

An Introduction to C++

from if statements, functions, classes, libraries, to building your own interactive story driven game.

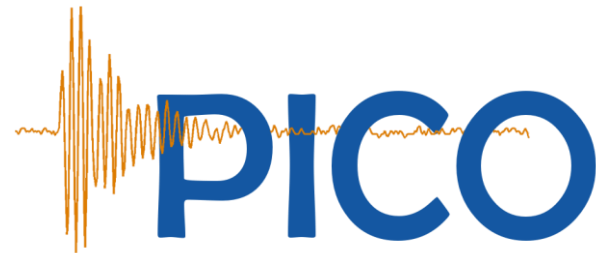
Remington Hill

Contact Info: remington.hill@queensu.ca

Supervisor: Dr. Stephen Sekula

Queen's University

May 8th, 2024





Outline

- Introduction.
- Formatting.
- Compiler(s).
- Libraries.
- Namespaces.
- Control statements (if, else, while, etc.).
- Functions.
- Classes.
- Vectors
- Zork project



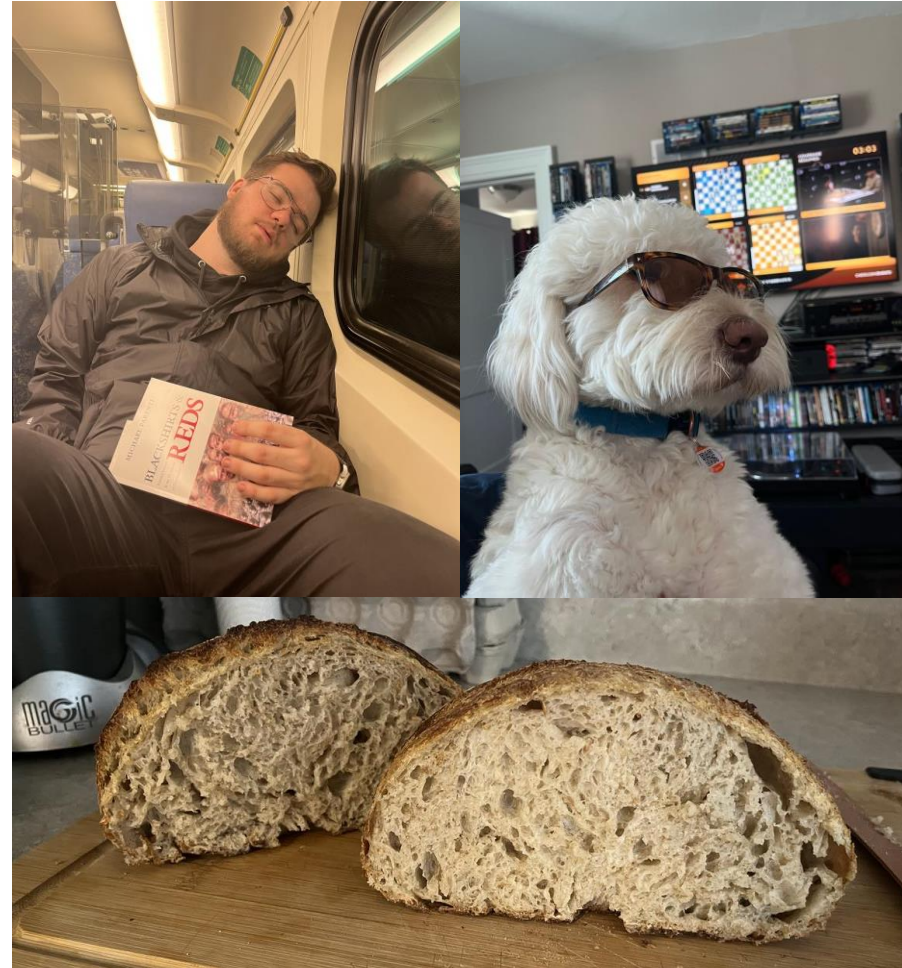
What is and what isn't this talk?

- This is not a formal lecture on C++.
 - I am not a computer scientist.
- I assume that each of you have no experience coding in C++.
 - If you have experience in C++, there is a 85% chance I put you to sleep.
- My hope is that after today, you have enough of the basics to do the following:
 - Understand collaboration analysis tools and write code in the existing framework.
 - Run Geant4 simulations (or other equivalent C++ simulations).
- A couple of questions before we get started:
 - How many of you have taken a computer science course before?
 - How many of you have written software in C++ before?
 - Be honest! No judgement. I am using this to gauge pace.
 - Who has written in Python before?



