C++ Part 1

Object oriented programming

- Designed around data and objects rather than functions and logic
  - Object classes can contain member properties and functions
- Programs can be compiled or interpreted
  - Compilation creates machine-readable commands from human-readable code
    - Typically runs faster but isn’t universally readable across all computers
  - Interpretation uses human-readable code
    - Not machine-dependent but typically executes more slowly
- C++ is typically compiled and Python is an interpreted language
- C++ is based on C, but is an object oriented language

C++ files

- *.cpp or *.cxx: source code where the main code lives
- *.h: header file where declarations are typically done
- *.out or *.exe: executable to run the compiled program
- Note that these are conventions and not required

Compilation

- 4 steps in compilation
  - Preprocessing: remove comments, expand macros and included files
  - Compiling: generate assembly language from c++ code
  - Assembly: convert assembly code into pure binary code (known as object code)
  - Linking: merge object code from multiple modules and link library function code
- Many compilers available - we will be using GCC
- Syntax and other errors can be found when compiling
  - Logical errors and other issues typically only show up at run time

Compilation II

- Compile source code mycode.cxx with:

  g++ mycode.cxx

- This creates an executable a.out that can be run using:

  ./a.out

- The following command allows you to name the output e.g., main.exe:

  g++ -o main.exe mycode.cxx

Basic c++ syntax

- Whitespace is ignored
  - Indentation is useful but not required
- Lines end with semicolon (;)
- Single line comments are denoted with //
- Block comments denoted as /* … */
- Scopes are defined using { … }
- Arguments are defined with (…)
- Preprocessor directives (such as include statements) begin with #

The basic source code structure

```
// hello.cxx file
int main()
{
    return 0;
}
```

Including libraries

- Include standard or user-defined libraries to make use of functionality
  - List of standard libraries available here: https://en.cppreference.com/w/cpp/header
- Include directives should appear at the top of the source code
- Syntax:
  - #include <blah> for standard libraries
  - #include “myblah.h” for user-defined libraries

Output messages

- It is useful to add print out statements so you can track what your code does
- Typically done using the iostream library and std::cout statements
  - Formatted output (printf) is also possible, but primarily for special cases

```
// hello.cxx file
#include <iostream>
int main()
{
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

Variable and primitive data types

- c++ makes use of variables that temporarily hold values
- Variables must be explicitly declared before they can be used
  - It is good practice to initialize variables to avoid undefined behavior
- Variables must have a type, either primitive or non-primitive
- Primitive data types:
  - int: Integer value
  - float: Floating point value (i.e., decimal value)
  - double: Double precision float (twice the precision)
  - fool: True or False
  - char: Single ASCII character
  - void: No data (empty)
- Common modifiers:
  - unsigned
  - long

Check the size of each data type on your machine using e.g., sizeof(int)

User input

- The iostream library allows you to read in user input to variables

```
// hello.cxx file
#include <iostream>
int main()
{
    int first = -1; // my first number
    int second = -1; // my second number

    std::cout << "Hello world" << std::endl;
    std::cout << "Please type two numbers:" << std::endl;
    std::cin >> first >> second;
    std::cout << "You typed: " << first << " and " << second << std::endl;

    return 0;
}
```

Mathematical operations

- Arithmetic
  - + : addition
  - - : subtraction
  - * : multiplication
  - / : division
  - % : modulus (remainder divide)
  - ++ : increment by 1
  - -- : decrement by 1
- Logical
  - && : logical AND
  - || : logical OR
  - ! : logical NOT
- Assignment
  - = : assign value
  - += : increase by value
  - -= : decrease by value
  - *= : multiply by value
  - /= : divide by value
  - %= : modulus by value
- Comparison
  - == : equal to
  - != : not equal to
  - > : greater than
  - < : less than
  - >= : greater than or equal to
  - <= : less than or equal to

Logical flow controls

```
if (<condition 1>) {
  <do something>
  if (<another condition>) {
    <do something>
  }
}
else if (<condition 2>) {
  <do something>
}
else {
  <do something>
}
```

```
switch (<expression>) {
  case <value 1>:
    <do something>
    break;
  case <value 2>:
    <do something>
    break;
  case <value 3>:
    <do something>
    break;
  default:
    <do something>
}
```

Resources

- https://www.w3schools.com/ - Great online learning resource
- https://www.youtube.com/@codebreakthrough - Excellent tutorial videos
- https://en.cppreference.com/w/ - Thorough documentation
- https://stackoverflow.com/ - Ask questions to experts