C++: A Quick Intro

Summer Particle Astrophysics Workshop
(EIEIOO)

Ian Lam

12th May 2022

Arthur B. McDonald
Canadian Astroparticle Physics Research Institute

Carleton University

About me

- Academic journey:
    - B.Math. @ University of Waterloo (2013)
    - Ph.D. @ Queen's University (2020)
    - Postdoc @ Carleton University (since 2021)

Source: https://imgur.com/gallery/aNmXJZu

Goal of this presentation

- Not a complete tutorial on C++.
- Just enough to get you started.
- Share with you key things I learned over the years.
- My setup:
  - Local machine: Windows 10
  - MobaXterm SSH Client
  - Neutrino server @ Queens (thanks Prof. Ryan Martin and Mark Anderson)
  - Editor: Emacs

Intro

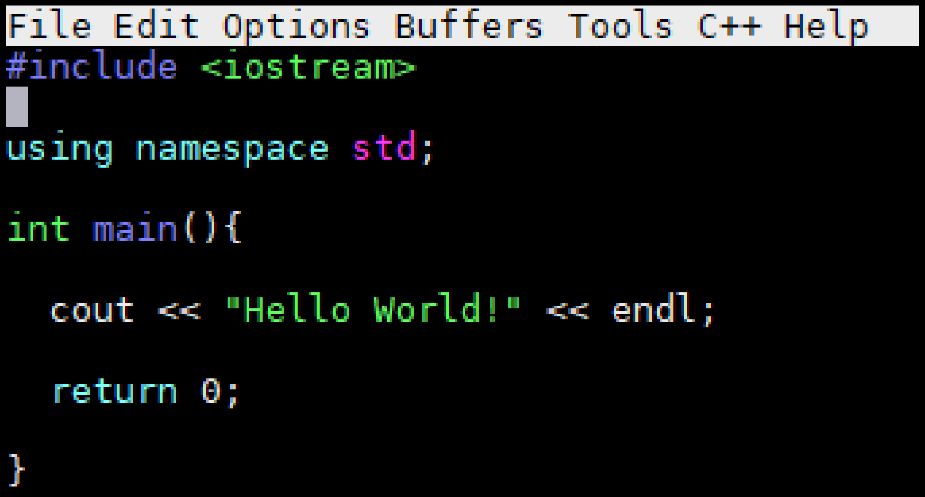
- object-oriented programming language i.e. everything related to classes and objects, along with attribute and methods, similar to Python
- more complicated than Python
  - Python handles many things implicitly like type declarations.
- more syntax to be aware off than Python
  - must declare data type with variables
  - must end statement with semicolon
- uses pointers (more later)

General workflow

1. Write code with human readable alphabets.
2. Compile the code i.e. translate the human readable alphabets into instructions the computer can understand.
3. Compiler will create an executable (.exe) file which you can run.
   - A successful compile does not mean you are 100% problem free. Running the .exe file can reveal more issues.

Hello World

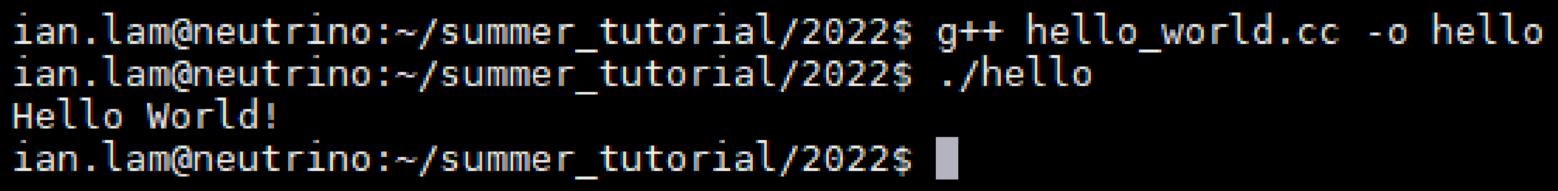
Purpose: Prints “Hello World!” to screen.



```cpp
#include <iostream>

using namespace std;

int main(){

  cout << "Hello World!" << endl;

  return 0;

}
```

Hello World

1. Write the code as shown. Save file as hello_world.cc
2. Compile by doing: g++ hello_world.cc –o hello
   - ‘hello.exe’ file will be created
3. Run by doing: ./hello



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ hello_world.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Hello World!
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

Note: g++ is one of few compilers. ROOT uses a different compiler.

Hello World

#include <iostream>

When writing programs, you might want to reuse functions from other sources. You can think of #include as ‘copy-pasting’ code to where the #include was called. In this case, it is a header file called ‘iostream’ (input-output stream) which contains the function definitions of ‘cout’ and ‘end’

Python:
import library_name

Hello World

using namespace std;

Namespaces can be thought of as a collection of functions, grouped under a larger umbrella (namespace) in order to prevent variable definition conflicts.

In this case, we are using the ‘std’ or ‘standard’ namespace. ‘cout’ and ‘endl’ functionalities are defined in here.

You can create your own namespaces but I won’t cover it here. I personally never found it necessary.

Hello World

int main(){…}

Declare a function called ‘main’ that returns a type int. ‘main’ is a special name, which the compiler looks for specifically and executes the functions called in ‘main’ in order.

Hello World

cout << “Hello World!” << endl;

cout prints whatever follows after << to the screen. It is equivalent to Python’s print() function. You can cout variables as well.

endl means ‘end line’; creates a new line.