Introduction

- My name is Remington W. Hill.
- First year PhD Student at Queen's University (Sept. 2023).
 - Supervisor: Dr. S. Sekula.
 - Working on commissioning PICO-500 and analyzing PICO-40L data.
- I did my MSc at Laurentian University.
 - Supernova neutrino detection (Supervisor: Dr. C. Virtue).
- You can find me in Sudbury at SNOLAB (2nd floor cubical space, C15).
- Hobbies:
 - Baking and cooking .
 - Reading (sci-fi/fantasy, history, etc.).
 - Walking my dog on the local trails in Sudbury.
 - I enjoy playing and watching chess (the most recent candidates was spectacular).



Some general remarks

- Don't be afraid to ask for help!
 - Can always speak to your coworkers, supervisor, etc.
 - If you need help, I can be reached at remington.hill@queensu.ca or at C-15 on the 2nd floor at SNOLAB (for those who are on site this summer).
- You are going to make mistakes; learn from them.
 - Here are my highlights:
 - 2018: Undeclared variable I assumed was out of scope, turned out to be spelt wrong.
 - 2019: Insufficient memory storage resulting in me creating 4-5 TBs of data and crashing my simulation.
- Make sure you keep a good reference handy.
 - I use “C++: How to program” by Deitel and Deitel (in fact, most examples in this talk are largely inspired by this textbook, as well as last years talk by J. Hucker).
- Understanding the limits of googling and using someone else's code.
 - Stack overflow is an **asset**. However, do not just copy the code without understanding why it is written the way that it is.
 - This will make you a better programmer. Will also help with readability; whomever takes over your code will thank you.

* Here, when we say every line, we exclude control operators such as if, else, while, etc. Classes, structs (see later slides), will require semi-colons.

C++ structure

- Every line must be ended with a semi-colon* (;).
- Variable types need to be specified upon declaration.
 - Is it an int, double, float, TFile, TTree, G4int? You need to tell the compiler.
- Indentation is optional; however, please respect your fellow programmers.
 - When entering a conditional statement, class, struct, etc. it is good practice to increase the indentation by 1 tab or 4 spaces.
- Make sure you leave useful comments.
 - Each of you will work on code that someone else will then use (potentially).
 - Do not make them pull their hair out.
 - For today only, I am breaking this rule to avoid clutter.
- Choose appropriate variable names.
 - Do not store an energy value in a float called “temp”.
 - This means nothing.
 - Suppose you were looking to store both the electrical charge of an electron, and the permittivity of free space, what would be a good name for each?



A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      cout << "Hello World" << endl;
7
8      return 0;
9  }
10 }
```

A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6
7      cout << "Hello World" << endl;
8
9      return 0;
10 }
```

The method to include standard libraries or your files.

If enclosed by `< >`, compiler will look in your path for the library.

If included by double quotes (i.e. `#include "experiments.h"`), local directories are checked.

A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...  
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int main(){  
6  
7      cout << "Hello World" << endl;  
8  
9      return 0;  
10 }
```

Tells the compiler which namespace is being used throughout program.

For example, cout and endl belong to the std namespace.

This line allows us to use these two functions without specifying their namespace.

Without this line, all calls to functions of this namespace need to include std::



A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6
7      cout << "Hello World" << endl;
8
9      return 0;
10 }
```

This is the main function; it is the entry point for your C++ program.

Every function will have its arguments defined in (), and be closed by {}.

Function definitions before this will not be called or used unless called within the main() function.

int tells the compiler that this function returns an int value when it exits.

Can return any type (double, int, vector<int>, etc.).

A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...  
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int main(){  
6  
7      cout << "Hello World" << endl;  
8  
9      return 0;  
10 }
```

Print to the terminal the string
“Hello World”.

cout and endl are objects from
std; more on this later (see
Namespaces).

cout stands for ‘character
output’. endl ends the current
line.



A classic example

- The first thing you, as a programmer, will typically do in a new language is print the classic, “Hello World”.
 - What does this look like in C++?

