
          expr
        | expr COMMA expr_list  { $$ = tree2(OP_COMMA, $1, $3); }
        ;


expr:     INF                   { $$ = tree0(OP_INF); }
        | INUMBER               { $$ = tree0(OP_INUMBER);
                                  $$->type = INT;
                                  Ivalue($$) = $1;
                                }
        | RNUMBER               { $$ = tree0(OP_RNUMBER);
                                  $$->type = REAL;
                                  Rvalue($$) = $1;
                                  display2("rnumber(%lf) ", $1);
                                }
        | LPAREN expr RPAREN    { $$ = tree1(OP_PAREN, $2); }
        | expr PLUS expr        { $$ = tree2(OP_PLUS, $1, $3); }
        | expr MINUS expr       { $$ = tree2(OP_MINUS, $1, $3); }
        | expr STAR expr        { $$ = tree2(OP_MULT, $1, $3); }
        | expr MOD expr         { $$ = tree2(OP_MOD, $1, $3); }
        | expr DIV expr         { $$ = tree2(OP_DIV, $1, $3); }
        | MINUS expr      %prec UMINUS
                                { $$ = tree1(OP_UMINUS, $2); }
        | expr DOTX             { $$ = tree1(OP_DOTX, $1); }
        | expr DOTY             { $$ = tree1(OP_DOTY, $1); }
        | expr DOT1             { $$ = tree1(OP_DOT1, $1); }
        | expr DOT2             { $$ = tree1(OP_DOT2, $1); }
        | expr DOTRAD           { $$ = tree1(OP_DOTRAD, $1); }
        | expr DOTARG           { $$ = tree1(OP_DOTARG, $1); }
        | LCURLY expr COMMA expr RCURLY
                                { $$ = tree2(OP_CART, $2, $4); }
        | LCURLY expr ATSIGN expr RCURLY
                                { $$ = tree2(OP_POLAR, $2, $4); }
        | LBRACK expr COMMA expr RBRACK
                                { $$ = tree2(OP_LINE, $2, $4); }
        | ROT LPAREN expr COMMA expr COMMA expr RPAREN
                                { $$ = tree3(OP_ROT, $3, $5, $7); }
        | SCALE LPAREN expr COMMA expr RPAREN
                                { $$ = tree3(OP_SCALE, $3, $5); }
        | IF expr THEN expr ELSE expr %prec ELSE
                                { $$ = tree3(OP_IF, $4, $2, $6); }
        | identifier      %prec LID
        ;

identifier:
          local_id SLASH identifier
                                { $$ = tree2(OP_SLASH, $1, $3); }
        | local_id
```

```
        ;

local_id:
          id                %prec LID
        ;

id:       ID                      { $$ = tree1(OP_ID, $1); }
        | TILDE                   { $$ = tree1(OP_ID, 0); }
        ;

%%
```

lex.l

```
%{
#include "error.h"
#include "oper.h"
#include "tree.h"
#include "y.tab.h"
%}
D                       [0-9]
E                       [Ee][+-]?{D}+
A                       [A-Za-z]
%%
[ \t]                   ;
\\\n                    print_prompt2();
"#".*\n                 return NEWLINE;
\n                      return NEWLINE;
inf                     return INF;
div                     return DIV;
mod                     return MOD;
if                      return IF;
then                    return THEN;
else                    return ELSE;
int                     return INT;
real                    return REAL;
point                   return POINT;
line                    return LINE;
shape                   return SHAPE;
openshape               return OPENSHAPE;
within                  return WITHIN;
trunc                   return TRUNC;
float                   return FLOAT;
rot                     return ROT;
scale                   return SCALE;

","                     return COMMA;
"+"                     return PLUS;
"-"                     return MINUS;
"*"                     return STAR;
"/"                     return SLASH;
"="                     return EQUALS;
"~"                     return TILDE;
"@"                     return ATSIGN;
"("                     return LPAREN;
")"                     return RPAREN;
"["                     return LBRACK;
"]"                     return RBRACK;
"{"                     return LCURLY;
"}"                     return RCURLY;
".x"                    return DOTX;
".y"                    return DOTY;
".arg"                  return DOTARG;
".rad"                  return DOTRAD;
".1"                    return DOT1;
".2"                    return DOT2;
"?".*                   {
                          char      *strsave();
```

```
                        yylval.s = strsave(yytext+1);
                        return QUERY;
                      }

{A}({A}|{D})*         { /* idenifier */
                        char     *strsave();

                        yylval.s = strsave(yytext);
                        return ID;
                      }

{D}*"."{D}+({E})?     |
{D}+"."{D}*({E})?     |
{D}+{E}               { /* floating point constant */
                        double   atof();

                        yylval.r = atof(yytext);
                        return RNUMBER;
                      }

{D}+                  { /* integer constant */
                        yylval.i = atoi(yytext);
                        return INUMBER;
                      }

.                     return *yytext;
```

error.h

```
/*----- error code definitions -----*/

enum errorcodes {
      Unclassified      ,
      Impossible        ,
      SyntaxError       ,
      OutOfMemory       ,
      UndefinedID       ,
      IdListExpect      ,
      IdExprUnmatch     ,
      RedeclareID       ,
      UndeclareID       ,
      StackOverflow     ,
      StackUnderflow    ,
      NotOpenshape      ,
      TypeMismatch
};

/*
** ERROR MESSAGE STRINGS
**    corresponds to the errorcodes above
*/
#define ERRORSTRINGS                          \
      "%s"                                   ,\
      "impossible error occurs in %s"        ,\
      "syntax error"                         ,\
      "out of memory"                        ,\
      "undefined id: %s"                     ,\
      "id list expected"                     ,\
      "id list and expr list unmatched"      ,\
      "redeclared identifier: %s"            ,\
      "undeclared identifier: %s"            ,\
      "%sstack overflow"                     ,\
      "%sstack underflow"                    ,\
      "not openshape: %s"                    ,\
      "type mismatched"
```

oper.h

```
/*----- operator code -----*/

#define OP_INF            0
#define OP_ASSIGN         1
#define OP_PLUS           2
#define OP_MINUS          3
#define OP_MULT           4
#define OP_MOD            5
#define OP_DIV            6
#define OP_UMINUS         7
#define OP_TILDE          8
#define OP_DOTX           9
#define OP_DOTY          10
#define OP_DOT1          11
#define OP_DOT2          12
#define OP_DOTRAD        13
#define OP_DOTARG        14
#define OP_CART          15
#define OP_POLAR         16
#define OP_LINE          17
#define OP_ROT           18
#define OP_SCALE         19
#define OP_SLASH         20
#define OP_ID            21
#define OP_INUMBER       22
#define OP_COMMA         23
#define OP_PAREN         24
#define OP_IF            25
#define OP_VECT_ADD      26
#define OP_VECT_SUB      27
#define OP_SCALAR_MULT   28
#define OP_SCALAR_DIV    29
#define OP_SCALAR_MOD    30
#define OP_RNUMBER       31

#define UNARY            -1
#define BINARY           -2
#define TRIARY           -3
#define INFIX            -4
```

```
struct trans {
      int  order;
      char *op_str;
};

/*
** translation table
**    format: { operator-type , string } ,
**    op-type indicates whether it is unary or infix etc.
**    string is the EDEN translation.
*/
#define TRANSLAT   \
{                              \
   /* OP_INF     */       { OP_INF,       "[ INF ]"       }, \
   /* OP_ASSIGN  */       { INFIX,        " = "           }, \
   /* OP_PLUS    */       { INFIX,        " + "           }, \
   /* OP_MINUS   */       { INFIX,        " - "           }, \
   /* OP_MULT    */       { INFIX,        " * "           }, \
   /* OP_MOD     */       { INFIX,        " % "           }, \
   /* OP_DIV     */       { INFIX,        " / "           }, \
   /* OP_UMINUS  */       { UNARY,        "- "            }, \
   /* OP_TILDE   */       { UNARY,        0               }, \
   /* OP_DOTX    */       { UNARY,        "dotx("         }, \
   /* OP_DOTY    */       { UNARY,        "doty("         }, \
   /* OP_DOT1    */       { UNARY,        "dot1("         }, \
   /* OP_DOT2    */       { UNARY,        "dot2("         }, \
   /* OP_DOTRAD  */       { UNARY,        "rad("          }, \
   /* OP_DOTARG  */       { UNARY,        "arg("          }, \
   /* OP_CART    */       { BINARY,       "cart("         }, \
   /* OP_POLAR   */       { BINARY,       "polar("        }, \
   /* OP_LINE    */       { BINARY,       "line("         }, \
   /* OP_ROT     */       { TRIARY,       "rot("          }, \
   /* OP_SCALE   */       { BINARY,       "scale("        }, \
   /* OP_SLASH   */       { OP_SLASH,     ""              }, \
   /* OP_ID      */       { OP_ID                         }, \
   /* OP_INUMBER*/        { OP_INUMBER                    }, \
   /* OP_COMMA   */       { INFIX,        ", "            }, \
   /* OP_PAREN   */       { UNARY,        "("             }, \
   /* OP_IF      */       { OP_IF                         }, \
   /* OP_VECT_ADD */      { BINARY,       "vector_add("   }, \
   /* OP_VECT_SUB */      { BINARY,       "vector_sub("   }, \
   /* OP_SCALAR_MULT */   { BINARY,       "scalar_mult("  }, \
   /* OP_SCALAR_DIV */    { BINARY,       "scalar_div("   }, \
   /* OP_SCALAR_MOD */    { BINARY,       "scalar_mod("   }, \
   /* OP_RNUMBER */       { OP_RNUMBER                    } \
}
```

symbol.h

```
/*
** symbol structure:
**    name - name of object
**    type - type of object, i.e. int, real, point, etc.
**    derived - for openshapes only
**            - points to the component objects
**    next - next symbol
*/
struct  symbol  {
     char             *name;      /* symbol name */
     int              type;       /* symbol type */
     struct  symbol   *derived;   /* sub-directory */
     struct  symbol   *next;      /* next symbol entry */
};

typedef  struct  symbol  symbol;

extern  symbol  *SymbolTable;

#define    UNDEFINED   0

#define    SymEnd      (symbol*)0
```

tree.h

```
/*
** node0-3 are the structures of parse tree nodes
** which contains 0-3 children respectively
*/
struct node3 {
      int           type;
      int           op;
      struct node3  *left;
      struct node3  *right;
      struct node3  *mid;
};

struct node2 {
      int           type;
      int           op;
      struct node3  *left;
      struct node3  *right;
};

struct node1 {
      int           type;
      int           op;
      struct node3  *left;
};

struct node0 {
      int  type;
      int  op;
      union {
            int     i;
            double  r;
      } d;
};

typedef struct  node3 node3;
typedef struct  node2 node2;
typedef struct  node1 node1;
typedef struct  node0 node0;
typedef node3 *           tree;

extern tree tree0(), tree1(), tree2(), tree3();

#define EMPTYTREE     (tree)0
```

Ivalue
Rvalue

```
#define Ivalue(expr)   ((node0*)expr)->d.i
#define Rvalue(expr)   ((node0*)expr)->d.r
```

Lexpr
Mexpr
Rexpr

```
#define Lexpr          expr->left
#define Mexpr          expr->mid
#define Rexpr          expr->right
```

Ltype
Mtype
Rtype
Etype

```
#define Ltype          Lexpr->type
#define Mtype          Mexpr->type
#define Rtype          Rexpr->type
#define Etype          expr->type
```

check_left
check_right
check_mid

```
#define check_left()   count_expr(Lexpr)
#define check_right()        count_expr(Rexpr)
#define check_mid()    count_expr(Mexpr)
```

check.c

```
#include "error.h"
#include "oper.h"
#include "tree.h"
#include "symbol.h"
#include "y.tab.h"

#define ERROR -1
```

TypeClash

```
TypeClash()                     /* a type clash error */
{
      don_err(TypeMismatch, 0);
}
```

count_id

```
count_id(expr)                  /* return no. of id in an id-list */
      tree expr;
{
      int    lcount, rcount;

      if (expr == EMPTYTREE)
            return 0;

      switch (expr->op) {
      case OP_ID:
      case OP_SLASH:
            check_id(expr);
            return 1;

      case OP_COMMA:
            lcount = count_id(Lexpr);
            rcount = count_id(Rexpr);
            return (lcount == ERROR) || (rcount == ERROR)
                  ? ERROR : lcount + rcount;

      default:
            return ERROR;
      }
}
```

check_children_type

```
check_children_type(expr)     /* check the semantics of children nodes */
      tree expr;
{
      if  (expr->type != UNDEFINED)       /* already checked */
            return;

      switch (nary(expr->op)) {           /* no. of children */
            case 3: count_expr(expr->mid);   /* count_expr will     */
            case 2: count_expr(expr->right); /* check the semantics */
            case 1: count_expr(expr->left);
```

```
        }
}

/*
** semantic1, semantic2 & semantic3
** check expression types:
**      E = result data type (if ok)
**      L = left expr data type expected
**      M = middle expr data type expected
**      R = right expr type expected
** return    0 if ok
**           1 if not match
*/
```

semantic_1

```
semantic_1(expr, E, L)
        tree expr;
{
        return      (Ltype != L) ||
                    ((Etype = E), 0);
}
```

semantic_2

```
semantic_2(expr, E, L, R)
        tree expr;
{
        return      (Ltype != L) ||
                    (Rtype != R) ||
                    ((Etype = E), 0);
}
```

semantic_3

```
semantic_3(expr, E, L, M, R)
        tree expr;
{
        return      (Ltype != L) ||
                    (Mtype != M) ||
                    (Rtype != R) ||
                    ((Etype = E), 0);
}
```

count_expr

```
count_expr(expr)          /* return no. of expr in an expr-list */
        tree expr;
{
        if  (expr == EMPTYTREE)
              return 0;

        check_children_type(expr);
        switch (expr->op) {
        case OP_COMMA:
              return count_expr(Lexpr) + count_expr(Rexpr);

        case OP_INF:
              Etype = INT;
              break;

        case OP_PLUS:
              if  (semantic_2(expr, Ltype, Ltype, Ltype) &&
                    semantic_2(expr, REAL, REAL, INT) &&
                    semantic_2(expr, REAL, INT, REAL))
```

```
            TypeClash();
        else if   (Etype == POINT)
            expr->op = OP_VECT_ADD;
        break;

    case OP_MINUS:
        if (semantic_2(expr, Ltype, Ltype, Ltype) &&
            semantic_2(expr, REAL, REAL, INT) &&
            semantic_2(expr, REAL, INT, REAL))
            TypeClash();
        else if (Etype == POINT)
            expr->op = OP_VECT_SUB;
        break;

    case OP_MULT:
    case OP_MOD:
    case OP_DIV:
        if (semantic_2(expr, Ltype, Ltype, INT) &&
            semantic_2(expr, REAL, INT, REAL))
            TypeClash();
        else if (Etype == POINT) {
            switch (expr->op) {
            case OP_MULT:
                expr->op = OP_SCALAR_MULT;
            break;
            case OP_DIV:
                expr->op = OP_SCALAR_DIV;
            break;
            case OP_MOD:
                expr->op = OP_SCALAR_MOD;
            break;
            }
        }
        break;

    case OP_UMINUS:
        if (semantic_1(expr, INT, INT) &&
            semantic_1(expr, REAL, REAL))
            TypeClash();
        break;

    case OP_DOTX:
    case OP_DOTY:
        if (semantic_1(expr, POINT, POINT))
            TypeClash();
        break;

    case OP_DOT1:
    case OP_DOT2:
        if (semantic_1(expr, REAL, POINT) &&
            semantic_1(expr, POINT, LINE))
            TypeClash();
        break;

    case OP_CART:
    case OP_POLAR:
        if (semantic_2(expr, POINT, INT, INT) &&
            semantic_2(expr, POINT, REAL, REAL) &&
            semantic_2(expr, POINT, INT, REAL) &&
            semantic_2(expr, POINT, REAL, INT))
            TypeClash();
        break;
```

```
        case OP_LINE:
                if (semantic_2(expr, LINE, POINT, POINT))
                        TypeClash();
                break;

        case OP_IF:
                if (semantic_3(expr, Ltype, Ltype, INT, Ltype))
                        TypeClash();
                break;

        case OP_ROT:
                switch (Ltype) {
                case SHAPE:
                case OPENSHAPE:
                        if (semantic_3(expr, SHAPE, Ltype, POINT, INT))
                                TypeClash();
                        break;

                case INT:
                case REAL:
                        TypeClash();
                        break;

                default:
                        if (semantic_3(expr, Ltype, Ltype, POINT, INT))
                                TypeClash();
                        break;
                }
                break;

        case OP_PAREN:
                Etype = Ltype;
                break;

        case OP_ID:
        case OP_SLASH:
                check_id(expr);
                break;
        }
        return 1;               /* default */
}
```

check_id

```
check_id(expr)                  /* Is variable declared ? */
        tree expr;              /* id-expr is assumed */
{
        char *id;
        symbol      *sym;
        symbol      *look_up();
        char *donald_full_name();

        switch (expr->op) {
        case OP_ID:
                id = (char *) Lexpr;
                if (id == 0)
                        don_err(Unclassified, "illegal use of `~'");
                if ((sym = look_up(id)) == SymEnd)
                        don_err(UndeclareID, donald_full_name(id));
                Etype = sym->type;
                break;

        case OP_SLASH:
```

```
            id = (char *) Lexpr->left;
            if (id) {
                  if ((sym = look_up(id)) == SymEnd)
                  don_err(UndeclareID, donald_full_name(id));
                  if (sym->type != OPENSHAPE)
                  don_err(NotOpenshape, donald_full_name(id));
                  change_context(sym);
            } else {          /* id == ~ */
                  change_context(SymEnd);
            }
            check_id(Rexpr);
            Etype = Rtype;
            resume_context();
            break;

      default:
            don_err(Impossible, "check.c");
            break;
      }
}
```

is_shape

```
is_shape(expr)          /* is object named by expr a shape? */
      tree        expr;
{
      return expr->type == SHAPE;
}
```

is_openshape

```
is_openshape(expr)      /* is object named by expr an openshape? */
      tree        expr;
{
      return expr->type == OPENSHAPE;
}
```

error.c

```
#include <stdio.h>
#include <setjmp.h>
#include "error.h"

static char *errorstring[] = { ERRORSTRINGS };      /* see error.h */
```

don_err

```
don_err(error_code, s)                    /* DoNaLD error handler */
      enum errorcodes  error_code;
{
      extern  char     *prog_name;        /* program name */
      extern  jmp_buf  don_start;         /* DoNaLD re-entry points */
      extern           reset_context();
      extern  FILE     *yyin;             /* lex input */
      extern  char     yysbuf[];          /* lex input buffer */
      extern  char     *yysptr;           /* lex input buffer pointer */

      fprintf(stderr, "%s: ", prog_name);
      fprintf(stderr, errorstring[(int)error_code], s);
      fprintf(stderr, "\n");
      fseek(yyin, 0L, 2);                 /* skip rest of input */
      yysptr = yysbuf;
      if (reset_context() > 0)
         fprintf(stderr, "%s: warning: reset context\n", prog_name);
      longjmp(don_start, 1);
}
```

main.c

```
#include <stdio.h>
#include <setjmp.h>
#include "error.h"

jmp_buf    don_start;

char *prompt  = "(!) ";
char *prompt2 = "(.) ";
char *prog_name;
```

main

```
main(argc, argv)
      char        *argv[];
{
      int         error_flag;

      prog_name = *argv;

      setbuf(stdout, 0);                    /* unbuffer the output */

      error_flag = setjmp(don_start);       /* set resume point */

program_start:

      switch (error_flag) {
      case 0:                               /* beginning of program */
            print_prompt();
            yyparse();
            break;
      default:                              /* resume from longjmp */
            error_flag = 0;                 /* reset error_flag */
            goto program_start;
            break;
      }
      printf("quit();\n");                  /* quit SunCore in EDEN */
      fprintf(stderr, "\n");
}
```

print_prompt

```
print_prompt()                              /* print a prompt */
{
      fprintf(stderr, "%s", prompt);
}
```

print_prompt2

```
print_prompt2()               /* second prompt for continuous line */
{
      fprintf(stderr, "%s", prompt2);
}
```

yyerror

```
yyerror(s)                                  /* error handler */
      char        *s;
{
      don_err(Unclassified, s);
}
```

symbol.c

```
#include <stdio.h>
#include "error.h"
#include "symbol.h"
#include "oper.h"
#include "tree.h"
#include "y.tab.h"

#define MAXCONTEXT 64

/* "indx" always points at the toppest context */
static  int       indx = 0;

static symbol   *GlobalSymbols = SymEnd;

static struct c {
            symbol      *sym;
            symbol      **table;
            int         prev_scope;
      } context[MAXCONTEXT] = { SymEnd, &GlobalSymbols, 0 } ;

#define SymbolTable     *context[indx].table
#define PARENT          0

static int        backindx = 0;
static struct c backcontext[MAXCONTEXT];
static struct c NORMAL = { SymEnd };
```

change_context

```
change_context(sym)           /* change context to sym */
      symbol      *sym;
{
      if (sym == SymEnd) {
            if (backindx > MAXCONTEXT)
                  don_err(StackOverflow, "context ");
            backcontext[backindx++] = context[indx];
            if (indx == 0)
                  don_err(StackUnderflow, "context ");
            else
                  --indx;
      } else {
            if (backindx > MAXCONTEXT)
                  don_err(StackOverflow, "context ");
            backcontext[backindx++] = NORMAL;
            if (indx++ >= MAXCONTEXT)
                  don_err(StackOverflow, "context ");
            context[indx].sym = sym;
            context[indx].table = &sym->derived;
            context[indx].prev_scope = context[indx-1].prev_scope;
      }
}
```

resume_context

```
resume_context()                        /* cd .. */
{
      struct c   cnt;

      if (backindx <= 0)
            don_err(StackUnderflow, "context ");
      cnt = backcontext[--backindx];
      if (cnt.sym == SymEnd) {
            if (indx == 0)
                  don_err(StackUnderflow, "context ");
            else
                  --indx;
      } else {
            if (indx++ >= MAXCONTEXT)
                  don_err(StackOverflow, "context ");
            context[indx] = cnt;
      }
}
```

reset_context

```
reset_context()                         /* cd root */
{
      int ix = indx;

      indx = 0;                         /* indx==0 means the root */
      backindx = 0;
      return ix;
}
```

change_scope

```
change_scope(expr)              /* change the scope to the context ... */
      tree        expr;         /* ... expressed by expr */
{
      symbol      *context_search();
      int         ix = indx;              /* so we can resume_scope */

      switch (expr->op) {
      case OP_ID:
            change_context(context_search(Lexpr));
            break;

      case OP_SLASH:
            change_context(context_search(Lexpr->left));
            change_scope(Rexpr);
            break;

      default:
            don_err(Impossible, "change_scope");
      }
      context[indx].prev_scope = ix;      /* so we can resume_scope */
}
```

resume_scope

```
resume_scope()                  /* undo last change_scope */
{
      indx = context[indx].prev_scope;
}
```

symbol_search

```
symbol *
symbol_search(name)           /* find the symbol entry w/ object name */
      char        *name;      /* within the current context */
{
      symbol      *sym, *look_up();
      char        *donald_full_name();

      if (! (sym = look_up(name)))        /* error if not found */
            don_err(UndeclareID, donald_full_name(name));
      return sym;
}
```

context_search

```
symbol *
context_search(name)          /* search for a context (openshape) */
{                             /* within the current context */
      symbol      *sym;
      char        *donald_full_name();

      if (name == PARENT)
            return SymEnd;
      if (! (sym=look_up(name)))    /* not found -> error */
            don_err(UndeclareID, donald_full_name(name));
      if (sym->type != OPENSHAPE)   /* not openshape -> error */
            don_err(NotOpenshape, donald_full_name(name));
      return sym;
}
```

new_symbol

```
symbol *
new_symbol(name)        /* creat a new symbol in current context */
      char        *name;      /* name of object */
{
      symbol      *sym;
      char        *donald_full_name();

      /* Make sure it is new */
      if (look_up(name))
            don_err(RedeclareID, donald_full_name(name));
      sym = (symbol *) getmem(sizeof(symbol));
      sym->name = name;
      sym->type = UNDEFINED;        /* not defined */
      sym->derived = SymEnd;        /* no component */
      sym->next = SymbolTable;
      return SymbolTable = sym;
}
```

look_up

```
symbol *
look_up(name)           /* look up the symbol in current context */
      char        *name;      /* name of object to be searched */
{
      symbol      *sym;

      for (sym = SymbolTable; sym != SymEnd; sym = sym->next) {
            if (strcmp(sym->name, name) == 0)
                  break;            /* found */
      }
      return sym;             /* sym == SymEnd if not found */
```

```
}
```

declare

```
declare(type, expr)                    /* declare variables */
      int         type;                /* type of variable */
      tree        expr;                /* identifier expr(-list) */
{
      char        *eden_full_name();
      symbol      *look_up(), *context_search(), *new_symbol();
      symbol      *sym;

      if (expr != EMPTYTREE) {
            switch (expr->op) {
            case OP_ID:
                  sym = new_symbol(Lexpr);
                  sym->type = type;
                  eden_declare(eden_full_name(Lexpr), type);
                  break;

            case OP_SLASH:
                  change_context(context_search(Lexpr->left));
                  declare(type, Rexpr);
                  resume_context();
                  break;

            case OP_COMMA:
                  declare(type, Lexpr);
                  declare(type, Rexpr);
                  break;

            default:
                  don_err(IdListExpect, 0);
                  break;
            }
      }
}
```

declare_segment

```
/* create a new graphics segment to display the object xxx */
/* the pointer "&xxx" is used as the unique segment id */
static char new_object[] = "new_object(&%s);\n";
#define declare_segment(id) printf(new_object,id)
```

declare_attribute

```
/* create a new attribute variable for the segment &xxx */
/* the variable is named Axxx */
static char new_attr[] = "A%s = [];\n";
#define declare_attribute(id) printf(new_attr,id)
```

declare_action

```
/*
** proc P_xxx : xxx, A_xxx
** {
**         plot_point(&xxx, &A_xxx);
** }
**
** Either plot_point, plot_line or plot_shape.
** "xxx" will be replaced by the EDEN full name.
** "A_xxx" is the attribute of "xxx".
*/
static char new_action[]
      = "proc P%s : %s, A%s\n{\n\t%s(&%s, &A%s);\n}\n";

#define declare_action(id, action_name) \
      printf(new_action, id, id, id, action_name, id, id)
```

eden_declare

```
eden_declare(id, type)
      char        *id;
      int         type;
{
      switch (type) {
      case POINT:
            declare_attribute(id);
            declare_segment(id);
            declare_action(id, "plot_point");
            break;

      case LINE:
            declare_attribute(id);
            declare_segment(id);
            declare_action(id, "plot_line");
            break;

      case SHAPE:
            declare_attribute(id);
            declare_segment(id);
            declare_action(id, "plot_shape");
            break;

      case OPENSHAPE:
            break;
      }
      declare_openshape();
}
```

declare_openshape

```
            /* sym is [ OPENSHAPE, sym/a, sym/b, ... ] */
declare_openshape()     /* rewrite the openshape definition */
{                       /* to build the structural dependency */
      symbol      *s;
      char        *name;

      /*----- print the context name -----*/
      /* indx == 0 --> root */
      printf("%s is [ OPENSHAPE", indx ? eden_full_name(0): "_");

      s = indx ? context[indx].sym->derived : *context[0].table;

      for ( ; s != SymEnd; s = s->next)
```

```
            printf(", &%s", eden_full_name(s->name));
      printf(" ];\n");
}
```

print_all_symbols

```
print_all_symbols()                          /* for debug only */
{
      fprintf(stderr, "Symbol Table :-\n");
      print_symbol_table(0, *context[0].table);
      fprintf(stderr, "Symbol Table End.\n");
}
```

print_symbol_table

```
print_symbol_table(level, symtab)            /* for debug only */
      int         level;
      symbol      *symtab;
{
      symbol      *sym;
      int         i;

      for (sym = symtab; sym != SymEnd; sym = sym->next) {
            for (i = level; i; --i)
                  fprintf(stderr, "|   ");
            fprintf(stderr, "%s %d\n", sym->name, sym->type);
            if (sym->type == OPENSHAPE)
                  print_symbol_table(level + 1, sym->derived);
      }
}
```

donald_full_name

```
char *
donald_full_name(id)          /* returns the DoNaLD full name */
      char        *id;        /* prepend the current context to id */
{
      static char name[256];      /* tmp string */
      char        *s = name;
      char        *t;
      int         i;
      int         len;
      int         id_len;

      id_len = id ? strlen(id) + 2 : 1;

      for (i = 1; i <= indx; i++) {
            t = context[i].sym->name;
            len = strlen(t);
            if (&name[256] - s < id_len + len)
                  don_err(Unclassified, "variable name too long");
            strcpy(s, t);
            s += len;
            *s++ = '/';
      }
      strcpy(s, id);
      return name;
}
```

eden_full_name

```
char *
eden_full_name(id)              /* returns the EDEN full name */
      char        *id;          /* prepend the current context to id */
{
      static char name[256];        /* tmp string */
      char        *s = name;
      char        *t;
      int         i;
      int         len;
      int         id_len;

      /* if (id == 0) ==> context name */
      id_len = id ? strlen(id) + 2 : 0;

      for (i = 1; i <= indx; i++) {
            t = context[i].sym->name;
            len = strlen(t);
            if (&name[256] - s < id_len + len)
                  don_err(Unclassified, "variable name too long");
            *s++ = '_';
            strcpy(s, t);
            s += len;
      }
      if (id) {
            *s++ = '_';
            strcpy(s, id);
      }
      return name;
}
```

expr_to_eden_name

```
char *
expr_to_eden_name(expr)
      tree        expr;         /* turn an id-expr into a string */
{                               /* assume expr was checked */
      int         count;
      char        *name;

      for (count = 0; expr->op == OP_SLASH; count++, expr = Rexpr)
            change_context(context_search(Lexpr->left));

      /* Now, expr->op should be OP_ID */
      name = eden_full_name(Lexpr);
      while (count--)
            resume_context();
      return name;
}
```

tree.c

```
#include <stdio.h>
#include "symbol.h"
#include "error.h"
#include "oper.h"
#include "tree.h"

/*
** tree0, tree1, tree2 & tree3 create a new tree node
** having 0-3 children respectively
*/
```

tree3

```
tree
tree3(op, l, m, r)
      int         op;         /* if...then...else/rot(,,)/etc. */
      tree        l, m, r;    /* left, mid and right children */
{
      tree        ptr;

      ptr = (tree) getmem(sizeof(node3));
      ptr->type  = UNDEFINED;
      ptr->op    = op;
      ptr->left  = l;
      ptr->right = r;
      ptr->mid   = m;
      return ptr;
}
```

tree2

```
tree
tree2(op, l, r)
      int         op;         /* operator: + - * / etc. */
      tree        l, r;       /* left and right children */
{
      tree        ptr;

      ptr = (tree) getmem(sizeof(node2));
      ptr->type  = UNDEFINED;
      ptr->op    = op;
      ptr->left  = l;
      ptr->right = r;
      return ptr;
}
```

tree1

```
tree
tree1(op, l)
      int         op;         /* id, inumber, etc. */
      tree        l;          /* left child */
{
      tree        ptr;

      ptr = (tree) getmem(sizeof(node1));
```

```
        ptr->type = UNDEFINED;
        ptr->op   = op;
        ptr->left = l;
        return ptr;
}
```

tree0

```
tree
tree0(op)
        int         op;         /* e.g. inf */
{
        tree        ptr;

        ptr = (tree) getmem(sizeof(node0));
        ptr->type = UNDEFINED;
        ptr->op   = op;
        return ptr;
}
```

match_operator

```
static struct trans trans[] = { TRANSLAT } ;  /* see oper.h */
#define match_operator(op, s) \
        (s = trans[(int)op].op_str, trans[(int)op].order)
```

nary

```
nary(opcode)
        int         opcode;
{
        switch (trans[opcode].order) {
        case UNARY:         return 1;
        case BINARY:
        case INFIX:         return 2;
        case OP_IF:
        case TRIARY:        return 3;
        default:            return 0;
        }
}
```

freetree

```
freetree(expr)              /*----- free the memory of a tree -----*/
        tree        expr;
{
        char        *op_string;         /* dummy */

        switch (match_operator(expr->op, op_string)) {
        case OP_ID:
                            if (Lexpr) free(Lexpr);
        case OP_RNUMBER:
        case OP_INUMBER:
        case OP_INF:
                            break;
        case OP_IF:
        case TRIARY:
                            freetree(Mexpr);  /* tri-ary */
        case OP_SLASH:
        case INFIX:
        case BINARY:
                            freetree(Rexpr);  /* binary */
        case UNARY:
```

```
                        freetree(Lexpr);  /* unary */
                        break;
        default:
                        don_err(Impossible, "free_tree");
                        break;
        }
        free(expr);             /* free itself */
}
```

lastchar

```
#define lastchar(s) s[strlen(s)-1]
```

print_comma

```
print_comma() { printf(", "); }
```

print_rparen

```
print_rparen() { printf(")"); }
```

print_tree

```
print_tree(expr)
        tree        expr;
{
        int         order;
        char        *op_string;
        char        *expr_to_eden_name();

        if (expr == EMPTYTREE) return;

        switch (expr->op) {
        case OP_ID:
        case OP_SLASH:
                printf("%s", expr_to_eden_name(expr));
                return;

        case OP_INUMBER:        /* integer number */
                printf("%d", Ivalue(expr));
                return;

        case OP_RNUMBER:        /* integer number */
                printf("%lf", Rvalue(expr));
                return;

        case OP_IF:                     /* if .. then .. else .. */
                print_tree(Mexpr);
                printf(" ? ");
                print_tree(Lexpr);
                printf(" : ");
                print_tree(Rexpr);
                return;
        }

        /* otherwise, node is operator */
        switch (order = match_operator(expr->op, op_string)) {
        case OP_INF:
                printf("%s", op_string);
                return;

        case UNARY:                     /* unary operator */
                printf("%s", op_string);
                print_tree(Lexpr);
                if (lastchar(op_string) == '(')
                        print_rparen();
```

```
            return;

        case BINARY:                    /* binary operator */
            printf("%s", op_string);
            print_tree(Lexpr);
            print_comma();
            print_tree(Rexpr);
            print_rparen();
            return;

        case INFIX:
            print_tree(Lexpr);
            printf("%s", op_string);
            print_tree(Rexpr);
            return;

        case TRIARY:                    /* 3-ary */
            printf("%s", op_string);
            print_tree(Lexpr);
            print_comma();
            print_tree(Mexpr);
            print_comma();
            print_tree(Rexpr);
            print_rparen();
            return;

        default:
            fprintf(stderr, "%d", order);
            don_err(Impossible, "print_tree");
        }
}
```

define

```
define(expr1, expr2)
        tree        expr1;
        tree        expr2;
{
        if (expr1 != EMPTYTREE) {
            switch (expr1->op) {
            case OP_ID:
            case OP_SLASH:
                if (is_shape(expr1) && is_openshape(expr2)){
                    print_tree(expr1);
                    printf(" is open2shape(");
                    print_tree(expr2);
                    printf(");\n");
                } else if (expr1->type == expr2->type) {
                    print_tree(expr1);
                    printf(" is ");
                    print_tree(expr2);
                    printf(";\n");
                } else
                    don_err(TypeMismatch, 0);
                break;

            case OP_COMMA:
                define(expr1->left, expr2->left);
                define(expr1->right, expr2->right);
                break;

            default:
                don_err(Impossible, "define");
```

```
                    break;
                }
            }
        }
```

utility.c