<< : stream operator

Note: This is my go-to method to debug code. If your code is throwing errors and you want to systematically work through your code, just ‘cout’ stuff at various lines and see where the code fails or returns something that does not makes sense.

Hello World

```
console.log("im still working");
console.log("asdjoauhefefefAsda");
console.log("peanut");
```

Hello World - Advanced

- I’ll now make some modifications to the hello world program to illustrate some common concepts.

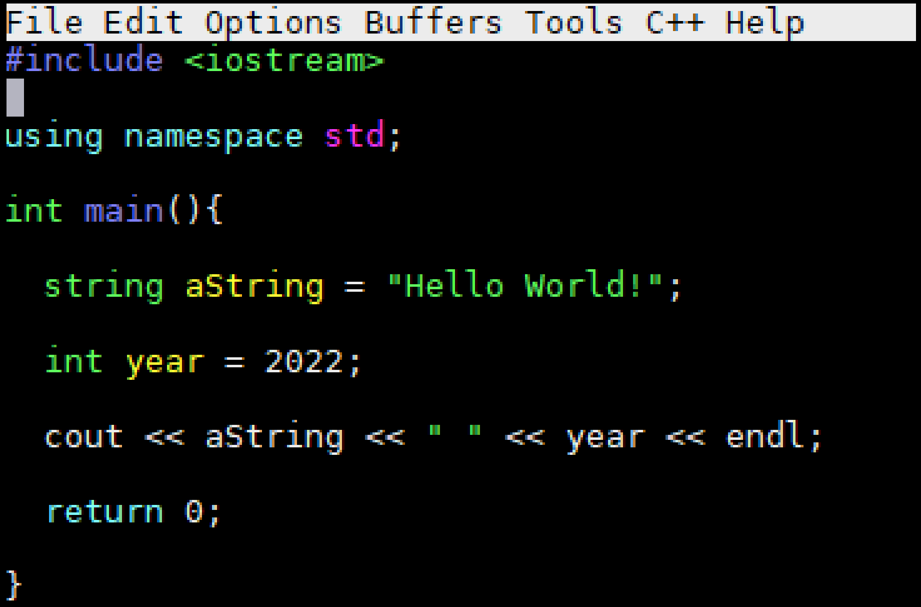
Topic
Declare Variables
Declare Functions
'for' loop
'if' statement
'if…else' statement
'break'
Variable Scope
Global Variable
More C++

Included some exercises at certain points to get you to explore on your own. (Not for course credit unfortunately ☹)

Quick link table. Click to jump to relevant slides.

Hello World – declare variables

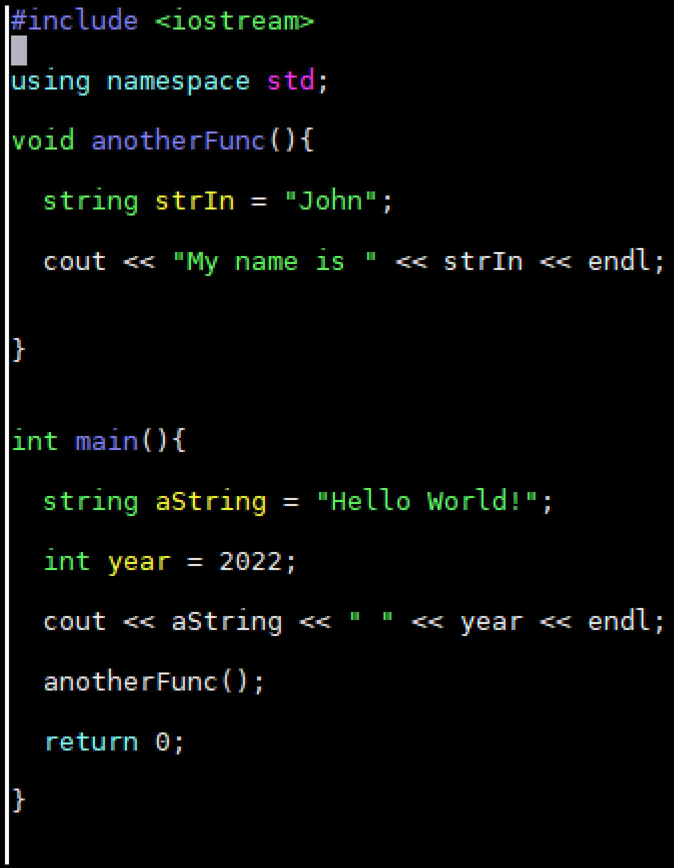
- Modify slightly to see how to declare variables.



```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  cout << aString << " " << year << endl;

  return 0;

}
```

Note: the ‘string’ datatype is under the ‘std’ namespace. If you do not use ‘using namespace std’, you’ll have to declare string variables with ‘std::string aString = “Hello World!” ;

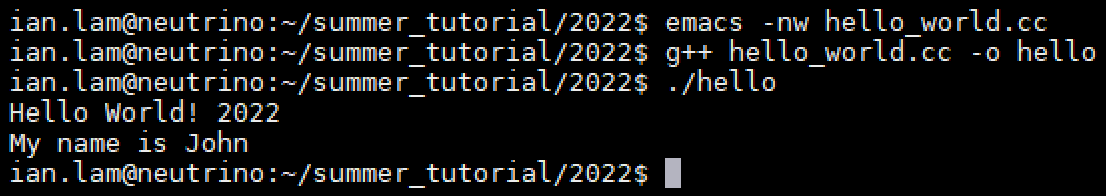
Hello World – declare functions



```
#include <iostream>

using namespace std;

void anotherFunc(){

  string strIn = "John";

  cout << "My name is " << strIn << endl;


}


int main(){

  string aString = "Hello World!";

  int year = 2022;

  cout << aString << " " << year << endl;

  anotherFunc();

  return 0;

}
```

