Glossary - General Programming

algorithm A sequence of steps to go from some input to some specified output.

atomicity Most code operations map into several instructions to the computer. Atomicity is the property that either the entire action will be performed, or none of it. This is commonly encountered in databases: for example if you overwrite a record with a new one you want to be sure not to end up with a mashup of the original and new record if something goes wrong. In some cases writing to a variable is not atomic: you could end up with one byte in memory from the old value and 3 bytes from the new, giving garbage. This is mainly a concern with multi-threaded programming or interrupts.

automagically (humorous) Automatically, as if by magic. Used mostly for systems with nice properties of doing what is actually needed rather than following a simple recipe, or when the details of how it works are tedious and boring, but the outcome very useful.

compiler flag (Aka directive) Command line arguments passed to the compiler to control compilation. For example in C you can define a value (for use with #ifdef etc) using -D[arg_name][= value]. Optimisation levels (how hard the compiler works to speed up or reduce memory use of your program) are usually set with a directive like -O[level number].

heap Program memory that can be used dynamically (determined as the program runs), for example anything used with malloc in C, ALLOCATABLEs in Fortran etc.

interface The functions available, including their signatures. The bare minimum somebody would need to use a chunk of code.

interpreter The interpreter, sometimes called a REPL (read-evaluate-print loop) is the program which runs your code in interpreted languages. Usually you can get a prompt at which you can type code directly, or you can invoke the interpreter with a script, and it will run.

language standard Rules specifying what valid code is in a given language, and what must be guaranteed by a compiler or interpreter about how this is implemented.

mutability In languages like Python, some data types are fixed when created, and cannot be changed later. These are called immutable. In practise you will mainly notice this with tuples. You can create a new tuple from some values, but you can't change a single element. Similarly, with strings you cannot change a single character, you have to create a new string with the change included.

pass(ed) by reference Different languages pass parameters into functions differently. When passed by reference, a reference to the variable is given, so any changes will affect the named variable in the calling code. For example a function

```
FUNCTION   inc(x)
x = x+1
END FUNCTION
y=1
inc(y)
PRINT y
```

would give 2.

pass(ed) by value Different languages pass parameters into functions differently. When passed by value, the current value (at call time) of the variable is copied to a dummy variable inside the function. For example a function

```
FUNCTION   inc(x)
x = x+1
END FUNCTION
y=1
inc(y)
PRINT y
```

would give 1 as y is not changed by the call to inc.

scope Scope of a variable is the region of the program in which it exists and can be used. Most languages have "function scope" so variables you create inside a function can't be used outside it. C-like languages add "block scope" so a variable defined, for example, inside an if-block is lost when the block ends.

source (code) Your program text. This is distinct from the runnable executable, or the bytecode or tokenised code produced by e.g. Python.

stack Program memory used for static variables (where the memory needed is known at compile time and can't change) such as numbers, strings etc.

subroutine (c.f. function) In languages like Fortran, subroutines are sections of code which can be used like a function but have no return value.

undefined behaviour Things which are not specified by a language standard can do whatever they want - their behaviour is undefined. Beware that this can mean doing exactly what you expect. Until it doesn't.

Glossary - Software Development

agile development A (family of) software development method(s) where design is adapted to requirements regularly and no planning is done further ahead than a chosen time, often as little as one week.

anti-pattern Patterns for the dark-side: models of code which are inflexible, restrictive, or unreliable common approaches to a problem.

bisection (AKA Binary bisection) Searching for something in an ordered collection (an item in a list, the version of code where something breaks) by repeatedly splitting into two halves, the one containing the target, and the one not, and then repeating the process on the containing half. For example, you have code that adds one to a particular variable in say ten places, you know that it starts at zero, but ends at 9, and you want to know which step is being missed.