```
/**********************************
 *                                *
 *      Utilities Functions       *
 *                                *
 **********************************/

#include <strings.h>
#include "error.h"
```

strsave

```
char *
strsave(s)              /* save a string in a permanent place */
      char       *s;
{
      extern  char      *strcpy();
      extern  char      *getmem();

      return strcpy(getmem(strlen(s)+1), s);
}
```

getmem

```
char *
getmem(size)                    /* get memory of size */
{
      char       *ptr;

      if (!(ptr = (char *) malloc(size)))
            don_err(OutOfMemory, 0);
      return ptr;
}
```

drawgfx.e

objtype

```
func objtype {                              /* return object type */
      switch (type($1)) {
      case "@":      return @;
      case "int":    return INT;
      case "list":   return $1[1] == INF ? INT : $1[1];
      }
}
```

new_object

```
proc new_object         /* create a graphics segment for the object */
{                       /* $1=pointer to the object */
      create_retained_segment($1);  /* use the pointer as unique id */
      close_retained_segment();
}
```

set_attributes

```
proc set_attributes           /* set attributes for segment */
{                             /* it is ad-hoc */
      auto i;
      auto attr;

      attr = *$1;
      for (i = 1; i <= attr#; i++) {
            execute(attr[i]);
      }
}
```

reset_attributes

```
proc reset_attributes         /* reset the default attributes */
{
      set_linestyle(SOLID);
      set_linewidth(0.5);
}
```

plot_point

```
plot_point is plot_shape;           /* the same as plot_shape */
```

plot_line

```
plot_line  is plot_shape;           /* the same as plot_shape */
```

plot_shape

```
proc plot_shape {
      auto       SegName;
      auto       obj, attr;

      SegName = $1;      /* segment id = pointer to object */
      obj     = *$1;     /* the object data, e.g. [LINE, ...] */
      attr    = $2;
```

```
        delete_retained_segment(SegName);    /* delete the old segment */

        /*
         * We can use the address of the variable (SegName) as
         * the segment name because the address is unique and
         * constant in EDEN environment.
         * Otherwise, we have to generate a unique id no.
         * for the segment and store in a value variable.
         */
        create_retained_segment(SegName);    /* create the new segment */
        set_attributes(attr);                /* set attributes (from A_xxx) */
        draw_shape(obj);                     /* draw primitives */
        close_retained_segment();            /* finish drawing, close segment */
        reset_attributes();                  /* reset attributes */
}
```

draw_shape

```
proc draw_shape
{
        auto    x1, y1, x2, y2, i, entity;

        entity = $1;
        switch (entity[TYPE]) {
        case 'C':                            /* a Cartesian point */
                get_xy_coord(entity, &x1, &y1);      /* get x, y coord */
                moveto(x1, y1);              /* two end-points of the line */
                lineto(x1, y1);              /* are the same, i.e. a point */
                break;

        case 'L':
                get_xy_coord(entity[X], &x1, &y1);  /* get 2 endpoints */
                get_xy_coord(entity[Y], &x2, &y2);
                moveto(x1, y1);
                lineto(x2, y2);
                break;

        case 'S':
                for (i=2; i <= entity#; i++)
                        draw_shape(entity[i]);       /* recursive draw */
                break;

        default:
                writeln("ERROR: draw_shape(",entity,");");
                break;
        }
}
```

get_xy_coord

```
proc get_xy_coord       /* get two coord of a point $1 */
{                       /* store in variables pointed to by $2, $3 */
        switch (objtype($1)) {
        case 'C':       /* Cartesian */
                *$2 = $1[X];
                *$3 = $1[Y];
                break;

        case 'P':       /* Polar */
                *$2 = *$3 = 0;    /* not implement polar coordinates yet */
                break;

        default:
```

```
            *$2 = *$3 = 0;
            break;
        }
}
```

moveto

```
func moveto { move_abs_2(float($1), float($2)); }
```

lineto

```
func lineto { line_abs_2(float($1), float($2)); }
```

init.e

```
setbuf(stdout, 0);      /* unbuffered the output */

OFF   = 0;
ON    = 1;

TYPE = 1;
X    = 2;
Y    = 3;

lib = "./";                     /* the directory the EDEN routines */
include(lib//"suncore.e.h");    /* useful constants for graphics */
include(lib//"drawgfx.e");      /* graphics utilities */

INF       = '!';                /* [ INF ]             */
REAL      = 'R';                /* [ REAL, ??? ]       */
CART      = 'C';                /* [ CART, X, Y ]      */
POLAR     = 'P';                /* [ POLAR, R, A ]     */
LINE      = 'L';                /* [ LINE, P1, P2 ]    */
SHAPE     = 'S';                /* [ SHAPE, ??? ]      */
OPENSHAPE = 'O';                /* [ OPENSHAPE, ... ] */
```

cart

```
      /* POINT */
func  cart                /* INT $1, $2; */
{
      return [CART, $1, $2];              /* [ CART, X, Y ] */
}
```

polar

```
      /* POINT */
func  polar               /* INT $1, $2; */
{
      return [POLAR, $1, $2];             /* [ POLAR, R, A ] */
}
```

line

```
      /* LINE */
func  line                /* POINT $1, $2; */
{
      return [LINE, $1, $2];              /* [ LINE, P1, P2 ] */
}
```

open2shape

```
      /* SHAPE */
func  open2shape          /* OPENSHAPE $1 */
{
      auto        shape, entity, i;

      shape = [SHAPE];
      i = 2;
      for (i = 2; i <= $1#; i++) {
```

```
            entity = *($1[i]);
            switch (entity[TYPE]) {
            case 'I':
            case 'R':   break;
            case 'O':   append shape, open2shape(entity);
                        break;
            default:    append shape, entity;
                        break;
            }
      }
      return shape;
}
```

rot

```
      /* ENTITY */
func  rot    /* ENTITY $1; POINT $2; INT(REAL) $3 */
{
      auto i;

      switch ($1[TYPE]) {
      case 'C':   $1[X] += 100;
      case 'P':   writeln("rot(",$1,",",$2,",",$3,")");
                  break;
      case 'I':
      case 'R':   writeln("rot(): Can't happen on ", $1[TYPE]);
                  break;
      case 'L':   $1[X] = rot($1[X],$2,$3);
                  $1[Y] = rot($1[Y],$2,$3);
                  break;
      case 'O':   $1 = open2shape($1);
      case 'S':   for (i=2; i<=$1#; i++)
                  $1[i] = rot($1[i], $2, $3);
                  break;
      }
      return $1;
}
```

scale

```
func  scale
{
      return $1;        /* NOT IMPLEMENTED YET */
}
```

line_reverse

```
            /* LINE */
func  line_reverse /* LINE $1; */
{
      auto        tmp;

      tmp = $1[Y];
      $1[Y] = $1[X];
      $1[X] = tmp;
      return $1;
}
```

dotx

```
            /* POINT */
func  dotx   /* POINT $1; */
{
      if ($1[TYPE] == CART) {
```

```
            $1[Y] = 0.0;
            return $1;
      }
}
```

doty

```
            /* POINT */
func doty   /* POINT $1; */
{
      if ($1[TYPE] == CART) {
            $1[X] = 0.0;
            return $1;
      }
}
```

dot1

```
func dot1
      { return $1[X]; }
```

dot2

```
func dot2
      { return $1[Y]; }
```

vector_add

```
func vector_add
      { return [CART, $1[X] + $2[X], $1[Y] + $2[Y]]; }
```

vector_sub

```
func vector_sub
      { return [CART, $1[X] - $2[X], $1[Y] - $2[Y]]; }
```

scalar_mult

```
func scalar_mult
      { return [CART, float($1[X]) * $2, float($1[Y]) * $2]; }
```

scalar_div

```
func scalar_div
      { return [CART, float($1[X]) / $2, float($1[Y]) / $2]; }
```

scalar_mod

```
func scalar_mod
      { return [CART, int($1[X]) % int($2), int($1[Y]) % int($2)]; }
```

start

```
proc start              /* start up procedure */
{
      auto        i, j;

      /* initialize SunCore graphics package */
      initialize_core(BUFFERED, SYNCHRONOUS, TWOD);
      surf = new_view_surface(TRUE, "100,100,500,500,0,0,64,64,0");
      initialize_view_surface(surf, FALSE);
      select_view_surface(surf);
      set_ndc_space_2(1.0, 1.0);    /* the whole surface is accessible */
      set_viewport_2(0.1, 0.9, 0.1, 0.9);    /* leave blank at 4 sides */
      set_window(0.0, 1000.0, 0.0, 1000.0); /* world is 1000x1000 */
      initialize_device(PICK, 1);           /* show the cursor */
      set_echo_surface(PICK, 1, surf);
      set_echo(PICK, 1, 2);

      create_retained_segment(&GRID);      /* draw grid lines */
            set_linestyle(DOTTED);
```

```
            for (i = 0; i <= 1000; i += 100) {
                  moveto(i, 0);
                  lineto(i, 1000);
                  moveto(0, i);
                  lineto(1000, i);
            }
      close_retained_segment();
      set_linewidth(1);
}
```

quit

```
proc quit                        /* terminate SunCore gracefully */
{
      save_raster(surf, "raster_file");    /* store image in a file */
      deselect_view_surface(surf);
      terminate_device(PICK, 1);
      terminate_view_surface(surf);
      terminate_core();
}

start();                         /* run the start up procedure */
```

suncore.e.h

```
        /*----- This is constant definitions header file -----*/

/* See /usr/include/usercore.h */
FALSE           = 0;
TRUE            = 1;

STRING          = 0;
CHARACTER       = 1;
MAXVSURF        = 5;      /* view surfaces; maximum number of */
PARALLEL        = 0;      /* transform constants */
PERSPECTIVE     = 1;
NONE            = 1;      /* segment types */
XLATE2          = 2;
XFORM2          = 3;
XLATE3          = 2;
XFORM3          = 3;
SOLID           = 0;      /* line styles */
DOTTED          = 1;
DASHED          = 2;
DOTDASHED       = 3;
CONSTANT        = 0;      /* polygon shading modes */
GOURAUD         = 1;
PHONG           = 2;
PICK            = 0;      /* input device constants */
KEYBOARD        = 1;
BUTTON          = 2;
LOCATOR         = 3;
VALUATOR        = 4;
STROKE          = 5;
ROMAN           = 0;      /* vector font select constants */
GREEK           = 1;
SCRIPT          = 2;
OLDENGLISH      = 3;
STICK           = 4;
SYMBOLS         = 5;
GALLANT         = 0;      /* raster font constants */
GACHA           = 1;
SAIL            = 2;
GACHABOLD       = 3;
CMR             = 4;
CMRBOLD         = 5;
OFF             = 0;      /* char justify constants */
LEFT            = 1;
CENTER          = 2;
RIGHT           = 3;
NORMAL          = 0;      /* rasterop selection */
XORROP          = 1;
ORROP           = 2;
PLAIN           = 0;      /* polygon interior style */
SHADED          = 1;
BASIC           = 0;      /* Core output levels */
BUFFERED        = 1;
DYNAMICA        = 2;
DYNAMICB        = 3;
```

```
DYNAMICC        = 4;
NOINPUT         = 0;      /* Core input levels */
SYNCHRONOUS     = 1;
COMPLETE        = 2;
TWOD            = 0;      /* Core dimensions */
THREED          = 1;
                          /* View Surface */
VWSURF_NEWFLG   = 1;
```

Sample DoNaLD Specification

```
#       The following definitions define a room ...
#       ... and some objects inside the room.
#

#       The room is a rectangle

int         width, length           # the dimension of room
point       NW, NE, SW, SE          # the 4 corners of room
line        N1, N2, S, E, W         # the 4 walls of room
                                    # North wall composites of 2 walls & a door

#       Every room has a door
#
openshape door                      # declare the door

within door {
      point       hinge, lock       # a door has a hinge, and a lock
      line        door              # the door itself
      int         width             # the size of the door
      int         open              # use it as a flag, telling whether ...
                                    # ... the door has opened
      door  = [hinge, lock]
      lock  = hinge + (if open then {0, -width} else {width, 0})
      hinge = ~/NW + {15, -10}      # set the coordinate of the hinge
      open  = 1                     # the door has opened
      width = 200                   # the size of the door
}

N1 = [NW, {door/hinge.1, NW.2}]
N2 = [{door/hinge.1+door/width, NW.2}, NE]
S  = [SW, SE]
W  = [NW, SW]
E  = [NE, SE]

SW = {100, 100}
SE = SW + {width, 0}                # The other 3 corners are relative
NE = SW + {width, length}           # to South-West corner.
NW = SW + {0, length}               #

width, length = 800, 800           # 800*800 pts

##########################################################################
openshape table  # There is a table inside the room.

within table {
      int         width, length     # the dimension of table
      point       NW, NE, SW, SE    # the 4 corners of table
      line        N, S, E, W        # the 4 sides of table

      N = [NW, NE]
      S = [SW, SE]
      W = [NW, SW]
      E = [NE, SE]
```

```
        SE = SW + {width, 0}            # the other 3 corners are
        NE = SW + {width, length}       # relative to SW corner
        NW = SW + {0, length}           #

        ##### table/lamp #####

        openshape lamp                  # There is a lamp on the table
        within lamp {
              point       centre        # center of the table
              int         size          # size of the lamp
              int         half          # half of size

              size   = 50
              half   = size div 2
              centre = (~/SW + ~/NE) div 2   # at the centre of table

              # L1-L8 forms an octagon.
              line L1, L2, L3, L4, L5, L6, L7, L8

              L1 = [centre + {size, -half}, centre + {size, half}]
              L2 = [L1.2, centre + {half, size}]
              L3 = [L2.2, centre + {-half, size}]
              L4 = [L3.2, centre + {-size, half}]
              L5 = [L4.2, centre + {-size, -half}]
              L6 = [L5.2, centre + {-half, -size}]
              L7 = [L6.2, centre + {half, -size}]
              L8 = [L7.2, L1.1]
        }

        width, length = 300, 300       # let the size of table be 300*300 pts.
        SW = (~/SW + ~/NE) div 2        # place the SW of table at the center of the room.
}

#############################################################################
point       plug
point       plug1, plug2                # there are 2 plugs.
plug1 = (S.1+S.2) div 2                 # middle of South wall
plug2 = (E.1+E.2) div 2                 # middle of East wall

line        cable                       # cable connects the table/lamp and the plug
int         cablelength
cable = [plug, table/lamp/centre]
plug  = plug1                           # let cable connects to plug1
cablelength = 600
#
#       Now set the line style of cable to dotted line.
#       Because many DoNaLD commands hasn't been implemented,
#       I just directly access EDEN using the non-standard '?' command.
#       A_cable is the attribute of the line cable.
? append A_cable, "set_linestyle(DASHED);";


#############################################################################

openshape desk                          # there is a desk in the room

within desk {
      int         width, length         # the size of the desk
      point       NW, NE, SW, SE        # the 4 corners of the desk
      line        N, S, E, W            # the 4 edges of the desk

      N = [NW, NE]
      S = [SW, SE]
```

```
        W = [NW, SW]
        E = [NE, SE]

        width, length = 250, 350

        # initially the desk is placed at (15,15) of relative to the room's SW.
        SW = ~/SW + {15, 15}
        SE = SW + {width, 0}            # the other 3 corners are ...
        NW = SW + {0, length}           # ... relative to the room's SW corner
        NE = NW + {width, 0}            #

        ##### desk/drawer #####

        openshape drawer                # the desk has a drawer

        within drawer {
              int         k             # k --   a parameter describes
                                        #        the condition of drawer
                                        #               1 --> closed
                                        #               larger k --> more open

              int         width, length     # the size of drawer
              point       NW, NE, SW, SE    # the 4 corners of the drawer
              line        N, S, W, E        # the 4 edges of the drawer

              # the size of the drawer is always a ratio k to the desk
              width  = ~/length div 3
              length = ~/width - ~/width div k
              k      = 2                    # initially, the draw is half open

              NW = ~/NE                     # NW is always at desk's NE
              SW = NW - {0, width}          # the other 3 corners are ...
              SE = SW + {length, 0}         # ... relative to NW
              NE = NW + {length, 0}         #

              N = [NW, NE]
              S = [SW, SE]
              W = [NW, SW]
              E = [NE, SE]
        }
}

#########################################################################
# The following sets up auto-checking action.
# *** It is NOT a standard DoNaLD program.
# *** But it shows the ability of doing so.
#
# Since many features of DoNaLD hasn't been implemented,
# ... the program simply defines an EDEN action directly
# ... through the '?' statement (non-standard DoNaLD feature).
#
# Note that the naming convention is different in DoNaLD & in EDEN.
#
# inside_rectangle is a function testing whether a point is inside
# a rectangle, if it is then return 1 (true), otherwise 0 (false).

?       func inside_rectangle           /* test (x,y) inside a rectangle */
?            /* x, y, xmin, xmax, ymin, ymax */
?       {
?            return ($1 >= $3 && $1 < $4 && $2 >= $5 && $2 < $6);
?       }
```

'door_hit_table' tests whether the door will hit the table.
If so, a warning message is printed on screen.
It is an action depends on `door' and `table'.
Only these two objects are modified will invoke this action.

```
?       proc door_hit_table: _door, _table
?       {
?               /* local variables */
?               auto h_x, h_y;     /* hinge */
?               auto l_x, l_y;     /* lock */
?               auto xmin, ymin;  /* table SW corner */
?               auto xmax, ymax;
?
?               get_xy_coord(_table_SW, &xmin, & ymin);
?               xmax = xmin + _table_width;
?               ymax = ymin + _table_length;
?               get_xy_coord(_door_lock, & l_x, & l_y);
?               get_xy_coord(_door_hinge, & h_x, & h_y);
?               /* if door/hinge or door/lock inside table */
?               /* then implies hit. */
?               if (   inside_rectangle(l_x, l_y, xmin, xmax, ymin, ymax)
?                   || inside_rectangle(h_x, h_y, xmin, xmax, ymin, ymax)
?               )       writeln("!!!  DOOR HITS THE TABLE  !!!");
?       }
```

The following action detects whether ...
... the cable is too long (which may not be available).
#

```
?       proc check_cable: _cable, _cablelength
?       {
?               auto v, xx, yy;
?               v = vector_sub(_cable[2], _cable[3]);
?               xx = v[2]*v[2];
?               yy = v[3]*v[3];
?               if (xx + yy > _cablelength * _cablelength) {
?                       writeln("\n!!!  CABLE NOT LONG ENOUGH  !!!");
?                       writeln("!!!  Cable length is ", _cablelength, ".");
?               }
?       }
```

Sample DoNaLD Translator Output

This is a part of the translation of the Sample DoNaLD Specification.
Comments (in *Times Roman Italic* font) are added.

```
_ is [ OPENSHAPE, &_width ];                    Declare integer width
_ is [ OPENSHAPE, &_length, &_width ];          Declare integer length
                                    Note: no actions are associated with integers
                                                Declare point NW
A_NW = [];                                      Attribute of point NW
                                                Empty list [] means default attr
new_object(&_NW);                               Create a new graphics segment
proc P_NW : _NW, A_NW                           Specify action to draw point NW
{
      plot_point(&_NW, &A_NW);                  Call plot_point to draw
}
_ is [ OPENSHAPE, &_NW, &_length, &_width ];    Add NW to the room "_"

A_NE = [];                                      Delcare point NE
new_object(&_NE);
proc P_NE : _NE, A_NE
{
      plot_point(&_NE, &A_NE);
}
_ is [ OPENSHAPE, &_NE, &_NW, &_length, &_width ];
      .
      .
      .
A_door_door = [];                               Delcare line door/door
new_object(&_door_door);                        Create a new graphics segment
proc P_door_door : _door_door, A_door_door
{
      plot_line(&_door_door, &A_door_door);     Call plot_line to draw a line
}
_door is [ OPENSHAPE, &_door_door,              Construct structural dependency
           &_door_lock, &_door_hinge ];         by adding _door_door to _door
      .
      .
      .
                                                Give definition to door/door
_door_door is line(_door_hinge, _door_lock);    door/door=[door/hinge,door/lock]

                                Give definition to door/lock:
                                lock = hinge + (if open then {0, -width} else {width, 0})
_door_lock is vector_add(_door_hinge,
      (_door_open ? cart(0, - _door_width) : cart(_door_width, 0)));

                                Give definition to hinge, open, width, etc:
_door_hinge is vector_add(_NW, cart(15, - 10));
_door_open is 1;
_door_width is 200;
_N1 is line(_NW, cart(dot1(_door_hinge), dot2(_NW)));
_N2 is line(cart(dot1(_door_hinge) + _door_width, dot2(_NW)), _NE);
_S is line(_SW, _SE);
_W is line(_NW, _SW);
```

```
_E is line(_NE, _SE);
_SW is cart(100, 100);
_SE is vector_add(_SW, cart(_width, 0));
_NE is vector_add(_SW, cart(_width, _length));
_NW is vector_add(_SW, cart(0, _length));

autocalc = OFF;                    By switching off the auto-calculate mechanism,
_width is 800;                     we can change more than one definition at a single trial.
_length is 800;
autocalc = ON;                     Switch on the auto-calculate mechanism to update changes.
      .
      .
      .
                                   The following actions monitor conditions and report to the user.
      func inside_rectangle  /* test (x,y) inside a rectangle */
      /* x, y, xmin, xmax, ymin, ymax */
      {
            return ($1 >= $3 && $1 < $4 && $2 >= $5 && $2 < $6);
      }
      proc door_hit_table: _door, _table
      {
            /* local variables */
            auto h_x, h_y;   /* hinge */
            auto l_x, l_y;   /* lock */
            auto xmin, ymin; /* table SW corner */
            auto xmax, ymax;

            get_xy_coord(_table_SW, &xmin, & ymin);
            xmax = xmin + _table_width;
            ymax = ymin + _table_length;
            get_xy_coord(_door_lock, & l_x, & l_y);
            get_xy_coord(_door_hinge, & h_x, & h_y);
            /* if door/hinge or door/lock inside table */
            /* then implies hit.                       */
            if (   inside_rectangle(l_x, l_y, xmin, xmax, ymin, ymax)
                || inside_rectangle(h_x, h_y, xmin, xmax, ymin, ymax)
            )     writeln("!!!  DOOR HITS THE TABLE  !!!");
      }
      proc check_cable: _cable, _cablelength
      {
            auto v, xx, yy;
            v = vector_sub(_cable[2], _cable[3]);
            xx = v[2]*v[2];
            yy = v[3]*v[3];
            if (xx + yy > _cablelength * _cablelength) {
                  writeln("\n!!!  CABLE NOT LONG ENOUGH  !!!");
                  writeln("!!!  Cable length is ", _cablelength, ".");
            }
      }

                                   Human interactions:
_door_open is 0;
_table_SW is vector_add(_SW, cart(400, 500));
quit();                            End the DoNaLD session, quit SunCore graphics package gracefully.
```

Appendix E

Listing of the EDEN Interpreter

The source code of the EDEN interpreter is divided into two modules: **main module**—contains the main code (includes lexical analyser, parser, pseduo machine), and **custom module**—contains user-customizable code (e.g. user can bind C functions).

Codes of main module

Imakefile	1	makefile for compiling EDEN main module
lex.c	6	lexical analyser
builtin.c	9	builtin function codes
code.c	15	EDEN-to-pseudo-machine translator
eval.c	18	definition manager (evaluation strategy)
heap.c	20	heap memory manager
global.q.c	21	support functions
lib.c	22	EDEN/C interface
main.c	23	main program/error handler etc.
machine.c	26	most of the pseudo machine
refer.c	36	dependeny maintainer
symbol.c	37	symbol table manager
type.c	39	type conversion functions
eden.h	41	header file for data structures and macros
builtin.h	43	builtin function bindings
global.q.h	44	support functions
symptr.q.h	45	support functions
hash.h	46	hash table declaration
inst.h	47	pseduo instruction declaration
keyword.h	48	EDEN keywords
symptr.q.c	49	support functions
entry.q.c	50	support functions
entry.q.h	51	support functions

Codes of custom module

Imakefile	52	makefile for compiling custom module
curses.c	53	EDEN/Curses I/F code
custom.c	55	user-customizable code
sungraf.c	56	EDEN/SunCore I/F code
ww_builtin.c	58	EDEN/WW I/F code
curses.h	64	EDEN/Curses I/F bindings
customlib.h	66	master EDEN/C bindings (includes Curses/SunCore etc.)
sungraf.h	68	EDEN/SunCore I/F bindings
ww_builtin.h	70	EDEN/WW I/F bindings
ww_info.h	71	used by WW I/F
wwinfo.h	72	used by WW I/F

Index **78** index of functions in the listing.

```
/***** Imakefile for making EDEN main module *****/

#ifdef sparc
EdenLib=eden-v3s4
#else
EdenLib=eden-v3
#endif

LibDir=$(HOME)/lib
L=$(LibDir)/lib$(EdenLib).a

YFLAGS  = -d
LFLAGS  =
CFLAGS  = -g -DDEBUG
AR      = ar

LIBOBJ = $L(yacc.o)     \
         $L(lex.o)      \
         $L(builtin.o)  \
         $L(code.o)     \
         $L(eval.o)     \
         $L(heap.o)     \
         $L(global.q.o) \
         $L(lib.o)      \
         $L(main.o)     \
         $L(machine.o)  \
         $L(refer.o)    \
         $L(symbol.o)   \
         $L(type.o)

LEXSRC  = lex.c
YACCSRC = yacc.y

INCLUDE = eden.h builtin.h global.q.h symptr.q.h hash.h inst.h keyword.h

LIBSRC  = builtin.c     \
          code.c        \
          eval.c        \
          heap.c        \
          global.q.c    \
          lib.c         \
          main.c        \
          machine.c     \
          refer.c       \
          symbol.c      \
          type.c

OTHERS  = symptr.q.c entry.q.c entry.q.h

SOURCE  = $(YACCSRC) $(LEXSRC) $(LIBSRC) $(INCLUDE) $(OTHERS)

printer=cs/psc
PRINTOPT = -n -uh -w78 -f ''

$(LibDir)/LIBEDEN: $(LIBOBJ)
        ranlib $L
        touch $(LibDir)/LIBEDEN

x.tab.h: y.tab.h
        -cmp -s x.tab.h y.tab.h || cp y.tab.h x.tab.h

$(LIBOBJ): x.tab.h eden.h

$L(yacc.o): $(YACCSRC)
/*
#       $(YACC) $(YFLAGS) $(YACCSRC)
```



```
#       $(CC) $(CFLAGS) -c y.tab.c
#       mv y.tab.o yacc.o
#       ar rv $@ yacc.o
#       rm -f y.tab.c yacc.o
*/

$L(lex.o)      : keyword.h
$L(refer.o)    : symptr.q.h
$L(eval.o)     : symptr.q.h
$L(symbol.o)   : global.q.h symptr.q.h symptr.q.c hash.h entry.q.h entry.q.c
$L(builtin.o)  : hash.h builtin.h

print:  $(SOURCE) Imakefile
        pg $(PRINTOPT) -h 'EDEN|(File: *  Page %)|PAGE #' $? \
        | lpr -P$(printer)
        @ touch print

printall: Imakefile $(SOURCE)
        print $?

save: $(SOURCE) Imakefile
        cp $? save /cs/acad/wmb/public/eden/MAIN
        chmod -R ug+rwX /cs/acad/wmb/public/eden/MAIN
        # rcp $? save ey@snow:EDEN/MAIN
        # rcp $? save csrrf@sol:EDEN/MAIN
        @ touch save

EDENsrc.a: $(SOURCE) Imakefile custom.c.sample customlib.h.sample
        @ echo SAVING $?
        @- chmod +w EDENsrc.a
        @ ar r EDENsrc.a $?
        @ chmod -w EDENsrc.a

clean:
        rm -f *.BAK *.bak *.o
```

```
%{
#include "eden.h"
#define code2(c1,c2)            code(c1); code(c2)
#define code3(c1,c2,c3)         code(c1); code(c2); code(c3)
#define fromto(F,T)             (F)[1] = (Inst)((T)-((F)+2))
extern char *textptr, textcode[];
extern symptr_QUEUE break_q, cont_q;
extern patch(), dispatch();
#define dispatch_break()        dispatch(progp, &break_q); code2(jmp, 0)
#define dispatch_continue()     dispatch(progp, &cont_q); code2(jmp, 0)
#define patch_break(mark,p)     patch(mark, p, &break_q)
#define patch_continue(mark,p)  patch(mark, p, &cont_q)
#define rts                     (Inst) 0
%}
%union {
        Datum *dp;      /* constants */
        symptr  sym;    /* symbol table ptr */
        Inst    *inst;  /* machine instruction */
        Int     narg;   /* number of arguments */
        Inst    fun;    /* binop */
}

%token  <sym>   UNDEF
%token  <sym>   REAL
%token  <sym>   INTEGER
%token  <sym>   CHAR
%token  <sym>   STRING
%token  <sym>   LIST
%token  <sym>   FORMULA
%token  <sym>   FUNCTION
%token  <sym>   PROCEDURE

%token  <sym>   CONST
%token  <sym>   AUTO PARA
%token  <narg>  LOCAL
%token  <sym>   FUNC PROC
%token  <sym>   VAR BLTIN LIB RLIB
%token  <sym>   BREAK CONTINUE
%token  <sym>   SWITCH CASE DEFAULT
%token  <sym>   DO FOR WHILE IF ELSE
%token  <sym>   SHIFT APPEND INSERT DELETE RETURN
%token  <narg>  ARG

%type   <sym>   id identifier
%type   <inst>  lvalue primary secondary
%type   <fun>   asgnop
%type   <inst>  and or colon
%type   <inst>  expr
%type   <inst>  expr1 expr2 end_expr2 expr3 end_expr3
%type   <narg>  switch
%type   <inst>  cases
%type   <inst>  stmt compound
%type   <inst>  asgn stmtlist
%type   <inst>  then else begin
%type   <narg>  action is def_begin def_end
%type   <inst>  defn declare_formula declare_action declare_relation
%type   <narg>  declare_local local_list
%type   <narg>  refer_opt id_list id_list_opt
%type   <narg>  arglist argument_list
%nonassoc IS TILDE_GT
%left   ','
%right  '=' PLUS_EQ MINUS_EQ
%right  '?'
%left   OR_OR  or
%left   AND_AND and
%left   EQ_EQ NOT_EQ
```



```
%left   '>' GT_EQ '<' LT_EQ
%left   '+' '-' SLASH_SLASH
%left   '*' '/' '%'
%nonassoc  NEGATE '!' PLUS_PLUS MINUS_MINUS '|' '&' ASTERISK
%%
program:    /* nothing */
        | stmt                  { code2(freeheap, rts);
                                  textptr = textcode;
                                  return 1; }
        | error                 { yyerrok; }
        ;

lvalue:
          id                    { $$ = code2(addr, $1); }
        | LOCAL                 { defnonly("local variable");
                                  $$ = code2(localaddr, $1);
                                }
        | '$'                   { defnonly("$");
                                  $$ = code2(localaddr, (Inst)0);
                                }
        | ARG                   { defnonly("$");
                                  $$ = code2(localaddr, (Inst)0);
                                  code3(pushint, (Inst)$1, indexcalc);
                                }
        | lvalue '[' expr ']'           { code(indexcalc); }
        | '*' primary    %prec ASTERISK { $$ = $2; }
        | '*' '&' lvalue                { $$ = $3; }
        | '(' expr ')'                  { $$ = $2; code(lookup_address); }
        | '(' lvalue ')'                { $$ = $2; }
        ;

asgn:     lvalue asgnop expr            { code($2); }
        | PLUS_PLUS lvalue              { code(pre_inc); $$ = $2; }
        | lvalue PLUS_PLUS              { code(post_inc); }
        | MINUS_MINUS lvalue            { code(pre_dec); $$ = $2; }
        | lvalue MINUS_MINUS            { code(post_dec); }
        ;

asgnop:   '='                           { $$ = assign; }
        | PLUS_EQ                       { $$ = inc_asgn; }
        | MINUS_EQ                      { $$ = dec_asgn; }
        ;

primary:
          lvalue                        { code(getvalue); }
        | secondary
        | '(' expr ')'                  { $$ = $2; }
        ;

secondary:
          primary '(' arglist ')'
                                        { code3(makelist, $3, eval); }
        | secondary '[' expr ']'
                                        { code(sel); }
        ;

stmt:
          defn
        | expr ';'              { code2(popd, freeheap); }
        | RETURN ';'            { defnonly("return");
                                  $$ = code2(pushUNDEF, rts); }
        | RETURN expr ';'       { defnonly("return");
                                  $$ = $2; code(rts); }
        | SHIFT ';'             { defnonly("shift");
                                  $$ = code3(localaddr, 0, shift);
                                }
```

```
        | SHIFT lvalue ';'          { $$ = $2; code(shift); }
        | APPEND lvalue ',' expr ';'
                                    { $$ = $2; code2(append, freeheap); }
        | INSERT lvalue ',' expr ',' expr ';'
                                    { $$ = $2; code2(insert, freeheap); }
        | DELETE lvalue ',' expr ';'
                                    { $$ = $2; code2(delete, freeheap); }
        | CASE CONST ':'            { if (inswitch)
                                        addentry($2, (Int) progp);
                                      else
                                        error("'case' used outside switch");
                                      $$ = progp;
                                    }
        | DEFAULT ':'               { if (inswitch)
                                        addentry(0, (Int) progp);
                                      else
                                        error("'default' used outside switch");
                                      $$ = progp;
                                    }
        | BREAK ';'                 { if (!inloop && !inswitch)
                                        error("'break' used outside loop/switch");
                                      $$ = progp;
                                      dispatch_break();
                                    }
        | CONTINUE ';'                      { if (!inloop)
                                                error("'continue' used outside loop");
                                              $$ = progp;
                                              dispatch_continue();
                                            }
        | switch expr cases stmt            { dispatch_break();
                                              fromto($3, progp);
                                              code(switchcode);
                                              codeswitch($1);
                                              patch_break($2, progp);
                                              $$ = $2;
                                              --inswitch;
                                            }

        | for
          expr1
          expr2 end_expr2
          expr3 end_expr3
          stmt                      { Inst * p = code(jmp);
                                      code($5 - (p + 2));
                                      fromto($4, $7);
                                      if ($3 != $4 - 2)
                                        fromto($4-2, progp);
                                      fromto($6, $3);
                                      patch_continue($2, $5);
                                      patch_break($2, progp);
                                      $$ = $2;
                                      --inloop;
                                    }

        | do stmt WHILE '(' expr ')' ';'
                                    { Inst * p = code(jpnz);
                                      code($2 - (p + 2));
                                      patch_continue($2, $5);
                                      patch_break($2, progp);
                                      $$ = $2;
                                      --inloop; }

        | while '(' expr ')' then stmt
                                    { Inst * p = code(jmp);
                                      code($3 - (p + 2));
                                      fromto($5, progp);
```