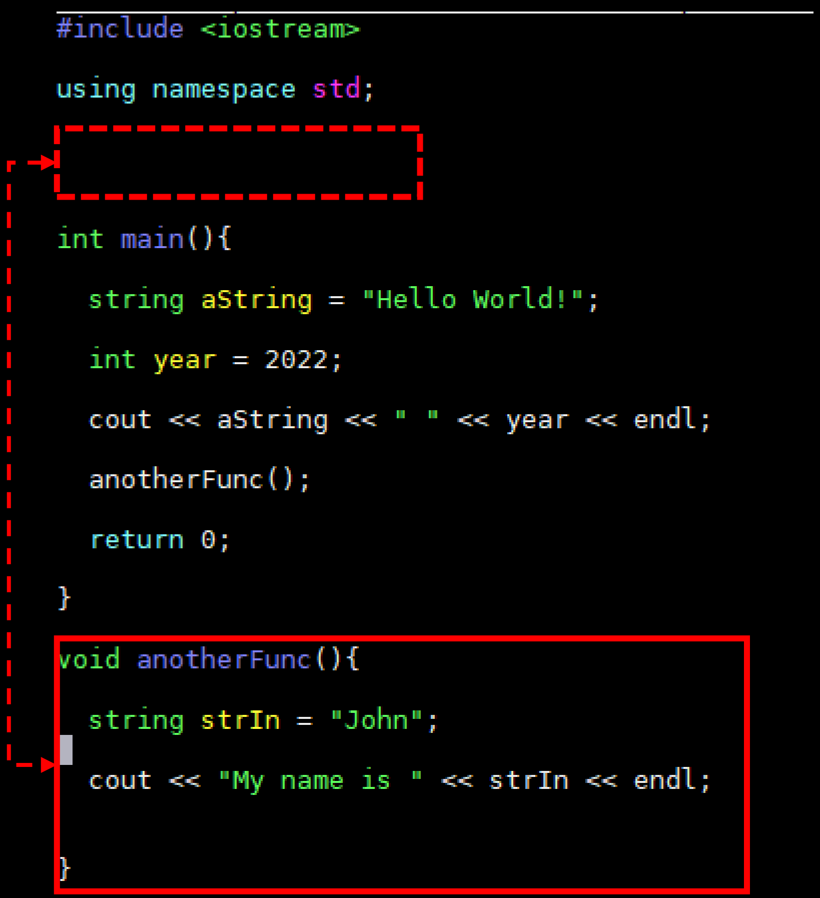
Python:
def anotherFunc():

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ emacs -nw hello_world.cc
ian.lam@neutrino:~/summer_tutorial/2022$ g++ hello_world.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Hello World! 2022
My name is John
ian.lam@neutrino:~/summer_tutorial/2022$ █
```

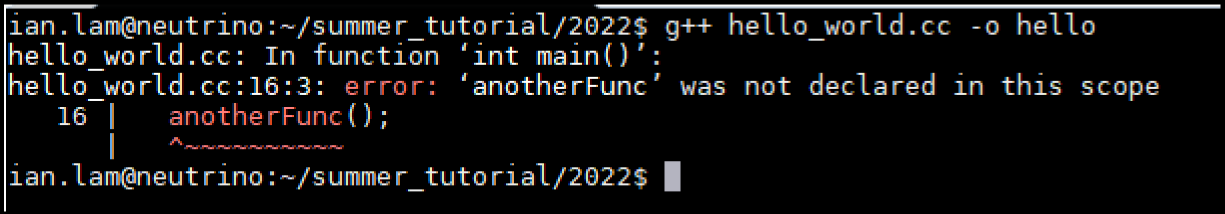
Hello World – declare functions



```
#include <iostream>

using namespace std;



int main(){

  string aString = "Hello World!";

  int year = 2022;

  cout << aString << " " << year << endl;

  anotherFunc();

  return 0;

}

void anotherFunc(){

  string strIn = "John";

  cout << "My name is " << strIn << endl;