```
hello_world.cpp > ...  
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int main(){  
6  
7      cout << "Hello World" << endl;  
8  
9      return 0;  
10 }
```

Return value of the function.

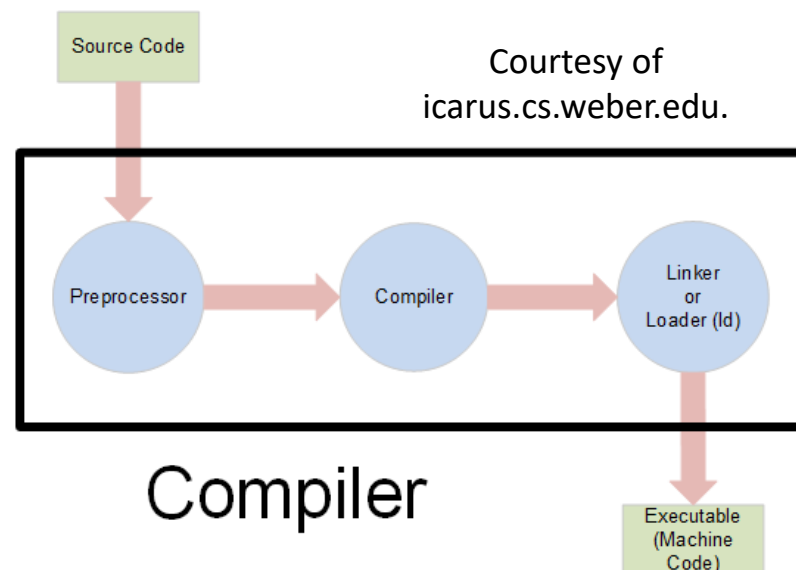
Returning zero tells the compiler that main exited without issue.

Any other value is some error status you need to identify.

User defined functions return whatever the user provides and

Compiling

- How do we run this code?
 - Will need to convert this from human readable to machine readable.
- Right now, we have what is often referred to as the **source code**.
 - For most daily applications, you don't have access to this (i.e. whatever video games you are playing right now).
- To get the machine readable file, also called the **executable**, we must look to compiling this C++ script.
- **Preprocessor**: process all statements with #; copies C++ statements to temp. files.
- **Compiler**: translates source code to machine code.
- **Linker**: will link together object code files (headers, more on this later).



Compiling pt. 2

- Here, we are going to use the command line and g++ (on LINUX).
- To compile, run: `C:\Users\User>g++ Hello_World.cpp -o Hello_World|`
- This will yield an executable of the name “Hello_World”.
 - It is fairly standard to use the same source code name, but drop the file extension.
 - If no name is supplied, default is a.out.
- There are other arguments that can be included:
 - You will most certainly use `-I/path/to/headers/`, which tells the compiler to include the supplied directory.
- To run this file, simply put: `C:\Users\User> ./Hello_World|`
- If you do not have access to the command line and g++ today (or a windows compiler), can use this: <https://www.programiz.com/cpp-programming/online-compiler/>
 - A temporary solution; you should seek access to a permanent solution.
- For large files or projects, I encourage you to use ‘make’ (or GNUmake).
 - Will automatically check which parts of the code need updating.
 - i.e. unchanged files are not recompiled.
 - You can find documentation here: <https://www.gnu.org/software/make/manual/make.html>.

Data types

- C++ requires you to be exact when defining variables.
 - You must tell the machine what it is handling.
- So what are they?
- **Declaration:** this is telling the compiler what the name **and** data type of your variable is.
- **Initialization:** This is supplying an initial value to the variable.

Key word	Size in bytes	Interpretation	Possible values
bool	1	boolean	true and false
unsigned char	1	Unsigned character	0 to 255
char (or signed char)	1	Signed character	-128 to 127
wchar_t	2	Wide character (in windows, same as unsigned short)	0 to $2^{16}-1$
short (or signed short)	2	Signed integer	-2^{15} to $2^{15} - 1$
unsigned short	2	Unsigned short integer	0 to $2^{16}-1$
int (or signed int)	4	Signed integer	-2^{31} to $2^{31} - 1$
unsigned int	4	Unsigned integer	0 to $2^{32} - 1$
Long (or long int or signed long)	4	signed long integer	-2^{31} to $2^{31} - 1$
unsigned long	4	unsigned long integer	0 to $2^{32} - 1$
float	4	Signed single precision floating point (23 bits of <u>significand</u> , 8 bits of exponent, and 1 sign bit.)	$3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$ (both positive and negative)
long long	8	Signed long long integer	-2^{63} to $2^{63} - 1$
unsigned long long	8	Unsigned long long integer	0 to $2^{64} - 1$
double	8	Signed double precision floating point(52 bits of <u>significand</u> , 11 bits of exponent, and 1 sign bit.)	$1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$ (both positive and negative)
long double	8	Signed double precision floating point(52 bits of <u>significand</u> , 11 bits of exponent, and 1 sign bit.)	$1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$ (both positive and negative)

Courtesy of go4expert.



Namespaces

hello_world_no_namespace.cpp > main()