```
                                      patch_continue($3, $3);
                                      patch_break($3, progp);
                                      $$ = $3;
                                      --inloop; }

        | IF '(' expr ')' then stmt
                                    { fromto($5, progp);
                                      $$ = $3; }

        | IF '(' expr ')' then stmt
          else stmt                 { fromto($5, $8);
                                      fromto($7, progp);
                                      $$ = $3; }

        | compound
        | ';'                       { $$ = progp; }
        ;

while:    WHILE                     { inloop++;
                                      dispatch(progp, &cont_q);
                                      dispatch(progp, &break_q); }
        ;

do:       DO                        { inloop++;
                                      dispatch(progp, &cont_q);
                                      dispatch(progp, &break_q); }
        ;

for:      FOR '('                   { inloop++;
                                      dispatch(progp, &cont_q);
                                      dispatch(progp, &break_q); }
        ;

expr1:    ';'                       { $$ = progp; }
        | expr ';'                  { code(popd); }
        ;

expr2:    ';'                       { $$ = code2(jmp, 0); }
        | expr ';'                  { code2(jpz, 0); }
        ;

end_expr2: /* nothing */            { $$ = code2(jmp, 0); }
        ;

expr3:    ')'                       { $$ = code2(jmp, 0); }
        | expr ')'                  { code3(popd, jmp, 0); }
        ;

end_expr3: /* nothing */            { $$ = code2(jmp, 0); }
        ;

then:     /* nothing */             { $$ = code2(jpz, 0); }
        ;

else:     ELSE                      { $$ = code2(jmp, 0); }
        ;

compound: '{' stmtlist '}'          { $$ = $2; }
        ;

stmtlist: /* nothing */             { $$ = progp; }
        | stmtlist stmt
        ;

switch:   SWITCH '('                { $$ = (Int) entry_ptr;
                                      inswitch++;
```

```
                                dispatch(progp, &break_q);
                              }
        ;

cases:    ')'                 { $$ = code2(jmp, 0); }
        ;

begin:    /* nothing */       { $$ = progp; }
        ;

id:       identifier          { if (informula) addID($1); }
        | BLTIN
        | LIB
        | RLIB
        ;

identifier:
          VAR
        | FORMULA
        | FUNCTION
        | PROCEDURE
        ;

expr:
          CONST                         { $$ = code2(constpush, (Inst)$1); }
        | asgn          { if (informula)
                            error("assignment used inside formula");
                        }
        | primary       /* already coded in primary */
        | '&' lvalue    %prec NEGATE    { $$ = $2; }
        | '{' begin arglist '}'         { code2(makelist, (Inst)$3); $$ = $2; }
        | expr '+' expr                 { code(add); }
        | expr '-' expr                 { code(sub); }
        | expr '*' expr                 { code(mul); }
        | expr '/' expr                 { code(div); }
        | expr '%' expr                 { code(mod); }
        | expr SLASH_SLASH expr         { code(concat); }
        | expr '>' expr                 { code(gt); }
        | expr GT_EQ expr               { code(ge); }
        | expr '<' expr                 { code(lt); }
        | expr LT_EQ expr               { code(le); }
        | expr EQ_EQ expr               { code(eq); }
        | expr NOT_EQ expr              { code(ne); }
        | '?' lvalue    %prec NEGATE    { $$ = $2; code(query); }
        | '-' expr      %prec NEGATE    { $$ = $2; code(negate); }
        | expr '#'                      { code(listsize); }
        | '!' expr                      { $$ = $2; code(not); }
        | expr '?' then expr colon expr { fromto($3, $6);
                                          fromto($5, progp);
                                        }
        | expr and expr                 { fromto($2, progp);
                                          code(cnv_2_bool);
                                        }
        | expr or expr                  { fromto($2, progp);
                                          code(cnv_2_bool);
                                        }
        ;

colon:    ':'                           { $$ = code2(jmp, 0); }
        ;

and:      AND_AND                       { code(ddup);
                                          $$ = code2(jpz, 0);
                                          code(popd);
                                        }
        ;
```



```
or:       OR_OR                         { code(ddup);
                                          $$ = code2(jpnz, 0);
                                          code(popd);
                                        }
        ;

defn:     declare_formula
        | declare_action
        | declare_relation
        ;

declare_relation:
          identifier TILDE_GT '[' id_list_opt ']' ';'
                                        { $$ = code_related_by($1); }
        ;

declare_formula:
          identifier is expr ';'        { informula = 0;
                                          code(rts);
                                          $$ = code_definition(
                                            FORMULA, $1, $3, progp, 0, $2);
                                        }
        ;

is:       IS                            { $$ = (Int) textptr;
                                          informula = 1;
                                          addID((symptr) 0);
                                        }
        ;

declare_action:
          action identifier             { addID((symptr) 0); }
          refer_opt
          def_begin
                declare_para
                declare_local
                stmtlist
          def_end                       { code2(pushUNDEF, rts);
                                          $$ = code_definition(
                                                $1 ? FUNCTION : PROCEDURE,
                                                $2, $8, progp, $7, $5);
                                          delete_local_level($9);
                                        }
        ;

action:   FUNC                          { $$ = 1; }
        | PROC                          { $$ = 0; }
        ;

def_begin:
          '{'                           {
                                          static char c = '{';
                                          if (indef)
                                            $$ = (Int) (textptr - 1);
                                          else {
                                            $$ = (Int) textptr;
                                            push_text(&c,1);
                                          }
                                          insert_level_marker(++indef);
                                        }
        ;

def_end:
          '}'                           { $$ = indef--; }
        ;
```

```
refer_opt:
          /* nothing */                 { $$ = 0; }
        | ':' id_list                   { $$ = $2; }
        ;

id_list_opt:
          /* nothing */                 { $$ = 0; }
        | id_list
        ;

id_list:
          identifier                    { $$ = 1; addID($1); }
        | id_list ',' identifier        { $$ = $1 + 1; addID($3); }
        ;

declare_para:   /* nothing */
        | PARA                          { inpara = 1; }
          local_list ';'                { inpara = 0; }
        ;

declare_local:
          /* nothing */                 { $$ = 0; }
        | declare_local
          AUTO                          { inauto = 1; }
          local_list ';'                { inauto = 0; $$ = $1 + $4; }
        ;

local_list:
          LOCAL                         { $$ = 1; }
        | local_list ',' LOCAL          { $$ = $1 + 1; }
        ;

arglist:  /* nothing */                 { $$ = 0; }
        | argument_list
        ;

argument_list:
          expr                          { $$ = 1; }
        | argument_list ',' expr        { $$ = $1 + 1; }
        ;
%%
    /*------------------------ end of grammar ---------------------------*/
```

```
#include <ctype.h>
#include "eden.h"
#include "y.tab.h"

extern string prompt, prompt1, prompt2;
extern char *index();
extern Datum *makedatum();
extern symptr lookup();
extern symptr install();

#define BUFSIZ 1024

int         nextc = ' ';
char        yytext[BUFSIZ];
int         yyleng = 0;
YYSTYPE     yylval;

#define app_char(c)     (yyleng>=BUFSIZ?buffer_overflow():(yytext[yyleng++]=c))
#define input()         (app_char(nextc),nextc=keyin())
#define safe_input()    (append_char(nextc),nextc=keyin())
#define unput(c)        (pop_char(),ungetkey(c))
#define pop_char()      --yyleng
#define last_char       yytext[yyleng-1]
#define end_text()      (accept_text(yytext,yyleng);yytext[yyleng]='\0';)
#define clear_text()    yyleng=0
#define accept_text(Text,Leng) if(indef||informula)push_text(Text,Leng)
#define RETURN_TOKEN(T) (end_text(); return T;)

append_char(c)
    int         c;
{
    if (yyleng >= BUFSIZ - 1) {
        end_text();
        clear_text();
    }
    yytext[yyleng++] = c;
}

buffer_overflow()
{
    error("input buffer overflow");
}

init_lex()
{
    input();
}

keyword_token(name)             /* Is name a keyword ? 0 = no; lexeme =
                                   yes) */
    string      name;
{
    static struct keyword_table {       /* Keywords */
        string      name;
        int         kval;
    }           keywords[] = {
#include "keyword.h"
        0, 0
    };

    int         i;

    for (i = 0; keywords[i].name; i++) {
        if (strcmp(name, keywords[i].name) == 0)
            return keywords[i].kval;
    }
```



```
    return 0;
}

number_token()
{
    int         i;
    double      r;
    int         m, n, e, is_float;
    string      format;
    int         is_hex;

    is_float = m = n = e = FALSE;
    while (isdigit(nextc)) {
        input();
        m = TRUE;
    }
    if (nextc == '.') {
        input();
        is_float = TRUE;
    }
    while (isdigit(nextc)) {
        input();
        n = TRUE;
    }

    if (is_float && !m && !n)   /* a single dot */
        return '.';

    if (nextc == 'E' || nextc == 'e') {
        input();
        is_float = TRUE;
        if (nextc == '+' || nextc == '-')
            input();
        while (isdigit(nextc)) {
            input();
            e = TRUE;
        }
        if (!e)
            error("Floating-point format error");
    }
    if (is_float) {             /* floating point */
        end_text();
        sscanf(yytext, "%lf", &r);
        yylval.dp = makedatum(REAL, 0);
        yylval.dp->u.r = r;
        return CONST;
    } else if (*yytext == '0') {/* octal or hexdecimal */
        is_hex = nextc == 'x' || nextc == 'X';
        format = is_hex ? "%x" : "%o";
        if (is_hex) {
            do {
                input();        /* read the 'x' */
            } while (isxdigit(nextc));
        }
        end_text();
        sscanf((is_hex ? yytext + 2 : yytext), format, &i);
        yylval.dp = makedatum(INTEGER, i);
        return CONST;
    } else {                    /* decimal */
        end_text();
        yylval.dp = makedatum(INTEGER, atoi(yytext));
        return CONST;
    }
}

id_token()
```

```
{
    int         i;
    symptr      sp;

    while (isalnum(nextc) || nextc == '_')
        input();

    end_text();

    if (i = keyword_token(yytext))      /* keyword ? */
        return i;                       /* return the lexeme */

    if (inauto || inpara) {     /* declaring local variables ? */
        yylval.narg = local_declare(yytext);
        return LOCAL;
    } else {                            /* in statements */
        if (i = lookup_local(yytext)) { /* local variable ? */
            if (i < 0) {                /* it is an argument */
                yylval.narg = -i;
                return ARG;
            } else {                    /* it is a local variable */
                yylval.narg = i;
                return LOCAL;
            }
        } else {
            if ((sp = lookup(yytext)) == 0)     /* must be global */
                sp = install(yytext, VAR, UNDEF, 0);
            yylval.sym = sp;
            return sp->stype;
        }
    }
}

multi_symbol_token()
{
    static struct {
        char        c1, c2;
        int         token;
    }           *tab, table[] = {
        '+', '=', PLUS_EQ,
        '+', '+', PLUS_PLUS,
        '-', '=', MINUS_EQ,
        '-', '-', MINUS_MINUS,
        '/', '/', SLASH_SLASH,
        '=', '=', EQ_EQ,
        '!', '=', NOT_EQ,
        '~', '>', TILDE_GT,
        '>', '=', GT_EQ,
        '<', '=', LT_EQ,
        '&', '&', AND_AND,
        '|', '|', OR_OR,
        0, 0, 0
    };

    for (tab = table; tab->c1; tab++) {
        if (*yytext == tab->c1 && nextc == tab->c2) {
            input();
            RETURN_TOKEN(tab->token);
        }
    }
    RETURN_TOKEN(*yytext);
}

skip_comment()
{
    safe_input();               /* '/' has already skipped, skip '*' */
```



```
    for (;;) {
        switch (nextc) {
        case '*':
            safe_input();       /* '*' */
            if (nextc == '/') {
                safe_input();   /* skip '/' */
                return;
            }
            break;
        case '/':
            safe_input();       /* skip '/' */
            if (nextc == '*') {
                skip_comment();
                end_text();
                clear_text();
            }
            break;
        case 0:         /* EOF */
            error("unexpected end-of-file in comment");
            return;
        default:
            safe_input();
        }
    }
}

backslash(text)                 /* interpret backslashs in a string */
    string      text;
{
    int         i, n;
    string      s = text;
    static char transtab[] = "b\bf\fn\nr\rt\t";

    while (*s) {
        if (*s == '\\') {
            s++;
            if (isdigit(*s)) {
                for (n = 3, i = 0; n && isdigit(*s); --n, s++)
                    i = (i << 3) | (*s - '0');
                *text++ = i;
            } else if (islower(*s) && index(transtab, *s))
                *text++ = index(transtab, *s++)[1];
            else
                *text++ = *s++;
        } else {
            *text++ = *s++;
        }
    }
    *text = '\0';
}

yylex()
{
    string      s;

    /* initial for a token */
    clear_text();

restart:

    while (isspace(nextc))      /* skip blanks */
        input();

    end_text();
    clear_text();
```

```
    if (nextc == 0)             /* EOF */
        return 0;               /* don't do input() so next token is EOF */

    prompt = prompt2;

    if (isdigit(nextc) || nextc == '.') /* a number or a dot */
        return number_token();

    if (isalpha(nextc) || nextc == '_') /* a keyword or an identifier */
        return id_token();

    if (nextc == '\'') {
        while (input(), nextc != '\'') {
            if (nextc == '\\')
                input();
            else if (nextc == 0)
                error("unexpected end-of-file in chararcter constant");
        }
        input();                /* read the quote */

        /* mustn't alter the following two lines */
        end_text();             /* accept the text */
        last_char = '\0';       /* delete the last quote */

        backslash(yytext + 1);  /* resolve the backslashs */
        if (yytext[2])          /* more than one char */
            error("single char expected");
        yylval.dp = makedatum(CHAR, yytext[1]);
        return CONST;
    }
    if (nextc == '\"') {
        while (input(), nextc != '\"') {
            if (nextc == '\\')
                input();
            else if (nextc == 0)
                error("unexpected end-of-file in string constant");
        }
        input();                /* read the quote */

        /* mustn't alter the following two lines */
        end_text();             /* accept the text */
        last_char = '\0';       /* delete the last quote */

        backslash(yytext + 1);  /* resolve the backslashs */
        s = (string) emalloc(strlen(yytext));
        strcpy(s, yytext + 1);
        yylval.dp = makedatum(STRING, s);
        return CONST;
    }
    input();                    /* accept the char, must be the 1st char */

    switch (*yytext) {          /* the 1st char */
    case '@':
        yylval.dp = &UndefDatum;
        RETURN_TOKEN(CONST);
        break;

    case '$':
        while (isdigit(nextc))
            input();
        if (yyleng == 1)
            RETURN_TOKEN('$');
        yylval.narg = atoi(yytext + 1);
        RETURN_TOKEN(ARG);

    case '/':
```



```
        if (nextc == '*') {     /* nested comment */
            skip_comment();     /* return next token */
            goto restart;
        }
        break;
    }

    return (multi_symbol_token());
}
```

```
/*************************************
 *      SOME BUILTIN FUNCTIONS       *
 *************************************/

#include <stdio.h>
#include <string.h>
#include "eden.h"
#include "builtin.h"
#include "y.tab.h"
#include "symptr.q.h"
#include "hash.h"

struct BLIBTBL blibtbl[] = {
#define INCLUDE 'T'
#include "builtin.h"
#undef INCLUDE
        0, 0
};

usage(s)                            /* internal subroutine */
    char        *s;
{
    error2("usage: ", s);
}

#ifdef DEBUG
int         Debug = 0;              /* 1 for debugging */

void
debug()
{                                   /* debug mode on/off */
    extern int  Debug;

    if (paracount > 0) {
        mustint(para(1));
        Debug = para(1).u.i;
    } else
        Debug = 1;
    pushUNDEF();
}

#endif

void
substr()
{                                   /* sub-string function */
    Datum       d;
    int         from, to, len, i;
    char        *s, *t, *str;
    Datum       ctos();

    if (paracount >= 3) {
        para(1) = ctos(para(1));
        str = para(1).u.s;          /* the string */
        len = strlen(str);          /* the strlen */
        mustint(para(2));           /* from */
        mustint(para(3));           /* to   */
        from = para(2).u.i - 1;
        to = para(3).u.i - 1;

        if (from < 0)
            error("substr: index out of range");

        i = (from <= to) ? (to - from + 1) : 0;

        for (t = s = getheap(i + 1); from <= to; from++)
```



```
            *t++ = (from < len) ? str[from] : ' ';
        *t = '\0';
    } else
        usage("substr(s, from, to)");
    dpush(d, STRING, s);
}

void
sublist()
{
    Datum       *a;
    int         i, listlen, from, to;

    if (paracount >= 3) {
        mustlist(para(1));
        mustint(para(2));
        mustint(para(3));
        a = para(1).u.a;
        listlen = a[0].u.i;
        from = para(2).u.i;
        to = para(3).u.i;
        if (from < 1)
            error("sublist: index out of range");
        for (i = from; i <= to; i++)
            push(i <= listlen ? a[i] : UndefDatum);
        makearr(from <= to ? to - from + 1 : 0);
    } else
        usage("sublist(L, from, to)");
}

void
writeln()
{                                   /* pascal-like writeln */
    b_write();
    printf("\n");
}

void
b_write()
{                                   /* pascal-like write */
    int         i;

    for (i = 1; i <= paracount; i++)
        print(para(i));
    pushUNDEF();
}

void
nameof()
{                                   /* built-in function: symbol name of a
                                       pointer */
    Datum       d, addr;
    string      s, name;

    if (paracount != 1)
        usage("s = nameof(&var);");

    addr = para(1);
    if (!is_symbol(addr))
        error("address needed");

    name = symbol_of(addr)->name;
    s = getheap(strlen(name) + 1);
    strcpy(s, name);

    dpush(d, STRING, s);
```

```
}

print(d)                            /* internal function: print datum */
    Datum       d;
{
    int         i;

    if (is_symbol(d)) {
        printf("(%s is symbol name, ", symbol_of(d)->name);
        printf(address_type(d) == DPTR
               ? "data pointer: %d)" : "char pointer: %d)",
               d.u.v.x);
    } else if (is_local(d)) {
        printf("(local variable #%d, ", local(d));
        printf(address_type(d) == DPTR
               ? "data pointer: %d)" : "char pointer: %d)",
               d.u.v.x);
    } else {
        switch (d.type) {
        case REAL:
            printf("%lg", d.u.r);
            break;

        case INTEGER:
            printf("%d", d.u.i);
            break;

        case CHAR:
            printf("%c", d.u.i);
            break;

        case STRING:
            if (d.u.s) printf("%s", d.u.s);
            break;

        case LIST:
            printf("[");
            for (i = 1; i <= d.u.a[0].u.i; i++) {
                print(d.u.a[i]);
                if (i < d.u.a[0].u.i)
                    printf(",");
            }
            printf("]");
            break;

        case VAR:
            printf("(%s is variable)", d.u.sym->name);
            break;

        case BLTIN:
            printf("(%s is built-in function)", d.u.sym->name);
            break;

        case LIB:
        case RLIB:
            printf("(%s is C function)", d.u.sym->name);
            break;

        case FORMULA:
            printf("(%s is formula)", d.u.sym->name);
            break;

        case FUNCTION:
            printf("(%s is user-defined function)", d.u.sym->name);
            break;
```



```
        case PROCEDURE:
            printf("(%s is user-defined procedure)", d.u.sym->name);
            break;

        case UNDEF:
            printf("@");
            break;

        default:
            printf("(type=%d, val=%d)", d.type, d.u.i);
            break;
        }
    }
}

void
scat()
{                                   /* string concatenation */
    Datum       d;
    int         i;
    int         len;
    string      s;

    len = 0;
    for (i = 1; i <= paracount; i++) {
        Datum       ctos();

        para(i) = ctos(para(i));
        len += strlen(para(i).u.s);
    }

    s = getheap(len + 1);

    for (i = 1, *s = '\0'; i <= paracount; i++)
        strcat(s, para(i).u.s);

    dpush(d, STRING, s);
}

void
lcat()
{
    int         i, j, total, listlen;
    Datum       *a;

    total = 0;
    for (i = 1; i <= paracount; i++) {
        mustlist(para(i));
        a = para(i).u.a;
        total += (listlen = a[0].u.i);
        for (j = 1; j <= listlen; j++)
            push(a[j]);
    }
    makearr(total);
}

void
b_exit()
{                                   /* builtin: exit the interpreter */
    exit(paracount > 0 && isint(para(1)) ? para(1).u.i : 0);
    pushUNDEF();
}

void
exec_string()
{                                   /* builtin: execute a string as a program */
```

```
    int         result;
    Datum       d;

    if (paracount == 0)
        usage("execute(string_expr);");
    muststr(para(1));
    result = run(STRING_DEV, para(1).u.s, 0);
    dpush(d, INTEGER, result);
}

void
exec_file()
{                               /* builtin: execute a file as a program */
    FILE        *filein;
    string      name;
    Datum       d;
    int         result;

    if (paracount == 0)
        usage("include(filename);");
    muststr(para(1));
    name = para(1).u.s;
    if (filein = fopen(name, "r")) {
        result = run(FILE_DEV, filein, name);
        /* ``run'' will close the file */
    } else {
        error2("can't read file", name);
    }
    dpush(d, INTEGER, result);
}

        /* call a function with a list as its argument */
void
apply()
{
    if (paracount >= 2) {
        switch (para(1).type) {
        case FUNCTION:
        case PROCEDURE:
        case BLTIN:
        case LIB:
        case RLIB:
            if (para(2).type == LIST)
                call(para(1).u.sym, para(2));
            break;

        default:
            usage("apply(function, list);");
            break;
        }
    } else
        usage("apply(function, list);");
}

        /* transform the internal symbol table into a list */
void
symtbl2list()
{
    extern symptr hashtable[];
    symptr      sp;
    Datum       d;
    Datum       *p;
    Int         count;
    Int         i;

    count = 0;
```



```
    for (i = 0; i <= HASHSIZE; i++)
        for (sp = hashtable[i]; sp != (symptr) 0; sp = sp->next)
            count++;
    p = (Datum *) getheap((count + 1) * sizeof(Datum));
    p[0].type = INTEGER;
    p[0].u.i = count;

    count = 1;
    for (i = 0; i <= HASHSIZE; i++) {
        for (sp = hashtable[i]; sp != (symptr) 0; sp = sp->next) {
            dpush(d, STRING, sp->name);
            dpush(d, STRING, typename(sp->stype));
            if (sp->text) {
                dpush(d, STRING, sp->text);
            } else {
                dpush(d, STRING, "");
            }
            pushlist(&sp->targets);
            pushlist(&sp->sources);
            makearr(5);
            p[count++] = pop();
        }
    }
    d.type = LIST;
    d.u.a = p;
    push(d);
}

void
printhash()                     /* for debugging */
{
    extern symptr hashtable[];
    symptr      sp;
    int         i;

    for (i = 0; i <= HASHSIZE; i++) {
        printf("%d ----------------------\n", i);
        for (sp = hashtable[i]; sp != (symptr) 0; sp = sp->next) {
            printf("%s\n", sp->name);
        }
    }
    pushUNDEF();
}

/* forget a symbol named by the string (1st arg) from internal
   symbol table into a list */
#define         OK              0
#define         NOTFOUND        1
#define         FAIL            2

void
forget()
{
    extern symptr hashtable[];
    Datum       d;
    int         error_code;
    string      s;
    symptr      sp, last;
    int         i;

    if (paracount < 1 || para(1).type != STRING)
        usage("forget(\"object_name\");");

    s = para(1).u.s;
    last = (symptr) 0;
    error_code = NOTFOUND;
```

```
    i = hashindex(s);
    for (sp = hashtable[i]; sp != (symptr) 0; sp = (last = sp)->next)
        if (strcmp(sp->name, s) == 0) {
            /* find sp */ ;
            if (Q_EMPTY(&sp->targets)) {
                if (sp->name)
                    free(sp->name);

                /* if (sp->inst) free(sp->inst); */

                if (sp->text)
                    free(sp->text);
                freedatum(sp->d);
                /* sp->targets is already EMPTY */
                refer_to(sp, &sp->targets);     /* free sources */

                if (last)
                    last->next = sp->next;
                else
                    hashtable[i] = sp->next;

                free(sp);
                error_code = OK;/* OK */
            } else {            /* some objects refers to this var */
                /* FAIL */
                /* error("can't forget", sp->name); */
                error_code = FAIL;
            }
        }
    dpush(d, INTEGER, error_code);
}

static
pushlist(Q)
    symptr_QUEUE *Q;
{
    Int         count;
    Datum       d;
    symptr_ATOM A;

    count = 0;
    FOREACH(A, Q) {
        dpush(d, STRING, A->obj->name);
        count++;
    }
    makearr(count);
}

static
unsigned
packpara(d, err_msg)
    Datum       d;
    char        *err_msg;
{
    if (is_address(d))
        *((Int *) getheap(sizeof(Int))) = (Int) &(dptr(d)->u.i);
    else
        switch (d.type) {
        case REAL:
            *((float *) getheap(sizeof(float))) = d.u.r;
            break;

        case INTEGER:
            *((int *) getheap(sizeof(int))) = d.u.i;
            break;
```



```
        case CHAR:
            *((char *) getheap(sizeof(char))) = d.u.i;
            break;

        case STRING:
            *((string *) getheap(sizeof(string *))) = d.u.s;
            break;

        case LIST:
            {
                Int         n, i;

                n = d.u.a[0].u.i;       /* no. of items */
                for (i = 1; i <= n; i++)
                    packpara(d.u.a[i], err_msg);
            }
            break;

        default:
            usage(err_msg);
            break;
        }
}

#define parameters (*fp->stackp)
void
pack()
{
    char        *mem;
    Datum       d;

    mem = getheap(0);           /* find the begin of memory block */
    packpara(parameters, "address=pack(data,...);");
    dpush(d, INTEGER, mem);
}

void
array()
{
    Int         n;
    Datum       d;

    if (paracount < 1)
        usage("list = array(n, data);");

    mustint(para(1));
    n = para(1).u.i;
    if (n < 0)
        error("listcat: -ve number in 1st argument");

    if (paracount > 1)
        d = para(2);
    else
        d = UndefDatum;

    while (n--)
        push(d);

    makearr(para(1).u.i);
}

void
user_error()                    /* generate an error */
{
    if (paracount > 0) {
        muststr(para(1));
```

```
        error(para(1).u.s);
    } else
        error("run-time error");
    /* don't need a push since "error" never return */
}

        /*-      touch variables (pointed to by a pointer)
                only their parents shall be re-evaluated
        -*/
void
touch()
{
    extern void schedule_parents_of();
    int     i;

    for (i = 1; i <= paracount; i++)
        schedule_parents_of((symptr) para(i).u.v.y);
    eval_formula_queue();
    pushUNDEF();
}

void
get_environ()
{
        Datum   d;
        string  s;
        char    *getenv();

        if (paracount < 1)
                usage("string = getenv(\"env_name\")");

        muststr(para(1));
        if (s = getenv(para(1).u.s)) {
                dpush(d, STRING, s);
        } else
                pushUNDEF();
}

void
put_environ()
{
        Datum   d;
        char    *s;
        if (paracount < 1)
                usage("ok = putenv(\"env_name=value\")");
        muststr(para(1));
        s = getheap(strlen(para(1).u.s) + 1);
        strcpy(s, para(1).u.s);
        dpush(d, INTEGER, putenv(s));
}

/*** errno ***/
#include <errno.h>
void
error_no()
{
        Datum   d;

        dpush(d, INTEGER, errno);
}

/*** process ***/
#undef FAIL

#define FAIL (-1)
```



```
void
backgnd()
{
        int     pid;
        Datum   d;
        Datum   *p;
        int     argc, i;
        char    **argv;

        p = &para(0);
        argc = p[0].u.i;

        if (argc < 2)
                usage("pid = backgnd(\"cmd\", \"cmd\", \"arg1\", ...);");

        argv = (char **) emalloc(sizeof(char*) * argc);
        for (i = 2; i <= argc; i++) {
                muststr(p[i]);
                argv[i-2] = p[i].u.s;
        }
        argv[argc-1] = (char *)0;

        pid = fork();
        switch (pid) {
        case 0:                 /* child process */
                {
                FILE    *f;

                f = fopen("/dev/null", "r");
                close(0);               /* close stdin */
                dup2(fileno(f), 0);     /* redirect stdin to /dev/null */
                fclose(f);
                }
                pid = execvp(para(1).u.s, argv);
                if (pid = FAIL) {               /* can't execute */
                        fprintf(stderr, "execvp: can't execute %s\n",
                                para(1).u.s);
                }
                break;

        default:                /* parent process */
                dpush(d, INTEGER, pid);
                break;
        }
        free(argv);
}

void
pipe_process()
{
        int     pid;
        Datum   d;
        Datum   *p;
        int     argc, i;
        char    **argv;
        FILE    *fdes[2];

        p = &para(0);
        argc = p[0].u.i;

        if (argc < 2)
                usage("pid = pipe_process(\"cmd\", \"cmd\", \"arg1\", ...);");

        muststr(p[1]);

        argv = (char **) getheap(sizeof(char *) * argc);
```

```
        for (i = 2; i <= argc; i++) {
                muststr(p[i]);
                argv[i-2] = p[i].u.s;
        }
        argv[argc-1] = (char *)0;

        if (pipe(fdes) != FAIL) {
            pid = fork();
            switch (pid) {
              case 0:                   /* child process */
                i = fileno(stdin);
                close(i);               /* close stdin */
                dup2(fdes[0], i);       /* redirect stdin */
                close(fdes[1]);         /* close other end */
                pid = execvp(p[1].u.s, argv);
                if (pid = FAIL) {       /* can't execute */
                        fprintf(stderr, "execvp: can't execute %s\n",
                                p[1].u.s);
                }
                break;

              default:                  /* parent process */
                i = fileno(stdout);
                close(i);               /* close stdout */
                dup2(fdes[1], i);       /* redirect stdout */
                close(fdes[0]);         /* close other end */
                dpush(d, INTEGER, pid);
                break;
            }
        }
}

/*=================== message ======================*/

#ifdef sun

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSG_SIZE 1024

struct {
        long    mtype;
        char    mtext[MSG_SIZE];
} msg;

void
get_msgq()
{
        Datum   d;
        if (paracount < 2)
                usage("msqid=get_message_queue(key,msgflg);");
        mustint(para(1));
        mustint(para(2));
        dpush(d, INTEGER, msgget((key_t) para(1).u.i, para(2).u.i));
}

void
send_message()
{
        int     msqid, mtype, msgflg;
        string  mtext;
        int     result;
        int     msize;
        Datum   *p;
```



```
        Datum   d;
        static  struct msgbuf msg;

        p = &para(0);
        if (p[0].u.i < 3)       /* paracount */
                usage("ok=send_message(msqid,[mtype,mtext],msgflg);");

        mustint(p[1]);
        mustlist(p[2]);
        if (p[2].u.a[0].u.i < 2)
                usage("ok=send_message(msqid,[mtype,mtext],msgflg);");
        mustint(p[2].u.a[1]);
        muststr(p[2].u.a[2]);
        mustint(p[3]);

        msqid  = p[1].u.i;
        msgflg = p[3].u.i;
        msg.mtype = p[2].u.a[1].u.i;
        strncpy(msg.mtext, p[2].u.a[2].u.s, MSG_SIZE);

        msize = strlen(msg.mtext)+1;
        if (msize > MSG_SIZE) msize = MSG_SIZE;

        result = msgsnd(msqid, &msg, msize, msgflg);
        dpush(d, INTEGER, result);
}

void
receive_message()
{
        Datum   *p;
        Datum   d;
        int     i;
        static  struct msgbuf msg;

        p = &para(0);
        if (p[0].u.i < 3)       /* paracount */
                usage("[mtype,mtext]=recieve_message(msqid,mtype,msgflg);");

        for (i = 1; i <= 3; i++)
                mustint(p[i]);

        if (msgrcv(p[1].u.i,&msg,MSG_SIZE,(long)p[2].u.i,p[3].u.i) == -1) {
                pushUNDEF();
        } else {
                dpush(d, INTEGER, msg.mtype);
                dpush(d, STRING, msg.mtext);
                makearr(2);
        }
}

void
remove_msgq()
{
        Datum   d;

        if (paracount < 1)
                usage("remove_msgq(msqid);");
        mustint(para(1));
        dpush(d, INTEGER, msgctl(para(1).u.i, IPC_RMID, 0));
}

#endif sun
```

```
#include <stdio.h>
#include "eden.h"
#include "y.tab.h"

#ifdef sun
#include <memory.h>
#define FASTMEM
#endif sun

#ifdef DEBUG
#define DEBUGPRINT(X,Y,Z) if(Debug){fprintf(stderr,X,Y,Z);}
#else
#define DEBUGPRINT(X,Y,Z)
#endif

Datum   UndefDatum = {UNDEF};

Datum   stack[NSTACK];          /* the stack */
Datum  *stackp;                 /* next free spot on stack */

Inst    prog[NPROG];            /* the machine */
Inst   *progp = prog;           /* next free spot for code generation */
Inst   *pc;                     /* program counter during execurtion */

union compiler_flags compiler_flag;

Frame   frame[NFRAME];
Frame  *fp;                     /* frame pointer */

struct t entry_tbl[MAXENTRY];   /* case entry table */
struct t *entry_ptr;

/* SUBROUTINE INITIALIZE THE RUN-TIME STATUS */
initcode()
{
    reset_pseudo_machine_status();
    reset_frames();
    reset_compiler_status();
}

reset_compiler_status()
{
    reset_prog_ptr();
    clear_IDlist();
    reset_entry_tbl();
    reset_compiler_flags();
}

reset_pseudo_machine_status()
{
    reset_stack();
    reset_heap();
    reset_eval();
}

stack_overflow_err()
{
    error("stack overflow");
}

Datum
stack_underflow_err()
{
    error("stack underflow");
    return UndefDatum;          /* dummy */
}
```



```
/* RUN-TIME CODE EVALUATE FUNCTION ID(...) WHERE ID = BLTIN, LIB, FUNCTION OR
   PROCEDURE. */
void
eval()
{
    Datum   args;
    Datum   lvalue;

    args = pop();               /* arguments (a list) */
    lvalue = pop();             /* lvalue */

    switch (lvalue.type) {
    case BLTIN:
    case LIB:
    case RLIB:
    case FUNCTION:
    case PROCEDURE:
        call(lvalue.u.sym, args);
        break;

    default:
        error("func/proc needed");
        break;
    }
}

call(sp, args)
    symptr  sp;
    Datum   args;
{
    Int     i;
    Datum   d;

    if (fp++ >= &frame[NFRAME - 1])
        error2(sp->name, "call nested too deeply");
    DEBUGPRINT("** call %s (level %d)\n", sp->name, fp - frame);
    fp->sp = sp;
    fp->retpc = pc;
    fp->stackp = stackp;
    fp->hptr = hptr;

    switch (sp->stype) {
    case FUNCTION:
    case PROCEDURE:
        d = newdatum(args);
        push(d);                /* put the arguments back */
        i = sp->nauto;          /* creat local variables */
        while (i--)
            pushUNDEF();
        execute(sp->inst);
        break;

    case BLTIN:
        push(args);             /* don't copy the whole structure ! */
        ((Inst) sp->inst) ();
        break;

    case LIB:
        push(args);             /* don't copy the whole structure ! */
        call_lib((Inst) sp->inst);      /* call the lib interface */
        break;

    case RLIB:                  /* floating point C-function */
        push(args);
```

```
        call_float((Inst) sp->inst);
        break;

    default:
        error("func/proc needed");
    }

    d = newhdat(pop());         /* preserve function return value */
    ret_call();
    push(d);