}
```

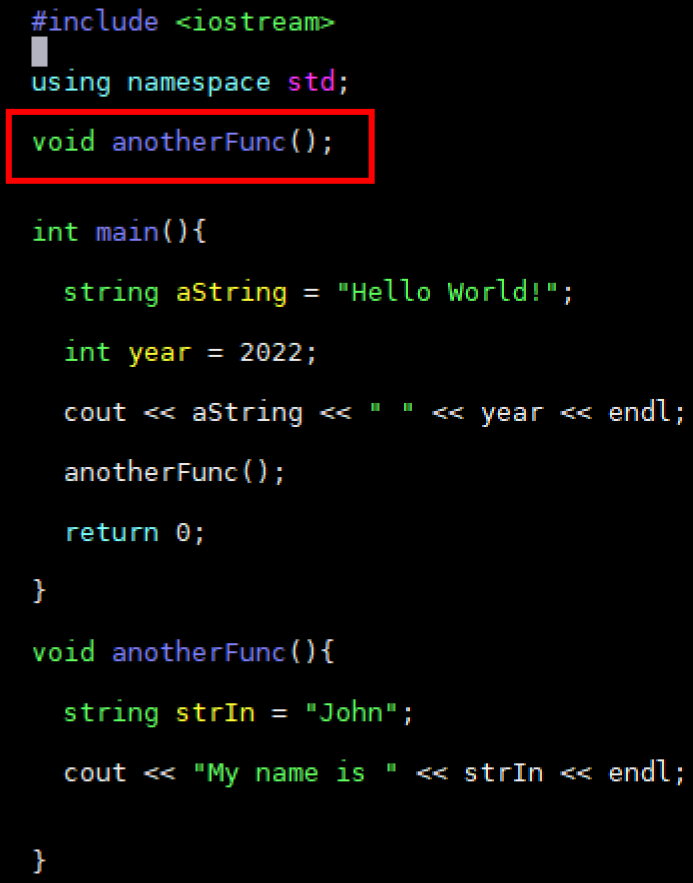
- Order of declaration and calling matters.
- What if we swapped the function to come after main?

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ hello_world.cc -o hello
hello_world.cc: In function ‘int main()’:
hello_world.cc:16:3: error: ‘anotherFunc’ was not declared in this scope
   16 |   anotherFunc();
      |   ^~~~~~~~~~~
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

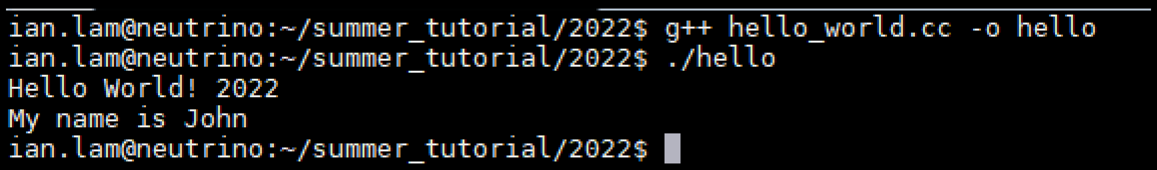
Hello World – declare functions



```
#include <iostream>

using namespace std;

void anotherFunc();

int main(){
  string aString = "Hello World!";
  int year = 2022;
  cout << aString << " " << year << endl;
  anotherFunc();
  return 0;
}
void anotherFunc(){
  string strIn = "John";
  cout << "My name is " << strIn << endl;

}
```

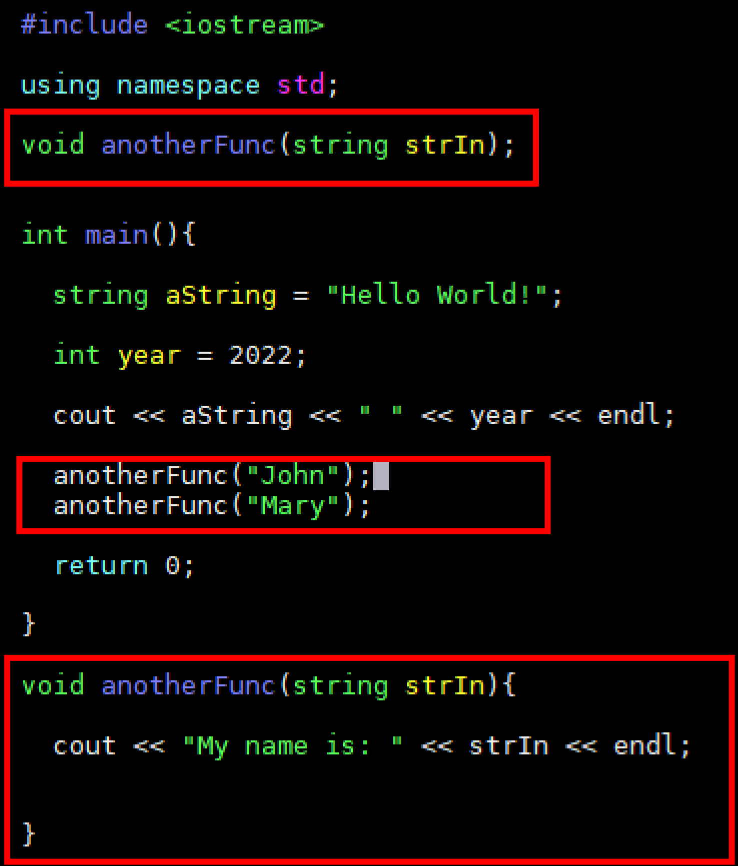
- Aesthetically, you don’t want to scroll all the way down your code to search for ‘main’.
- Can declare the function first and define it later.
- Let the compiler know that such a function exists so it won’t throw a fit.

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ hello_world.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Hello World! 2022
My name is John
ian.lam@neutrino:~/summer_tutorial/2022$
```

Hello World – declare functions

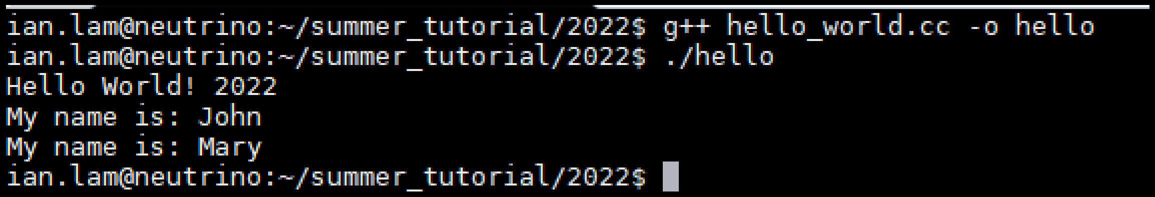


```
#include <iostream>

using namespace std;

void anotherFunc(string strIn);

int main(){

  string aString = "Hello World!";

  int year = 2022;

  cout << aString << " " << year << endl;

  anotherFunc("John");
  anotherFunc("Mary");

  return 0;

}

void anotherFunc(string strIn){

  cout << "My name is: " << strIn << endl;


}
```

Functions can take in inputs/arguments/parameters.