```
1 #include <iostream>
2
3
4
5 int main(){
6     std::cout << "Hello World" << std::endl;
7
8     return 0;
9
10 }
```

hello_world.cpp > ...

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     cout << "Hello World" << endl;
7
8     return 0;
9
10 }
```

- What are namespaces?
 - Suppose you include two header files, Alice.h and Bob.h.
 - Each are a different namespace.
 - They both function within them called person().
 - How do we specify which one we are using?
- Namespaces: define the scope where global identifiers and global variables are placed.
 - I.e. Within the namespace, Alice, access to all identifiers is possible.
- To use functions within a namespace, two approach's.
 - Specify namespace followed by function or variable name (Alice::person()).
 - Or use the namespace (see left).



Control statements

- Control statements allow you to control where a program will go and what actions it will take.
- The most common are:
 - `if (condition){..code..}`
 - `else if (condition){..code..}`
 - `else{..code..}`
 - `for(start; stop; increment){..code..}`
 - `while (start/condition){..code..}`
 - `do while(condition){..code..}`
- Examples of each can be found on the right.
 - There are critical differences between all of them (lets discuss).

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6
7     int a = 1;
8     int b = 10;
9
10    // Example of if statements and conditions.
11    if (a < b) cout << "a < b" << endl;
12    else if (a > b) cout << "a > b" << endl;
13    else{
14        cout << "a = b" << endl;
15    }
16
17    // While loop. Stop before a = b
18    while (a < b){
19        cout << "a = " << a << endl;
20        a++
21    }
22
23    // Do while loop. Stop before a = b
24    a = 1;
25    do{
26        cout << "a = " << a << endl;
27        a++
28    }while(a < b);
29
30    // For loop. Equivalent to the while loop above.
31    for (a = 1; a < b; a++){
32        cout << "a = " << a << endl;
33    }
34
35    return 0;
36 }
```



Find the errors!

- We should have enough information to now begin to identify errors in the code.
 - There are 5/6 errors in the code below.
 - Take 1-2 minutes amongst yourself to identify what they are.

```
etiquette_example_1.cpp > main()
1  #include <iostream>
2
3  int main(){
4
5      a = 50;
6
7      while (a < 100){
8          cout << "a = " << a << endl
9      }
10
11 }
```



Corrected code

etiquette_example_1_corrected.cpp > main()

```
1 #include <iostream>
```

```
2  
3 int main(){
```

```
4  
5     int a = 50;
```

```
6  
7     while (a < 100){
```

```
8         std::cout << "a = " << a << std::endl;
```

```
9         a++;
```

```
10     }
```

```
11
```

```
12
```

```
13     return 0;
```

```
14 }
```

1) Missing variable type in declaration.

2/3) Missing namespace identifier(s).

4) Missing semi-colon at the end of the line.

5) No increment resulting in infinite loop.

6) Missing return statement.
(This technically won't give an error message, but good practice is to always return a value for non-void functions)

Listen to your compiler!

- Listen to your compiler.
 - If the compiler encounters an error, it will be very explicit on where it is.
- Let us look at the log from our previous example.
 - Some things it won't catch (i.e. infinite loop; that is your responsibility!).