0, 1, 2, 3, 4, **5**, 6, 6, 7, 8, 9	Target is above 6th element, value 5
6, 6, **7**, 8, 9	Target is at or below 3rd element, value 7
6, **6**, 7	Target is at or below 2nd element, value 6
6, 6	Target is above 1st element, value 6
6	Length is one, target found

Note that we select "at or below" and "above", and when the length is even, we choose the lower element as the "middle". These are not the only choices, but it is vital to be consistent or you will sometimes get the wrong answer.

coding standard A set of rules dictating anything from techniques (some companies forbid pointers), naming conventions, source code layout (2 spaces or 4? Do braces go on a new line?) and every other element of code style. The coding standard may select a language standard but shouldn't cover matters of source code validity.

DRY Don't Repeat Yourself, the idea that you should try and reuse code as functions, modules, libraries etc rather than copy paste and edit.

input sanitation Removing active code or invalid characters from input. For example, suppose you were to (please don't ever do this without extreme caution!!) take some user input and execute it, as all sorts of online bots do. Now suppose you feed this directly to the system and your user enters "firefox http://my_super_virus_downloader.co.uk".

MoSCoW method A prioritization method where requirements are grouped into things that Must be, Should be, Could be and Wont be done, and then treated as such, often in a given iteration of a piece of code.

MUU Minimum Useful Unit; the smallest functioning, useful, releasable version of a program or feature.

pattern Models of code which are flexible, general purpose, reliable common approaches to a problem.

PLS Principle of Least Surprise; do the least surprising thing in case of ambiguity.

prototype A partial product, giving useful information. For example, a look-and-feel prototype for a webpage may be a simple image showing how the page will be arranged. A prototype code may work on only a restricted set of inputs, or produce no output, or use a slower but simpler algorithm.

refactor .

refactoring Changing the content of code without changing its results, c.f. rewriting. For example moving some chunk of code into a function; placing several named variables into a single structure; renaming things for better consistency etc.

rewrite .

rewriting Rewriting differs from refactoring as you may actually change results. Changing to a different inexact algorithm would generally be a rewrite, as would changing from plain-text output files to a special file format.

UML Unified Modeling Language; a standard way of creating program description diagrams.

waterfall development A (family of) software development method(s) where design and creation steps are fully completed before the next stage can begin, so information flows only downwards.

WSIWYG What You See Is What You Get, editors like Word where you see formatting and arrangement as you go, as opposed to e.g. Latex where you must compile your document to see it laid out.

YAGNI You Aren't Going to Need It, a feature which may seem cool but isn't actually required right now, and probably never will be. If it ever is, your current version of it probably wont work anyway.

Glossary - Testing and Debugging

code path Aka control flow path. The path taken through your source code when your program runs. For example, each "if" statement causes a branch into two paths, true and false. The complexity grows exponentially: a second if following the first gives up to 4 paths and a 3rd gives up to 8. The use of "up-to" is because in general many paths will be indistinguishable because the branches are independent. Ideally all code paths should be tested.

correct A program that uses all language features that it uses correctly. It does not necessarily give the answer that you expect.

floating point numbers Decimals, where the position of the decimal point is allowed to vary. This gives them the same relative precision over a very large range of values. Scientific notation is technically a floating point notation: although you always write a fixed number of decimals, the scaling factor 10^x means the absolute accuracy changes.

garbage collector When variables go out of scope the memory they occupy should be made available for reuse. In languages that allow references or pointers to variables, there is a problem with this, as you only want this to happen after the last reference to a given item is lost. This is done by the "garbage collector" in languages which have one. Usually, this runs every so often, or when available memory is getting tight, and cleans up all the now dead values. Note that languages like C don't have this, you must free memory when the last pointer or reference goes. In Fortran you must manually clean up Pointers, but Allocatables are automatically deallocated (since F95) and cleaned up like regular variables.

invariant A condition which is always fulfilled. For example, within any for loop (FOR i =0, 10) you know that i is between 0 and 10. Often it is very useful to know that a number is positive, non-zero etc as this allows you use code that relies on this (e.g. if you're passing the number to a sqrt function, or dividing by it respectively).