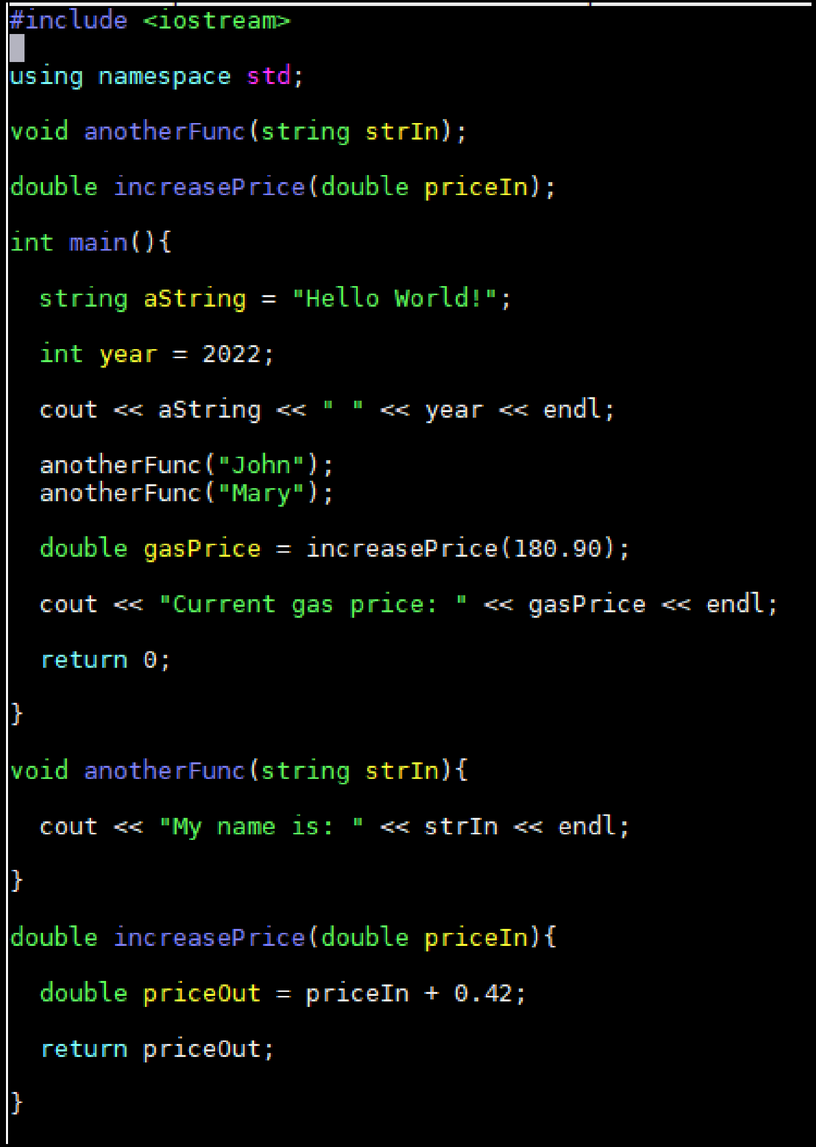
1) Don’t have to hardcode.
2) Can reuse same function with multiple inputs.

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ hello_world.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Hello World! 2022
My name is: John
My name is: Mary
ian.lam@neutrino:~/summer_tutorial/2022$
```

Hello World – declare functions



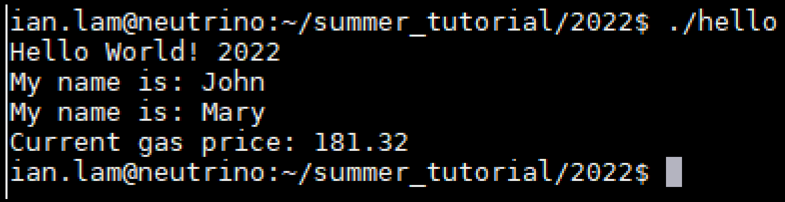
```
#include <iostream>

using namespace std;

void anotherFunc(string strIn);

double increasePrice(double priceIn);

int main(){

  string aString = "Hello World!";

  int year = 2022;

  cout << aString << " " << year << endl;

  anotherFunc("John");
  anotherFunc("Mary");

  double gasPrice = increasePrice(180.90);

  cout << "Current gas price: " << gasPrice << endl;

  return 0;

}

void anotherFunc(string strIn){

  cout << "My name is: " << strIn << endl;

}

double increasePrice(double priceIn){

  double priceOut = priceIn + 0.42;

  return priceOut;