```
(base) [rhill@nearline-login EIEI00_2024]$ g++ etiquette_example_1.cpp -o etiquette_example_1
etiquette_example_1.cpp: In function 'int main()':
etiquette_example_1.cpp:5:5: error: 'a' was not declared in this scope
   5 |     a = 50;
     |     ^
etiquette_example_1.cpp:8:9: error: 'cout' was not declared in this scope; did you mean 'std::cout'?
   8 |     cout << "a = " << a << endl
     |     ~~~~
     |     std::cout
In file included from etiquette_example_1.cpp:1:
/cvmfs/soft.computecanada.ca/easybuild/software/2020/Core/gcccore/9.3.0/include/c++/9.3.0/iostream:61:18: note: 'std::cout' declared here
   61 |     extern ostream cout; /// Linked to standard output
     |     ~~~~
etiquette_example_1.cpp:8:32: error: 'endl' was not declared in this scope; did you mean 'std::endl'?
   8 |     cout << "a = " << a << endl
     |                                ~~~~
     |                                std::endl
In file included from /cvmfs/soft.computecanada.ca/easybuild/software/2020/Core/gcccore/9.3.0/include/c++/9.3.0/iostream:39,
   from etiquette_example_1.cpp:1:
/cvmfs/soft.computecanada.ca/easybuild/software/2020/Core/gcccore/9.3.0/include/c++/9.3.0/ostream:599:5: note: 'std::endl' declared here
   599 |     endl(basic_ostream<_CharT, _Traits>& __os)
     |     ~~~~
```

Command Line / User Inputs

- Yesterday, you would have learned how to navigate the command line at Mark's talk.
 - I will assume you are familiar with the basics.
- Suppose I wanted to check which two numbers, a and b , was smaller.
 - Can I avoid having to recompile each time? Yes.
 - Two solutions we will cover:
 - Command line arguments
 - User input during execution (cin).
- For command line arguments, modify the main function to include two arguments (which we will talk about later):

```
int main(){}  
int main(int argc, char* argv[]){}  
int main(int argc, char** argv){}
```

Command line (cont.)

- The code on the right will take two integers as inputs.
- It then uses three conditional statements to check if:
 - $a < b$
 - $a > b$
 - $a == b$
- The pointer to the array of `argv[]` needs to be converted to the appropriate type.
 - `atoi()` for ints.
 - `atof()` for doubles.
- `argv[0]` is the filename of the executable.

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int main(int argc, char* argv[]) {

    if (argc != 3) {
        cout << "ERROR: Incorrect number of arguments given." << endl;
        return 1;
    }

    int a = atoi(argv[1]);
    int b = atoi(argv[2]);

    if (a < b) cout << "a < b" << endl;
    else if (a > b) cout << "a > b" << endl;
    else {
        cout << "a = b" << endl;
    }

    return 0;
}
```



Character inputs

- Another variation of this program is character input.
 - Called cin; opposite of cout.
 - Like cout, it is apart of the iostream header.
- This asks the user for **input** during execution.
- The general structure is to declare some variable of a given type, then use the overloaded cin operator to store it

```
int val = 0;
cin >> val;
```

- Let us see this in action.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6
7      int val = 0;
8
9      while (val > 10 || val < 1){
10         cout << "Please enter a number between 1 and 10." << endl;
11         cin >> val;
12     }
13
14     cout << "Thank you for entering " << val;
15     cout << ", which is in the correct bounds!" << endl;
16
17     return 0;
18 }
```

```
Please enter a number between 1 and 10.
0
Please enter a number between 1 and 10.
11
Please enter a number between 1 and 10.
12
Please enter a number between 1 and 10.
-1
Please enter a number between 1 and 10.
1
Thank you for entering 1, which is in the correct bounds!
```



Exercise (basic intro.)

- Take the next 5 minutes to do the following:
 - Write a program that asks the user for two numbers.
 - Stipulate that the one number must be greater than the other.
 - Loop over the range that these two numbers span (incrementing by 1 at each step).
 - At each point, compute the product of the increment and upper bound and output the information to the terminal to update the user.
 - BONUS: Instead of printing the product, cout the division and the remainder for the two numbers. For the remainder, you will need to use the % operator.



Functions

- Thus far, we have made use of functions from the standard library.
 - You are free to create your own.
- As is the case with our main, our custom functions will have the following format:
type function_name(*args*){}
- *type*: Denotes the type of value returned by the function.
- *function_name*: The name you give to the function.
- *args*: the arguments supplied to the function (types must be specified and separated by commas).
- Lets talk about some of the examples on the right.