machine epsilon Floating point numbers are stored in the exponential format $a \times 2^b$. With a limited number of bits, there is a finite step from one number to the next that can be represented. For example, in base-10 with a 5 digit significand (a) we have 1.0000×10^b and the next number we can show is 1.0001×10^b. For $b = 0$ the difference between these is 0.0001. For $b = 4$ the difference is 1. This step-size is called the machine epsilon, and is usually quoted for numbers close to 1. For very large numbers, the step size can be much greater than 1. This is one reason why you should not using floating-point numbers for large integers.

minimum working example A small program that demonstrates something with as little extra code as possible. They are very useful when trying out a new library or code, or when reporting a problem or bug. If somebody is trying to help, they wont appreciate wading through reams of code to find the problem, so producing a small program that demonstrates your issue is very useful.

no-op Code that does nothing. Stands for no-operation, and can exist for many reasons. Can include incomplete statements, like `a;` in C or conditionals like `if(false)`.

overflow A numerical error caused by exceeding the largest number (positive or negative) a type can store.

precondition and postcondition Guarantees about the inputs (precondition) or the outputs (postcondition) of a function. E.g. for a sqrt function, you may have to be sure that the input is positive, and the function may promise to return only the positive branch ($\sqrt{(}9)$ is plus OR minus 3).

regression Going backwards in code fitness, e.g. reintroducing a bug which was already fixed, making answer quality worse, breaking a working feature etc.

segmentation (seg) fault A severe error in code causing it to attempt to read or write invalid memory.

syntax error An error in the sequence of characters in a piece of code. For example, a misspelled name, or a missing bracket, making the piece invalid and unreadable to the computer.

underflow A numerical error caused by attempting to store a number that is too small, often causing it to be rounded to 0.

unreachable code Code that never runs. This can be a function that is never called, or a condition that is always true or false, so its body is unreachable.

Glossary - Testing and Debugging 2

backtrace (Aka stacktrace) A readout of the call stack of your program. The call stack, loosely, is the path from where you are in the code back up to the main level.

breakpoint A command to the debugger to stop and wait for your input (i.e. to "break" execution).

call graph A tree showing which functions call, and are called by, others.

call stack The chain of function calls made by your program. A stack is a particular data structure which is "Last In First Out" so elements are taken off in reverse of the order they were put on, like a stack of real objects. Compare a "queue" which, like a real queue is FIFO: the first person to queue up is the first to be served. Each level of the call stack is called a stack frame.

continuous integration The process of continually combining code from multiple developers, usually with some automatic testing process which disallows contributing code that doesn't work.

debug symbols During compilation, the variable and function names you used are replaced with internal symbols to allow the compiler to link together all of their occurrences. For debugging, you can attach extra information, such as the name used in the source, the filename and line number where the definition was made etc. This information is used by symbolic debugger to map between your source code and the actual running executable.

entry point (function) The start of a function. In theory, and in some old code, you could jump into your function somewhere other than the start. This is generally accepted to have been a Bad Idea. Note that you can generally return from a function in several places, but be sensible.

inlining To call a function, a program will construct a stack frame, copy (where necessary) variables into place in it, jump to the function code, run the content and then jump back to the calling point, copying the return value into place if needed. To avoid this, the compiler may *inline* a function instead, replacing the function call with the content of the function.

profiling Analysing where code spends its time, to identify performance bottlenecks and hotspots for optimization.

stack frame A single level of the call stack of your program, each frame contains information on the function called, its parameters and in particular where the computer should go to when the function ends..

symbolic debugger A debugger relying on debug symbols to allow you to run your code and to link it to the source you wrote. For example, you can print the value of a variable by name, even though that name may have been altered (mangled) in the executable. Also, you can set things like breakpoints at specific points in the source code, or you can ask the debugger to continue until the next function call etc.

test runner Processes (threads) that can run tests etc. Usually these are idling until they're called upon. Automated testing systems need runners to perform tests.