}
```

Functions can return a value.
Need to specify the type and make sure it is consistent throughout. If it isn't, you'll know when you compile and run.

Notice the function declarations neatly grouped at the top. This can be collected into a header file (shown later in slide 49).

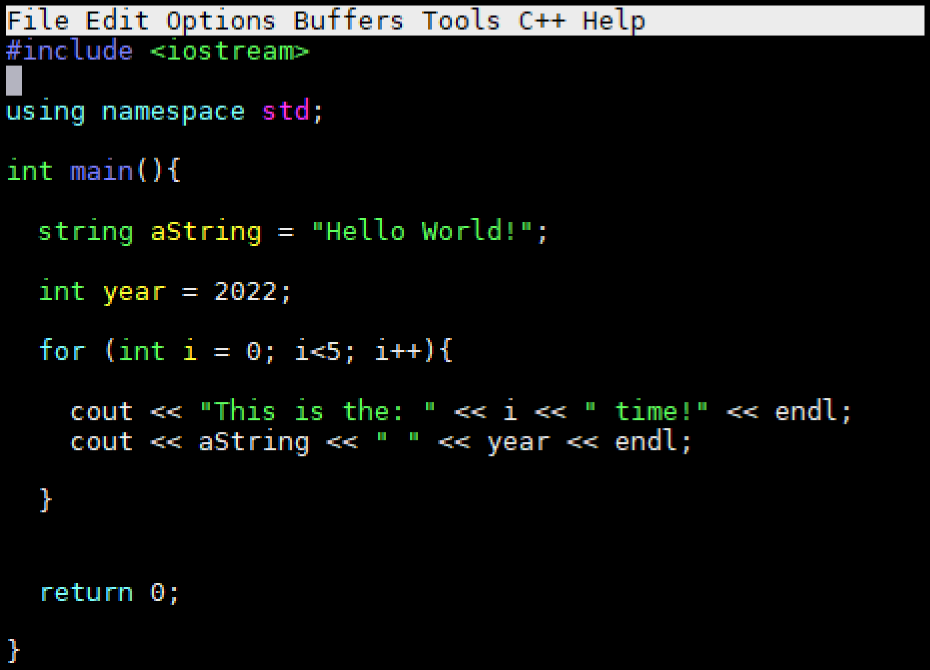
Exercise:
For the function increasePrice, rewrite it such that 0.42 is passed as an argument.
Hint: increasePrice(double priceIn, double argIn){…}

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Hello World! 2022
My name is: John
My name is: Mary
Current gas price: 181.32
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

Hello World – for loop



```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

Statement 1 (int i=0): Declare variable and start with 0, executed once at start.

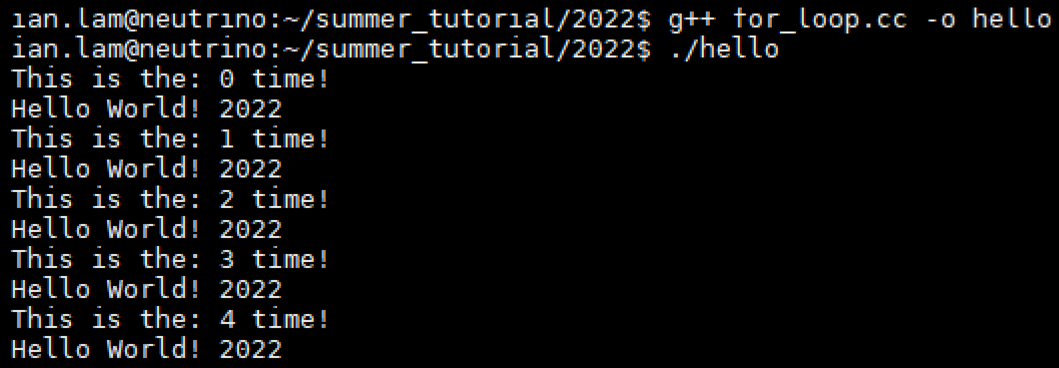
Statement 2 (i<5): Boolean condition. If True, continue execution of code block.

Statement 3 (i++): executed every time after code block has been executed.

Notes:

- Doesn't have to be 'i'. Can be any variable name.
- i++ means 'increment 'i' by 1 but return value of 'i' before incrementation.
  In for loop implementation like this, it is similar to i = i + 1, or i+=1