```
1  #include <iostream>
2
3  using namespace std;
4
5  double add(double a, double b){
6      return a + b;
7  }
8
9  double subtract(double a, double b){
10     return a - b;
11 }
12
13 double multiple(double a, double b){
14     return a*b;
15 }
16
17 double f(double x){
18     return x*x - 13*x + 42;
19 }
20
21 int main(){
22
23     cout << add(1,2) << endl;
24     cout << subtract(1,2) << endl;
25     cout << multiple(1,2) << endl;
26     cout << f(2) << endl;
27
28     return 0;
29 }
30
```

Question time!

- Will the following code give me an error during compilation?

NO!

```
1  #include <iostream>
2
3  using namespace std;
4
5  double multiply(double x, double y){
6      return x*y;
7  }
8
9  int multiply(int x, int y){
10     return x*y;
11 }
12
13 int main(){
14
15     int i_product = multiply(2, 4);
16     double d_product = multiply(2.1, 4.2);
17
18     cout << "Product using integer multiply function: " << i_product << endl;
19     cout << "Product using double multiply function: " << d_product << endl;
20
21
22     return 0;
23
24 }
```



Function overloading

- Function overloading is a method of having the same function name but with different arguments.
- This is allowed, if and only if, there arguments are different.
 - Compiler will look for appropriate function call.
- An unrealistic example is on the right.
 - In practice, a sum function for array or vector of length N would be used.
- See this in ROOT, Geant4, etc. very often.
 - TH1, Tfile, TTree, TBranch.

```
1  #include <iostream>
2
3  using namespace std;
4
5  ✓ double add(double a, double b){
6      |     return a + b;
7  }
8
9  ✓ double add(double a, double b, double c){
10     |     return a + b + c;
11 }
12
13 ✓ double add(double a, double b, double c, double d){
14     |     return a + b + c + d;
15 }
16
17 ✓ int main(){
18     |
19     |     cout << add(1,2) << endl;
20     |     cout << add(1,2,3) << endl;
21     |     cout << add(1,2,3,4) << endl;
22     |
23     |     return 0;
24 }
25
```



Function prototyping

- Something you will encounter when working with classes is **function prototyping**.
 - A method of defining a function and its arguments prior to specifying its operation.
- Useful for when functions depend on one another.
 - Or for the sake of tidiness, a class is defined in a header file outside the current file.

```
1  #include <iostream>
2
3  using namespace std;
4
5  double f(double x);
6
7  int main(){
8
9      for (int i = -50; i < 50; i++){
10         cout << "x = " << i;
11         cout << ", f(x) = " << f(i) << endl;
12     }
13
14     return 0;
15 }
16
17 double f(double x){
18     return x*x + 3*x + 5;
19 }
20
21
22
```

Namespaces pt. 2

- You can create your own namespace.
- Suppose you have a set of custom functions that you use in your analysis.
 - Can place these in a header file and include this in future analysis to have access to the functions.
- To define a namespace:
`namespace nsp_name{..code..}`
- An example is on the right.

```
1 #include <iostream>
2
3 namespace analysis{
4
5     const double energy = 2.614;
6     const double ratio = 0.99;
7
8     double f(double x){
9         return x*x;
10    }
11
12    namespace sub_analysis{
13        const double momentum = 3.14;
14        const double e2 = energy*energy;
15
16        double g(double x){
17            return x*x*x;
18        }
19    }
20 }
21
22
23 int main(){
24
25     double a = 2.0;
26     double a2 = analysis::f(a);
27     double a3 = analysis::sub_analysis::g(a);
28     std::cout << "A = " << a << ", A^2 = " << a2 << ", A^3 = " << a3 << std::endl;
29
30     double e = analysis::energy;
31     double e2 = analysis::sub_analysis::e2;
32     std::cout << "Energy = " << e << ", Energy squared = " << e2 << std::endl;
33
34
35     return 0;
36 }
```

Main namespace (analysis).

Nested namespace (sub_analysis).

sub_analysis has access to members of analysis.

A brief aside on scope

- Let's talk about scope.
- You might often hear programmers talk about scope. What does this mean?
- Scope tells us where a function, variable, struct, class, **is or is not** defined.
- We saw earlier namespaces, functions:
 - The rule of thumb is that when a variable is defined within one of these, it is **local** to this scope.
 - You cannot access it outside of this namespace of function.
- It is good practice to avoid global variables.



Scope error

- Let us take a couple minutes to find the scope error.
- Where is the error?

FOUND IT!

If we try to declare that `d` is equal to `a`, the compiler will recognize that `a` is “not declared in this scope”.