    DEBUGPRINT("** end call %s (level %d)\n", sp->name, fp - frame);
}

ret_call()
{
    Int     i;

    switch (fp->sp->stype) {
    case FUNCTION:
    case PROCEDURE:
        i = fp->sp->nauto;
        if (stackp != &fp->stackp[i+1])
            fprintf(stderr,"WARNING: stack pointer was inconsistent\n");
        while (i >= 0)
            freedatum(fp->stackp[i--]);
        break;
    }
    pc = fp->retpc;
    stackp = fp->stackp;
    --fp;
}

execute(p)
    Inst   *p;
{
    Inst   *resume_point;

    DEBUGPRINT("** execute %d (stack level %d)\n", p, stackp - stack);
    resume_point = pc;
    pc = p;
    while (*pc)
        (*(*pc++)) ();
    pc = resume_point;
    DEBUGPRINT("** end exec %d (stack level %d)\n", p, stackp - stack);
}

/* ENCODER encode ``CASE'' or ``DEFAULT'' entry */
addentry(dp, ip)
    Datum  *dp;
    Inst   *ip;
{
    if (entry_ptr < &entry_tbl[MAXENTRY]) {
        entry_ptr->dp = dp;
        entry_ptr->ip = ip;
        entry_ptr++;
    } else
        error("no. of 'case' overflow");
}

#include <memory.h>
Inst   *
saveprog(p_begin, p_end)                /* save a piece of program code at a
                                           permanent place */
    Inst   *p_begin;
    Inst   *p_end;
```



```
{
    Int     size;
    Inst   *p;

#ifndef FASTMEM
    Inst   *q;

#endif

    size = ((char *) p_end) - ((char *) p_begin);
    p = (Inst *) emalloc(size);
#ifdef FASTMEM
    memcpy((char *)p, (char *)p_begin, size);
#else
    for (q = p; p_begin != p_end;)
        *q++ = *p_begin++;
#endif
    return p;
}

/* ENCODER install one instruction or operand */
Inst   *
code(f)
    Inst    f;
{
    Inst   *oprogp = progp;

    if (progp >= &prog[NPROG])
        error("program too big");
    *progp++ = f;
    return oprogp;
}

string
savetext(text)                  /* save text buffer permanently */
    string  text;               /* start position of text buffer */
{
    extern string textptr;      /* end position of text buffer */
    string  s;

    *textptr = '\0';            /* force a null */
    s = (string) emalloc((textptr - text) + 1);
    return (string) strcpy(s, text);
}

/* ENCODER encode the definition commands sequences */
Inst   *
code_definition(type, id, prog_start, prog_end, nauto, text)
    Int     type;
    symptr  id;
    Inst   *prog_start, *prog_end;
    Int     nauto;
    string  text;
{
    Inst   *p_begin, *p_end;
    Inst   *oprogp;
    symptr_QUEUE *splist, *save_IDlist();

    p_begin = saveprog(prog_start, prog_end);
    p_end = p_begin + (prog_end - prog_start);
    oprogp = progp = prog_start;
    splist = save_IDlist();

    code((Inst) definition);
    code((Inst) type);
    code((Inst) id);
```

```
    code((Inst) splist);
    code((Inst) p_begin);
    code((Inst) p_end);
    code((Inst) nauto);
    code((Inst) savetext(text));

    return oprogp;
}

/* ENCODER code the ``id -> [...]'' command */
Inst    *
code_related_by(sp)
    symptr  sp;
{
    symptr_QUEUE *splist, *save_IDlist();
    Inst    *ip;
    extern  related_by_code();

    splist = save_IDlist();
    ip = code((Inst) related_by_code);
    code((Inst) sp);
    code((Inst) splist);
    return ip;
}

/* RUN-TIME CODE check a symbol whether is cyclic defined if ok then add it
   to the by_list */
related_by_code()
{
    symptr  sp;
    symptr_QUEUE *splist;

    sp = (symptr) (*pc++);
    splist = (symptr_QUEUE *) (*pc++);

    if (!checkok(sp, splist))
        error2(sp->name, " : CYCLIC DEF: ABORTED");
    change_targets(sp, splist);
}

/* SUBROUTINE add a symbol to by_list then change the related objects */
change_targets(sp, splist)
    symptr  sp;
    symptr_QUEUE *splist;
{
    symptr_QUEUE *A;

    refer_by(sp, splist);
    FOREACH(A, splist) {
        change(A->obj, FALSE /* A->obj->changed */ );
    }
}

/* SUBROUTINE add a symbol to the to_list of formula, function/procedure then
   change the related objects */
change_sources(sp, splist)
    symptr  sp;
    symptr_QUEUE *splist;
{
    switch (sp->stype) {
    case VAR:
    case FORMULA:
        refer_to(sp, splist);
        change(sp, TRUE);
        break;
```



```
    case FUNCTION:
    case PROCEDURE:
        refer_to(sp, splist);
        change(sp, !Q_EMPTY(splist));
        break;

    default:
        error("???");
    }
}

/* RUN-TIME CODE actual execution code of formula/function/procedure
   definition */

codeswitch(tbl)
    struct t *tbl;
{
    Inst    *ip, *defp = (Inst *) 0;
    struct t *t = tbl;

    for (; tbl != entry_ptr; tbl++) {
        if (tbl->dp) {
            ip = code((Inst) tbl->dp);
            code((Inst) (tbl->lp - ip));
        } else
            defp = tbl->lp;
    }
    entry_ptr = t;              /* reset */
    ip = code((Inst) 0);
    if (defp)
        code((Inst) (defp - ip));
    else
        code((Inst) 2);         /* there is no default case */
}
```

```
#include <stdio.h>
#include "eden.h"
#include "y.tab.h"

#ifdef DEBUG
#define XPRINT(s,t) if (Debug)fprintf(stderr,s,t);
#else
#define XPRINT(s,t)
#endif

extern void eval_formula_queue();
extern void invoke_action_queue();

static symptr_QUEUE formula_queue = EMPTYQUEUE(formula_queue);
static symptr_QUEUE action_queue = EMPTYQUEUE(action_queue);

static
void
schedule(sp)
    symptr  sp;
{
    register symptr_ATOM   P;
    register symptr_QUEUE *Q;

    switch (sp->stype) {
    case FORMULA:
        Q = &formula_queue;
        break;

    case BLTIN:
    case PROCEDURE:
    case FUNCTION:
        Q = &action_queue;
        break;

    default:                    /* Is it an error ? */
        return;                 /* don't queue up */
    }

    if (P = SEARCH_symptr(Q, sp, 1)) {
        /* Already there --> reschedule */
        DELETE_ATOM(Q, P);
        APPEND_Q(Q, P);
    } else {
        /* Not there --> put it at the end */
        switch (sp->stype) {
        case FORMULA:
            APPEND_symptr_Q(Q, sp);
            break;

        case BLTIN:
        case PROCEDURE:
        case FUNCTION:          /* if it's an action */
            if (!Q_EMPTY(&sp->sources))
                APPEND_symptr_Q(Q, sp);
            break;
        }
    }
}

        /*-----------------------------------------------**
        **      queue up all formulae and actions        **
        **      which depends on v                       **
        **-----------------------------------------------*/
void
schedule_parents_of(v)          /* builtin function */
```



```
    symptr  v;
{
    register symptr id;
    register symptr_ATOM P;
    register symptr_QUEUE *Q;

    Q = &v->targets;
    FOREACH(P, Q) {
        mark_changed(P->obj);
        schedule(P->obj);
    }
}

static int lock = FALSE;        /* prevent nested eval & invoke */
                                /* of formula and action        */

        /***** evaluate all formulae in the formula queue *****/
void
eval_formula_queue()            /* internal */
{
    register symptr_QUEUE *Q;
    register symptr sp;

    if (lock)
        return;
    lock = TRUE;
    Q = &formula_queue;

    while (!Q_EMPTY(Q)) {
        sp = FRONT(Q)->obj;
        DELETE_FRONT_symptr(Q);
        if (sp->changed) {
            if (ready(sp)) {
                XPRINT("* Eval formula %s *\n", sp->name);
                /* evaluate the formula */
                update(sp);
            } else {
                XPRINT("* Formula %s is not calculated *\n", sp->name);
            }
        } else {
            XPRINT("* Formula %s is unchanged *\n", sp->name);
        }
    }
    lock = FALSE;
    if (!Q_EMPTY(&action_queue))
        invoke_action_queue();
}

        /***** invoke all actions in the action queue *****/
void
invoke_action_queue()           /* internal */
{
    register symptr_QUEUE *Q;
    register symptr sp;
    Datum   d;

    if (lock)
        return;
    lock = TRUE;
    Q = &action_queue;

    while (!Q_EMPTY(Q)) {
        sp = FRONT(Q)->obj;
        DELETE_FRONT_symptr(Q);
        if (ready(sp)) {
            XPRINT("* Invoke action %s *\n", sp->name);
```

```
            makearr(0);
            d = pop();          /* creat null arguments */
            call(sp, d);        /* invoke action */
            popd();             /* drop the value returned */
            /*
            eval_formula_queue();
            */
        } else {
            XPRINT("* Action %s not invoked *\n", sp->name);
        }
    }
    lock = FALSE;
    if (! Q_EMPTY(&formula_queue))
        eval_formula_queue();
}


        /* force formula and action queue to evaluate (unless autocalc off) */
void
eager()                                 /* builtin C function */
{
    int     lockvalue;

    lockvalue = lock;
    lock = FALSE;
    eval_formula_queue();
    lock = lockvalue;
}


        /** put 'sp' in schedule                        ***
        *** and if 'autocalc' is enabled evaluate it    **/
void
change(sp, flag)
    symptr  sp;
    int     flag;
{
    extern Int *autocalc;
    register Inst *savepc;
    register symptr_ATOM P;

    sp->changed = flag;
    schedule(sp);
    schedule_parents_of(sp);

    if (*autocalc) {
        savepc = pc;
        eval_formula_queue();
        /*
        invoke_action_queue();
        */
        pc = savepc;
    }
}

void
mark_changed(sp)
    symptr  sp;
{
    register symptr_QUEUE *P, *Q;

    if (!sp->changed) {         /* not already marked */
        sp->changed = TRUE;
        Q = &sp->targets;
        FOREACH(P, Q)
            mark_changed(P->obj);
```



```
    }
}

/* convert formula-queue to a list of pointers */
void
formula_list()                          /* builtin function */
{
    register int count;
    register symptr_ATOM P;
    register symptr_QUEUE *Q;

    count = 0;
    Q = &formula_queue;
    FOREACH(P, Q) {
        address(P->obj);
        count++;
    }
    makearr(count);
}

/* convert action-queue to a list of pointers */
void
action_list()                           /* builtin function */
{
    register symptr_ATOM P;
    register symptr_QUEUE *Q;
    register int count;

    count = 0;
    Q = &action_queue;
    FOREACH(P, Q) {
        address(P->obj);
        count++;
    }
    makearr(count);
}

void
reset_eval()
{
    lock = FALSE;
    CLEAR_symptr_Q(&formula_queue);
    CLEAR_symptr_Q(&action_queue);
}
```

```
/*********************************
 *      HEAP MEMORY MANAGER      *
 *********************************/

#include "eden.h"

char    heap[HEAPSIZE];
char   *hptr = heap;
#define heap_overflow(p)        ((p)>=&heap[HEAPSIZE])

char   *
getheap(size)                   /* allocate memory from heap */
    int     size;
{
    char   *p;

    if (heap_overflow(hptr + size))
        error("heap overflow");
    p = hptr;
    hptr = hptr + size;
#ifdef sparc
    hptr += (4 - (int)hptr & 3);        /* alignment */
#endif sparc
    return p;
}

freeheap()
{                               /* free heap by resetting the heap ptr */
    if (fp == frame)
        hptr = heap;
    else
        hptr = fp->hptr;
}
```

```
#include "global.q.h"

Q_LEN(Q)
QUEUE *Q;
{
        int     count = 0;
        QUEUE   *P;
        FOREACH (P, Q) count++;
        return count;
}

QUEUE *
WHERE_IS(Q, n)
QUEUE *Q;
int n;
{
        QUEUE   *P;

        for (P = FRONT(Q); P && --n; P = NEXT(Q,P))
                ;
        return P;
}
```

```
#include "eden.h"
#include "builtin.h"
#include "y.tab.h"

/*
 * call_lib --- C library interface
 *
 * It is a useful interface to include the C library routines into the
 * language.
 *
 * Weak points:
 *
 * (1)  Only the first 10 arguments can be passed.
 *
 * (2)  The interface cannot deal with list (cause the different internal
 * representation of data structures).
 *
 * (3)  All the values returned by library routines are integers.
 * Programmers must explicitly can write the type convertion codes. See
 * type.c for the type-convertion functions.
 *
 * (4)  The interface can handle ``true'' functions, NOT macros.
 *
 */

#define P(i) para(i)
#define PARSIZE 10
#define PushInteger(data) {Datum d;d.type=INTEGER;d.u.i=(Int)data;push(d);}
#define PushReal(data) {Datum d;d.type=REAL;d.u.r=(double)data;push(d);}

/*************************************/

call_lib(inst)
    Inst    inst;
{
    int     i, j;
    int     A[PARSIZE];
    Int     result;

    for (i = 1, j = 0; i <= paracount && j < PARSIZE; i++) {
        if (is_address(P(i)))
            A[j++] = (Int) &(((Datum *) P(i).u.v.x)->u.i);
        else
            switch (P(i).type) {
            case INTEGER:
            case CHAR:
            case STRING:
                A[j++] = P(i).u.i;
                break;

            case REAL:
                A[j++] = P(i).u.v.x;
                A[j++] = P(i).u.v.y;
                break;

            default:
                error("illegal parameter in C-lib");
                break;
            }
    }

    result = ((int (*) ()) inst) (A[0], A[1], A[2], A[3], A[4],
                                  A[5], A[6], A[7], A[8], A[9]);
    PushInteger(result);
}
```



```
call_float(inst)
    Inst    inst;
{
    int     i, j;
    int     A[PARSIZE];
    double  result;

    for (i = 1, j = 0; i <= paracount && j < PARSIZE; i++) {
        if (is_address(P(i)))
            A[j++] = (Int) &(((Datum *) P(i).u.v.x)->u.i);
        else
            switch (P(i).type) {
            case INTEGER:
            case CHAR:
            case STRING:
                A[j++] = P(i).u.i;
                break;

            case REAL:
                A[j++] = P(i).u.v.x;
                A[j++] = P(i).u.v.y;
                break;

            default:
                error("illegal parameter in C-lib");
                break;
            }
    }

    result = ((double (*) ()) inst) (A[0], A[1], A[2], A[3], A[4],
                                     A[5], A[6], A[7], A[8], A[9]);
    PushReal(result);
}
```

```
#include <stdio.h>
#include <ctype.h>
#include <signal.h>
#include "eden.h"
#include "y.tab.h"
#include "builtin.h"
#include "custom.h"

#define code2(c1,c2)     code(c1); code(c2)
#define code3(c1,c2,c3) code(c1); code(c2); code(c3)

#ifdef DEBUG
static
DEBUGPRINT(s, i)
    string  s;
    int     i;
{
    if (Debug)
        fprintf(stderr, s, i);
}
#else
#define DEBUGPRINT(s,i)
#endif

char   *progname;

#include <setjmp.h>

bool    interactive = TRUE;
string  prompt1 = ":> ";
string  prompt2 = "+> ";
string  prompt;
Inst   *saveprog();

char  **gargv;                  /* global argument list */
int     gargc;

Datum  *
makedatum(type, value)
{                               /* MAKE A NEW DATUM FOR CONSTANTS */
    Datum  *p;

    p = (Datum *) emalloc(sizeof(Datum));
    p->type = type;
    p->u.i = value;
    return p;
}

#define LARGE_TEXT      0x20000

char    textcode[LARGE_TEXT];   /* INPUT TEXT BUFFER */
char   *textptr = textcode;

push_text(text, len)
    string  text;               /* push some text on textcode */
    int     len;                /* number of char */
{                               /* orginal text is temp. stored */

    if (textptr + len >= &textcode[LARGE_TEXT])
        error("text buffer overflow");
    strncpy(textptr, text, len);
    textptr += len;
}

#define MAXBUF 256
static struct input_device {    /* INPUT DEVICE */
```



```
    char   *name;               /* FILE NAME */
    short   type;               /* DEVICE TYPE: FILE / STRING */
    char   *ptr;                /* FILE-PTR / CHAR-PTR */
    char    sbuf[MAXBUF];       /* I/O BUFFER */
    char   *sptr;               /* BUFFER PTR */
    bool    newline;            /* NEWLINE READ ? */
    int     lineno;             /* LINE NUMBER */
    int     lastc;              /* THE FORWARD CHAR OF LAST DEVICE */
    Frame  *frame;              /* FRAME */
    jmp_buf begin;              /* RESUME POINT WHEN ERROR OCCURRED */
}       Input_Devices[10];      /* 10 LEVELS OF INPUT */

static struct input_device *Inp_Dev = Input_Devices;

keyin()
{                               /* READ A CHAR FROM INPUT DEVICE */
    int     c;
    FILE   *filein;

    switch (Inp_Dev->type) {
    case FILE_DEV:
        if (Inp_Dev->sptr > Inp_Dev->sbuf)
            c = *--(Inp_Dev->sptr);
        else {
            filein = (FILE *) Inp_Dev->ptr;
            if (Inp_Dev->newline && filein == stdin)
                print_prompt();
            c = getc(filein);
            if (c == EOF)
                c = 0;
        }
        if (Inp_Dev->newline = (c == '\n'))
            Inp_Dev->lineno++;
        break;

    case STRING_DEV:
        c = *(Inp_Dev->ptr)++;
        break;
    }
    return c;
}

/*-
ungetkey(c)
{
    switch (Inp_Dev->type) {
    case FILE_DEV:
        *Inp_Dev->sptr++ = c;
        break;
    case STRING_DEV:
        *--Inp_Dev->ptr = c;
        break;
    }
    if (c == '\n') {
        Inp_Dev->newline = FALSE;
        --Inp_Dev->lineno;
    }
}
-*/

defnonly(s)                     /* warn if illegal definition */
    char   *s;
{
    if (!indef)
        error2(s, "used outside definition");
}
```

```
yyerror(s)                          /* report compile-time error */
    char   *s;
{
    error(s);
}

error2(s, t)                        /* PRINT ERROR MESSAGE */
    char   *s, *t;                  /* SKIP THE REST OF INPUT */
{
    extern void eden_interrupt_handler();
    extern int nextc;

    warning(s, t);
    switch (Inp_Dev->type) {
    case FILE_DEV:
        /* flush rest of file */
        fseek(Inp_Dev->ptr, 0L, 2);
        break;
    case STRING_DEV:
        /* empty the string */
        *Inp_Dev->ptr = '\0';
        break;
    }
    nextc = 0;  /* EOF */

    while (fp != Inp_Dev->frame) {      /* free the memory for frame */
        fprintf(stderr, "------- called by %s\n", fp->sp->name);
        ret_call();
    }

    reset_pseudo_machine_status();      /* except the frames */
    reset_compiler_status();

    prompt = prompt1;

    signal(SIGINT, eden_interrupt_handler);

    longjmp(Inp_Dev->begin, 1);
}

print_prompt()
{
    if (interactive)
        fprintf(stderr, prompt);
}

static
void
eden_interrupt_handler()
{
    /* signal(SIGINT, SIG_IGN); fprintf(stderr,"\n*** ^C ***\n"); */
    yyerror("interrupt");
    /* should not reach here because yyerror() never returns */
    /* signal(SIGINT, eden_interrupt_handler); */
}

init()          /* INITIALIZATION */
{               /* init builtin function, variables, etc */
    extern Int    *autocalc;
    extern struct CLIBTBL clibtbl[];
    extern struct RLIBTBL rlibtbl[];
    extern struct BLIBTBL blibtbl[];
    register int    i;
    register symptr    s;
```



```
    /* install builtin C library functions */
    for (i = 0; blibtbl[i].name; i++) {
        s = install(blibtbl[i].name, BLTIN, BLTIN, 0);
        s->inst = (Inst *) blibtbl[i].func;
        s->d.type = BLTIN;
        s->d.u.sym = s;
    }

    /* install integer C library functions */
    for (i = 0; ilibtbl[i].name; i++) {
        s = install(ilibtbl[i].name, LIB, LIB, 0);
        s->inst = (Inst *) ilibtbl[i].func;
        s->d.type = LIB;
        s->d.u.sym = s;
    }

    /* install real (floating-point) C library functions */
    for (i = 0; rlibtbl[i].name; i++) {
        s = install(rlibtbl[i].name, RLIB, RLIB, 0);
        s->inst = (Inst *) rlibtbl[i].func;
        s->d.type = RLIB;
        s->d.u.sym = s;
    }

    install_custom_variables(); /* custom-built */
}

main(argc, argv)                    /* MAIN PROGRAM */
    char   *argv[];

{
    FILE   *filein, *moreinput();
    char   *name;
    symptr  s;

#ifdef YYDEBUG
    extern int yydebug;

    yydebug = 1;
#endif
    progname = argv[0];
    prompt = prompt1;

    (s = install("autocalc", VAR, INTEGER, 1))->changed = FALSE;
    autocalc = &s->d.u.i;

    init();
    initcode();
    init_LocalVarList();
    gargc = argc - 1;
    gargv = argv + 1;
    signal(SIGINT, eden_interrupt_handler);
    while (filein = moreinput(&name)) {
        run(FILE_DEV, filein, name);
    }
    run(FILE_DEV, stdin, NULL);
    return 0;
}

#define streq(s1,s2) (strcmp(s1,s2)==0)

FILE   *
moreinput(name)
    string *name;
{
    FILE   *f;
```

```
    for (;;) {
        if (gargc-- <= 0) {
            *name = 0;
            return (FILE *) 0;
        }
        *name = *gargv++;
        if (streq(*name, "-"))
            return stdin;
        if (streq(*name, "-n")) {
            interactive = FALSE;
            continue;
        }
        if (streq(*name, "-i")) {
            interactive = TRUE;
            continue;
        }
        if (**name == '-')
            continue;           /* ignore unknown options */
        /* else a filename */
        if (f = fopen(*name, "r"))
            return f;
        fprintf(stderr, "%s: can't open %s\n", progname, *name);
    }
}

run(type, ptr, name)            /* RUN A PROGRAM */
    short   type;               /* SOURCE TYPE: FILE/STRING_DEV */
    char    *ptr;               /* FILE POINTER / STRING_DEV */
    char    *name;              /* FILE NAME (IF ANY) */
{                               /* EXECUTE UNTIL EOF OR ERROR OCCURRED */
    bool    errorflag = FALSE;
    Inst    *savepc = progp;

    Inp_Dev++;
    Inp_Dev->frame = fp;
    Inp_Dev->name = name;
    switch (Inp_Dev->type = type) {
    case FILE_DEV:
        Inp_Dev->ptr = ptr;
        Inp_Dev->sptr = Inp_Dev->sbuf;
        break;

    case STRING_DEV:
        Inp_Dev->sptr = (char *) emalloc(strlen(ptr) + 1);
        strcpy(Inp_Dev->sptr, ptr);
        Inp_Dev->ptr = Inp_Dev->sptr;
        break;
    }
    Inp_Dev->newline = TRUE;
    Inp_Dev->lineno = 1;
    Inp_Dev->lastc = nextc;

    DEBUGPRINT("** Run Level: %d **\n", Inp_Dev - Input_Devices);

    errorflag = setjmp(Inp_Dev->begin);

    for (init_lex(),run_init(savepc); yyparse(); run_init(savepc))
        execute(savepc);

    /* finished, remove the device */
    nextc = Inp_Dev->lastc;
    switch (Inp_Dev->type) {
    case FILE_DEV:
        /* close the file, but not stdin */
        if ((FILE *) Inp_Dev->ptr != stdin)
```



```
            fclose((FILE *) Inp_Dev->ptr);
        break;

    case STRING_DEV:
        free(Inp_Dev->sptr);
        break;
    }
    --Inp_Dev;                  /* remove device */

    progp = savepc;

    DEBUGPRINT("** Run Level: %d **\n", Inp_Dev - Input_Devices);

    return errorflag;           /* return error flag */
}

run_init(pc)                    /* init before running */
    Inst    *pc;
{
    progp = pc;
    reset_entry_tbl();
    reset_compiler_flags();
    prompt = prompt1;
}

warning(s, t)                   /* print warning message */
    char    *s, *t;
{
    fprintf(stderr, "%s: %s", progname, s);
    if (t)
        fprintf(stderr, " %s", t);
    if (Inp_Dev->name)
        fprintf(stderr, " in %s", Inp_Dev->name);
    if (Inp_Dev->type == FILE_DEV)
        fprintf(stderr, " near line %d", Inp_Dev->lineno);
    fprintf(stderr, "\n");
    /* Inp_Dev->sptr = Inp_Dev->sbuf; */
}

void
user_trace()                    /* EDEN function */
{
    Frame *p;

    fprintf(stderr, "\n");
    for (p = &frame[1]; p <= fp; p++)
        fprintf(stderr, p == fp ? "%s\n" : "%s->", p->sp->name);
}
```

```
/*************************************************
 *      RUN-TIME CODES OF BINARY OPERATORS       *
 *************************************************/

#include "eden.h"
#include "y.tab.h"
#include <stdio.h>

#ifdef DEBUG
#define DEBUGPRINT(s, t)        {if(Debug)fprintf(stderr,s,t);}
#else
#define DEBUGPRINT(s,t)
#endif

extern Datum ctos();

#define dpush(x, y, z)          x.u.i=(Int)(z); x.type = y; push(x);
#define CheckRange(n,low,up)    if(n<low||n>up)out_of_range_error()
#define get_address(addr)       addr=pop();mustaddr(addr)
#define sametype(d1, d2) \
        ((d1.type == d2.type) || (isnum(d1) && isnum(d2)))

#define ChangeSym(addr) if(is_symbol(addr))change(symbol_of(addr),FALSE)

#define address_error() type_clash_addr_err()

div_by_zero_error(op)
    string      op;
{
    error2(op, "divided by zero");
}

num_required_error()
{
    error("type clash: expecting number");
}

out_of_range_error()
{
    error("index error: out of range");
}

dimension_error()
{
    error("index error: data isn't a list or string");
}

mustint(d)
    Datum       d;
{
    if (d.type != INTEGER && d.type != CHAR)
        type_clash_err(INTEGER);
}

mustchar(d)
    Datum       d;
{
    if (d.type != INTEGER && d.type != CHAR)
        type_clash_err(CHAR);
}

muststr(d)
    Datum       d;
{
    if (d.type != STRING)
        type_clash_err(STRING);
}
```



```
}

mustlist(d)
    Datum       d;
{
    if (d.type != LIST)
        type_clash_err(LIST);
}

mustaddr(d)
    Datum       d;
{
    if (!is_address(d))
        type_clash_addr_err();
}

get2num(dp1, dp2)               /* get 2 integer from stack */
    Datum       *dp1, *dp2;
{
    *dp2 = pop();
    *dp1 = pop();

    switch (dp1->type) {
    case CHAR:
        dp1->type = INTEGER;
    case INTEGER:
    case REAL:
        break;
    case UNDEF:
        return UNDEF;           /* undefined value */
    default:
        num_required_error();
    }

    switch (dp2->type) {
    case CHAR:
        dp1->type = INTEGER;
    case INTEGER:
    case REAL:
        break;
    case UNDEF:
        return UNDEF;           /* undefined value */
    default:
        num_required_error();
    }

    if (dp1->type != REAL && dp2->type == REAL) {
        /* convert d1 to real */
        dp1->type = REAL;
        dp1->u.r = dp1->u.i;
        return REAL;
    }
    if (dp1->type == REAL && dp2->type != REAL) {
        /* convert d2 to real */
        dp2->type = REAL;
        dp2->u.r = dp2->u.i;
        return REAL;
    }
    return dp1->type;
}

void
add()
{                               /* operator + */
    Datum       d1, d2;
```

```
    switch (get2num(&d1, &d2)) {
    case INTEGER:
        d1.u.i += d2.u.i;
        break;
    case REAL:
        d1.u.r += d2.u.r;
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d1);
}

void
sub()
{                                   /* operator - */
    Datum       d1, d2;

    switch (get2num(&d1, &d2)) {
    case INTEGER:
        d1.u.i -= d2.u.i;
        break;
    case REAL:
        d1.u.r -= d2.u.r;
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d1);
}

void
mul()
{                                   /* operator * */
    Datum       d1, d2;

    switch (get2num(&d1, &d2)) {
    case INTEGER:
        d1.u.i *= d2.u.i;
        break;
    case REAL:
        d1.u.r *= d2.u.r;
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d1);
}

void
div()
{                                   /* operator / */
    Datum       d1, d2;

    switch (get2num(&d1, &d2)) {
    case INTEGER:
        if (d2.u.i == 0)
            div_by_zero_error("/");
        d1.u.i /= d2.u.i;
        break;
    case REAL:
        if (d2.u.r == 0.0)
            div_by_zero_error("/");
```



```
        d1.u.r /= d2.u.r;
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d1);
}

void
mod()
{                                   /* operator % */
    Datum       d1, d2;

    switch (get2num(&d1, &d2)) {
    case INTEGER:
        if (d2.u.i == 0)
            div_by_zero_error("%");
        d1.u.i %= d2.u.i;
        break;
    case REAL:
        error("operand of % have incompatible types");
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d1);
}

void
negate()
{                                   /* unary minus - */
    Datum       d;

    d = pop();
    switch (d.type) {
    case CHAR:
        d.type = INTEGER;
    case INTEGER:
        d.u.i = -d.u.i;
        break;
    case REAL:
        d.u.r = -d.u.r;
        break;
    case UNDEF:
        pushUNDEF();
        return;
    default:
        num_required_error();
        return;
    }
    push(d);
}

void
not()
{                                   /* logical not */
    Datum       d;

    d = pop();
    if (!isint(d))
        type_clash_err(INTEGER);
    switch (d.type) {
    case CHAR:
        d.type = INTEGER;
```

```
    case INTEGER:
        d.u.i = !d.u.i;
        break;
    case REAL:
        d.type = INTEGER;
        d.u.i = !d.u.r;
        break;
    default:
        pushUNDEF();
        return;
    }
    push(d);
}

#define is_string(d) (d.type == CHAR || d.type==STRING)
#define is_list(d) (d.type == LIST)

void
concat()
{                                   /* string concat opeator // */
    Datum       d1, d2;
    int         len;
    string      s;
    int         i, size1, size2;

    d2 = pop();
    d1 = pop();
    if (isundef(d1) || isundef(d2))
        pushUNDEF();
    else if (is_string(d1) && is_string(d2)) {
        d1 = ctos(d1);
        d2 = ctos(d2);
        len = strlen(d1.u.s) + strlen(d2.u.s);
        s = (string) getheap(len + 1);
        d1.u.s = (string) strcat(strcpy(s, d1.u.s), d2.u.s);
        push(d1);
    } else if (is_list(d1) && is_list(d2)) {
        size1 = d1.u.a[0].u.i;
        size2 = d2.u.a[0].u.i;
        for (i = 1; i <= size1; i++)
            push(d1.u.a[i]);
        for (i = 1; i <= size2; i++)
            push(d2.u.a[i]);
        makearr(size1 + size2);
    } else
        error("type clash: expecting strings or lists");
}

void
jmp()
{                                   /* unconditional jump */
    Int         i = (Int) (*pc++);

    pc += i;
}

void
jpz()
{                                   /* jump on zero */
    Int         i = (Int) (*pc++);
    Datum       d;

    mustint(d = pop());
    if (!d.u.i)
        pc += i;
}
```



```
void
jpnz()
{                                   /* jump on not zero */
    Int         i = (Int) (*pc++);
    Datum       d;

    mustint(d = pop());
    if (d.u.i)
        pc += i;
}

void
ddup()                              /* duplicate datum */
{
    Datum       d;

    d = top_of_stack;
    push(d);
}

void
popd()                              /* pop a datum */
{
    Datum       d;

    d = pop();
}

void
pushUNDEF()                         /* push @ */
{
    push(UndefDatum);
}

void
pushint()
{                                   /* interprete next instruction as an
                                       integer and push it onto stack */
    Datum       d;

    d.type = INTEGER;
    d.u.i = (Int) *pc++;
    push(d);
}

void
constpush()
{
    push(*((Datum *) (*pc++)));
}

datacmp(d1, d2)                     /* internal function */
    Datum       d1, d2;             /* compare two data */
{                                   /* return 0 = equal, +ve = greater than */
    int         i;                  /* -ve = less than */
    double      r;

    if (sametype(d1, d2)) {
        if (d1.type != REAL && d2.type == REAL) {
            /* convert d1 to real */
            d1.type = REAL;
            d1.u.r = d1.u.i;
        } else if (d1.type == REAL && d2.type != REAL) {
            /* convert d2 to real */
            d2.type = REAL;
```

```
            d2.u.r = d2.u.i;
        }
        switch (d1.type) {
        case UNDEF:
            return 0;
        case REAL:
            return (r = d1.u.r - d2.u.r) ? (r > 0.0 ? 1 : -1) : 0;
        case STRING:
            return strcmp(d1.u.s, d2.u.s);
        case LIST:
            if (d1.u.a[0].u.i != d2.u.a[0].u.i)
                return 1;       /* not equal */
            for (i = 1; i <= d1.u.a[0].u.i; i++)
                if (datacmp(d1.u.a[i], d2.u.a[i]))
                    return 1;   /* not equal */
            return 0;           /* equal */
        default:
            return (d1.u.i - d2.u.i);
        }
    } else {
        return 1;               /* not equal */
    }
}

void
cnv_2_bool()
{                               /* convert an integer to boolean value */
    mustint(top_of_stack);
    top_of_stack.type = INTEGER;
    top_of_stack.u.i = top_of_stack.u.i != 0;
}

void
gt()
{                               /* >= */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    if (sametype(d1, d2)) {
        if (d1.type != UNDEF && d1.type != LIST) {
            dpush(d1, INTEGER, datacmp(d1, d2) > 0);
            return;
        }
    }
    pushUNDEF();
}

void
lt()
{                               /* < */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    if (sametype(d1, d2)) {
        if (d1.type != UNDEF && d1.type != LIST) {
            dpush(d1, INTEGER, datacmp(d1, d2) < 0);
            return;
        }
    }
    pushUNDEF();
    push(d1);
}

void
```



```
ge()
{                               /* >= */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    if (sametype(d1, d2)) {
        if (d1.type != UNDEF && d1.type != LIST) {
            dpush(d1, INTEGER, datacmp(d1, d2) >= 0);
            return;
        }
    }
    pushUNDEF();
}

void
le()
{                               /* <= */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    if (sametype(d1, d2)) {
        if (d1.type != UNDEF && d1.type != LIST) {
            dpush(d1, INTEGER, datacmp(d1, d2) <= 0);
            return;
        }
    }
    pushUNDEF();
}

void
eq()
{                               /* == */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    d1.u.i = datacmp(d1, d2) == 0;
    d1.type = INTEGER;
    push(d1);
}

void
ne()
{                               /* != */
    Datum       d1, d2;

    d2 = pop();
    d1 = pop();
    d1.u.i = datacmp(d1, d2) != 0;
    d1.type = INTEGER;
    push(d1);
}

void
switchcode()
{
    Datum       d;

    d = pop();
    for (; *pc && datacmp(d, *(Datum *) (*pc)); pc += 2);
    pc += (Int) pc[1];
}

void
```

```
definition()
{                                   /* put func/proc in symbol table */
    symptr      sp;
    symptr_QUEUE *splist;
    Inst        *prog_begin, *prog_end;
    Int         type;
    Int         nauto;
    string      text;

    type = (Int) (*pc++);
    sp = (symptr) (*pc++);
    splist = (symptr_QUEUE *) (*pc++);
    prog_begin = (Inst *) (*pc++);
    prog_end = (Inst *) (*pc++);
    nauto = (Int) (*pc++);
    text = (string) (*pc++);

    if (!checkok(sp, splist)) {
        error2(sp->name, " : CYCLIC DEF : ABORTED");
    }
    if (prog_begin != sp->inst) {
        /*-
        if (sp->inst && !indef) {
            free(sp->inst);
            DEBUGPRINT("** free %s %d\n", sp->name);
        }
        -*/
        sp->nauto = nauto;
        sp->inst = prog_begin;
        sp->stype = type;
        sp->d.type = type;
        sp->d.u.sym = sp;
        sp->text = text;
        change_sources(sp, splist);
    }
    DEBUGPRINT("** define %s\n", sp->name);
}