Glossary - Workflow and Distribution

backwards compatibility A guarantee that anything possible or valid in an older version remains possible, so that e.g. input or output files from an older version can still be used. Sometimes this means that you can make the code behave exactly as it used to, sometimes it means only that you can use the files as a base for new files. For example Excel can read any Excel file, from any version correctly.

branch Repositories can have more than one branch, an independent set of changes, often for some purpose such as a specific feature or fix. Branches can be merged together to combine their changes.

commit (A commit) A chunk of changes, usually along with a message and an author etc. (To commit) To record files or changes in the version-control system, i.e. to add its existence and state to the history and store files and content however the system decides (often as incremental diffs).

dependency In general terms dependencies are the libraries, tools or other code which something uses (depends on). In build tools specifically, they are also known as prerequisites and are the things which must be built or done before a given item can be built or done. For example I may have a file-io module which I have to build before I can build my main code.

fetch To download changes to a repository. In git, this includes information on new branches but does not update your local copy of the code.

fork To split off a copy of code to work on independently. Often this implies some sort of difference of opinion as to how things should be done. In the Github model, forks are encouraged in normal development: one forks code, adds features and then may or may not make a pull request back to the original repository. This way only core developers can commit to the main repository, but anybody can easily modify the code.

forwards compatibility A guarantee that files etc from a newer code version will still work with the older version, although some features may be missing. E.g. a file format designed to be extended: extended behaviour will be missing, but will simply be ignored by old versions.

library Code to solve a problem, intended to be used by other programmers in their programs. For example, in C there is the Standard Library which contains things like mathematical functions, string handling and other core function which is not part of the language itself.

master A common name for the primary branch of a code. Master should always be working and ready to release, as it is the default branch in git. Some consider it a bad idea to commit changes directly to master, preferring to work on branches and then merge or rebase.

merge Combining one set of changes with another, usually by merging two branchs The combined version usually shares its name with the branch that has been merged "onto".

module A piece of a program, something like chapters in a book. In python modules are the things you import (possibly from a larger package). Fortran has modules explicitly, that are created using MODULE [name] and made available elsewhere with the USING statement. In C modules are closest to namespaces.

package A piece of software, usually stand-alone. In contrast a library is usually code only used by other programs, but there is a lot of overlap. This may contain multiple smaller modules. In context of OSs, packages are software that can be installed, whether they're available as compiled binaries or source code.

pull To download changes to a repository. In git this then integrates them with your local copy. If you have local changes, the remote changes are merged into yours.

pull request A request to pull-in code from elsewhere. In the Github model, one forks code, makes changes, and then raises a pull request for them to be integrated back to the original repo.

push To upload your local state to a remote repository (see repository). You can push commits, whole branches, git tags etc.

rebase Replay changes as though they had been applied to a different starting point. Because systems like git work in terms of changes to code, they can go back through history and redo changes from a different base. For example, one can "rebase" a branch onto another. The changes in the first branch are taken one by one, and applied to the second branch. This differs from merging mainly in how changes are interleaved in the history.

repository (Aka repo) A single project or piece of software under version control. In general a local repository is a working (in the sense of "to be worked on") copy of the code, whereas a remote repository is a copy everybody shares, pushing their work and combining changes. The remote copy can be sitting on somebody's machine - remote is a designation not a requirement. Note that git does not require a remote repo (or server), but some systems like subversion do.

sideways compatibility A guarantee that code remains compatible with other code. For example you may create files for another program to read, and you want to make sure that your output remains compatible with their input requirements, even when these may change.

stage Staging means preparing changes to be added (commited) and comes from the similar concept in transport, `https://en.wikipedia.org/wiki/Staging_area`.

VCS Version Control System; a tool for preserving versions of code and details about who, why and when they were created.