```
1  #include <iostream>
2
3  namespace top_namespace{
4
5      int b = 5;
6      double c = b*2.0;
7
8      namespace inner_namespace{
9          int a = b;
10         double f(double x){
11             return x*x + 2*x -3;
12         }
13     }
14
15     int d = a;
16     d = d*b + b;
17     double g(double x){
18         return 2*x;
19     }
20 }
21
22
23
24 int main(){
25
26     std::cout << "Just hanging out!" << std::endl;
27
28     return 0;
29 }
```

Exercise

- Take the next 10 minutes to do the following:
 - Write a program that prompts the user for two numbers.
 - The first will be a lower limit and the second will be an upper limit.
 - Create a function, some kind of polynomial (*i. e.* $f(x) = 2x^2 + 3x - 12$).
 - Use the function you have created, as well as a loop of your choice to evaluate the function at every point across range.
 - BONUS: Compute the average value of the function on this interval.:
 - Remember that the average value of a function is obtained through the mean value theorem.
$$\frac{1}{(b - a)} \int_b^a f(x) = f(c)$$
 - Don't bother numerically integrating the function, just use the fact it is a simple polynomial to find the anti-derivative.



Arrays and vectors

- Vectors and arrays serve a similar function. But are vastly different in execution.
 - Both can store multiple instances of the same type (i.e. int, double, etc.).
- Arrays are **static** entities (like structs).
 - This means that they **cannot** change size after declaration.

type array_name[size];

- Vectors are **dynamically** sized entities.
 - They **can** change size after declaration (although, be careful of memory constraints).

std::vector<type> vector_name;

- Each have their own respective functions.
 - std::vector reference (<https://cplusplus.com/reference/vector/vector/>).
 - std::array reference (<https://en.cppreference.com/w/cpp/container/array>).

```
1 #include <iostream>
```

```
2 #include <vector>
```

```
3  
4 int main(){
```

```
5  
6     int size = 50;  
7     int arr[size];
```

```
8  
9     std::cout << arr[0] << std::endl;
```

```
10  
11     arr[0] = 1;  
12     arr[1] = 2;
```

```
13  
14     for (int i = 0; i < size; i++){  
15         | arr[i] = i;  
16     }
```

```
17  
18     std::cout << arr[0] << std::endl;
```

```
19  
20     std::vector<double> vec;
```

```
21  
22     vec.push_back(1); // Appends 1 to the back.  
23     vec.push_back(2);  
24     vec.clear(); // Clears all elements.
```

```
25  
26  
27     for (int j = 0; j < size; j++){  
28         | vec.push_back(j);  
29     }
```

```
30  
31     std::cout << vec.at(0) << std::endl;  
32     std::cout << vec[0] << std::endl;
```

```
33  
34     return 0;
```

```
35 }
```



Structs

- “Structures are aggregate data types”.
 - Another way of saying collection.
- What are some examples of ‘structures’ in real life?
 - A person.
 - Height, name, hair colour, etc.
 - A SNOLAB experiment.
 - Remington activity, etc.
 - Any Brown
- General Blue structure is:


```
struct person {
    string name;
    string hair_colour;
    string eye_colour;
    double height; // Units of cm
    double weight; // Units of kg
};
```
- Do not forget to initialize!
- Let us go to the example.