/*****************************************************************
 *      RUN-TIME CODES OF ASSIGNMENT AND INCREMENT OPERATORS     *
 *****************************************************************/

static
need_rwv(addr, message)             /* check if lvalue a RWV ? */
    Datum       addr;
    char        *message;
{
    if (!(is_local(addr) ||
          (is_symbol(addr) && symbol_of(addr)->stype == VAR)))
        error2(message, "not read/write variable");
}

static
cnv_formula_to_rwv(addr)            /* if sym is a formula, convert to rwv */
    Datum       addr;
{
    symptr      sp;
    static symptr_QUEUE NullList = EMPTYQUEUE(NullList);

    if (is_symbol(addr) && (sp = symbol_of(addr))->stype == FORMULA) {
        /*-
        free(sp->inst);
        -*/                         /* remove the code */
        sp->inst = (Inst *) 0;
        sp->nauto = 0;
        sp->stype = VAR;
```



```
        /* free(sp->text); */
        sp->text = (char *) 0;
        change_sources(sp, &NullList);
    }
}

void
assign()
{                                   /* lvalue = expression */
    Datum       d, addr, tmp;

    d = pop();                      /* value */
    addr = pop();                   /* address */

    mustaddr(addr);

    cnv_formula_to_rwv(addr);

    need_rwv(addr, "'='");

    switch (address_type(addr)) {
    case DPTR:
        tmp = newdatum(d);
        freedatum(*dptr(addr));
        *dptr(addr) = tmp;
        break;

    case CPTR:
        mustchar(d);
        *cptr(addr) = d.u.i;
        break;

    default:
        address_error();
        break;
    }
    push(d);
    ChangeSym(addr);
}

void
inc_asgn()
{                                   /* lvalue += expression */
    Datum       d, addr;

    mustint(d = pop());             /* value */
    addr = pop();                   /* address */

    mustaddr(addr);

    need_rwv(addr, "'+='");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        dptr(addr)->u.i += d.u.i;
        break;

    case CPTR:
        mustchar(d);
        *cptr(addr) += d.u.i;
        break;

    default:
        address_error();
        break;
```

```
    }
    push(d);
    ChangeSym(addr);
}

void
dec_asgn()
{                                   /* lvalue -= expression */
    Datum       d, addr;

    mustint(d = pop());             /* value */
    addr = pop();                   /* address */

    mustaddr(addr);

    need_rwv(addr, "'-='");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        dptr(addr)->u.i -= d.u.i;
        break;

    case CPTR:
        mustchar(d);
        *cptr(addr) -= d.u.i;
        break;

    default:
        address_error();
        break;
    }
    push(d);
    ChangeSym(addr);
}

void
pre_inc()
{                                   /* ++lvalue */
    Datum       d, addr;

    addr = pop();                   /* address */

    mustaddr(addr);

    need_rwv(addr, "'++'");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        (dptr(addr)->u.i)++;
        push(*dptr(addr));
        break;

    case CPTR:
        (*cptr(addr))++;
        dpush(d, CHAR, *cptr(addr));
        break;

    default:
        address_error();
        break;
    }
    ChangeSym(addr);
}
```



```
void
post_inc()
{                                   /* lvalue++ */
    Datum       d, addr;

    addr = pop();                   /* address */

    mustaddr(addr);

    need_rwv(addr, "'++'");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        push(*dptr(addr));
        (dptr(addr)->u.i)++;
        break;

    case CPTR:
        dpush(d, CHAR, *cptr(addr));
        (*cptr(addr))++;
        break;

    default:
        address_error();
        break;
    }
    ChangeSym(addr);
}

void
pre_dec()
{                                   /* --lvalue */
    Datum       d, addr;

    addr = pop();                   /* address */

    mustaddr(addr);

    need_rwv(addr, "'--'");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        --(dptr(addr)->u.i);
        push(*dptr(addr));
        break;

    case CPTR:
        --(*cptr(addr));
        dpush(d, CHAR, *cptr(addr));
        break;

    default:
        address_error();
        break;
    }
    ChangeSym(addr);
}

void
post_dec()
{                                   /* lvalue-- */
    Datum       d, addr;

    addr = pop();                   /* address */
```

```
    mustaddr(addr);

    need_rwv(addr, "'--':");

    switch (address_type(addr)) {
    case DPTR:
        mustint(*dptr(addr));
        push(*dptr(addr));
        --(dptr(addr)->u.i);
        break;

    case CPTR:
        dpush(d, CHAR, *cptr(addr));
        --(*cptr(addr));
        break;

    default:
        address_error();
        break;
    }
    ChangeSym(addr);
}

string
erealloc(ptr, size)
    string      ptr;
{
    string      p;

    if ((p = (string) realloc(ptr, size)) == NULL)
        error("system error: realloc");
    return p;
}

freedatum(d)
    Datum       d;
{
    int         i;

    switch (d.type) {
    case STRING:
        if (d.u.s)
            free(d.u.s);
        break;

    case LIST:
        for (i = 1; i <= d.u.a[0].u.i; i++)
            freedatum(d.u.a[i]);
        free(d.u.a);
        break;
    }
}

Datum
newdatum(d)
    Datum       d;
{
    string      s;
    Datum       *a;
    int         i;

    switch (d.type) {
    case STRING:
        if (d.u.s) {
            s = (string) emalloc(strlen(d.u.s) + 1);
```



```
            d.u.s = (string) strcpy(s, d.u.s);
        }
        break;

    case LIST:
        a = (Datum *) emalloc((d.u.a[0].u.i + 1) * sizeof(Datum));
        for (i = 0; i <= d.u.a[0].u.i; i++)
            a[i] = newdatum(d.u.a[i]);
        d.u.a = a;
        break;
    }
    return d;
}

Datum
newhdat(d)                          /* similiar to newdatum, but save on heap */
    Datum       d;
{
    string      s;
    Datum       *a;
    int         i;

/*
        fprintf(stderr, "\nnewhdat:%d:",d.type);
        print(d);
        fflush(stdout);
        fprintf(stderr, "\n");
*/

    switch (d.type) {
    case STRING:
        if (d.u.s) {
            s = (string) getheap(strlen(d.u.s) + 1);
            d.u.s = (string) strcpy(s, d.u.s);
        } else
            s = d.u.s;
        break;

    case LIST:
        a = (Datum *) getheap((d.u.a[0].u.i + 1) * sizeof(Datum));
        for (i = 0; i <= d.u.a[0].u.i; i++) {
            /* DEBUG
            Datum x;
            x = newhdat(d.u.a[i]);
            a[i] = x;                               <------ here
            */
            /*-
                Sun4 has strange system error (BUS error) here when
                d is a large nested list.
                    Try eden:> symboltable();
                This error do not reproduce on Sun3.
                Not likely a heap overflow since there is still  a large
                spare heap space.
                May a bug on Sun4's cc compiler ?! who knows?
            -*/

            a[i] = newhdat(d.u.a[i]);
        }
        d.u.a = a;
        break;
    }

        /* DEBUG
        fprintf(stderr, "newhdat:-----");
        print(d);
        fflush(stdout);
```

```
        fprintf(stderr,"\n\n");
        */
    return d;
}

void
addr()                          /* push addr of data */
{
    symptr      sp;

    sp = (symptr) (*pc++);
    DEBUGPRINT("** addr %s\n", sp->name);
    if (sp->stype == FORMULA && sp->changed) {
        update(sp);
    }
    address(sp);
}

update(sp)
    symptr      sp;
{
    execute(sp->inst);
    freedatum(sp->d);
    sp->d = newdatum(pop());
    change(sp, FALSE);
}

address(sp)                     /* put the address of sp onto stack */
    symptr      sp;
{
    Datum       d;

    d.type = SYMBOL | DPTR;
    d.u.v.x = (Int) &sp->d;
    d.u.v.y = (Int) sp;
    push(d);
}

/**
   Take a variable name from stack, and look up from symbol table.
   Put the address onto stack.
   If symbol not found, create it.
**/

void
lookup_address()
{
    Datum       d;
    symptr      sp;

    muststr(d = pop());
    sp = lookup(d.u.s);         /* look up for name */
    if (sp == NULL)
        sp = install(d.u.s, VAR, UNDEF, 0);
    address(sp);                /* put the address of sp onto stack */
}

void
localaddr()
{
    Datum       d;
    Int         n;

    n = (Int) *pc++;
    DEBUGPRINT("** local addr %d\n", n);
    d.type = LOCALV | DPTR;
```



```
    d.u.v.x = (Int) &fp->stackp[n];
    d.u.v.y = n;
    push(d);
}

void
indexcalc()
{
    Datum       addr, *dp, index;
    Int         i;

    DEBUGPRINT("** index calc\n", 0);
    mustint(index = pop());     /* index */
    i = index.u.i;
    get_address(addr);

    if (address_type(addr) == DPTR) {
        dp = dptr(addr);

        switch (dp->type) {
        case LIST:
            CheckRange(i, 1, dp->u.a[0].u.i);
            addr.type = (addr.type & ADDRESSMASK) | DPTR;
            addr.u.v.x = (Int) &(dp->u.a[i]);
            break;

        case STRING:
            CheckRange(i, 1, strlen(dp->u.s));
            --i;
            addr.type = (addr.type & ADDRESSMASK) | CPTR;
            addr.u.v.x = (Int) &(dp->u.s[i]);
            break;

        default:
            dimension_error();
            break;
        }
    } else
        dimension_error();
    push(addr);
}

void
makelist()
{
    makearr((Int) *pc++);
}

makearr(n)
    Int         n;
{
    Datum       arr, d;

    arr.type = LIST;
    arr.u.a = (Datum *) getheap((n + 1) * sizeof(Datum));
    arr.u.a[0].type = INTEGER;
    arr.u.a[0].u.i  = n;
    while (n > 0) {
        d = pop();
        arr.u.a[n--] = d;
    }
    push(arr);                  /* return the list */
}

void
getvalue()
```

```
{                                       /* eval */
    Datum       d, addr;
    symptr      sp;
    extern Int *autocalc;

    DEBUGPRINT("** getvalue\n", 0);
    get_address(addr);
    sp = symbol_of(addr);

    if (*autocalc && is_symbol(addr)
        && sp->stype == FORMULA && sp->changed && ready(sp)) {
        update(sp);
    }
    switch (address_type(addr)) {
    case DPTR:
        push(*dptr(addr));
        break;

    case CPTR:
        dpush(d, CHAR, *cptr(addr) & 0xFF);
        break;

    default:
        address_error();
        break;
    }
}

void
sel()
{
    Datum       index, dat;
    Int         i;

    DEBUGPRINT("** select\n", 0);
    mustint(index = pop());
    i = index.u.i;

    dat = pop();
    switch (dat.type) {
    case LIST:
        CheckRange(i, 1, dat.u.a[0].u.i);
        push(dat.u.a[i]);
        break;

    case STRING:
        --i;
        CheckRange(i, 0, strlen(dat.u.s));
        dpush(dat, CHAR, dat.u.s[i] & 0xFF);
        break;

    default:
        dimension_error();
    }
}

void
listsize()
{                                       /* return the current size of list */
    Datum       d;

    DEBUGPRINT("** list size\n", 0);
    d = pop();
    switch (d.type) {
    case CHAR:
        dpush(d, INTEGER, 1);
```



```
        break;

    case STRING:
        dpush(d, INTEGER, strlen(d.u.s));
        break;

    case LIST:
        push(d.u.a[0]);
        break;

    default:
        pushUNDEF();
        break;
    }
}

void
shift()
{
    Datum       addr, *dp;
    Int         i;

    DEBUGPRINT("** shift\n", 0);
    get_address(addr);
    mustlist(*(dp = dptr(addr)));

    if (dp->u.a[0].u.i > 0) {
        freedatum(dp->u.a[1]);
        for (i = 1; i < dp->u.a[0].u.i; i++)
            dp->u.a[i] = dp->u.a[i + 1];
        --(dp->u.a[0].u.i);
        ChangeSym(addr);
    } else
        error("zero sized list found in 'shift'");
}

void
append()
{
    Datum       addr, *dp;
    Datum       d, *a;

    DEBUGPRINT("** append\n", 0);
    d = pop();                  /* value */

    get_address(addr);
    mustlist(*(dp = dptr(addr)));

    a = (Datum *) erealloc(dp->u.a, (dp->u.a[0].u.i + 2) * sizeof(Datum));
    a[++a[0].u.i] = newdatum(d);
    dp->u.a = a;
    ChangeSym(addr);
}

void
insert()
{
    Datum       addr, *dp;
    Datum       d, p, *a;
    Int         i, pos;

    DEBUGPRINT("** insert\n", 0);
    d = pop();                  /* value */
    mustint(p = pop());         /* position */
    pos = p.u.i;
```

```
    get_address(addr);
    mustlist(*(dp = dptr(addr)));

    CheckRange(pos, 1, dp->u.a[0].u.i + 1);
    a = (Datum *) erealloc(dp->u.a, (dp->u.a[0].u.i + 2) * sizeof(Datum));
    for (i = a[0].u.i; i >= pos; --i)   /* move backward */
        a[i + 1] = a[i];
    a[pos] = newdatum(d);       /* insert */
    ++a[0].u.i;
    dp->u.a = a;
    ChangeSym(addr);
}

void
delete()
{
    Datum       addr, *dp;
    Datum       p, *a;
    Int         i, pos;

    DEBUGPRINT("** delete\n", 0);
    mustint(p = pop());         /* position */
    pos = p.u.i;

    get_address(addr);
    mustlist(*(dp = dptr(addr)));

    CheckRange(pos, 1, dp->u.a[0].u.i);
    a = dp->u.a;
    freedatum(a[pos]);          /* delete */
    for (i = pos + 1; i <= a[0].u.i; i++)       /* move forward */
        a[i - 1] = a[i];
    --a[0].u.i;
    dp->u.a = a;
    ChangeSym(addr);
}

void
query()
{
    Datum       addr;
    symptr      sp;
    symptr_QUEUE *P, *Q;

    DEBUGPRINT("** query\n", 0);

    get_address(addr);
    sp = symbol_of(addr);

    switch (sp->stype) {
    case FORMULA:
        printf("%s is%s\n", sp->name, sp->text);
        break;

    case FUNCTION:
    case PROCEDURE:
        printf("%s %s",
               sp->stype == FUNCTION ? "func" : "proc",
               sp->name);
        Q = &sp->sources;
        for (P = FRONT(Q); P; P = NEXT(Q, P)) {
            printf("%c %s", P == Q->next ? ':' : ',', P->obj->name);
        }
        printf("\n");
        if (sp->text)
            printf("%s\n", sp->text);
```



```
        break;

    default:
        print(sp->d);
        printf("\n");
        break;
    }

    printf("%s -> {", sp->name);
    Q = &sp->targets;
    for (P = FRONT(Q); P; P = NEXT(Q, P)) {
        printf(P == Q->next ? "%s" : ", %s", P->obj->name);
    }
    printf("};\n");
    pushUNDEF();
}
```

```
/*************************************
 *      REFERENCE LIST MAINTAINER      *
 *************************************/

#include "eden.h"
#include "y.tab.h"

symptr_QUEUE IDlist = EMPTYQUEUE(IDlist);

Int    *autocalc;

addID(id)                       /* INTERNAL: ADD AN IDENTIFIER */
    symptr  id;                 /* USED BY PASER */
{
    if (!id || !IN_symptr_Q(&IDlist, id))
        APPEND_symptr_Q(&IDlist, id);
}

clear_IDlist()
{
    CLEAR_symptr_Q(&IDlist);
}

symptr_QUEUE *
save_IDlist()
{
    register symptr_ATOM A;
    register symptr_QUEUE *P, *Q;

    P = &IDlist;
    Q = (symptr_QUEUE *) emalloc(sizeof(symptr_QUEUE));
    CLEAN_Q(Q);
    while (A = LAST(P)) {
        DELETE_ATOM(P, A);
        if (A->obj == (symptr) 0) {
            DESTROY_symptr_ATOM(A);
            break;
        }
        INSERT_Q(Q, A);
    }
    return Q;
}

#ifdef DEBUG
printlist(Q)                    /* INTERNAL: PRINT A DEPENDENCY LIST */
    symptr_QUEUE *Q;            /* USED FOR DEBUGGING ONLY */
{
    register symptr_ATOM P;

    FOREACH(P, Q)
        printf(" %s", P->obj ? P->obj->name : "*");
    printf("\n");
}

#endif

refer_to(sp, Q)                 /* INTERNAL: APPEND SYMBOL TO SOURCES */
    symptr  sp;
    symptr_QUEUE *Q;
{
    register symptr_ATOM Symbol;
    register symptr_QUEUE *SourceTargets, *Sources;

    Sources = &sp->sources;

    FOREACH(Symbol, Sources) {
```



```
        SourceTargets = &Symbol->obj->targets;
        DELETE_FIRST_symptr(SourceTargets, sp);
    }

    CLEAR_symptr_Q(Sources);

    FOREACH(Symbol, Q) {
        APPEND_symptr_Q(Sources, Symbol->obj);
    }

    FOREACH(Symbol, Sources) {
        SourceTargets = &Symbol->obj->targets;
        APPEND_symptr_Q(SourceTargets, sp);
    }
}

refer_by(sp, Q)                 /* INTERNAL: APPEND SYMBOL TO TARGETS */
    symptr  sp;
    symptr_QUEUE *Q;
{
    register symptr id;
    register symptr_ATOM Symbol;
    register symptr_QUEUE *Targets, *Sources;

    FOREACH(Symbol, Q) {
        id = Symbol->obj;
        Sources = &id->sources;
        DELETE_FIRST_symptr(Sources, sp);
        APPEND_symptr_Q(Sources, sp);
        Targets = &sp->targets;
        DELETE_FIRST_symptr(Targets, id);
        APPEND_symptr_Q(Targets, id);
    }
}

checkok(sp, Q)                  /* INTERNAL: IF SYMBOL IN THE LIST ? */
    symptr  sp;                 /* TRUE  = 1 = NOT IN THE LIST */
    symptr_QUEUE *Q;            /* FALSE = 0 = ALREADY EXISTED */
{
    register symptr id;
    register symptr_ATOM P;

    FOREACH(P, Q) {
        id = P->obj;
        if (id == sp)
            return FALSE;
        if (!checkok(sp, &id->sources))
            return FALSE;
    }
    return TRUE;
}

ready(sp)                       /* INTERNAL: CHECK CHILDREN OF SYMBOL */
    symptr  sp;                 /* TRUE = ALL CHILDREN ARE UP-TO-DATE */
{
    register symptr_ATOM P;
    register symptr_QUEUE *Q;

    Q = &sp->sources;
    FOREACH(P, Q) {
        if (P->obj->changed)
            return FALSE;
    }
    return TRUE;
}
```

```
/*********************************************
 *      SYMBOL TABLE: INSTALL AND LOOK-UP     *
 *********************************************/

#include "eden.h"
#include "y.tab.h"
#include "symptr.q.c"    /* some of the code is used by other modules */
#include "hash.h"

char   *
emalloc(n)                          /* memory allocation with error checking */
    int     n;                      /* n = size */
{
    char   *p, *malloc();

    p = malloc(n);
    if (p == 0)
        error("out of memory");
    return p;
}

/*---------------------------------------------------------------------
                        SYMBOL HASH TABLE
---------------------------------------------------------------------*/
/* HASHSIZE is defined in "hash.h": must be power of 2 minus 1 */
hashindex(s)
    string  s;
{
    int     i = 0;

    while (*s)
        i ^= *s++;
    return i & HASHSIZE;
}

symptr  hashtable[HASHSIZE+1] = {(symptr) 0};   /* init to nulls */

symptr
lookup(s)                           /* FIND SYMBOL OF NAME S */
    char   *s;                      /* 0 = NOT FOUND, ELSE = SYMBOL POINTER */
{
    symptr  sp;

    for (sp = hashtable[hashindex(s)]; sp != (symptr) 0; sp = sp->next)
        if (strcmp(sp->name, s) == 0)
            return sp;
    return 0;                       /* 0 ==> not found */
}

symptr
install(s, st, t, i)                /* INTERNAL: INSTALL A SYMBOL IN SYMBOL TABLE */
    char   *s;                      /* s  = symbol name */
    int     st;                     /* st = symbol type */
    int     t;                      /* t  = initial data type */
    int     i;                      /* i  = initial data value */
{
    extern symptr hashtable[];
    int     indx;
    symptr  sp;
    char   *emalloc();

    sp = (symptr) emalloc(sizeof(symbol));
    sp->name = emalloc(strlen(s) + 1);  /* +1 for '\0' */
    strcpy(sp->name, s);
    sp->stype = st;
    sp->inst = (Inst *) 0;
```



```
    sp->nauto = 0;
    sp->changed = TRUE;
    sp->text = (string) 0;
    sp->d.type = t;
    sp->d.u.i = i;
    CLEAN_Q(&sp->targets);
    CLEAN_Q(&sp->sources);
    sp->next = hashtable[indx = hashindex(s)];  /* put at front of list */
    hashtable[indx] = sp;
    return sp;
}


/*---------------------------------------------------------------------
                        LOCAL VARIABLE LOOK UP
---------------------------------------------------------------------*/


struct entry {
    string  name;
    short   level;
    short   num;
};

typedef struct entry entry;

#define streq(X,Y)              (strcmp(X,Y)==0)
#define EQUAL_entry(A,B)        streq(A.name,B.name)
#define FREE_entry(A)           if(A.name)free(A.name)

#include "entry.q.c"

static entry_QUEUE      LocalVarList = EMPTYQUEUE(LocalVarList);

/*
 * look up the position of local variable
 * 0 = not in LocalVarList
 */
entry_ATOM
search_local(name)
    string  name;
{
    entry_ATOM A;

    FOREACH(A, &LocalVarList) {
        if (A->obj.name == (string) 0)
            break;
        if (streq(A->obj.name, name))   /* match */
            return A;
    }
    return (entry_ATOM) 0;
}

/*
 *      add a local variable to the list
 *      return the entry of the list
 */
entry_ATOM
add_local_variable(name)
    string  name;
{
    entry   E;
    entry_ATOM F;
#ifdef sun
    string  strdup();
#endif
```

```
    F = FRONT(&LocalVarList);
#ifdef sun
    E.name = strdup(name);
#else
    E.name = emalloc(strlen(name)+1);
    strcpy(E.name, name);
#endif
    E.level = indef;
    if (inpara) {
        E.num = F->obj.num - 1;
    } else {                        /* inauto */
        if (F->obj.num < 0)
            E.num = 1;
        else
            E.num = F->obj.num + 1;
    }
    INSERT_entry_Q(&LocalVarList, E);
    return FRONT(&LocalVarList);
}

/*
 *      initialist the LocalVarList
 */
init_LocalVarList()
{
    CLEAN_Q(&LocalVarList);
    insert_level_marker(0);
}

/*
 *      insert a level marker
 */
insert_level_marker(level)
{
    entry   E;

    E.name = (string) 0;
    E.level = level;
    E.num = 0;
    INSERT_entry_Q(&LocalVarList, E);
}

/* delete all entry not smaller than the level */
delete_local_level(level)
{
    entry_ATOM  E;
    entry_QUEUE *Q = &LocalVarList;

    for (E = FRONT(Q); E->obj.level >= level; E = FRONT(Q))
        DELETE_FRONT_entry(Q);
}

int
local_declare(name)
    string  name;
{
    entry_ATOM  E;

    if (search_local(name))
        error2("redeclare local variable:", name);
    E = add_local_variable(name);
    return E->obj.num;
}
```



```
int
lookup_local(name)
    string  name;
{
    entry_ATOM  E;
    if (E = search_local(name)) {
        if (E->obj.level != indef)
            error2("local variable is not in this level:", name);
        else
            return E->obj.num;
    } else
        return 0;
}

printlocal()
{
    entry_ATOM E;

    FOREACH(E, &LocalVarList)
        printf("%s %d %d\n", E->obj.name ? E->obj.name : "*",
               E->obj.level, E->obj.num);
}


symptr_QUEUE break_q = EMPTYQUEUE(break_q);
symptr_QUEUE cont_q = EMPTYQUEUE(cont_q);

dispatch(p, Q)
    Inst    *p;
    symptr_QUEUE *Q;
{
    APPEND_symptr_Q(Q, (symptr) p);
}

patch(mark, p, Q)
    Inst    *mark, *p;
    symptr_QUEUE *Q;
{
    symptr_ATOM A;
    Inst    *ip;

    while (A = LAST(Q)) {
        ip = (Inst *) A->obj;
        DELETE_symptr_ATOM(Q, A);
        if (ip == mark)
            return;
        ip[1] = (Inst) (p - (ip + 2));
    }
    /* should not get here */
    error("compiler error while patching code");
}
```

```
/*****************************************************************
 *                    TYPE CONVERSION FUNCTIONS                  *
 *****************************************************************/
#include "eden.h"
#include "builtin.h"
#include "y.tab.h"
#include <stdio.h>

extern Datum newhdat();

static struct {
    short   type;
    string  name;
}       datatype[] = {

    UNDEF, "@",
    REAL, "float",
    INTEGER, "int",
    CHAR, "char",
    STRING, "string",
    LIST, "list",
    FORMULA, "formula",
    FUNCTION, "func",
    PROCEDURE, "proc",
    BLTIN, "builtin",
    LIB, "C-func",
    RLIB, "C-func",
    VAR, "var",
    0, "???"
};

Datum
ctos(d)                             /* convert char to string */
    Datum   d;
{
    if (ischar(d)) {                /* only if Datum is char */
        string  s;

        s = (string) getheap(2);/* put the string on heap */
        *s = d.u.i;
        s[1] = '\0';
        d.u.s = s;
        d.type = STRING;
    }
    if (!isstr(d))
        error("string needed");
    return d;
}

string
typename(type)
{
    int     i;

    for (i = 0; datatype[i].type; i++) {
        if (type == datatype[i].type)
            break;
    }
    return datatype[i].name;
}

/* type(data): type of datum in the form of a string */
void
t_type()
{                                   /* data type name */
    Datum   d;
```



```
    short   type;
    int     i;

    if (paracount == 0)
        usage("s = type(expr);");
    type = para(1).type;
    for (i = 0; datatype[i].type; i++) {
        if (type == datatype[i].type)
            break;
    }
    dpush(d, STRING, datatype[i].name);
}

/* int(data): convert datum to an integer */
void
t_int()
{
    Datum   d;

    if (paracount == 0)
        usage("i = int(expr);");
    d = para(1);

    if (is_symbol(d)) {
        d.u.i = d.u.v.x;
    } else {
        switch (d.type) {
        case REAL:
            d.u.i = d.u.r;
        case INTEGER:
        case CHAR:
            break;

        case STRING:
            d.u.i = atoi(d.u.s);
            break;

        default:
            pushUNDEF();                /* return @ */
            return;
        }
    }

    d.type = INTEGER;
    push(d);
}

/* float(data): builtin function convert datum to an integer */
void
t_float()
{
    Datum   d;
    double  atof();

    if (paracount == 0)
        usage("f = float(expr);");
    d = para(1);

    if (is_symbol(d)) {
        d.u.r = d.u.v.x;
    } else {
        switch (d.type) {
        case INTEGER:
        case CHAR:
            d.u.r = d.u.i;
```

```
        case REAL:
            break;

        case STRING:
            d.u.r = atof(d.u.s);
            break;

        default:
            pushUNDEF();                /* return @ */
            return;
        }
    }
    d.type = REAL;
    push(d);
}

/* char(datum): builtin function convert datum to a character */
void
t_char()
{
                                /* convert to char */
    Datum   d;

    if (paracount == 0)
        usage("c = char(expr);");
    d = para(1);
    switch (d.type) {
    case REAL:
        d.u.i = d.u.r;
    case INTEGER:
    case CHAR:
        break;

    case STRING:                /* the 1st char of string */
        d.u.i = *d.u.s & 0377;  /* '\0' if s == "" */
        break;

    default:
        pushUNDEF();            /* return @ */
        return;
    }
    d.type = CHAR;
    push(d);
}

/* str(datum): builtin function convert datum to a string */
void
t_str()
{
    Datum   d;
    char    buf[STRLEN];

    if (paracount == 0)
        usage("c = str(expr);");
    d = para(1);
    switch (d.type) {
    case UNDEF:
        buf[0] = '@';
        buf[1] = '\0';
        break;

    case REAL:
        sprintf(buf, "%lf", d.u.r);
        break;

    case INTEGER:
        sprintf(buf, "%d", d.u.i);
```



```
        break;

    case CHAR:
        buf[0] = d.u.i;
        buf[1] = '\0';
        break;

    case STRING:
        push(d);
        return;

    default:
        pushUNDEF();            /* return @ */
        return;
    }
    d.type = STRING;
    d.u.s = buf;
    d = newhdat(d);             /* put it on heap */
    push(d);
}

/*----
        _type_convert(datum, typename): builtin function convert datum
        to type specified by typename.

        ! ! !   DANGER --- MAY CAUSE CLASH IF NOT CAREFUL   ! ! !

        no type checking. no memory allocation. extremely dangerous to
        convert into string or list.
----*/
void
t_super()
{                               /* super type convertion */
    Datum   d;
    Datum   t;
    char    *s;
    int     i;

    if (paracount < 2)
        usage("data = _type_convert(data, \"type\");");
    d = para(1);
    t = para(2);
    muststr(t);
    s = t.u.s;
    for (i = 0; datatype[i].type; i++) {
        if (strcmp(s, datatype[i].name) == 0)
            break;
    }
    d.type = datatype[i].type;
    push(d);
}

type_clash_err(type)
{
    error2("type clash: expecting ", typename(type));
}

type_clash_addr_err()
{
    error("type clash: expecting reference of variable");
}
```

```
#ifdef DEBUG
extern int Debug;
#endif Debug

#define bool char

#define FALSE  (0)
#define TRUE   (1)

typedef int Int;                /* a pointer compatible integer */
typedef void (*Inst) ();
typedef char * string;

#define STOP    (Inst)0

typedef struct symbol *symptr;

typedef struct Datum Datum;

#include "symptr.q.h"


typedef struct symbol {
    char   *name;               /* variable name */
    short   stype;              /* symbol type */
    Inst   *inst;               /* instruction entry */
    unsigned nauto;             /* no. of local variables */
    string  text;               /* original text of definition, if any */
    bool    changed;            /* flag */
    struct Datum {
        short   type;           /* data type */
        union {
            double  r;
            Int     i;          /* integer/char */
            string  s;          /* string */
            Datum  *a;          /* list */
            symptr  sym;        /* symbol pointer */
            struct {
                Int     x;
                Int     y;
            } v;
        } u;
    } d;
    symptr_QUEUE sources, targets;
    symptr  next;               /* to link to another */
} symbol;

/* bylist & tolist: for backward compatibility */
#define bylist targets
#define tolist sources

symptr  install(), lookup();

extern error2();
#define error(s)  error2(s, (char *)0)

extern Datum UndefDatum;
extern Inst *progp, *progbase, prog[];
extern Inst *code(), *code_related_by(), *code_definition();

extern union compiler_flags {
    struct {
        unsigned define_level   : 8;
        unsigned loop_level     : 8;
        unsigned switch_level   : 8;
        unsigned formula        : 1;
```



```
        unsigned local_declare  : 1;
        unsigned arg_declare    : 1;
    }       s;
    int all;            /* an int is 32 bit */
}       compiler_flag;

#define reset_compiler_flags()  (compiler_flag.all = 0)

#define indef           compiler_flag.s.define_level
#define informula       compiler_flag.s.formula
#define inauto          compiler_flag.s.local_declare
#define inpara          compiler_flag.s.arg_declare
#define inloop          compiler_flag.s.loop_level
#define inswitch        compiler_flag.s.switch_level

extern Datum newdatum(), newhdat();
extern  freedatum();

#include "inst.h"

/*----- typecheck.h -----*/
#define ADDRESSMASK     0xF000
#define SYMBOL          0x1000
#define LOCALV          0x2000
#define POINTERTYPE     0x0FFF
#define DPTR            0x0001
#define CPTR            0x0002
#define isundef(d)      (d.type == UNDEF)
#define ischar(d)       (d.type == CHAR)
#define isstr(d)        (d.type == STRING)
#define isint(d)        (d.type == INTEGER)
#define isnum(d)        (d.type==REAL||d.type==INTEGER||d.type==CHAR)
#define is_address(d)   (d.type & ADDRESSMASK)
#define is_symbol(d)    (is_address(d)==SYMBOL)
#define is_local(d)     (is_address(d)==LOCALV)
#define address_type(d) (d.type & POINTERTYPE)
#define dptr(d)         ((Datum*)d.u.v.x)
#define cptr(d)         ((string)d.u.s)
#define symbol_of(d)    ((symptr)d.u.v.y)
#define local(d)        (d.u.v.y)

/*----- code.h -----*/
#define NSTACK  256
extern Datum stack[];           /* the stack */
extern Datum *stackp;           /* next free spot on stack */
#define top_of_stack            stackp[-1]              /* top of stack */
#define reset_stack()           (stackp=stack)
extern Datum stack_underflow_err();
#define push(d) \
        if (stackp>=&stack[NSTACK]) stack_overflow_err(); else *stackp++ = d
#define pop()   \
        (stackp==stack? stack_underflow_err() : *--stackp)

#define NPROG   5000

extern Inst prog[];             /* the machine */
extern Inst *progp;             /* next free spot for code generation */
extern Inst *pc;                /* program counter during execurtion */
#define reset_prog_ptr()        (progp=prog)

typedef struct Frame {          /* proc/func call stack frame */
    symptr  sp;                 /* symbol table entry */
    Inst   *retpc;              /* where to resume after return */
    Datum  *stackp;             /* the stack to be resume */
    char   *hptr;               /* heap pointer */
} Frame;
```

```
#define NFRAME  100

extern Frame frame[];
extern Frame *fp;               /* frame pointer */
#define reset_frames()          (fp=frame)

/* INPUT DEVICE TYPES */
#define STRING_DEV      0
#define FILE_DEV        1

#define MAXENTRY 200
struct t {
    Datum  *dp;
    Inst   *ip;
};

extern struct t entry_tbl[];
extern struct t *entry_ptr;
#define reset_entry_tbl()       (entry_ptr=entry_tbl)

#define STRLEN  1024

/*----- refer -----*/
#define NID 100
extern symptr_QUEUE IDlist;
extern  addID(), refer(), clearID();