Output

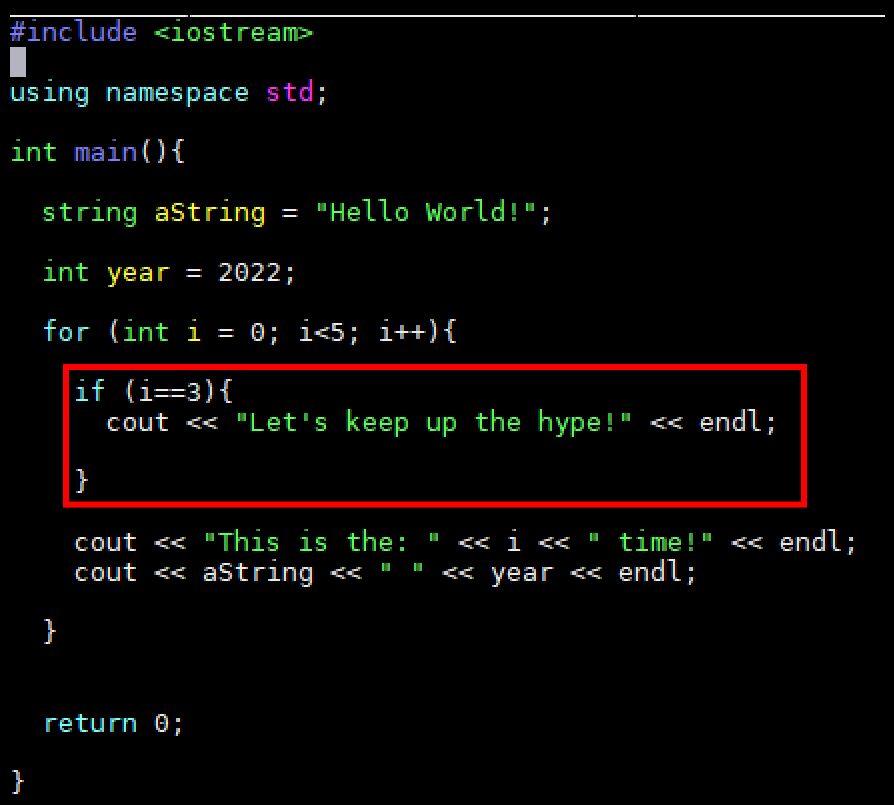


```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
This is the: 0 time!
Hello World! 2022
This is the: 1 time!
Hello World! 2022
This is the: 2 time!
Hello World! 2022
This is the: 3 time!
Hello World! 2022
This is the: 4 time!
Hello World! 2022
```

Python equivalent:
for i in range (0,5)

Exercise:
- Increment more than one step.
- Step backwards instead of forward.
- What if you did i<=5 ?
- Increment in steps of 0.1 (hint: change i from int to double)

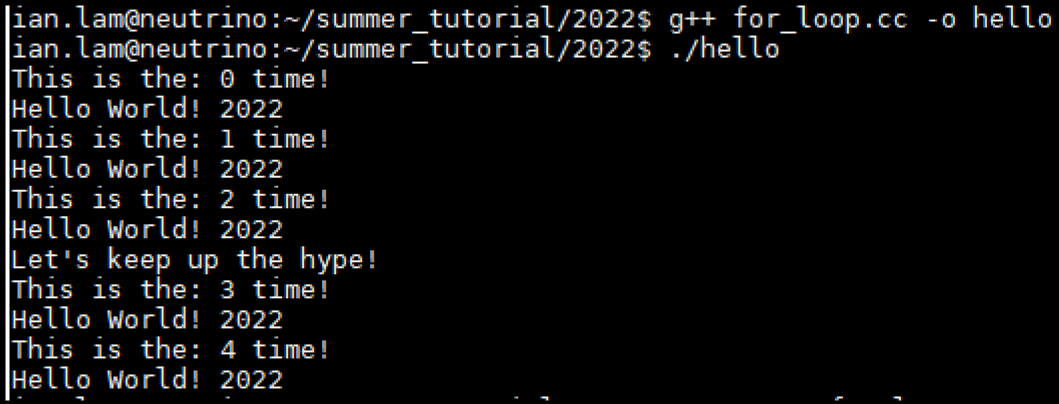
Hello World – if statement



```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

i==3 : Boolean statement can be read as “variable i is equal to 3”

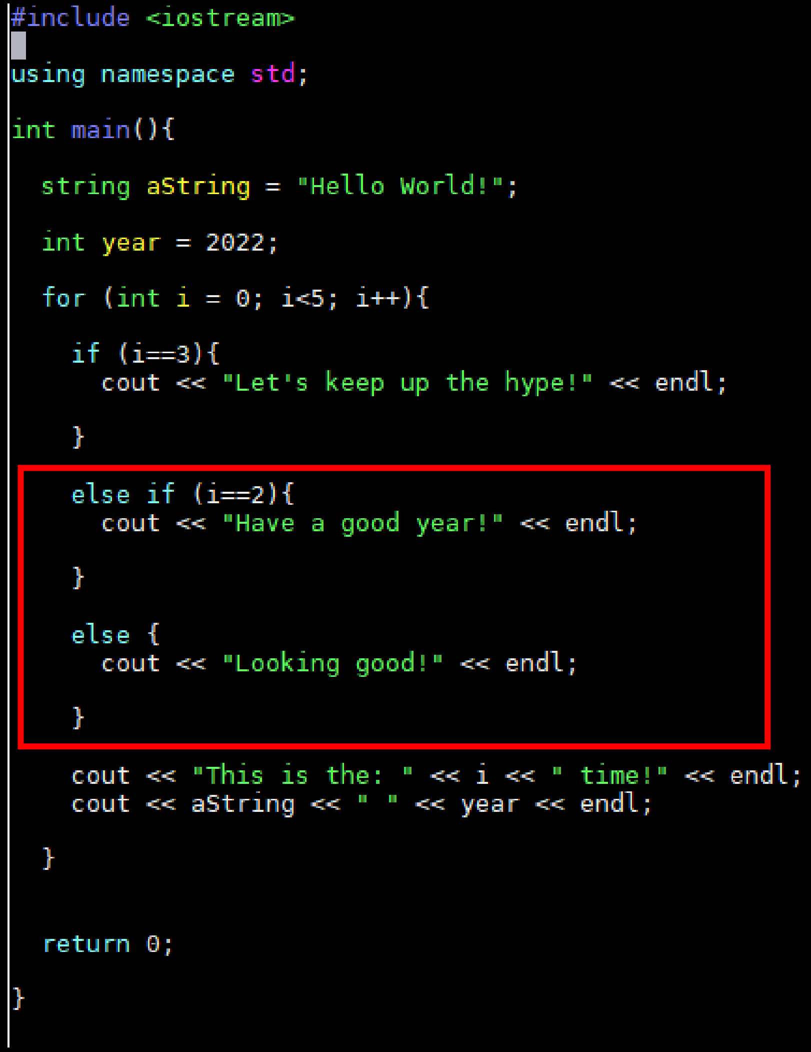
Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
This is the: 0 time!
Hello World! 2022
This is the: 1 time!
Hello World! 2022
This is the: 2 time!
Hello World! 2022
Let's keep up the hype!
This is the: 3 time!
Hello World! 2022
This is the: 4 time!
Hello World! 2022
```

