C/C++ – Lecture 1

AMS 595 / DCS 525

Stony Brook University – Applied Math & Statistics

Fall 2023

Final Projects

Final Project

- 25% of final grade
- Groups of 2-4
- Finalize teams by 11:59 PM November 9th
  - Discussion board will be posted on Brightspace
  - If you are having trouble finding someone, please email me ASAP
- Groups and topics must be approved by both instructors
  - Send both instructors an email once you have your group and topic finalized
- Last two classes devoted to presentations
  - December 5th and December 7th
  - Strict adherence to time
  - More information regarding presentations and submissions will be given later in the semester

Final Projects

- Project difficulty must scale with group size
- Project can be in MATLAB, Python, or C++
  - Teams that use C++ might accomplish less, this will be taken into account
- Example projects:
  - Numerical analysis (e.g. $Ax = b$ and/or differential equation solver)
  - Statistics (e.g. ARMA implementation)
  - Quantitative finance (e.g. portfolio optimization, options pricing, stock forecasting)
  - Machine learning
  - Other (e.g. a simple game)
  - ...etc
- Use GitHub to keep track of everyone's contributions

C/C++

A Brief History

- BCPL → B → C → C++
- C was developed by Dennis Ritchie at Bell Labs
  - Procedural, “low level” language
  - Many commercial applications written in C
- C++ was developed by Bjarne Stroustrup at Bell Labs
  - C is practically a subset of C++
  - Overcame several shortcomings of C
  - C++ provides support for OOP
  - The ++ is a pun (++ is the increment operator)

Applications of C++

- Scientific and numerical computations (e.g. simulations)
- Banking and financial sectors (e.g. high frequency finance)
- Systems programming (e.g. operating systems)
- Entertainment (e.g. video games)
- Libraries (e.g. back-end of machine learning libraries)
- GUI based applications (e.g. Adobe Photoshop)
- Database software (e.g SQL)
- Browsers (e.g. Mozilla Firefox)
- Robotics (e.g. NASA's Curiosity rover)
- ...and many more

Running C++

- IDEs:
  - Windows: Visual Studio
  - macOS: Xcode
- Compilers:
  - GNU
    - g++ - C++
    - gcc - C
  - Clang
- You may also configure some text editors to run C/C++
- Example using g++:

```
g++ myfile.cpp -o mynewname
```

- Instructions on configuring VS Code to compile C/C++ can be found here
- Instructions on installing the GNU compiler can be found here for Mac and here for Windows

Resources

- Websites:
  - `https://www.learncpp.com`
  - `https://www.cplusplus.com`
  - `https://en.cppreference.com/w/`
- Texts:
  - *C++ Primer* by Stanley Lippman
  - *Programming: Principles and Practice Using C++* by Bjarne Stroustrup

About C++

- General purpose language
- Compiled language
- Statically typed
- Very useful when we are concerned with performance
- Original C++ standard is referred to as C++98
- Current release is C++20
- The resulting executable is OS dependent

About C++

- Common file extensions: `.cpp` or `.hpp`
  - Common C file extensions: `.c` or `.h`
- Heavy use of braces {} and semicolons ;
- To comment use //
  - For multi-line comments, use `/* your comments here */`
- Execution always begins with the `main` function
  - By default, `main` will always return 0 and indicated that the program ran successfully
- Standard library

C++ Layout

- The C++ compiler accepts almost any pattern of line breaks or indentation
- It is good practice to format programs so they are easy to read
  - Opening '{' and closing '}' braces should go on a line by themselves
  - Indent statements when appropriate (e.g. contents within a function)
  - Use only one statement per line

Identifiers

- An **identifier** is given to any user-defined entity in your program
  - Variables, functions, classes...etc
- Identifiers are case-sensitive
  - "`A`" $\neq$ "`a`"
- Must start with a letter or an underscore, and may be followed with letters, underscores, or digits
- Some C++ compilers recognize only the first 32 characters of an identifier as significant
- Avoid identifiers that begin with a single or double underscore
- Cannot conflict with any of the reserved keywords (e.g. `int`, `float`, `while`,...etc)

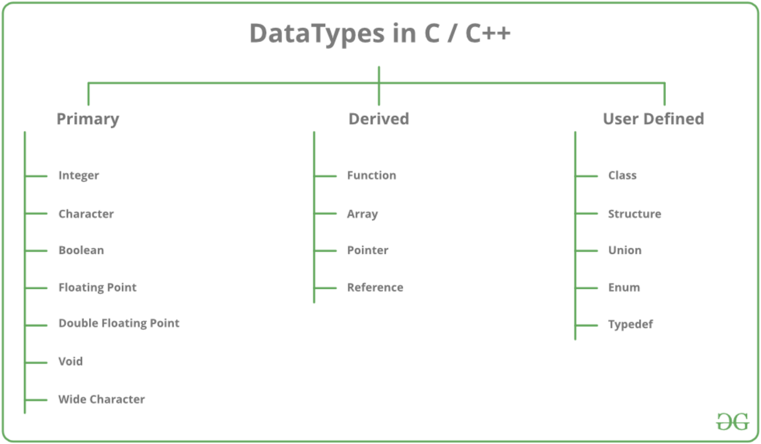
Variables

- Integral types
  - Integer types (both signed and unsigned): `int`, `short`, or `long`
  - `bool`
  - `char`
- Floating types
  - `float`, `double`, `long double`
- Character types
  - `char`
  - Single quotes
- Strings are **not** a built-in type
  - Provided by the C++ standard library, `#include <string>`
  - `std::string`
  - Double quotes
- Void type
  - `void`