/*----- heap -----*/
#define HEAPSIZE                0x30000
extern char *hptr, heap[];
extern char *getheap();
extern  freeheap();
#define reset_heap()            (hptr=heap)

/*----- eval -----*/
extern void reset_eval(), mark_changed(), change();
```

```
#if INCLUDE != 'T'

#define paracount       fp->stackp->u.a[0].u.i
#define para(n)         fp->stackp->u.a[n]
#define dpush(x,y,z)    x.type=y; x.u.i=(Int)(z); push(x)

struct BLIBTBL {
        string  name;
        void    (*func)();
};

extern struct BLIBTBL blibtbl[];

extern usage();

#endif

/*-------------------------------------------------------------*/

#if INCLUDE == 'T'
#define BIND(name,func) name,func,
#else
#define BIND(name,func) extern void func();
#endif

/*-------------------------------------------------------------*/

BIND(   "exit",                 b_exit          )
BIND(   "execute",              exec_string     )
BIND(   "include",              exec_file       )
BIND(   "include_file",         exec_file       )
BIND(   "apply",                apply           )
BIND(   "substr",               substr          )
BIND(   "strcat",               scat            )
BIND(   "sublist",              sublist         )
BIND(   "listcat",              lcat            )
BIND(   "type",                 t_type          )
BIND(   "int",                  t_int           )
BIND(   "float",                t_float         )
BIND(   "char",                 t_char          )
BIND(   "str",                  t_str           )
BIND(   "_type_convert",        t_super         )
BIND(   "write",                b_write         )
BIND(   "writeln",              writeln         )
BIND(   "array",                array           )
BIND(   "nameof",               nameof          )
BIND(   "formula_list",         formula_list    )
BIND(   "action_list",          action_list     )
BIND(   "symboltable",          symtbl2list     )
BIND(   "forget",               forget          )
BIND(   "pack",                 pack            )
BIND(   "error",                user_error      )
BIND(   "get_msgq",             get_msgq        )
BIND(   "send_msg",             send_message    )
BIND(   "receive_msg",          receive_message )
BIND(   "remove_msgq",          remove_msgq     )
BIND(   "error_no",             error_no        )
BIND(   "backgnd",              backgnd         )
BIND(   "pipe",                 pipe_process    )
BIND(   "getenv",               get_environ     )
BIND(   "putenv",               put_environ     )
BIND(   "printhash",            printhash       )
BIND(   "touch",                touch           )

#ifdef DEBUG
BIND(   "debug",                debug           )
```



```
#endif DEBUG

#undef BIND
```

```
#ifndef STAMP_QUEUE

struct queue {
        struct queue *prev;
        struct queue *next;
        char obj;       /* dummy */
} ;

typedef struct queue QUEUE;
#define EMPTYQUEUE(Q) {&Q, &Q}

#define CLEAN_Q(Q) ((Q)->prev=(Q)->next=(Q))
#define Q_EMPTY(Q) ((Q)->next==(Q))
#define DELETE_ATOM(Q,A) \
        (((A)->prev->next=(A)->next)->prev=(A)->prev,(A)->prev=(A)->next=0)
#define INSERT_ATOM(Q,A) \
        ((((A)->prev=((A)->next=(Q))->prev)->next=(A)),(Q)->prev=(A))
#define APPEND_ATOM(Q,A) \
        ((((A)->next=((A)->prev=(Q))->next)->prev=(A)),(Q)->next=(A))
#define APPEND_Q(Q,A) INSERT_ATOM(Q,A)
#define INSERT_Q(Q,A) APPEND_ATOM(Q,A)
#define NEXT(Q,A) ((Q)==(A)->next?0:(A)->next)
#define PREV(Q,A) ((Q)==(A)->prev?0:(A)->prev)
#define FRONT(Q) NEXT(Q,Q)
#define LAST(Q) PREV(Q,Q)

#ifdef vax
#define ALLOC_ATOM(A) (A=(void*)calloc(1,sizeof(*A)))
#else
#define ALLOC_ATOM(A) ((void*)A=(void*)calloc(1,sizeof(*A)))
#endif

#define FOREACH(A,Q) for(A=FRONT(Q);A;A=NEXT(Q,A))
#define STAMP_QUEUE
#endif
```

```
        /*
        **      HEADER FILE: symptr.q.h
        */

#include "global.q.h"

#ifndef STAMP_symptr_QUEUE

struct symptr_queue {
        struct symptr_queue *prev;
        struct symptr_queue *next;
        symptr obj;
} ;

typedef struct symptr_queue symptr_QUEUE;
typedef symptr_QUEUE * symptr_ATOM;
extern symptr_QUEUE *SEARCH_symptr();

#ifndef NEW_symptr
#define NEW_symptr(OBJ) OBJ
#endif

#ifndef FREE_symptr
#define FREE_symptr(OBJ)        /* do nothing */
#endif

#ifndef EQUAL_symptr
#define EQUAL_symptr(A,B) ((A)==(B))
#endif

#define DESTROY_symptr_ATOM(A) \
        { \
        FREE_symptr((A)->obj); \
        free(A); \
        }

#define DELETE_NTH_symptr(Q,OBJ,n) DELETE_symptr_ATOM(Q,SEARCH_symptr(Q,OBJ,n))
#define DELETE_FIRST_symptr(Q,OBJ) DELETE_symptr_ATOM(Q,SEARCH_symptr(Q,OBJ,1))
#define DELETE_FRONT_symptr(Q) DELETE_symptr_ATOM(Q,FRONT(Q))
#define DELETE_LAST_symptr(Q) DELETE_symptr_ATOM(Q,LAST(Q))

#define CLEAR_symptr_Q(Q) \
        while (!Q_EMPTY(Q)) DELETE_symptr_ATOM(Q,(Q)->next)

#define APPEND_symptr_Q(Q,OBJ) \
        { \
        symptr_ATOM A; \
        ALLOC_ATOM(A); \
        A->obj = NEW_symptr(OBJ); \
        APPEND_Q(Q,A); \
        }

#define CONCAT_symptr_Q(Qdst, Qsrc) \
        { \
        symptr_QUEUE *P; \
        FOREACH (P, Qsrc) APPEND_symptr_Q(Qdst, P->obj); \
        }

#define IN_symptr_Q(Q,OBJ) SEARCH_symptr(Q,OBJ,1)

#define STAMP_symptr_QUEUE
#endif
```

```
#define HASHSIZE 127    /* 1 smaller than power of 2 */
```

```
extern  void    eval(), add(), sub(), mul(), div(), mod(), negate();
extern  void    constpush(), varpush(), pushint(), pushUNDEF();
extern  void    localaddr(), addr(), indexcalc();
extern  void    getvalue(), sel(), makelist();
extern  void    concat();
extern  void    assign(), inc_asgn(), dec_asgn();
extern  void    pre_inc(), post_inc();
extern  void    pre_dec(), post_dec();
extern  void    cnv_2_bool(), gt(), lt(), eq(), ne(), ge(), le();
extern  void    not(), and(), or();
extern  void    popd(), ddup(), jmp(), jpz(), jpnz();
extern  void    assign();
extern  void    definition();
extern  void    shift(), append();
extern  void    insert(), delete();
extern  void    switchcode();
extern  void    makelist(), listsize();
extern  void    lookup_address(), query();
```

```
/* keyword translation table */

"auto",         AUTO,
"para",         PARA,
"return",       RETURN,
"break",        BREAK,
"continue",     CONTINUE,
"if",           IF,
"else",         ELSE,
"do",           DO,
"while",        WHILE,
"for",          FOR,
"switch",       SWITCH,
"case",         CASE,
"default",      DEFAULT,
"shift",        SHIFT,
"append",       APPEND,
"insert",       INSERT,
"delete",       DELETE,
"func",         FUNC,
"proc",         PROC,
"is",           IS,
```

```
        /*
        **      C FUNCTIONS: symptr.q.c
        */

#include "symptr.q.h"

        /* return the position of the n-th object in a symptr queue */
symptr_QUEUE *
SEARCH_symptr(Q, obj, n)
symptr_QUEUE *Q;
symptr obj;
int n;
{
        symptr_QUEUE *P;

        FOREACH (P, Q) {
                if (EQUAL_symptr(obj, P->obj) && !--n)
                        return P;
        }
        return (symptr_QUEUE*) 0;       /* NOT FOUND */
}

DELETE_symptr_ATOM(Q,A)
symptr_QUEUE *Q;
symptr_ATOM A;
{
        if (A && A != Q) {
                A->prev->next = A->next;
                A->next->prev = A->prev;
                DESTROY_symptr_ATOM(A);
        }
}

MOVE_symptr_Q(Qsrc, Qdst)
symptr_QUEUE *Qsrc, *Qdst;
{
        if (!Q_EMPTY(Qsrc)) {
                (Qdst->prev->next = Qsrc->next)->prev = Qdst->prev;
                (Qsrc->prev->next = Qdst)->prev = Qsrc->prev;
                CLEAN_Q(Qsrc);
        }
}
```

```
        /*
        **      C FUNCTIONS: entry.q.c
        */

#include "entry.q.h"

        /* return the position of the n-th object in a entry queue */
entry_QUEUE *
SEARCH_entry(Q, obj, n)
entry_QUEUE *Q;
entry obj;
int n;
{
        entry_QUEUE *P;

        FOREACH (P, Q) {
                if (EQUAL_entry(obj, P->obj) && !--n)
                        return P;
        }
        return (entry_QUEUE*) 0;        /* NOT FOUND */
}

DELETE_entry_ATOM(Q,A)
entry_QUEUE *Q;
entry_ATOM A;
{
        if (A && A != Q) {
                A->prev->next = A->next;
                A->next->prev = A->prev;
                DESTROY_entry_ATOM(A);
        }
}

MOVE_entry_Q(Qsrc, Qdst)
entry_QUEUE *Qsrc, *Qdst;
{
        if (!Q_EMPTY(Qsrc)) {
                (Qdst->prev->next = Qsrc->next)->prev = Qdst->prev;
                (Qsrc->prev->next = Qdst)->prev = Qsrc->prev;
                CLEAN_Q(Qsrc);
        }
}
```

```
        /*
        **      HEADER FILE: entry.q.h
        */

#include "global.q.h"

#ifndef STAMP_entry_QUEUE

struct entry_queue {
        struct entry_queue *prev;
        struct entry_queue *next;
        entry obj;
} ;

typedef struct entry_queue entry_QUEUE;
typedef entry_QUEUE * entry_ATOM;
extern entry_QUEUE *SEARCH_entry();

#ifndef NEW_entry
#define NEW_entry(OBJ) OBJ
#endif

#ifndef FREE_entry
#define FREE_entry(OBJ) /* do nothing */
#endif

#ifndef EQUAL_entry
#define EQUAL_entry(A,B) ((A)==(B))
#endif

#define DESTROY_entry_ATOM(A) \
        { \
        FREE_entry((A)->obj); \
        free(A); \
        }

#define DELETE_NTH_entry(Q,OBJ,n) DELETE_entry_ATOM(Q,SEARCH_entry(Q,OBJ,n))
#define DELETE_FIRST_entry(Q,OBJ) DELETE_entry_ATOM(Q,SEARCH_entry(Q,OBJ,1))
#define DELETE_FRONT_entry(Q) DELETE_entry_ATOM(Q,FRONT(Q))
#define DELETE_LAST_entry(Q) DELETE_entry_ATOM(Q,LAST(Q))

#define CLEAR_entry_Q(Q) \
        while (!Q_EMPTY(Q)) DELETE_entry_ATOM(Q,(Q)->next)

#define APPEND_entry_Q(Q,OBJ) \
        { \
        entry_ATOM A; \
        ALLOC_ATOM(A); \
        A->obj = NEW_entry(OBJ); \
        APPEND_Q(Q,A); \
        }

#define INSERT_entry_Q(Q,OBJ) \
        { \
        entry_ATOM A; \
        ALLOC_ATOM(A); \
        A->obj = NEW_entry(OBJ); \
        INSERT_Q(Q,A); \
        }

#define CONCAT_entry_Q(Qdst, Qsrc) \
        { \
        entry_QUEUE *P; \
        FOREACH (P, Qsrc) APPEND_entry_Q(Qdst, P->obj); \
        }
```



```
#define IN_entry_Q(Q,OBJ) SEARCH_entry(Q,OBJ,1)

#define STAMP_entry_QUEUE
#endif
```

```
/***** Imakefile for making EDEN custom module *****/

#define EXEOBJ eden

#ifdef sparc
#define MACHINE s4
#else
#define MACHINE
#endif sparc

EDENVER=eden-v3
LibDir=$(HOME)/lib
L=$(LibDir)/lib$(EDENVER)MACHINE.a

YFLAGS  = -d
LFLAGS  =
CFLAGS  = -g -DDEBUG -I$(HOME)/EDEN/MAIN/$(EDENVER) $(Cflags)
LDFLAGS = -L$(LibDir) -l$(EDENVER)MACHINE $(LIBS) -lm

compile: EXEOBJ

#if COMPILE == 'c'
Hfiles  = curses.h
Ofiles  = curses.o
Cfiles  = curses.c
LIBS    = -lcurses -ltermlib
Cflags  = -DCURSES
curses.o  : curses.h
#endif

#if COMPILE == 's'
Hfiles  = sungraf.h
Ofiles  = sungraf.o
Cfiles  = sungraf.c
Cflags  = -fswitch -DSUNCORE
LIBS    = -lcore -lsunwindow -lpixrect
#endif

#if COMPILE == 'w'
Hfiles  = ww_info.h ww_builtin.h
Ofiles  = ww_builtin.o
Cfiles  = ww_builtin.c
Cflags  = -fswitch -DWWINFO
LIBS    = -lww -lsuntool -lsunwindow -lcore -lpixrect
#endif

OBJECT  = custom.o $(Ofiles)
CFILE   = custom.c $(Cfiles)
INCLUDE = customlib.h $(Hfiles)

SOURCE  = $(CFILE) $(INCLUDE)
printer=gl
PRINTOPT = -n -uh -w80 -f ''

EXEOBJ: $(OBJECT) $(L)
        $(CC) $(CFLAGS) $(OBJECT) -o EXEOBJ $(LDFLAGS)
        @ chmod a+rx EXEOBJ

custom.o: customlib.h

print:  $(SOURCE) Imakefile
        $(HOME)/bin/pg $(PRINTOPT) -h 'EDEN|(File: *  Page %)|PAGE #' $? \
        | lpr -P$(printer)
        @ touch print

printall: $(SOURCE) Imakefile
```



```
        $(HOME)/bin/pg $(PRINTOPT) -h 'EDEN|(File: *  Page %)|PAGE #' $? \
        | lpr -P$(printer)

Other.a: $(SOURCE) Imakefile
        @ echo SAVING $?
        @- chmod +w Other.a
        @ ar r Other.a $?
        @ chmod -w Other.a

save: $(SOURCE) Imakefile
        cp $? save /cs/acad/wmb/public/eden
        @ touch save
```

```
/*********************************************************
 *       MACROS FUNCTIONS OF "CURSES" ARE REWRITTEN      *
 *                  INTO TRUE FUNCTIONS                  *
 *********************************************************/

#ifdef CURSES

#include        "eden.h"

#define INCLUDE 'H'
#include        "curses.h"
#undef INCLUDE

w_initscr()
{
        symbol *sp;

        initscr();
        sp = lookup("stdscr");
        sp->d.u.i = (Int) stdscr;
        change(sp, TRUE);
        sp = lookup("curscr");
        sp->d.u.i = (Int) curscr;
        change(sp, TRUE);
}

clearok(win, bf)
WINDOW *win;
int    bf;
{
        win->_clear = bf;
}

leaveok(win, bf)
WINDOW *win;
int    bf;
{
        win->_leave = bf;
}

scrollok(win, bf)
WINDOW *win;
int    bf;
{
        win->_scroll = bf;
}

flushok(win, bf)
WINDOW *win;
int    bf;
{
        if (bf)
                win->_flags |= _FLUSH;
        else
                win->_flags &= ~_FLUSH;
}

gety(win)
WINDOW *win;
{
        return win->_cury;
}

getx(win)
WINDOW *win;
{
```



```
        return win->_curx;
}

winch(win)
WINDOW *win;
{
        win->_y[win->_cury][win->_curx] & 0177;
}

raw()
{
        _tty.sg_flags |= RAW;
        _pfast = _rawmode = TRUE;
        stty(_tty_ch, &_tty);
}

noraw()
{
        _tty.sg_flags &= ~RAW;
        _rawmode = FALSE;
        _pfast = !(_tty.sg_flags & CRMOD);
        stty(_tty_ch, &_tty);
}

crmode()
{
        _tty.sg_flags |= CBREAK;
        _rawmode = TRUE;
        stty(_tty_ch, &_tty);
}

nocrmode()
{
        _tty.sg_flags &= ~CBREAK;
        _rawmode = FALSE;
        stty(_tty_ch, &_tty);
}

echo()
{
        _tty.sg_flags |= ECHO;
        _echoit = TRUE;
        stty(_tty_ch, &_tty);
}

noecho()
{
        _tty.sg_flags &= ~ECHO;
        _echoit = FALSE;
        stty(_tty_ch, &_tty);
}

nl()
{
        _tty.sg_flags |= CRMOD;
        _pfast = _rawmode;
        stty(_tty_ch, &_tty);
}

nonl()
{
        _tty.sg_flags &= ~CRMOD;
        _pfast = TRUE;
        stty(_tty_ch, &_tty);
}
```

```
savetty()
{
        gtty(_tty_ch, &_tty);
        _res_flg = _tty.sg_flags;
}

resetty()
{
        _tty.sg_flags = _res_flg;
        stty(_tty_ch, &_tty);
}

#endif
```

```
/***********************************
 *      SYSTEM INITIALIZATION      *
 ***********************************/

#include "eden.h"
#include "y.tab.h"
#include <stdio.h>
#include "custom.h"

#define INCLUDE 'H'
#include "builtin.h"
#include "customlib.h"
#undef  INCLUDE

struct ILIBTBL ilibtbl[] = {
#define INCLUDE 'I'
#include "customlib.h"
#undef  INCLUDE
        0, 0
};

struct RLIBTBL rlibtbl[] = {
#define INCLUDE 'R'
#include "customlib.h"
#undef  INCLUDE
        0, 0
};

#ifdef WWINFO
#define INCLUDE 'H'
#include "ww_builtin.h"
#undef  INCLUDE
struct {
        string  name;
        int     (*func)();
} wwlibtbl[] = {
#define INCLUDE 'T'
#include "ww_builtin.h"
#undef  INCLUDE
        0, 0
};
#endif WWINFO

install_custom_variables()
{
        /* PRE-DEFINED VARIABLES */
#define INSTALL_VAR(name, value) \
        (install(name, VAR, INTEGER, value))->changed = FALSE

#ifdef CURSES
        INSTALL_VAR("stdscr", stdscr);
        INSTALL_VAR("curscr", curscr);
#endif CURSES

        /** these are standard input/output file pointers **/
        /** don't change them **/
        INSTALL_VAR("stdin",  stdin);
        INSTALL_VAR("stdout", stdout);
        INSTALL_VAR("stderr", stderr);

#ifdef WWINFO
        {
            int         i;
            symptr      s;

            for (i = 0; wwlibtbl[i].name; i++) {
```



```
                s = install(wwlibtbl[i].name, BLTIN, BLTIN, 0);
                s->inst = (Inst *) wwlibtbl[i].func;
                s->d.type = BLTIN;
                s->d.u.sym = s;
            }
        }
#endif WWINFO
}
```

```
/************************************************
**                                             **
**      EDEN/Sun Core Graphics Interface       **
**                                             **
************************************************/

#include <stdio.h>
#include <usercore.h>

int     pixwindd();
/*
int     cgpixwindd();
*/

/*---------------------------------------+--------------------+
                                        | SC_new_view_surface |
                                        +--------------------*/
      /*
       * This function allocates space for a view surface.
       */
struct vwsurf *
SC_new_view_surface(flags, ptr)
int     flags;
char    *ptr;
{
        extern char   *emalloc();
        struct vwsurf *vs;

        vs = (struct vwsurf *) emalloc(sizeof(struct vwsurf));
        vs->ptr = (char **) emalloc(2*(sizeof(char *)));
        vs->ptr[0] = ptr;
        vs->ptr[1] = 0;
        vs->flags = flags;
        vs->dd = pixwindd;
        vs->screenname[0] = vs->windowname[0] = vs->cmapname[0] = '\0';
        vs->windowfd = vs->instance = vs->cmapsize = 0;
        return vs;
}

/*-----------------------------------+-------------------------+
                                    | SC_new_color_view_surface |
                                    +-------------------------*/
      /*
       * This function allocates space for a view surface.
       */
struct vwsurf *
SC_new_color_view_surface(flags, ptr)
int     flags;
char    *ptr;
{
/*
        char   *emalloc();
        struct vwsurf *vs;

        vs = (struct vwsurf *) emalloc(sizeof(struct vwsurf));
        vs->ptr = (char **) emalloc(2*sizeof(char *));
        vs->ptr[1] = 0;
        vs->flags = flags;
        vs->ptr[0] = ptr;
        strcpy(vs->screenname, "");
        strcpy(vs->windowname, "");
        strcpy(vs->cmapname, "");
        vs->dd = cgpixwindd;
        vs->windowfd = vs->instance = 0;
        vs->cmapsize = 256;
        return vs;
```



```
*/
}

/*---------------------------------------+----------------------+
                                        | SC_initialize_Basic2D |
                                        +----------------------*/
SC_initialize_Basic2D()
{
        initialize_core(BASIC, NOINPUT, TWOD);
}

/*-----------------------------------------------+----------------+
                                                | SC_save_raster |
                                                +----------------*/
SC_save_raster(vsurf, filename)
struct vwsurf *vsurf;
char *filename;
{
        struct  suncore_raster raster;
        FILE    *f;
        int     fd;
        int     error_number;
        float   xmin, xmax, ymin, ymax;
        struct  {
                int     type;
                int     nbytes;
                char    *data;
                } map;

        map.type = 1;
        map.nbytes = 0;
        map.data = NULL;

        inquire_viewport_2(&xmin, &xmax, &ymin, &ymax);
        size_raster(vsurf, xmin, xmax, ymin, ymax, &raster);
        allocate_raster(&raster);
        if (raster.bits == 0) {
                report_most_recent_error(&error_number);
                print_error("raster.bits == 0:", error_number);
        }
        get_raster(vsurf, xmin, xmax, ymin, ymax, 0, 0, &raster);
        f = fopen(filename, "w");
        if (f == 0) {
                fprintf(stderr, "%s: file cannot open\n", filename);
        } else {
                fd = fileno(f);
                if (fd == -1) exit(1);
                if (raster_to_file(&raster, &map, fd, 1)) {
                        report_most_recent_error(&error_number);
                        print_error("Error:", error_number);
                }
                fclose(f);
        }
        free_raster(&raster);
}

/*-----------------------------------------+--------------------+
                                          | SC_restore_raster |
                                          +--------------------*/
SC_restore_raster(filename)
char    *filename;
{
        struct  suncore_raster raster;
        FILE    *f;
        int     fd;
        float   xmin, xmax, ymin, ymax;
```

```
        struct  {
                int     type;
                int     nbytes;
                char    *data;
        } map;

        map.type = 1;
        map.nbytes = 0;
        map.data = NULL;

        f = fopen(filename, "r");
        if (f == 0) {
                fprintf(stderr, "%s: file not found\n", filename);
        } else {
                fd = fileno(f);
                file_to_raster(fd, &raster, &map);
                fclose(f);
                put_raster(&raster);
                free_raster(&raster);
        }
}
```

```
/*****************************************************************
 *                                                               *
 *                        ww_builtin.c                           *
 *                                                               *
 *   This file contains the definitions of the EDEN builtin      *
 *   function version of those ww functions having their         *
 *   arguments or return values are structures                   *
 *                                                               *
 *****************************************************************/

#include "eden.h"
#include "builtin.h"
#include "y.tab.h"
#include <stdio.h>
#include "wwinfo.h"

#define P(i)        para(i)
#define RET         pushUNDEF()
#define RETINT(j)   {Datum d;d.type=INTEGER;d.u.i = (int)j;push(d);}
#define RETBOX(b)   {Datum d;d=UNBOX(b);push(d);}
#define ARGNUM(i)   if(paracount<i)error("insufficient argument")
#define GETSTR(d)   (muststr(d),d.u.s)
#define RETWWSTATE(b)  {Datum d;d=b;push(d);}
#define RETFONTINFO(b) {Datum d;d=b;push(d);}

GETINT(d)
Datum d;
{
        switch (d.type) {
        case INTEGER:
        case CHAR:
        case STRING:
                return d.u.i;
        default:
                error("GETINT: type mismatch");
        }
}

/*
   Ii returns the ith argument, that should be of type integer
*/
#define I(i)    GETINT(P(i))
#define I1      I(1)
#define I2      I(2)
#define I3      I(3)
#define I4      I(4)
#define I5      I(5)
#define I6      I(6)

/*
   Bi returns the ith argument, that should be of type box.
*/
#define B(i)    BOX(P(i))
#define B1      B(1)
#define B2      B(2)
#define B3      B(3)
#define B4      B(4)

/*
   Ei returns the ith argument, that should be of type exrep.
   union {
        bitmap* ex_bitmap;
        cursor* ex_cursor;
        int     ex_gray;
   } exrep
*/
```



```
#define E(i)    GETINT(P(i))
#define E1      E(1)
#define E2      E(2)
#define E3      E(3)
#define E4      E(4)

/*
   Si returns the ith argument, that should be of type string
*/
#define S(i)    GETSTR(P(i))
#define S1      S(1)

/*
   Convert a list of 4 integers to a box
*/
box
BOX(d)
Datum d;
{
        box     b;

        if (d.type != LIST || d.u.a[0].u.i < 4)
                error("BOX: Conversion Error");
        b.b_left   = GETINT(d.u.a[1]);
        b.b_top    = GETINT(d.u.a[2]);
        b.b_right  = GETINT(d.u.a[3]);
        b.b_bottom = GETINT(d.u.a[4]);
        return b;
}

/*
   Convert a box into a list of 4 integers
*/
Datum
UNBOX(b)
box     b;
{
        Datum d;
        int     i;

        d.type = LIST;
        d.u.a = (Datum *) getheap(5*(sizeof (Datum)));
        for (i = 0; i < 5; i++)
                d.u.a[i].type = INTEGER;
        d.u.a[0].u.i = 4;
        d.u.a[1].u.i = b.b_left;
        d.u.a[2].u.i = b.b_top;
        d.u.a[3].u.i = b.b_right;
        d.u.a[4].u.i = b.b_bottom;
        return d;
}

/*
   Convert a list of 4 integers into a magic structure
*/
magic
MAGIC(d)
Datum d;
{
        magic   m;

        if (d.type != LIST || d.u.a[0].u.i < 4)
                error("MAGIC: Conversion Error");
        m.mc_erase   = GETINT(d.u.a[1]);
        m.mc_kill    = GETINT(d.u.a[2]);
        m.mc_delword = GETINT(d.u.a[3]);
```

```
        m.mc_retype  = GETINT(d.u.a[4]);
        return m;
}

/*
   Comvert a magic structure into a list of 4 integers
*/
Datum
UNMAGIC(m)
magic m;
{
        Datum d;
        int     i;

        d.type = LIST;
        d.u.a = (Datum *) getheap(5*(sizeof (Datum)));
        for (i = 0; i < 5; i++)
                d.u.a[i].type = INTEGER;
        d.u.a[0].u.i = 4;
        d.u.a[1].u.i = m.mc_erase;
        d.u.a[2].u.i = m.mc_kill;
        d.u.a[3].u.i = m.mc_delword;
        d.u.a[4].u.i = m.mc_retype;
        return d;
}

/*
   Returns a list of 16 elements which corresponds to the structure
   pointed by state(wwstate *)
*/
Datum
POPWWSTATE(state)
int     state;
{
        Datum d;
        wwstate         *s;

        d.type = LIST;
        d.u.a = (Datum *) getheap(17*(sizeof (Datum)));
        s = (wwstate *)state;
        d.u.a[0].type = INTEGER; d.u.a[0].u.i = 16;
        d.u.a[1].type = INTEGER; d.u.a[1].u.i = s->d_line;
        d.u.a[2].type = INTEGER; d.u.a[2].u.i = s->d_rop;
        d.u.a[3].type = INTEGER; d.u.a[3].u.i = s->d_buttons;
        d.u.a[4].type = INTEGER; d.u.a[4].u.i = s->d_x;
        d.u.a[5].type = INTEGER; d.u.a[5].u.i = s->d_y;
        d.u.a[6] = UNMAGIC(s->d_line);
        d.u.a[7].type = STRING;  d.u.a[7].u.s = s->d_errmsg;
        d.u.a[8].type = INTEGER; d.u.a[8].u.i = s->d_flags;
        d.u.a[9].type = INTEGER; d.u.a[9].u.i = s->d_event;
        d.u.a[10].type = CHAR;    d.u.a[10].u.i = s->d_char;
        d.u.a[11].type = INTEGER; d.u.a[11].u.i = (int)(s->d_newwindow);
        d.u.a[12].type = INTEGER; d.u.a[12].u.i = s->d_select;
        d.u.a[13].type = INTEGER; d.u.a[13].u.i = s->d_fore;
        d.u.a[14].type = INTEGER; d.u.a[14].u.i = s->d_back;
        d.u.a[15].type = INTEGER; d.u.a[15].u.i = s->d_colours;
        d.u.a[16].type = INTEGER; d.u.a[16].u.i = (int)(s->d_ipwait);
        return d;
}

/*
   Change the fields of the structure pointed by state(wwstate *)
   to the values of the elements in slist
*/
PUSHWWSTATE(state, slist)
int     state;
```



```
Datum slist;
{
        wwstate *s;

        if (slist.type != LIST || slist.u.a[0].u.i < 16)
                error("pustwwstate: Conversion Error");
        s = (wwstate *)state;
        s->d_line = GETINT(slist.u.a[1]);
        s->d_rop = GETINT(slist.u.a[2]);
        s->d_buttons = GETINT(slist.u.a[3]);
        s->d_x = GETINT(slist.u.a[4]);
        s->d_y = GETINT(slist.u.a[5]);
        s->d_magic = MAGIC(slist.u.a[6]);
        strcpy(s->d_errmsg, GETSTR(slist.u.a[7]));
        s->d_flags = GETINT(slist.u.a[8]);
        s->d_event = GETINT(slist.u.a[9]);
        s->d_char = (char)GETINT(slist.u.a[10]);
        s->d_newwindow = (window *)GETINT(slist.u.a[11]);
        s->d_select = GETINT(slist.u.a[12]);
        s->d_fore = GETINT(slist.u.a[13]);
        s->d_back = GETINT(slist.u.a[14]);
        s->d_colours = GETINT(slist.u.a[15]);
        s->d_ipwait = (int (*)())GETINT(slist.u.a[16]);
}

/*
   Returns a list of 5 elements which corresponds to the structure
   pointed by font(fontinfo *)
*/
Datum
POPFONTINFO(font)
int     font;
{
        Datum d;
        fontinfo        *f;

        d.type = LIST;
        d.u.a = (Datum *) getheap(6*(sizeof (Datum)));
        f = (fontinfo *)font;
        d.u.a[0].type = INTEGER; d.u.a[0].u.i = 5;
        d.u.a[1].type = STRING;  d.u.a[1].u.s = f->f_name;
        d.u.a[2].type = INTEGER; d.u.a[2].u.i = f->f_height;
        d.u.a[3].type = INTEGER; d.u.a[3].u.i = f->f_width;
        d.u.a[4].type = INTEGER; d.u.a[4].u.i = (int)(f->f_cursor);
        d.u.a[5].type = INTEGER; d.u.a[5].u.i = (int)(f->f_junk);
        return d;
}

/*
   Change the fields of the structure pointed by font(fontinfo *)
   to the values of the elements in finfo
*/
PUSHFONTINFO(font, finfo)
int     font;
Datum finfo;
{
        fontinfo *f;

        if (finfo.type != LIST || finfo.u.a[0].u.i < 5)
                error("pushfontinfo: Conversion Error");
        f = (fontinfo *)font;
        f->f_name   = GETSTR(finfo.u.a[1]);
        f->f_height = GETINT(finfo.u.a[2]);
        f->f_width  = GETINT(finfo.u.a[3]);
        f->f_cursor = (cursor *)GETINT(finfo.u.a[4]);
        f->f_junk   = (int *)GETINT(finfo.u.a[5]);
```

```
}

/*
   The actual function called by EDEN which performs POPWWSTATE
*/
popwwstate()
{
        ARGNUM(1);
        RETWWSTATE(POPWWSTATE(I1));
}

/*
   The actual function called by EDEN which performs PUSHWWSTATE
*/
pushwwstate()
{
        ARGNUM(2);
        PUSHWWSTATE(I1, P(2));
        RET;
}

/*
   The actual function called by EDEN which performs POPFONTINFO
*/
popfontinfo()
{
        ARGNUM(1);
        RETFONTINFO(POPFONTINFO(I1));
}

/*
   The actual function called by EDEN which performs PUSHFONTINFO
*/
pushfontinfo()
{
        ARGNUM(2);
        PUSHFONTINFO(I1, P(2));
        RET;
}

/*
   EDEN version of bmbox()
*/
E_bmbox()
{
        void    bmbox();

        ARGNUM(2);
        bmbox(B1, I2);
        RET;
}

/*
   EDEN version of bmclip()
*/
E_bmclip()
{
        box     bmclip();

        ARGNUM(2);
        RETBOX(bmclip(I1, B2));
}

/*
   EDEN version of bmcopy()
*/
```



```
E_bmcopy()
{
        bitmap *bmcopy();

        ARGNUM(3);
        RETINT(bmcopy(I1, B2, I3));
}

/*
   EDEN version of bmexchange()
*/
E_bmexchange()
{
        void    bmexchange();

        ARGNUM(4);
        bmexchange(I1, B2, I3, B4);
        RET;
}

/*
   EDEN version of bmgrey()
*/
E_bmgrey()
{
        void    bmgrey();

        ARGNUM(2);
        bmgrey(B1, I2);
        RET;
}

/*
   EDEN version of bmmove()
*/
E_bmmove()
{
        void    bmmove();

        ARGNUM(3);
        bmmove(B1, I2, I3);
        RET;
}

/*
   EDEN version of bmscroll()
*/
E_bmscroll()
{
        void    bmscroll();

        ARGNUM(3);
        bmscroll(B1, I2, I3);
        RET;
}

/*
   EDEN version of bmslide()
*/
E_bmslide()
{
        void    bmslide();

        ARGNUM(3);
        bmslide(B1, I2, I3);
        RET;
```

```
}

/*
   EDEN version of bmxbox()
*/
E_bmxbox()
{
        void    bmxbox();

        ARGNUM(3);
        bmxbox(I1, B2, I3);
        RET;
}

/*
   EDEN version of bmxcopy()
*/
E_bmxcopy()
{
        void    bmxcopy();

        ARGNUM(5);
        bmxcopy(I1, B2, I3, B4, I5);
        RET;
}

/*
   EDEN version of boxbuild()
*/
E_boxbuild()
{
        box     boxbuild();

        ARGNUM(4);
        RETBOX(boxbuild(I1, I2, I3, I4));
}

/*
   EDEN version of boxInside()
*/
E_boxInside()
{
        ARGNUM(3);
        RETINT(boxInside(B1, I2, I3));
}