```
Remington
Brown
Blue
185
99
Samwise the Brave
Dirty blonde
Blue
100
99
```

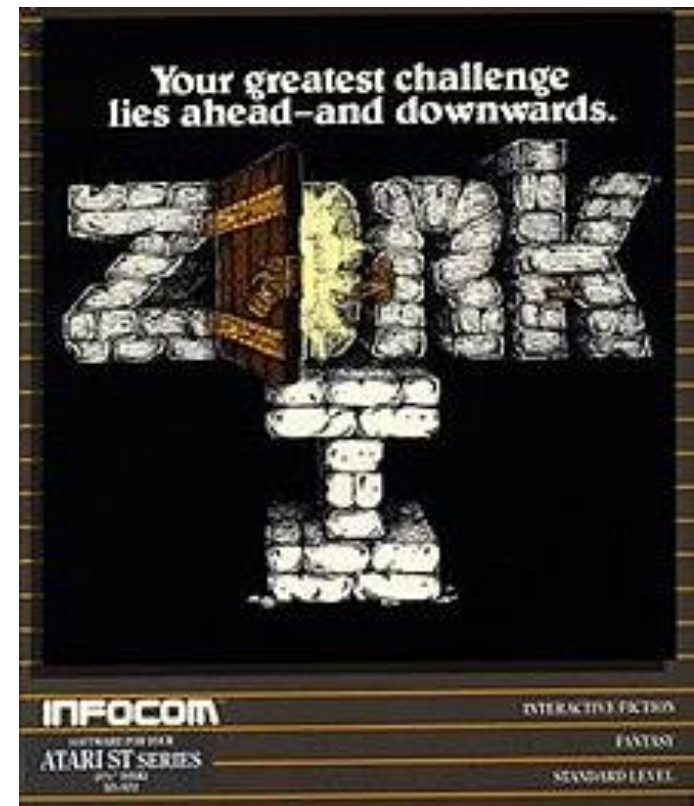
```
1 #include <iostream>
2
3 using namespace std;
4
5 struct person{
6
7     string name = "Remington";
8     string hair_colour = "Brown";
9     string eye_colour = "Blue";
10    double height = "185"; // Units of cm
11    double weight = "99"; // Units of kg
12
13 };
14
15
16 int main(){
17
18     person Remington;
19     cout << Remington.name << endl;
20     cout << Remington.hair_colour << endl;
21     cout << Remington.eye_colour << endl;
22     cout << Remington.height << endl;
23     cout << Remington.weight << endl;
24
25     // I dye my hair dirty blonde, lose 80 cm in height
26     // Also change my name to Samwise the Brave.
27     Remington.name = "Samwise the Brave";
28     Remington.height = 100;
29     Remington.hair_colour = "Dirty blonde";
30
31     cout << Remington.name << endl;
32     cout << Remington.hair_colour << endl;
33     cout << Remington.eye_colour << endl;
34     cout << Remington.height << endl;
35     cout << Remington.weight << endl;
36
37     return 0;
38 }
```

Classes

- Classes are a more complicated struct.
 - Very similar construction.
- They allow you to **privatize** or **publicize** variables and functions.
- Like structs, classes have an optional requirement(s):
 - **Constructor**: allows the user to specify what functions are called, variables are set, etc. when the class object is first created.
 - **Destructor**: allows the user to specify what happens to the class variables at the end of the program.
 - We won't talk about this much here.
- Let us consider an example:
 - Suppose we wanted to create a class that represented each experiment at SNOLAB.
 - What sort of variables, functions, etc. would we need?
- Let us do this together!

Zork project! (30 minutes)

- Ever heard of Zork?
 - Text-based adventure game; released back in the 1970s and 80s.
- Design a program that includes the following:
 - Create a C++ class for the character.
 - Include health, attributes, items.
 - Create a set of narrative conditions that asks the user to choose what to do.
 - Each narrative condition can be represented by a function.
 - Use control statements to change to different functions.
 - Make sure to ask for user input at each step.
 - In the time that you have, try and have at least three prompts; update you character if they take damage, acquire an item, etc.
- I will be walking around and helping.
 - Feel free to work in groups.





Conclusions

- Throughout this talk we have established the **basics** of how to write in C++.
 - We looked at the different data types in C++ and how to perform basic numerical operations on them.
 - We established some of the basics for defining your own functions, while also talking about prototyping and overloading.
 - Arrays and vectors were introduced to give you an introduction on where each might be more appropriate or better to use.
 - We discussed classes and structs that will allow you to interpret more complex C++ code and organize your own analysis work.
- I strongly encourage you to all attend the talk by Philippe next week on ROOT.
 - Between now and then play around with C++ and see if you can begin to write more complex scripts.
 - ROOT is a toolset used by many experiments; it will help to have experience with it.
- If you ever have any questions or comments, feel free to stop by my cubical or reach out via email: remington.hill@queensu.ca.

Thank you for listening! Questions?



Queen's
UNIVERSITY