Variables

- A *named constant* is a location in memory that we can refer to by an identifier, but **cannot** be changed
  - Ex. `const int voting_age = 18`
- Literals:
  - Integer literals
    - Decimial
    - Octal, 0 follows by zero or more octal digits
    - Hexadecimal, 0x or 0X followed by one or more hexadecimal digits
    - Binary, 0b or 0B followed by one or more binary digits
  - Float literals
    - Scientific notation, exponent is indicated by either E or e
    - A suffix may be included to indicate the type of literal, for example, by default the type is `double`
    - `float`: `.f` or `.F`
    - `long double`: `.l` or `.L`
  - Character literals
  - String literals

Variables



DataTypes in C / C++

- Primary
  - Integer
  - Character
  - Boolean
  - Floating Point
  - Double Floating Point
  - Void
  - Wide Character
- Derived
  - Function
  - Array
  - Pointer
  - Reference
- User Defined
  - Class
  - Structure
  - Union
  - Enum
  - Typedef

Figure: Source

Variables

- In C++, variables **must** be declared before they can be used
- Declaration means specifying both its name *and* its data type
- Definition provides the actual information and causes the object to be created
- This can be done in the same line or separately

```
int a;
a = 100;
```

or

```
int a = 100;
```

Input and Output

- No input and output methods are built into C++
- This is instead provided to use by the standard library
  - `#include <iostream>`
- Output: `cout`
- Input: `cin`
  - Stops reading the input once it has reached a whitespace character (i.e. newline, tab, space...etc)
  - If you would like to include whitespace characters in your string, instead use `getline` from the standard library
- Insertion operator: <<
- Extraction operator: >>
- To end a line: `std::endl` or '\n'
  - Note these are *not* exactly the same

Input and Output in C

- C has a library to perform input/output and uses *formatted strings*
- `#include <stdio.h>`
- Output: `printf()`
  - Ex: `printf("%5.2f", n)`
  - This would print out the float `n` and only display the first 2 digits after the decimal, and would ensure the output's character width is at least 5
  - Adding a dash '-' after the % changes the justification from right to left
- Input: `scanf()`
  - Ex: `scanf("%d", &n)`
  - Note we store the input in `n` by calling its *address*
- Some common specifiers:
  - `%d` - decimal integer
  - `%f` - floating point number
  - `%lf` - double number
  - `%c` - single character
  - `%s` - string of characters

Example Using Strings

In this example we take in a string from the user and store it in the variable MyString, we then output this back to the console

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string MyString;
    cin >> MyString;
    cout << MyString;
}
```

Python vs C++

Let's consider a simple program; we would like to take an integer n from the user and print out $n+1$ to the console

Python:

```
n = int(input("Enter a number: "))
print("Your number plus one is: {}".format(n+1))
```

Python vs C++

C++:

```
#include <iostream>
using namespace std;

int main(){
    int num;

    cout << "Enter a number: ";
    cin >> num;
    cout << "Your number plus one is: " << ++num << endl;
}
```

Using C

```c
#include <stdio.h>

int main(){
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Your number plus one is: %d \n", num+1);
}
```

The Compilation Process

- C++ is a complied language, but how does this process work? How do we go from a C++ source file to an executable?
- The compilation involves three main steps
  1. Preprocessing
     - Prepare our code for compilation by essentially "copy and pasting"
     - Example: replace `#include` with contents of corresponding file
     - **No** compilation occurs during this phase
  2. Compilation
     - Process the source code to produce an object file, `.o`
     - Compiles each file in the program individually (i.e. they are not yet linked)
     - This object file contains machine code along with extra information
  3. Linking
     - Create an executable file from multiple object files

The Compilation Process

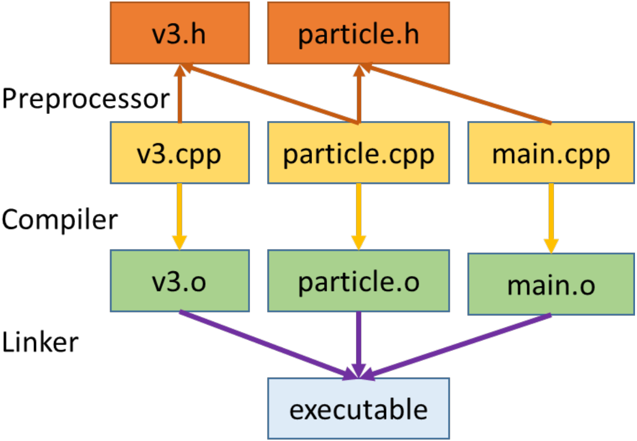


Figure: Source