Data and computation in C++

Assuming we know the rough structure of C++ programs, let's introduce the basics of data and computation

- data storage (constants and variables)
- data types (integers, reals, characters)
- size limits on data types
- the assignment operator
- literal (“hard-coded”) values in source code
- basic math operations and computation in C++

Data storage (variables)

- any data a program works with must be stored somewhere
- often the specific value to be used is unknown before the program starts, or changes as the program runs
- in our programs we can specify a name for a storage location, and the kind of data we will store there
- these named locations are referred to as *variables* (since their content can vary over time)

Declaring variables

- We must declare a variable before we can use it
- this involves specifying what kind of data it can hold, as well as the variable name
- e.g. for a variable named “age” that can hold integer (int) data, a valid declaration would be

```
int age;
```

- similarly, for a variable named “temperature” that can hold floating point data (float), a valid declaration would be

```
float temperature;
```

Assigning values to variables

- The = symbol represents the assignment operator in C++
- It is used to set/change the value stored in a variable, using the syntax

```
variablename = newvalue ;
```

- the new value may simply be a literal (hard-coded) value, e.g. 23, or it may be a complex expression

```
x = (3 + (7 - (y / z)) * 10;
```

- we can assign values as part of a variable declaration

```
int somevariable = 5;
```

Basic computation

- the usual math operations are supported (+, -, *, /) with typical order of operation rules and support for bracketing
- instructions are processed in sequence, always using the most recent value for a variable

```
int x = 3;  // x has value 3
int y;      // y has no value yet
y = 10 + x; // stores 10+3, i.e. 13, in y
x = x + 1;  // computes right side, x + 1 is 4, then
            // assigns to the variable on the left
int z = 5 * x; // uses latest value of x, so z = 20
```

The importance of initialization

- a variable has no value until one is assigned
- until that point, the value of the variable could be anything

```
int x;
int y = x; // we're using uninitialized variable x
// we really don't know what is in either variable
```

- compilers will often generate warning messages when they see you are using a potentially uninitialized variable

Basic data types (char, int, float)

- There are many different data types
- first, we'll introduce three types
  - int: for storing integers (whole numbers)
  - float: for storing floating point values (real numbers)
  - char: for storing single characters (e.g. 'x' or 'X' or '?')
- We'll introduce more complex data types as the course progresses

Size limits of data types

- Each data type has a fixed amount of space allocated for it in computer memory

  (on our system, 1 byte for a char, 4 for an int, 4 for a float)

- This means only a limited “size” of value can be stored

  (on our system, the largest int is 214783647,

  the largest float is 3.40282×10^{38})

Literal values and their downside

- We can code specific int, float, or char values directly into our C++ programs, e.g.

```
circumf = 3.1415 * diameter;
```

- these values (like 3.1415 above) are called literal values
- downside for maintainability:
  - if we use the same literal value multiple times in a program then later decide we need to change the value, we must find and correctly edit each instance of it (without accidentally changing anything else)

Constants, why they're useful

- instead of using these fixed, or constant, literal values (often called “magic numbers” by programmers), we can give the value a name and use that instead
- const float Pi = 3.1415;
- now we can use the value by name

```
circumf = Pi * diameter;
```

- this makes our code more readable and maintainable

Mixing data types in assignment

- if we assign an integer value to a floating point variable, the results are generally intuitive

```
float x = 3;  // x will hold 3.0
```

- if we assign a (small enough) floating point value to an integer, the results will be truncated

```
int y = 2.9; // y will hold 2
```

Mixing data types in operations

- when the compiler sees an expression like “x + y”, the compiler must decide what kind of addition to use (most computer chips have different circuitry to add integers than to add floating point values)
- if x and y are both integers then it uses integer addition, but if one or both are floats then it uses floating point addition
- 3 + 10 gives an integer result, 13
- 3.0 + 10 gives a floating point result, 13.0

int versus float division

- integer division in C++ drops any remainder
  - 7 / 3 gives 2 (dropping the remainder, 1)
  - 6 / 11 gives 0 (dropping the remainder, 6)
- floating point division computes the full result
  - 3.0/4 gives 0.75
- if you wish to know the remainder in integer division you must use the modulo (%) operator
  - 7 % 3 gives 1
  - 6 % 11 gives 6