/*
   EDEN version of boxop()
*/
E_boxop()
{
        box     boxop();

        ARGNUM(3);
        RETBOX(boxop(B1, B2, I3));
}

/*
   EDEN version of boxshift()
*/
E_boxshift()
{
        box     boxshift();

        ARGNUM(3);
        RETBOX(boxshift(B1, I2, I3));
```



```
}

/*
   EDEN version of boxzoom()
*/
E_boxzoom()
{
        box     boxzoom();

        ARGNUM(2);
        RETBOX(boxzoom(B1, I2));
}

/*
   EDEN version of exwrite()
*/
E_exwrite()
{
        ARGNUM(4);
        RETINT(exwrite(I1, I2, I3, E4));
}

/*
   EDEN version of ftbox()
*/
E_ftbox()
{
        box     ftbox();

        ARGNUM(4);
        RETBOX(ftbox(I1, I2, I3, I4));
}

/*
   EDEN version of ftprint()
*/
E_ftprint()
{
        void    ftprint();

        ARGNUM(3);
        ftprint(B1, I2, I3);
        RET;
}

/*
   EDEN version of treecreate()
*/
E_treecreate()
{
        treeinfo *treecreate();

        ARGNUM(2);
        RETINT(treecreate(I1, B2));
}

/*
   EDEN version of treecwd()
*/
E_treecwd()
{
        treeinfo *treecwd();

        ARGNUM(2);
        RETINT(treecwd(I1, B2));
}
```

```
/*
   EDEN version of txcreate()
*/
E_txcreate()
{
        txinfo *txcreate();

        ARGNUM(1);
        RETINT(txcreate(B1));
}

/*
   EDEN version of txrecreate()
*/
E_txrecreate()
{
        void    txrecreate();

        ARGNUM(2);
        txrecreate(I1, B2);
        RET;
}

/*
   EDEN version of wwfnbox()
*/
E_wwfnbox()
{
        void    wwfnbox();

        ARGNUM(2);
        wwfnbox(B1, I2);
        RET;
}

/*
   EDEN version of wwget()
*/
E_wwget()
{
        window *wwget();
        window *w;
        symptr  sym;

        ARGNUM(1);
        w = wwget(B1);                  /* perform wwget() */
        if ((sym = lookup("dd"))) {     /* is EDEN variable dd present? */
                sym->d.type = INTEGER;  /* No, create dd */
                sym->d.u.i = (int) dd;
                sym->changed = FALSE;
                change(sym);
        } else {                        /* Yes, update dd */
                sym = install("dd", VAR, INTEGER, dd);
                sym->changed = FALSE;
        }
        if ((sym = lookup("ddfont"))) { /* is EDEN variable ddfont present? */
                sym->d.type = INTEGER;  /* No, create ddfont */
                sym->d.u.i = (int) ddfont;
                sym->changed = FALSE;
                change(sym);
        } else {                        /* Yes, update ddfont */
                sym = install("ddfont", VAR, INTEGER, ddfont);
                sym->changed = FALSE;
        }
        RETINT(w);
```



```
}

/*
   EDEN version of wwinit()
*/
E_wwinit()
{
        symptr  sym;

        wwinit();                       /* perform wwinit() */
        if ((sym = lookup("dd"))) {     /* Is EDEN variable dd present? */
                sym->d.type = INTEGER;  /* No, create dd */
                sym->d.u.i = (int) dd;
                sym->changed = FALSE;
                change(sym);
        } else {                        /* Yes, update dd */
                sym = install("dd", VAR, INTEGER, dd);
                sym->changed = FALSE;
        }
        if ((sym = lookup("ddfont"))) { /* Is EDEN variable ddfont present? */
                sym->d.type = INTEGER;  /* No, create ddfont */
                sym->d.u.i = (int) ddfont;
                sym->changed = FALSE;
                change(sym);
        } else {                        /* Yes, update ddfont */
                sym = install("ddfont", VAR, INTEGER, ddfont);
                sym->changed = FALSE;
        }
        RET;
}

/*
   EDEN version of wwxget()
*/
E_wwxget()
{
        window *wwxget();
        window *w;
        symptr  sym;

        ARGNUM(4);
        w = wwxget(B1, I2, I3, I4);     /* perform wwxget() */
        if ((sym = lookup("dd"))) {     /* Is EDEN variable dd present? */
                sym->d.type = INTEGER;  /* No, create dd */
                sym->d.u.i = (int) dd;
                sym->changed=FALSE;
                change(sym);
        } else {                        /* Yes, update dd */
                sym = install("dd", VAR, INTEGER, dd);
                sym->changed = FALSE;
        }
        if ((sym = lookup("ddfont"))) { /* Is EDEN variable ddfont present? */
                sym->d.type = INTEGER;  /* No, create ddfont */
                sym->d.u.i = (int) ddfont;
                sym->changed=FALSE;
                change(sym);
        } else {                        /* Yes, update ddfont */
                sym = install("ddfont", VAR, INTEGER, ddfont);
                sym->changed = FALSE;
        }
        RETINT(w);
}

/*
   EDEN version of xxbar()
*/
```

```
E_xxbar()
{
        ARGNUM(6);
        RETINT(xxbar(B1, I2, I3, I4, I5, I6));
}

/*
   EDEN version of xxoutline()
*/
E_xxoutline()
{
        void    xxoutline();

        ARGNUM(2);
        xxoutline(I1, B2);
        RET;
}

/*
   EDEN version of xxrelease()
*/
E_xxrelease()
{
        ARGNUM(1);
        RETINT(xxrelease(B1));
}

/*
   EDEN version of xxrotate()
*/
E_xxrotate()
{
        void    xxrotate();

        ARGNUM(3);
        xxrotate(B1, I2, I3);
        RET;
}
```

```
#ifdef CURSES

#if INCLUDE == 'H'

# include       <stdio.h>
# include       <sgtty.h>

#ifndef bool
#define bool char
#endif

#ifndef FALSE
#define FALSE (0)
#endif

#ifndef TRUE
#define TRUE (1)
#endif

# define        reg     register

# define        ERR     (0)
# define        OK      (1)

# define        _ENDLINE        001
# define        _FULLWIN        002
# define        _SCROLLWIN      004
# define        _FLUSH          010
# define        _STANDOUT       0200
# define        _NOCHANGE       -1

typedef struct sgttyb   SGTTY;

/*
 * From the tty modes...
 */

extern bool     NONL, UPPERCASE, normtty, _pfast;

struct _win_st {
        short           _cury, _curx;
        short           _maxy, _maxx;
        short           _begy, _begx;
        short           _flags;
        short           _ch_off;
        bool            _clear;
        bool            _leave;
        bool            _scroll;
        char            **_y;
        short           *_firstch;
        short           *_lastch;
        struct _win_st  *_nextp, *_orig;
};

# define        WINDOW  struct _win_st

extern bool     My_term, _echoit, _rawmode, _endwin;

extern char     *Def_term, ttytype[];

extern int      LINES, COLS, _tty_ch, _res_flg;

extern SGTTY    _tty;

extern WINDOW   *stdscr, *curscr;
```



```
/* ------------------------ lib.h --------------------------- */

extern  w_initscr();
extern  waddch();       /* add char on window */
extern  waddstr();      /* add string on window */
extern  box();          /* draws a box around the window */
extern  clearok();      /* sets the clear flag */
extern  wclear();       /* clear the window */
extern  wclrtobot();    /* clear to bottom */
extern  wclrtoeol();    /* clear to end of line */
extern  wdelch();       /* delete a char */
extern  wdeleteln();    /* delete line */
extern  werase();       /* erase window */
extern  winsch();       /* insert char */
extern  winsertln();    /* insert line */
extern  wmove();        /* move the cursor */
extern  overlay();      /* overlay windows */
extern  overwrite();    /* overwrite windows */
extern  wprintw();      /* printf on windows */
extern  wrefresh();     /* refresh window */
extern  wstandout();    /* set standout mode */
extern  wstandend();    /* stop standout mode */
        /* I/O */
extern  crmode();       /* cbreak mode */
extern  nocrmode();     /* end cbreak mode */
extern  echo();         /* echo mode */
extern  noecho();       /* no echo mode */
extern  wgetch();       /* get a char thru' window */
extern  wgetstr();      /* get a string thru' window */
extern  raw();          /* raw mode */
extern  noraw();        /* end raw mode */
extern  wscanw();       /* scan window */
        /* miscellanous functions */
extern  newwin();       /* create a new window */
extern  delwin();       /* delete window */
extern  endwin();       /* end window before exit */
extern  getx();         /* get x coord */
extern  gety();         /* get y coord */
extern  winch();        /* char under cursor */
extern  initscr();      /* init screen */
extern  leaveok();
extern  mvcur();        /* actually move cursor */
extern  nl();           /* set nl mode */
extern  nonl();         /* stop nl mode */
extern  scrollok();     /* scroll ok */
extern  scroll();       /* scroll the window */

#endif  /* if INCLUDE == 'H' */

/* ------------------------------------------------------------ */

#if INCLUDE == 'T'

        /***** curses (window package) *****/

"waddch",       waddch,         /* add char on window */
"waddstr",      waddstr,        /* add string on window */
"box",          box,            /* draws a box around the window */
"clearok",      clearok,        /* sets the clear flag */
"wclear",       wclear,         /* clear the window */
"wclrtobot",    wclrtobot,      /* clear to bottom */
"wclrtoeol",    wclrtoeol,      /* clear to end of line */
"wdelch",       wdelch,         /* delete a char */
"wdeleteln",    wdeleteln,      /* delete line */
"werase",       werase,         /* erase window */
```

```
"winsch",       winsch,         /* insert char */
"winsertln",    winsertln,      /* insert line */
"wmove",        wmove,          /* move the cursor */
"overlay",      overlay,        /* overlay windows */
"overwrite",    overwrite,      /* overwrite windows */
"wprintw",      wprintw,        /* printf on windows */
"wrefresh",     wrefresh,       /* refresh window */
"wstandout",    wstandout,      /* set standout mode */
"wstandend",    wstandend,      /* stop standout mode */
        /* I/O */
"crmode",       crmode,
"nocrmode",     nocrmode,
"echo",         echo,
"noecho",       noecho,
"wgetch",       wgetch,
"wgetstr",      wgetstr,
"raw",          raw,
"noraw",        noraw,
"wscanw",       wscanw,
        /* miscellanous functions */
"newwin",       newwin,         /* create a new window */
"delwin",       delwin,         /* delete window */
"endwin",       endwin,         /* end window before exit */
"getx",         getx,           /* get x coord */
"gety",         gety,           /* get y coord */
"winch",        winch,          /* char under cursor */
"initscr",      w_initscr,      /* init screen */
"leaveok",      leaveok,
"mvcur",        mvcur,          /* actually move cursor */
"nl",           nl,             /* set nl mode */
"nonl",         nonl,           /* stop nl mode */
"scrollok",     scrollok,       /* scroll ok */
"scroll",       scroll,         /* scroll the window */

#endif  /* if INCLUDE == 'T' */

#endif  /* ifdef CURSES */
```

```
/* ------------------------ library.h --------------------------- */

   /*****************************************************************\
  *******************************************************************
 **                                                                 **
**       The following macros are intended to reduce the           **
**       complexity of binding EDEN functions with C functions.    **
**       Programmers are adviced to use these macros in their      **
**       header files. The macros are:                             **
**                                                                 **
**       Function(Ename,Cname) -- bind Eden name to Cname func.     **
**       SameFunc(Name) -- Eden's name is the same as C's name.    **
**       SpecialF(Ename,Type,Cname)                                **
**               -- if the type of C func isn't int,               **
**                  Type specifies the type.                       **
 **                                                               **
  *******************************************************************
   \*****************************************************************/

#if INCLUDE == 'H'
#define SpecialF(Ename,Type,Cname)extern Type Cname();
#define Function(Ename,Cname)extern Cname();
#define SameFunc(Name)extern Name();
#define RealFunc(Ename,Cname)extern double Cname();
#define SameReal(Name)extern double Name();
#endif

#if INCLUDE == 'T'
#define SpecialF(Ename,Type,Cname){"Ename",(int(*)())Cname},
#define Function(Ename,Cname){"Ename",Cname},
#define SameFunc(Name){"Name",Name},
#define RealFunc(Ename,Cname)
#define SameReal(Name)
#endif

#if INCLUDE == 'R'
#define SpecialF(Ename,Type,Cname)
#define Function(Ename,Cname)
#define SameFunc(Name)
#define RealFunc(Ename,Cname){"Ename",Cname},
#define SameReal(Name){"Name",Name},
#endif

   /*****************************************************************\
  *******************************************************************
 **                                                                 **
**       The following include files containes EDEN/C binding      **
**       header files.                                             **
**       If programmers do not use the macros, the format of the   **
**       header files should be:                                   **
**               #if INCLUDE == 'H'                                **
**               extern C_name();        C function declaration    **
**               ...                     etc.                      **
**               #endif                                            **
**               #if INCLUDE == 'T'                                **
**               {"Eden_name",C_name,0), binding declaration       **
**                                       the 0 means int function  **
**                                           1 means double        **
**               ...                     etc.                      **
**               #endif                                            **
 **                                                               **
  *******************************************************************
   \*****************************************************************/

#ifdef SUNCORE
#include "sungraf.h"
```



```
#endif

#ifdef CURSES
#include "curses.h"
#endif

/*      <<<<------ Any more header files should be inserted here */


   /*****************************************************************\
  *******************************************************************
 **                                                                 **
**       The following declarations bind some standard             **
**       C functions to EDEN names.                                **
**                                                                 **
**       Note: no commas or semicolons are needed                  **
**             after the macros.                                   **
 **                                                               **
  *******************************************************************
   \*****************************************************************/

SameFunc(system)
SameFunc(fprintf)
#ifdef vax
SpecialF(sprintf,char *,sprintf)
#else
SameFunc(sprintf)
#endif
SpecialF(popen,FILE *,popen)
SpecialF(pclose,FILE *,pclose)
SameFunc(fscanf)
SameFunc(sscanf)
SpecialF(fopen,FILE *,fopen)
SameFunc(fclose)
SameFunc(fgetc)
SameFunc(ungetc)
SpecialF(gets,char *,gets)
SpecialF(fgets,char *,fgets)
SpecialF(fputc,char *,fputc)
SameFunc(putw)
SameFunc(setbuf)
SpecialF(trace,void,user_trace) /* produce trace message */
SpecialF(eager,void,eager)      /* eagerly eval all def's and action's */

/* the followings are math functions */

SameReal(sqrt)
SameReal(sin)
SameReal(cos)
SameReal(tan)
SameReal(asin)
SameReal(acos)
SameReal(atan)
SameReal(atan2)
SameReal(exp)
SameReal(log)
SameReal(log10)
SameReal(pow)
#ifndef vax
SameReal(exp2)                  /* vax don't have these functions */
SameReal(exp10)
SameReal(log2)
#endif not vax

/* end of macros */
```

```
#undef SpecialF
#undef Function
#undef SameFunc
#undef RealFunc
#undef SameReal
```

```
/*-----------------------------------------------------------------------
**      Custom Interface
**      see "sungraf.c"
*/
Function(initialize_Basic2D,SC_initialize_Basic2D)
Function(terminate_Basic2D,terminate_core)
Function(new_view_surface,SC_new_view_surface)
Function(new_color_view_surface,SC_new_color_view_surface)
Function(save_raster,SC_save_raster)
Function(restore_raster,SC_restore_raster)

/*-----------------------------------------------------------------------
**      SunCore Interface
**      Commented functions are not compatible with EDEN/C interface
**      or not essential in the implementation of DoNaLD (e.g. inquire_xxx).
**      c.f. "SunCore Reference Manual"
*/

/*
SameFunc(allocate_raster)
SameFunc(await_any_button)
SameFunc(await_any_button_get_locator_2)
SameFunc(await_any_button_get_valuator)
SameFunc(await_keyboard)
SameFunc(await_pick)
SameFunc(await_stroke_2)
*/
SameFunc(begin_batch_of_updates)
SameFunc(close_retained_segment)
SameFunc(close_temporary_segment)
SameFunc(create_retained_segment)
SameFunc(create_temporary_segment)
/*
SameFunc(define_color_indices)
*/
SameFunc(delete_all_retained_segments)
SameFunc(delete_retained_segment)
SameFunc(deselect_view_surface)
SameFunc(end_batch_of_updates)
/*
SameFunc(file_to_raster)
SameFunc(free_raster)
SameFunc(get_mouse_state)
SameFunc(get_raster)
SameFunc(get_view_surface)
*/
SameFunc(initialize_core)
SameFunc(initialize_device)
SameFunc(initialize_view_surface)
/*
SameFunc(inquire_charjust)
SameFunc(inquire_charpath_2)
SameFunc(inquire_charpath_3)
SameFunc(inquire_charprecision)
SameFunc(inquire_charsize)
SameFunc(inquire_charspace)
SameFunc(inquire_charup_2)
SameFunc(inquire_charup_3)
SameFunc(inquire_color_indices)
SameFunc(inquire_current_position_2)
SameFunc(inquire_current_position_3)
SameFunc(inquire_detectability)
SameFunc(inquire_echo)
SameFunc(inquire_echo_position)
SameFunc(inquire_echo_surface)
SameFunc(inquire_fill_index)
```



```
SameFunc(inquire_font)
SameFunc(inquire_highlighting)
SameFunc(inquire_image_transformation_2)
SameFunc(inquire_image_transformation_3)
SameFunc(inquire_image_transformation_type)
SameFunc(inquire_image_translate_2)
SameFunc(inquire_image_translate_3)
SameFunc(inquire_inverse_composite_matrix)
SameFunc(inquire_keyboard)
SameFunc(inquire_line_index)
SameFunc(inquire_linestyle)
SameFunc(inquire_linewidth)
SameFunc(inquire_locator_2)
SameFunc(inquire_marker_symbol)
SameFunc(inquire_ndc_space_2)
SameFunc(inquire_ndc_space_3)
SameFunc(inquire_open_retained_segment)
SameFunc(inquire_open_temporary_segment)
SameFunc(inquire_pen)
SameFunc(inquire_pick_id)
SameFunc(inquire_polygon_edge_style)
SameFunc(inquire_polygon_interior_style)
SameFunc(inquire_primitive_attributes)
SameFunc(inquire_projection)
SameFunc(inquire_rasterop)
SameFunc(inquire_retained_segment_names)
SameFunc(inquire_retained_segment_surfaces)
SameFunc(inquire_segment_detectability)
SameFunc(inquire_segment_highlighting)
SameFunc(inquire_segment_image_transformation_2)
SameFunc(inquire_segment_image_transformation_3)
SameFunc(inquire_segment_image_translate_2)
SameFunc(inquire_segment_image_translate_3)
SameFunc(inquire_segment_visibility)
SameFunc(inquire_stroke)
SameFunc(inquire_text_extent_2)
SameFunc(inquire_text_extent_3)
SameFunc(inquire_text_index)
SameFunc(inquire_valuator)
SameFunc(inquire_view_depth)
SameFunc(inquire_view_plane_distance)
SameFunc(inquire_view_plane_normal)
SameFunc(inquire_view_reference_point)
SameFunc(inquire_view_up_2)
SameFunc(inquire_view_up_3)
SameFunc(inquire_viewing_control_parameters)
SameFunc(inquire_viewing_parameters)
SameFunc(inquire_viewport_2)
SameFunc(inquire_viewport_3)
SameFunc(inquire_visibility)
SameFunc(inquire_window)
SameFunc(inquire_world_coordinate_matrix_2)
SameFunc(inquire_world_coordinate_matrix_3)
*/
SameFunc(line_abs_2)
SameFunc(line_abs_3)
SameFunc(line_rel_2)
SameFunc(line_rel_3)
SameFunc(map_ndc_to_world_2)
SameFunc(map_ndc_to_world_3)
SameFunc(map_world_to_ndc_2)
SameFunc(map_world_to_ndc_3)
SameFunc(marker_abs_2)
SameFunc(marker_abs_3)
SameFunc(marker_rel_2)
SameFunc(marker_rel_3)
```

```
SameFunc(move_abs_2)
SameFunc(move_abs_3)
SameFunc(move_rel_2)
SameFunc(move_rel_3)
SameFunc(new_frame)
SameFunc(polygon_abs_2)
SameFunc(polygon_abs_3)
SameFunc(polygon_rel_2)
SameFunc(polygon_rel_3)
SameFunc(polyline_abs_2)
SameFunc(polyline_abs_3)
SameFunc(polyline_rel_2)
SameFunc(polyline_rel_3)
SameFunc(polymarker_abs_2)
SameFunc(polymarker_abs_3)
SameFunc(polymarker_rel_2)
SameFunc(polymarker_rel_3)
SameFunc(print_error)
/*
SameFunc(put_raster)
SameFunc(raster_to_file)
*/
SameFunc(rename_retained_segment)
SameFunc(report_most_recent_error)
SameFunc(restore_segment)
SameFunc(save_segment)
SameFunc(select_view_surface)
SameFunc(set_back_plane_clipping)
SameFunc(set_charjust)
SameFunc(set_charpath_2)
SameFunc(set_charpath_3)
SameFunc(set_charprecision)
SameFunc(set_charsize)
SameFunc(set_charspace)
SameFunc(set_charup_2)
SameFunc(set_charup_3)
SameFunc(set_coordinate_system_type)
SameFunc(set_detectability)
SameFunc(set_drag)
SameFunc(set_echo)
/*
SameFunc(set_echo_group)
*/
SameFunc(set_echo_position)
SameFunc(set_echo_surface)
SameFunc(set_fill_index)
SameFunc(set_font)
SameFunc(set_front_plane_clipping)
SameFunc(set_highlighting)
SameFunc(set_image_transformation_2)
SameFunc(set_image_transformation_3)
SameFunc(set_image_transformation_type)
SameFunc(set_image_translate_2)
SameFunc(set_image_translate_3)
SameFunc(set_keyboard)
SameFunc(set_light_direction)
SameFunc(set_line_index)
SameFunc(set_linestyle)
SameFunc(set_linewidth)
SameFunc(set_locator_2)
SameFunc(set_marker_symbol)
SameFunc(set_ndc_space_2)
SameFunc(set_ndc_space_3)
SameFunc(set_output_clipping)
SameFunc(set_pen)
SameFunc(set_pick_id)
```



```
SameFunc(set_polygon_edge_style)
SameFunc(set_polygon_interior_style)
/*
SameFunc(set_primitive_attributes)
*/
SameFunc(set_projection)
SameFunc(set_rasterop)
SameFunc(set_segment_detectability)
SameFunc(set_segment_highlighting)
SameFunc(set_segment_image_transformation_2)
SameFunc(set_segment_image_transformation_3)
SameFunc(set_segment_image_translate_2)
SameFunc(set_segment_image_translate_3)
SameFunc(set_segment_visibility)
SameFunc(set_shading_parameters)
SameFunc(set_stroke)
SameFunc(set_text_index)
SameFunc(set_valuator)
/*
SameFunc(set_vertex_indices)
SameFunc(set_vertex_normals)
*/
SameFunc(set_view_depth)
SameFunc(set_view_plane_distance)
SameFunc(set_view_plane_normal)
SameFunc(set_view_reference_point)
SameFunc(set_view_up_2)
SameFunc(set_view_up_3)
/*
SameFunc(set_viewing_parameters)
*/
SameFunc(set_viewport_2)
SameFunc(set_viewport_3)
SameFunc(set_visibility)
SameFunc(set_window)
/*
SameFunc(set_world_coordinate_matrix_2)
SameFunc(set_world_coordinate_matrix_3)
SameFunc(set_zbuffer_cut)
SameFunc(size_raster)
*/
SameFunc(terminate_core)
SameFunc(terminate_device)
SameFunc(terminate_view_surface)
SameFunc(text)
```

```
/*****************************************************************
 *                                                               *
 *                      ww_builtin.h                             *
 *                                                               *
 *   This file contains declarations for the functions           *
 *   corresponding to the builtin EDEN version of                *
 *   ww library functions                                        *
 *                                                               *
 *****************************************************************/

#if INCLUDE == 'T'
#define BIND(name,func) name,func,
#else
#define BIND(name,func) extern func();
#endif

BIND(   "popwwstate",   popwwstate      )
BIND(   "pushwwstate",  pushwwstate     )
BIND(   "popfontinfo",  popfontinfo     )
BIND(   "pushfontinfo", pushfontinfo    )
BIND(   "bmbox",        E_bmbox         )
BIND(   "bmclip",       E_bmclip        )
BIND(   "bmcopy",       E_bmcopy        )
BIND(   "bmexchange",   E_bmexchange    )
BIND(   "bmgrey",       E_bmgrey        )
BIND(   "bmmove",       E_bmmove        )
BIND(   "bmscroll",     E_bmscroll      )
BIND(   "bmslide",      E_bmslide       )
BIND(   "bmxbox",       E_bmxbox        )
BIND(   "bmxcopy",      E_bmxcopy       )
BIND(   "boxbuild",     E_boxbuild      )
BIND(   "boxinside",    E_boxinside     )
BIND(   "boxop",        E_boxop         )
BIND(   "boxshift",     E_boxshift      )
BIND(   "boxzoom",      E_boxzoom       )
BIND(   "exwrite",      E_exwrite       )
BIND(   "ftbox",        E_ftbox         )
BIND(   "ftprint",      E_ftprint       )
BIND(   "treecreate",   E_treecreate    )
BIND(   "treecwd",      E_treecwd       )
BIND(   "txcreate",     E_txcreate      )
BIND(   "txrecreate",   E_txrecreate    )
BIND(   "wwfnbox",      E_wwfnbox       )
BIND(   "wwget",        E_wwget         )
BIND(   "wwinit",       E_wwinit        )
BIND(   "wwxget",       E_wwxget        )
BIND(   "xxbar",        E_xxbar         )
BIND(   "xxoutline",    E_xxoutline     )
BIND(   "xxrelease",    E_xxrelease     )
BIND(   "xxrotate",     E_xxrotate      )

#undef BIND
```

```
/*****************************************************************
 *                                                               *
 *                        ww_info.h                              *
 *                                                               *
 *   This file contains the interface of those ww functions      *
 *   having none of their arguments or return values is          *
 *   a structure                                                 *
 *                                                               *
 *   See lib.h for the meanings of SpecialF and SameFunc         *
 *                                                               *
 *****************************************************************/

#if INCLUDE == 'H'
#include "wwinfo.h"
#endif

SpecialF(bmcircle,void,bmcircle)
SpecialF(bmdecode,bitmap*,bmdecode)
SpecialF(bmencode,char*,bmencode)
SpecialF(bmfree,void,bmfree)
SpecialF(bmget,bitmap*,bmget)
SpecialF(bmstack,void,bmstack)
SpecialF(bmxget,bitmap*,bmxget)
SameFunc(copack)
SameFunc(corep)
SameFunc(cosize)
SpecialF(cudecode,cursor*,cudecode)
SpecialF(cuencode,char*,cuencode)
SpecialF(cufree,void,cufree)
SpecialF(custack,void,custack)
SameFunc(exread)
SpecialF(ftcursor,cursor*,ftcursor)
SpecialF(ftfree,void,ftfree)
SpecialF(ftload,fontinfo*,ftload)
SpecialF(ftstack,void,ftstack)
SpecialF(ftxprint,void,ftxprint)
SpecialF(ipset,void,ipset)
SameFunc(ipwait)
SpecialF(ipxset,void,ipxset)
SameFunc(ipxwait)
SpecialF(line,void,line)
SpecialF(lnstack,void,lnstack)
SameFunc(treefollow)
SpecialF(treefree,void,treefree)
SpecialF(txdelete,void,txdelete)
SameFunc(txfollow)
SpecialF(txfree,void,txfree)
SpecialF(txgroup,void,txgroup)
SpecialF(txinsert,void,txinsert)
SameFunc(txmenu)
SameFunc(txselect)
SpecialF(txset,void,txset)
SpecialF(txvisible,txinfo*,txvisible)
SameFunc(unportask)
SpecialF(winstack,void,winstack)
SameFunc(wordpopup)
SameFunc(wwask)
SpecialF(wwexit,void,wwexit)
SpecialF(wwfree,void,wwfree)
SpecialF(wwgetscreen,window*,wwgetscreen)
SpecialF(wwnoise,void,wwnoise)
SpecialF(wwpanic,void,wwpanic)
SpecialF(wwshare,char*,wwshare)
SpecialF(wwstack,void,wwstack)
SpecialF(xxneutral,void,xxneutral)
SameFunc(xxpopup)
```



```
SameFunc(xxpopx)
```

```
/* copyright (c) Mark M Martin. RAL. 1986 */
/*
 * C include file for window information in ww library.
 * 28 June 85. 28 July 85. 24 Oct 85. etc
 * Beware: automatic extraction of #defines and define comments for man page.
 * # define will not be found. * define will be.
 * define Useful constants, machine type, and return codes.
 */
#ifdef ns16000
#define WWFORWC
#define WWFORWCORIG
#endif ns16000
#ifdef ns32000
#define WWFORWC
#define WWFORWCNIX
#endif ns32000
#ifdef perq
#ifdef CSI
#define WWFORCSI
#else CSI
#define WWFORPNX
#ifdef m68k
#define WWFORAGW
#else
#define WWFORPERQ2
#endif m68k
#endif CSI
#endif perq
#ifdef mc68000
#define WWFORSUN
#define WWFORCOLOUR
#endif mc68000
#ifndef TRUE
#define TRUE            1
#define FALSE           0
#endif TRUE
# define NULLPTR(x)     ((x *)0)        /* null (coerced) */
#define WWERRLEN        200     /* length of errmsg buffer */
#define WWNOCLIP        (-1)    /* an uninteresting value to mark noclipbox out */
#define WWERROR         (-2)    /* failure */
#define WWOK            0       /* ok */
/*
 * structure that holds the corners of a box
 */
typedef struct{
        int     b_left;
        int     b_top;
        int     b_right;
        int     b_bottom;
}box;
/*
 * info about a bitmap. sadly cannot forward reference the window typdef
 */
typedef struct{
        box             bm_box;         /* initially box enclosing whole bitmap */
        int             bm_colours;     /* max num colours in bitmap (eg 2!) */
        struct _window  *bm_window;     /* point to window it is for, or null */
        int             *bm_junk;       /* more irrelevant info */
}bitmap;
/*
 * a cursor with hot spot and offset from usual cursor image
 */
typedef struct{
        bitmap  *c_bm;
        int     c_rop;
        int     c_xhot,c_yhot;
```



```
        int     c_xoffset,c_yoffset;
}cursor;
/*
 * info about a window
 */
typedef struct _window{
        bitmap  *w_bm;          /* the bitmap that is the window */
        int     w_xrel,w_yrel;  /* relative posn of window to parent */
        box     w_fnbox;        /* posn of fnbox if to be used */
        int     w_flags;
        bitmap  *w_icon;        /* -> the icon or null */
        int     *w_junk;        /* more irrelevant info */
}window;
/*
 * define w_flags bit settings
 */
#define WICONSEEN       01      /* the icon is visible */
#define WSEEN           02      /* window can be seen */
/*
 * define bmbox arguments that can be or-ed together
 */
#define BMCLEARALL      01      /* clear box, including edges */
#define BMCLEAR         02      /* clear inside of box, not including edges */
#define BMNOTALL        04      /* not all of box */
#define BMNOT           010     /* not inside of box */
#define BMEDGES         020     /* draw edges of box, using dd->d_line */
#define BMWIN           04000   /* use ddwin->w_bm instead of ddbm */
/*
 * define bmcopy arguments that can be or-ed together.
 */
#define BMGET           0100    /* get the returned bitmap (implies BMTO) */
#define BMFREE          0200    /* free the given bitmap (implies BMFROM) */
#define BMTO            01000   /* copy to the given bitmap */
#define BMFROM          02000   /* copy from the given bitmap */
/* and  BMWIN           04000      use ddwin->w_bm instead of ddbm */
/*
 * define bmgrey arguments are 16 bit values with this or-ed in
 */
#define WWREGISTER      0x80000000      /* top bit set for tiling to register */
# define SHADE1 41120   /* 0120240. 4 in 16 */
# define SHADE2 42405   /* 0122645. every other pixel, 8 in 16 */
# define SHADE3 33410   /* 0101202. 4 in 16 */
# define SHADE4 1025    /* 02001. 2 in 16 */
# define SHADE5 260     /* 0404. 2 in 16 */
# define HATCHR 33825   /* 0102041. right diagonal lines, 4 in 16 */
# define HATCHL 4680    /* 011110. left diagonal lines, 4 in 16 */
/*
 * define boxop arguments can be one of these
 */
#define BOXINTERSECT    1       /* return common box between two boxes. if none retur
n 0,0,0,0 */
#define BOXCENTRE       2       /* return first box when centred in 2nd box, with red
uced size if 2nd box smaller */
#define BOXENCLOSING    3       /* smallest box enclosing both boxes */
#define BOXREDUCE       4       /* reduce a to the size of b, unless is it smaller al
ready */
#define BOXADD          5       /* offset first box by 2nd box top/left */
#define BOXBOUNCE       6       /* bounce first box off walls of 2nd, and reduce if n
ecessary */
#define BOXBRLAP        7       /* 2nd box moved to overlap 1st as much as possible,
        (to bottom right) but leaving a border so that either can be on top & picked
*/
#define BOXBLLAP        8       /* overlap to bottom left */
#define BOXTLLAP        9       /* overlap to top left */
#define BOXTRLAP        10      /* overlap to top right */
/*
```

```
 * define corep copack arguments can be one of these
 */
#define COGET   1       /* get colour map entries */
#define COSET   2       /* set them */
/*
 * info about a font
 */
typedef struct{
        char            *f_name;                /* mallocd filename */
        int             f_height,f_width;       /* of biggest char */
        cursor          *f_cursor;              /* -> suitable text cursor, if any */
        int             *f_junk;
}fontinfo;
/*
 * define ftprint arguments that can be or-ed together
 */
#define FTNOT           02      /* not box after putting in text */
#define FTOVER          04      /* dont clear box before use */
#define FTWIN           010     /* use ddwin->w_bm instead of ddbm */
#define FTVERT          020     /* centre text vertically in height */
#define FTHORIZ         040     /* centre text horizontally in width */
#define FTCENTRE        (FTVERT|FTHORIZ)        /* centre text */
#define FTRIGHT         0100    /* right justify */
#define FTBOTTOM        0200    /* bottom justify */
#define FTPROP          0400    /* use proportional spacing if available */
#define FTCONTROL       01000   /* show control chars as inverse video */
/*
 * all special info about editing chars to use
 */
typedef struct{
        char    mc_erase;
        char    mc_kill;
        char    mc_delword;
        char    mc_retype;
}magic;
/*
 * global info about the state of ww
 */
typedef struct{
        int     d_line;         /* line style in use */
        int     d_rop;          /* raster op in use */
        int     d_buttons;      /* which mouse buttons down */
        int     d_x,d_y;        /* x and y position of mouse in pixels */
        magic   d_magic;        /* chars to use in standard editing */
        char    d_errmsg[WWERRLEN];     /* last error message */
        int     d_flags;        /* more about this state */

        int     d_event;        /* last input event giving this state */
        char    d_char;         /* char typed in */
        window  *d_newwindow;   /* window selected by mouse */
        int     d_select;       /* fds you can use in select before ipwait */
        int     d_fore;         /* foreground colour (for lines eg) */
        int     d_back;         /* background colour (eg for BMCLEAR) */
        int     d_colours;      /* default num colours in bitmaps and windows */
        int     (*d_ipwait)();  /* point to input wait filter */
}wwstate;
/*
 * define dd->d_buttons settings that can be or-ed together
 */
#define ITEMBUTTON      01
#define MENUBUTTON      02
#define SHOWBUTTON      04
/*
 * define dd->d_event can be one of these
 */
#define IPOTHER         0       /* mouse position or button change */
```



