C++ Data Structures

Tom Latham

(based on material from Matt Williams)

Data structures

- Until now, if we wanted to return multiple values from a function, the only option was via reference arguments
- This gets unwieldy and difficult to maintain
- It makes sense to bundle related things together into one object

The problem

- If we want to clone a person, we have to pass all the input information and get all the outputs by reference.

```
void clone_taller(const std::string& a_name, const float a_height,
                  std::string& b_name, float& b_height)
{
    b_name = a_name + "'s taller clone";
    b_height = a_height + 0.1;
}

int main()
{
    std::string clone_name;
    int clone_height;
    clone_taller("Dave", 1.74, clone_name, clone_height);
    std::cout << clone_name << " " << clone_height << std::endl;
}
```

Person

- In that example a person is defined by their name and height
- Adding more attributes will make the function signature longer and longer
- Imagine that later we might want to modify the code so that a person is defined by their name, height, age, etc.
- We want to be able to bundle all that information into a single object, in C++ this is a structure

A data structure: struct

- A structure is created using the `struct` keyword, followed by a unique name
- Together, these define a new type
- The new type can be used like any other, e.g. to create instances, to specify function arguments, etc.

```cpp
// Define a new structure called "Person"
struct Person {
  std::string name; ///< Name of person
  int age; ///< Age in years
  double height; ///< Height in metres
};
```

- A `struct` contains a list of data members
    - Listed with their types and names
- It is enclosed in curly brackets and ended with a semi-colon

A data structure: struct

- In the example below, a struct is declared and used to make an object
- Members are accessed with the dot operator (just like you have been doing with std::vector and std::string to see if they are empty(), to get their size(), etc.)

```cpp
struct Person {
  std::string name; ///< Name of person
  int age; ///< Age in years
  double height; ///< Height in metres
};

int main()
{
  Person dave {"Dave", 24, 1.74}; //Set in order declared
  std::cout << dave.name << std::endl;
  dave.age = 25; //It's his birthday!
  std::cout << dave.age << std::endl;
}
```

Passing structs

- We can simplify our previous "cloning" example using the new Person structure

```
Person clone_taller(const Person& a){
  Person b {a};
  b.height += 0.1;
  b.name = a.name + "'s taller clone";
  return b;
}

int main()
{
  Person dave {"Dave", 24, 1.74}; //Initialised in order declared
  Person clone {clone_taller(dave)};
  std::cout << clone.name << " " << clone.height << std::endl;
}
```

Exercise 1: command-line information struct

- Create a new type, called `ProgramSettings`, that is a `struct` that holds all the command-line information
- This should be declared in the same header (.hpp) file as the `processCommandLine` function declaration
- Edit the `processCommandLine` function
  - Use your new type as a reference argument to replace many of the current ones (it should be the second of only two arguments)
  - Simply set the values of its data members instead of setting the values of the individual objects that you had before
- Edit the `main` function accordingly