```
#define IPKEY           1       /* keyboard input in d_char */
#define IPSIZE          3       /* window size has changed */
#define IPLEAVE         4       /* window not selected. guarantee buttons=0 */
#define IPENTER         5       /* window selected. sets d_newwindow */
#define IPSEEN          6       /* window visible. icon may not be */
#define IPNOTSEEN       7       /* window no longer visible. icon may be */
/*
 * define dd->d_flags bit settings
 */
#define PRINTERR        01      /* d_flags: print error messages */
/*
 * define dd->d_rop and raster operations values. WWNOT can be or-ed with one.
 * one of, but WWNOT can be ored in with any of the others.
 * they have slightly limited meaning for line drawing.
 */
#define WWCOPY          0       /* d <- s */
#define WWXOR           01      /* d <- d ^ s */
#define WWOR            02      /* d <- d | s */
#define WWAND           03      /* d <- d & s */
#define WWNOT           04      /* use ~s instead of s */
/*
 * define dd->d_fore and d_back colours can be (using ww bit-op colour map)
 */
#define COBLACK         0
#define CORED           01
#define COGREEN         02
#define COBLUE          04
#define COYELLOW        03
#define COCYAN          06
#define COMAGENTA       05
#define COWHITE         07
/*
 * structure for each terminal emulator pane on screen
 * giving current cursor posn, etc
 */
typedef struct{
        int     p_flags;
        int     p_x,p_y;        /* cursor posn in chars */
        int     p_xmax,p_ymax;  /* size of page in chars(-1) */
        int     p_height,p_width;       /* font stuff at time of creation */
        fontinfo *p_font;
        box     p_box;          /* area in which we operate, inclusive */
        box     p_select;       /* highlighted selection in char/line units */
        char    *p_text;        /* dynamic array char[xmax+1,ymax+1] with screen text
 */
        int     *p_length;      /* dynamic array int[ymax+1] with length of each scre
en line */
        int     *p_junk;
}emupane;
/*
 * define emupane p_flags bit settings
 */
#define PCRMOD          01      /* cr means cr/lf */
#define PBUTTONS        02      /* return on button up */
# define P_TEXT(pp,x,y) pp->p_text[x+y*(pp->p_xmax+1)]  /* access to char at x,y */
/*
 * external representation of ww objects
 */
typedef union{
        bitmap *ex_bitmap;
        cursor *ex_cursor;
        int     ex_grey;
}exrep;
/*
 * define exrep external representations
 */
```

```
#define EXBITMAP        0
#define EXCURSOR        1
#define EXGREY          2
/*
 * define ipset arguments can be one of these
 */
#define IPON            1
#define IPOFF           2
#define IPSAMPLE        3
#define IPREQUEST       4
#define IPNOKEY         5
/*
 * define line routine arguments can be one of these
 */
#define LNDRAWABS 00
#define LNMOVEABS 01
#define LNDRAWREL 02
#define LNMOVEREL 03
/*
 * info about a text-entry-and-editing area
 */
typedef struct _txinfo{
        box             tx_box;         /* box where text is entered */
        char            *tx_start;      /* ->text string, null terminated */
        char            *tx_next;       /* ->terminating null */
        char            *tx_term;       /* 0 or ->chars to cause txfollow to return *
/
        int             tx_length;      /* length of text, not inc extra null at end
*/
        int             tx_left;        /* current user selected char(s): left */
        int             tx_right;       /* and right (0=>left of first char of text)
*/
        int             tx_flags;
        int             tx_xmax;        /* max num columns in display */
        int             tx_ymax;        /* max num lines in display */
        int             tx_offset;      /* number of chars to start of display */
        struct _txinfo  *tx_group;      /* 0 or next in circle list of single selecti
ons */
        struct _txinfo  *(*tx_check)(); /* 0 or routine called when something about t
o happen */
        int             *tx_junk;       /* more irrelevant info */
}txinfo;
/*
 * define tx_flags bit settings
 */
#define TXVISIBLE       01              /* selection is visible */
#define TXALLON         02              /* text can all be seen without scrolling */
#define TXVERT          04              /* centre the text vertically */
#define TXHORIZ         010             /* centre the text horizontally */
#define TXCENTRE        (TXHORIZ|TXVERT) /* centre the text */
#define TXCRMOD         020             /* cr means lf */
#define TXONHIT         040             /* make selection visible only after button h
it */
/*
 * define txinsert arguments can be one of these, or a character position
 */
#define TXLAST          (-1)    /* at end of text line */
#define TXCURRENT       (-2)    /* before current selection in text line */
/*
 * define txvisible arguments can be one of these
 */
#define TXSET           1       /* set this txinfo visible */
#define TXCLEAR         2       /* set this txinfo not visible */
#define TXCLEARALL      3       /* set all this circular list not visible */
#define TXWHICH         4       /* return which txinfo in list is visible */
/*
```



```
 * define tx_check will be given one of these
 */
#define TXCOPY          1       /* user wants to copy selection. etc */
#define TXMOVE          2       /* (values prev fixed for txmenu ajs) */
#define TXDELETE        3

#define TXSELECT        4
#define TXEXTEND        5
#define TXKEY           6       /* input when selection not here */
/*
 * info about an extension of the txinfo area that is used to hold a filename and whe
re
 * you can wander around the filesystem
 */
typedef struct _treeinfo{
        box             tr_box;         /* the whole area to display in */
        box             tr_hitbox;      /* the sensitive area for popups */
        txinfo          *tr_tx;         /* the absolute or relative pathname in a txi
nfo area */
        int             tr_flags;
        struct _treeinfo *tr_base;      /* the current working dir for this filename
*/
}treeinfo;
/*
 * define treeinfo tr_flags bit settings
 */
#define TRABSOLUTE      01      /* use absolute pathnames */
#define TRCWDOK         02      /* this is the correct current dir */
/*
 * define stack routine arguments can be one of these
 */
#define WWPUSH          1
#define WWPUSHOFF       2
#define WWPOP           3
#define WWSET           4
#define WWPUSHBUSY      5       /* custack */
#define WWPUSHON        6
/*
 * define bmdecode arguments can be one of these
 */
#define ENLPSTYLE       1       /* lineprinter image of a raster */
#define ENWWSTYLE       2       /* octal bit image in serial bytes. length&width in 1
st 4 bytes */
#define ENRUNSTYLE      3       /* runlength encoding. length&width&bytelength in 1st
 6 bytes */
/*
 * define wwask arguments can be one of these
 */
#define ASKXPPI         1
#define ASKYPPI         2
#define ASKXSCREEN      3
#define ASKYSCREEN      4
/*
 * define wwxget arguments can be these ored together
 */
#define WWSIZEFROMUSER  01      /* get better size for window from user */
#define WWOPENICONISED  02      /* start window iconised */
/*
 * define xxpopx arguments can be one of these
 */
#define POPREGULAR      0       /* conventional popup linear menu */
#define POPNOCURSOR     01      /* similar, but no cursor, and x and y movements note
d */
#define POPROTATE       02      /* similar, but text moves! awful */
#define POPSHIFT        03      /* similar, but whole menu moves! awful */
#define POPRIGHT        04      /* ored in: menu is to right of cursor */
```

```
/*
 * define xxbar arguments can be these ored together
 */
#define BARVERTICAL     01      /* use a vertical not horizontal bar */
#define BARSELECTION    02      /* use a |___| type presentation */
/*
 * define unportask arguments can be one of these
 */
#define ASKBMMEMORY     01      /* (bitmap *) -> (char *) perq */
                                /* (bitmap *) -> (struct pixrect *) sun */
#define ASKWINCANVAS    02      /* (window *) -> (Canvas) sun */
#define ASKWINFRAME     04      /* (window *) -> (Frame) sun */
#define ASKWINFILE      010     /* (window *) -> file descriptor (int) perq */
/*
 * default pointers set by wwinit.
 */
extern wwstate  *dd;
extern window   *ddwin;
extern bitmap   *ddbm;
extern fontinfo *ddfont;
extern box       noclipbox;
extern treeinfo *treebase;
#ifdef WWFORPERQ2
# define void   int
#endif WWFORPERQ2
/*
 * External Typing: rest of file generated automatically. BEWARE
 */
#ifndef c_plusplus
void     bmbox();
void     bmcircle();
box      bmclip();
bitmap  *bmcopy();
bitmap  *bmdecode();
char    *bmencode();
void     bmexchange();
void     bmfree();
bitmap  *bmget();
void     bmgrey();
void     bmmove();
void     bmscroll();
void     bmslide();
void     bmstack();
void     bmxbox();
void     bmxcopy();
bitmap  *bmxget();
box      boxbuild();
int      boxinside();
box      boxop();
box      boxshift();
box      boxzoom();
int      copack();
int      corep();
int      cosize();
cursor  *cudecode();
char    *cuencode();
void     cufree();
void     custack();
box      emucharbox();
emupane *emucreate();
void     emufollow();
void     emufree();
char    *emuget();
void     emuprint();
void     emurecreate();
void     emushow();
```



```
char    *emutermcap();
int      exread();
int      exwrite();
box      ftbox();
cursor  *ftcursor();
void     ftfree();
fontinfo        *ftload();
void     ftprint();
void     ftstack();
void     ftxprint();
void     ipset();
int      ipwait();
void     ipxset();
int      ipxwait();
void     line();
void     lnstack();
treeinfo        *treecreate();
treeinfo        *treecwd();
int      treefollow();
void     treefree();
txinfo  *txcreate();
void     txdelete();
int      txfollow();
void     txfree();
void     txgroup();
void     txinsert();
int      txmenu();
void     txrecreate();
void     txscroll();
int      txselect();
void     txset();
txinfo  *txvisible();
int      unportask();
void     winstack();
int      wordpopup();
int      wwask();
void     wwexit();
void     wwfnbox();
void     wwfree();
window  *wwget();
window  *wwgetscreen();
int      wwinit();
void     wwnoise();
void     wwpanic();
char    *wwshare();
void     wwstack();
window  *wwxget();
int      xxbar();
void     xxneutral();
void     xxoutline();
int      xxpopup();
int      xxpopx();
int      xxrelease();
void     xxrotate();
#else c_plusplus

// void bmbox(b,flags)box b;int flags;
   void bmbox(box,int);
// void bmcircle(bm,x,y,radius,flags)bitmap *bm;int x,y,radius,flags;
   void bmcircle(bitmap*,int,int,int,int);
// box bmclip(bm,b)bitmap *bm;box b;
   box bmclip(bitmap*,box);
// bitmap *bmcopy(bm,b,flags)bitmap *bm;box b;int flags;
   bitmap *bmcopy(bitmap*,box,int);
// bitmap *bmdecode(pattern,style)char *pattern;int style;
   bitmap *bmdecode(char*,int);
```

```
// char *bmencode(bm,style,length)bitmap *bm;int style;int *length;
   char *bmencode(bitmap*,int,int*);
// void bmexchange(from,frombox,to,tobox)bitmap *from,*to;box frombox,tobox;
   void bmexchange(bitmap*,box,bitmap*,box);
// void bmfree(bm)bitmap *bm;
   void bmfree(bitmap*);
// bitmap *bmget(width,height)int width,height;
   bitmap *bmget(int,int);
// void bmgrey(b,sh)box b;int sh;
   void bmgrey(box,int);
// void bmmove(b,x,y)box b;int x,y;
   void bmmove(box,int,int);
// void bmscroll(b,xdist,ydist)box b;int xdist,ydist;
   void bmscroll(box,int,int);
// void bmslide(b,x,y)box b;int x,y;
   void bmslide(box,int,int);
// void bmstack(flag)int flag;
   void bmstack(int);
// void bmxbox(bm,b,flags)bitmap *bm;box b;int flags;
   void bmxbox(bitmap*,box,int);
// void bmxcopy(from,frombox,to,tobox,rop)bitmap *from,*to;box frombox,tobox;int rop;
   void bmxcopy(bitmap*,box,bitmap*,box,int);
// bitmap *bmxget(width,height,colours)int width,height,colours;
   bitmap *bmxget(int,int,int);
// box boxbuild(left,top,right,bottom)int left,top,right,bottom;
   box boxbuild(int,int,int,int);
// int boxinside(b,x,y)box b;int x,y;
   int boxinside(box,int,int);
// box boxop(a,b,flag)box a,b;int flag;
   box boxop(box,box,int);
// box boxshift(b,x,y)box b;int x,y;
   box boxshift(box,int,int);
// box boxzoom(b,dist)box b;int dist;
   box boxzoom(box,int);
// int copack(rep,colour,flags)int rep,colour,flags;
   int copack(int,int,int);
// int corep(bm,r,g,b,step,start,count,flags)bitmap *bm;char *r,*g,*b;int step,start,
count,flags;
   int corep(bitmap*,char*,char*,char*,int,int,int,int);
// int cosize(bm,colours)bitmap *bm;int colours;
   int cosize(bitmap*,int);
// cursor *cudecode(pattern,style)char *pattern;int style;
   cursor *cudecode(char*,int);
// char *cuencode(cp,style,length)cursor *cp;int style,*length;
   char *cuencode(cursor*,int,int*);
// void cufree(cp)cursor *cp;
   void cufree(cursor*);
// void custack(cp,flag)cursor *cp;int flag;
   void custack(cursor*,int);
// box emucharbox(ep,fromx,tox,y)emupane *ep;int fromx,tox,y;
   box emucharbox(emupane*,int,int,int);
// emupane *emucreate(b)box b;
   emupane *emucreate(box);
// void emufollow(pp)emupane *pp;
   void emufollow(emupane*);
// void emufree(ep)emupane *ep;
   void emufree(emupane*);
// char *emuget(pp,b)emupane *pp;box b;
   char *emuget(emupane*,box);
// void emuprint(ep,cp,len)emupane *ep;char *cp;int len;
   void emuprint(emupane*,char*,int);
// void emurecreate(ep,b)emupane *ep;box b;
   void emurecreate(emupane*,box);
// void emushow(pp,b)emupane *pp;box b;
   void emushow(emupane*,box);
// char *emutermcap(ep)emupane *ep;
```



```
   char *emutermcap(emupane*);
// int exread(file,style,ex)int file,*style;exrep *ex;
   int exread(int,int*,exrep*);
// int exwrite(file,what,style,ex)int file,what,style;exrep ex;
   int exwrite(int,int,int,exrep);
// box ftbox(fp,str,len,flags)fontinfo *fp;char *str;int len;
   box ftbox(fontinfo*,char*,int,int);
// cursor *ftcursor(width,height)
   cursor *ftcursor(int,int);
// void ftfree(fp)fontinfo *fp;
   void ftfree(fontinfo*);
// fontinfo *ftload(file)char *file;
   fontinfo *ftload(char*);
// void ftprint(b,str,flags)box b;char *str;int flags;
   void ftprint(box,char*,int);
// void ftstack(flag)int flag;
   void ftstack(int);
// void ftxprint(x,y,str,len,rop,font,bm,flags)int x,y,len,rop,flags;char *str;fontin
fo *font;bitmap *bm;
   void ftxprint(int,int,char*,int,int,fontinfo*,bitmap*,int);
// void ipset(flag)int flag;
   void ipset(int);
// int ipwait()
   int ipwait();
// void ipxset(win,flag)window *win;int flag;
   void ipxset(window*,int);
// int ipxwait()
   int ipxwait();
// void line(x,y,flag)int x,y,flag;
   void line(int,int,int);
// void lnstack(flag)int flag;
   void lnstack(int);
// treeinfo *treecreate(tr,area)treeinfo *tr;box area;
   treeinfo *treecreate(treeinfo*,box);
// treeinfo *treecwd(base,area)char *base;box area;
   treeinfo *treecwd(char*,box);
// int treefollow(tr)treeinfo *tr;
   int treefollow(treeinfo*);
// void treefree(tr)treeinfo *tr;
   void treefree(treeinfo*);
// txinfo *txcreate(b)box b;
   txinfo *txcreate(box);
// void txdelete(tx,from,count)txinfo *tx;int from,count;
   void txdelete(txinfo*,int,int);
// int txfollow(tx)txinfo *tx;
   int txfollow(txinfo*);
// void txfree(tx)txinfo *tx;
   void txfree(txinfo*);
// void txgroup(list,tx)txinfo *list,*tx;
   void txgroup(txinfo*,txinfo*);
// void txinsert(tx,where,str,len)txinfo *tx;char *str;int where,len;
   void txinsert(txinfo*,int,char*,int);
// int txmenu(tx,txfrom)txinfo *tx,*txfrom;
   int txmenu(txinfo*,txinfo*);
// void txrecreate(tx,b)txinfo *tx;box b;
   void txrecreate(txinfo*,box);
// void txscroll(tx,by)txinfo *tx;int by;
   void txscroll(txinfo*,int);
// int txselect(tx,extend)txinfo *tx;int extend;
   int txselect(txinfo*,int);
// void txset(tx,left,right)txinfo *tx;int left,right;
   void txset(txinfo*,int,int);
// txinfo *txvisible(tx,what)txinfo *tx;int what;
   txinfo *txvisible(txinfo*,int);
// int unportask(pointer,what)void *pointer;int what;
   int unportask(void*,int);
```

```
// void winstack(flag)int flag;
   void winstack(int);
// int wordpopup(words,lens,startdepth,index,retndepth)char ***words;int **lens;int s
tartdepth,index;int *retndepth;
   int wordpopup(char***,int**,int,int,int*);
// int wwask(what)int what;
   int wwask(int);
// void wwexit()
   void wwexit();
// void wwfnbox(posn,str)box posn;char **str;
   void wwfnbox(box,char**);
// void wwfree(win)window *win;
   void wwfree(window*);
// window *wwget(size)box size;
   window *wwget(box);
// window *wwgetscreen()
   window *wwgetscreen();
// int wwinit()
   int wwinit();
// void wwnoise()
   void wwnoise();
// void wwpanic(s)char *s;
   void wwpanic(char*);
// char *wwshare(what,length,filename,returnlen)char *what,*filename;int length,*retu
rnlen;
   char *wwshare(char*,int,char*,int*);
// void wwstack(flag)int flag;
   void wwstack(int);
// window *wwxget(size,colours,label,flags)box size;int colours,flags;char *label;
   window *wwxget(box,int,char*,int);
// int xxbar(b,min,max,from,to,flags)box b;int min,max,from,to,flags;
   int xxbar(box,int,int,int,int,int);
// void xxneutral()
   void xxneutral();
// void xxoutline(bm,b)bitmap *bm;box b;
   void xxoutline(bitmap*,box);
// int xxpopup(list,start)char **list;int start;
   int xxpopup(char**,int);
// int xxpopx(list,start,style,printflags)char **list;int start,style,printflags;
   int xxpopx(char**,int,int,int);
// int xxrelease(b)box b;
   int xxrelease(box);
// void xxrotate(b,size,rot)box b;int size,rot;
   void xxrotate(box,int,int);
#endif c_plusplus
```

accept_text 6 (lex.c 1)
action_list 19 (eval.c 4)
add 26 (machine.c 2)
addentry 16 (code.c 3)
addID 36 (refer.c 1)
addr 32 (machine.c 14)
address 32 (machine.c 14)
address_error 26 (machine.c 1)
address_type 41 (eden.h 2)
add_local_variable 37 (symbol.c 2)
ALLOC_ATOM 44 (global.q.h 1)
append 34 (machine.c 17)
APPEND_ATOM 44 (global.q.h 1)
append_char 6 (lex.c 1)
APPEND_entry_Q 51 (entry.q.h 1)
APPEND_Q 44 (global.q.h 1)
APPEND_symptr_Q 45 (symptr.q.h 1)
apply 11 (builtin.c 5)
app_char 6 (lex.c 1)
ARGNUM 58 (ww_builtin.c 1)
array 12 (builtin.c 8)
assign 30 (machine.c 9)
B 58 (ww_builtin.c 1)
backgnd 13 (builtin.c 9)
backslash 7 (lex.c 4)
BIND 43 (builtin.h 1)
BIND 70 (ww_builtin.h 1)
BOX 58 (ww_builtin.c 1)
buffer_overflow 6 (lex.c 1)
b_exit 10 (builtin.c 4)
b_write 9 (builtin.c 2)
call 15 (code.c 2)
call_float 22 (lib.c 1)
call_lib 22 (lib.c 1)
change 19 (eval.c 3)
ChangeSym 26 (machine.c 1)
change_sources 17 (code.c 5)
change_targets 17 (code.c 5)
checkok 36 (refer.c 2)
CheckRange 26 (machine.c 1)
CLEAN_Q 44 (global.q.h 1)
clearok 53 (curses.c 1)
CLEAR_entry_Q 51 (entry.q.h 1)
clear_IDlist 36 (refer.c 1)
CLEAR_symptr_Q 45 (symptr.q.h 1)
clear_text 6 (lex.c 1)
cnv_2_bool 29 (machine.c 7)
cnv_formula_to_rwv 30 (machine.c 9)
code 16 (code.c 4)
code2 23 (main.c 1)
code3 23 (main.c 1)
codeswitch 17 (code.c 6)
code_definition 16 (code.c 4)
code_related_by 17 (code.c 5)
concat 28 (machine.c 5)
CONCAT_entry_Q 51 (entry.q.h 1)
CONCAT_symptr_Q 45 (symptr.q.h 1)
constpush 28 (machine.c 6)
cptr 41 (eden.h 2)
crmode 53 (curses.c 2)
ctos 39 (type.c 1)
datacmp 28 (machine.c 6)
ddup 28 (machine.c 6)
debug 9 (builtin.c 1)
DEBUGPRINT 15 (code.c 1)
DEBUGPRINT 23 (main.c 1)

DEBUGPRINT 26 (machine.c 1)
dec_asgn 30 (machine.c 10)
definition 29 (machine.c 8)
defnonly 23 (main.c 2)
delete 34 (machine.c 18)
DELETE_ATOM 44 (global.q.h 1)
DELETE_entry_ATOM 50 (entry.q.c 1)
DELETE_FIRST_entry 51 (entry.q.h 1)
DELETE_FIRST_symptr 45 (symptr.q.h 1)
DELETE_FRONT_entry 51 (entry.q.h 1)
DELETE_FRONT_symptr 45 (symptr.q.h 1)
DELETE_LAST_entry 51 (entry.q.h 1)
DELETE_LAST_symptr 45 (symptr.q.h 1)
delete_local_level 38 (symbol.c 3)
DELETE_NTH_entry 51 (entry.q.h 1)
DELETE_NTH_symptr 45 (symptr.q.h 1)
DELETE_symptr_ATOM 49 (symptr.q.c 1)
DESTROY_entry_ATOM 51 (entry.q.h 1)
DESTROY_symptr_ATOM 45 (symptr.q.h 1)
dimension_error 26 (machine.c 1)
dispatch 38 (symbol.c 4)
div 27 (machine.c 3)
div_by_zero_error 26 (machine.c 1)
dptr 41 (eden.h 2)
dpush 26 (machine.c 1)
dpush 43 (builtin.h 1)
E 58 (ww_builtin.c 1)
eager 19 (eval.c 3)
echo 53 (curses.c 2)
eden_interrupt_handler 24 (main.c 3)
emalloc 37 (symbol.c 1)
EMPTYQUEUE 44 (global.q.h 1)
end_text 6 (lex.c 1)
eq 29 (machine.c 8)
EQUAL_entry 37 (symbol.c 2)
EQUAL_entry 51 (entry.q.h 1)
EQUAL_symptr 45 (symptr.q.h 1)
erealloc 32 (machine.c 13)
error 41 (eden.h 1)
error2 23 (main.c 2)
error_no 13 (builtin.c 9)
eval 15 (code.c 2)
eval_formula_queue 18 (eval.c 2)
execute 16 (code.c 3)
exec_file 11 (builtin.c 5)
exec_string 10 (builtin.c 4)
E_bmbox 59 (ww_builtin.c 4)
E_bmclip 60 (ww_builtin.c 5)
E_bmcopy 60 (ww_builtin.c 5)
E_bmexchange 60 (ww_builtin.c 5)
E_bmgrey 60 (ww_builtin.c 5)
E_bmmove 60 (ww_builtin.c 5)
E_bmscroll 60 (ww_builtin.c 5)
E_bmslide 60 (ww_builtin.c 5)
E_bmxbox 60 (ww_builtin.c 6)
E_bmxcopy 60 (ww_builtin.c 6)
E_boxbuild 60 (ww_builtin.c 6)
E_boxinside 60 (ww_builtin.c 6)
E_boxop 60 (ww_builtin.c 6)
E_boxshift 60 (ww_builtin.c 6)
E_boxzoom 60 (ww_builtin.c 6)
E_exwrite 61 (ww_builtin.c 7)
E_ftbox 61 (ww_builtin.c 7)
E_ftprint 61 (ww_builtin.c 7)
E_treecreate 61 (ww_builtin.c 7)
E_treecwd 61 (ww_builtin.c 7)

E_txcreate 61 (ww_builtin.c 7)
E_txrecreate 61 (ww_builtin.c 7)
E_wwfnbox 61 (ww_builtin.c 7)
E_wwget 61 (ww_builtin.c 8)
E_wwinit 61 (ww_builtin.c 8)
E_wwxget 61 (ww_builtin.c 8)
E_xxbar 62 (ww_builtin.c 9)
E_xxoutline 62 (ww_builtin.c 9)
E_xxrelease 62 (ww_builtin.c 9)
E_xxrotate 62 (ww_builtin.c 9)
flushok 53 (curses.c 1)
FOREACH 44 (global.q.h 1)
forget 11 (builtin.c 6)
formula_list 19 (eval.c 3)
freedatum 32 (machine.c 13)
freeheap 20 (heap.c 1)
FREE_entry 37 (symbol.c 2)
FREE_entry 51 (entry.q.h 1)
FREE_symptr 45 (symptr.q.h 1)
FRONT 44 (global.q.h 1)
Function 66 (customlib.h 1)
Function 68 (sungraf.h 1)
ge 29 (machine.c 7)
get2num 26 (machine.c 2)
getheap 20 (heap.c 1)
GETINT 58 (ww_builtin.c 1)
GETSTR 58 (ww_builtin.c 1)
getvalue 33 (machine.c 16)
getx 53 (curses.c 1)
gety 53 (curses.c 1)
get_address 26 (machine.c 1)
get_environ 13 (builtin.c 9)
get_msgq 14 (builtin.c 11)
gt 29 (machine.c 7)
hashindex 37 (symbol.c 1)
heap_overflow 20 (heap.c 1)
I 58 (ww_builtin.c 1)
id_token 6 (lex.c 2)
inc_asgn 30 (machine.c 10)
indexcalc 33 (machine.c 15)
init 24 (main.c 3)
initcode 15 (code.c 1)
init_lex 6 (lex.c 1)
init_LocalVarList 38 (symbol.c 3)
input 6 (lex.c 1)
insert 34 (machine.c 18)
INSERT_ATOM 44 (global.q.h 1)
INSERT_entry_Q 51 (entry.q.h 1)
insert_level_marker 38 (symbol.c 3)
INSERT_Q 44 (global.q.h 1)
install 37 (symbol.c 1)
install_custom_variables 55 (custom.c 1)
INSTALL_VAR 55 (custom.c 1)
invoke_action_queue 18 (eval.c 2)
IN_entry_Q 51 (entry.q.h 1)
IN_symptr_Q 45 (symptr.q.h 1)
ischar 41 (eden.h 2)
isint 41 (eden.h 2)
isnum 41 (eden.h 2)
isstr 41 (eden.h 2)
isundef 41 (eden.h 2)
is_address 41 (eden.h 2)
is_list 28 (machine.c 5)
is_local 41 (eden.h 2)
is_string 28 (machine.c 5)
is_symbol 41 (eden.h 2)

jmp 28 (machine.c 5)
jpnz 28 (machine.c 6)
jpz 28 (machine.c 5)
keyin 23 (main.c 2)
keyword_token 6 (lex.c 1)
LAST 44 (global.q.h 1)
lcat 10 (builtin.c 4)
le 29 (machine.c 8)
leaveok 53 (curses.c 1)
listsize 34 (machine.c 17)
local 41 (eden.h 2)
localaddr 33 (machine.c 15)
local_declare 38 (symbol.c 3)
lookup 37 (symbol.c 1)
lookup_address 33 (machine.c 15)
lookup_local 38 (symbol.c 3)
lt 29 (machine.c 7)
MAGIC 58 (ww_builtin.c 2)
main 24 (main.c 4)
makearr 33 (machine.c 16)
makedatum 23 (main.c 1)
makelist 33 (machine.c 16)
mark_changed 19 (eval.c 3)
mod 27 (machine.c 4)
moreinput 24 (main.c 4)
MOVE_entry_Q 50 (entry.q.c 1)
MOVE_symptr_Q 49 (symptr.q.c 1)
mul 27 (machine.c 3)
multi_symbol_token 7 (lex.c 3)
mustaddr 26 (machine.c 2)
mustchar 26 (machine.c 1)
mustint 26 (machine.c 1)
mustlist 26 (machine.c 2)
muststr 26 (machine.c 1)
nameof 9 (builtin.c 2)
ne 29 (machine.c 8)
need_rwv 30 (machine.c 9)
negate 27 (machine.c 4)
newdatum 32 (machine.c 13)
newhdat 32 (machine.c 14)
NEW_entry 51 (entry.q.h 1)
NEW_symptr 45 (symptr.q.h 1)
NEXT 44 (global.q.h 1)
nl 53 (curses.c 2)
nocrmode 53 (curses.c 2)
noecho 53 (curses.c 2)
nonl 53 (curses.c 2)
noraw 53 (curses.c 2)
not 27 (machine.c 4)
number_token 6 (lex.c 2)
num_required_error 26 (machine.c 1)
out_of_range_error 26 (machine.c 1)
P 22 (lib.c 1)
P 58 (ww_builtin.c 1)
pack 12 (builtin.c 8)
packpara 12 (builtin.c 7)
para 43 (builtin.h 1)
patch 38 (symbol.c 4)
pipe_process 13 (builtin.c 10)
pop 41 (eden.h 2)
popd 28 (machine.c 6)
POPFONTINFO 59 (ww_builtin.c 3)
popfontinfo 59 (ww_builtin.c 4)
POPWWSTATE 58 (ww_builtin.c 2)
popwwstate 59 (ww_builtin.c 4)
pop_char 6 (lex.c 1)

post_dec 31 (machine.c 12)
post_inc 31 (machine.c 11)
PREV 44 (global.q.h 1)
pre_dec 31 (machine.c 12)
pre_inc 31 (machine.c 11)
print 9 (builtin.c 2)
printhash 11 (builtin.c 6)
printlist 36 (refer.c 1)
printlocal 38 (symbol.c 3)
print_prompt 24 (main.c 3)
push 41 (eden.h 2)
pushfontinfo 59 (ww_builtin.c 4)
pushint 28 (machine.c 6)
PushInteger 22 (lib.c 1)
pushlist 12 (builtin.c 7)
PushReal 22 (lib.c 1)
pushUNDEF 28 (machine.c 6)
PUSHWWSTATE 59 (ww_builtin.c 3)
pushwwstate 59 (ww_builtin.c 4)
push_text 23 (main.c 1)
put_environ 13 (builtin.c 9)
query 35 (machine.c 19)
Q_EMPTY 44 (global.q.h 1)
Q_LEN 21 (global.q.c 1)
raw 53 (curses.c 2)
ready 36 (refer.c 2)
RealFunc 66 (customlib.h 1)
receive_message 14 (builtin.c 12)
refer_by 36 (refer.c 2)
refer_to 36 (refer.c 1)
related_by_code 17 (code.c 5)
remove_msgq 14 (builtin.c 12)
resetty 54 (curses.c 3)
reset_compiler_flags 41 (eden.h 2)
reset_compiler_status 15 (code.c 1)
reset_entry_tbl 42 (eden.h 3)
reset_eval 19 (eval.c 4)
reset_frames 42 (eden.h 3)
reset_heap 42 (eden.h 3)
reset_prog_ptr 41 (eden.h 2)
reset_pseudo_machine_status 15 (code.c 1)
reset_stack 41 (eden.h 2)
RETBOX 58 (ww_builtin.c 1)
RETFONTINFO 58 (ww_builtin.c 1)
RETINT 58 (ww_builtin.c 1)
RETURN_TOKEN 6 (lex.c 1)
RETWWSTATE 58 (ww_builtin.c 1)
ret_call 16 (code.c 3)
run 25 (main.c 5)
run_init 25 (main.c 6)
S 58 (ww_builtin.c 1)
safe_input 6 (lex.c 1)
SameFunc 66 (customlib.h 1)
SameFunc 68 (sungraf.h 1)
SameFunc 68 (sungraf.h 2)
SameFunc 71 (ww_info.h 1)
SameReal 66 (customlib.h 1)
SameReal 66 (customlib.h 2)
sametype 26 (machine.c 1)
saveprog 16 (code.c 3)
savetext 16 (code.c 4)
savetty 53 (curses.c 2)
save_IDlist 36 (refer.c 1)
scat 10 (builtin.c 4)
schedule 18 (eval.c 1)
schedule_parents_of 18 (eval.c 1)

scrollok 53 (curses.c 1)
SC_initialize_Basic2D 56 (sungraf.c 1)
SC_new_color_view_surface 56 (sungraf.c 1)
SC_new_view_surface 56 (sungraf.c 1)
SC_restore_raster 56 (sungraf.c 2)
SC_save_raster 56 (sungraf.c 1)
SEARCH_entry 50 (entry.q.c 1)
search_local 37 (symbol.c 2)
SEARCH_symptr 49 (symptr.q.c 1)
sel 33 (machine.c 16)
send_message 14 (builtin.c 11)
shift 34 (machine.c 17)
skip_comment 7 (lex.c 3)
SpecialF 66 (customlib.h 1)
SpecialF 71 (ww_info.h 1)
stack_overflow_err 15 (code.c 1)
stack_underflow_err 15 (code.c 1)
streq 24 (main.c 4)
streq 37 (symbol.c 2)
sub 27 (machine.c 3)
sublist 9 (builtin.c 2)
substr 9 (builtin.c 1)
switchcode 29 (machine.c 8)
symbol_of 41 (eden.h 2)
symtblZlist 11 (builtin.c 5)
touch 13 (builtin.c 9)
typename 39 (type.c 1)
type_clash_addr_err 40 (type.c 4)
type_clash_err 40 (type.c 4)
t_char 40 (type.c 3)
t_float 39 (type.c 2)
t_int 39 (type.c 2)
t_str 40 (type.c 3)
t_super 40 (type.c 4)
t_type 39 (type.c 1)
UNBOX 58 (ww_builtin.c 2)
UNMAGIC 58 (ww_builtin.c 2)
unput 6 (lex.c 1)
update 32 (machine.c 14)
usage 9 (builtin.c 1)
user_error 12 (builtin.c 8)
user_trace 25 (main.c 6)
warning 25 (main.c 6)
WHERE_IS 21 (global.q.c 1)
winch 53 (curses.c 2)
writeln 9 (builtin.c 2)
w_initscr 53 (curses.c 1)
XPRINT 18 (eval.c 1)
yyerror 23 (main.c 2)
yylex 7 (lex.c 4)