Exercise:
Try other conditionals like i<=3 , i>=3, i!=3 (i not equal to 3).

Hello World – if…else



```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

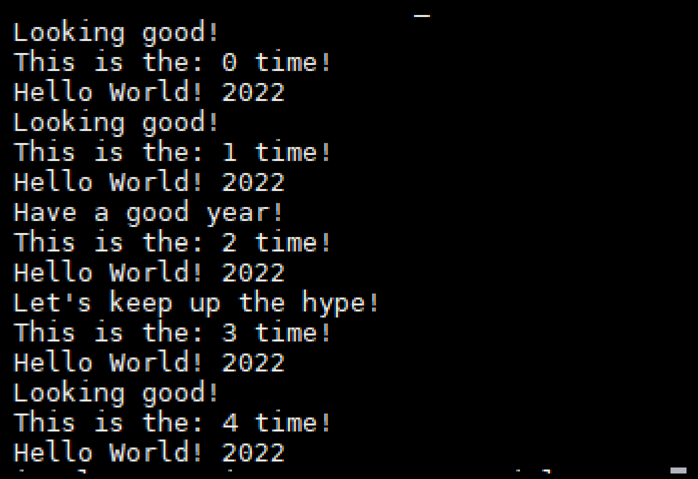
else if : check this if above conditional is false.

else: evaluate this if all above conditionals are false.

Note: not necessary to have ‘else if’, ‘else’ for code to run.

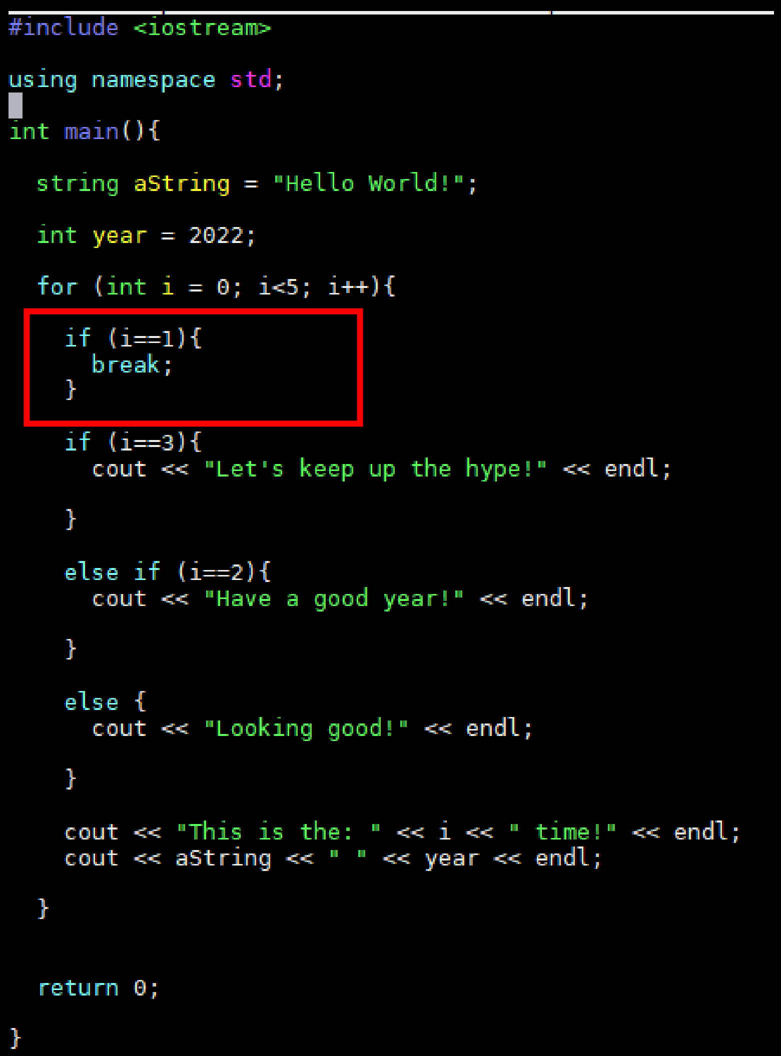
If want to check multiple conditionals, could also use multiple ‘if’ statements.

Output



```
Looking good!
This is the: 0 time!
Hello World! 2022
Looking good!
This is the: 1 time!
Hello World! 2022
Have a good year!
This is the: 2 time!
Hello World! 2022
Let's keep up the hype!
This is the: 3 time!
Hello World! 2022
Looking good!
This is the: 4 time!
Hello World! 2022
```

Hello World - break



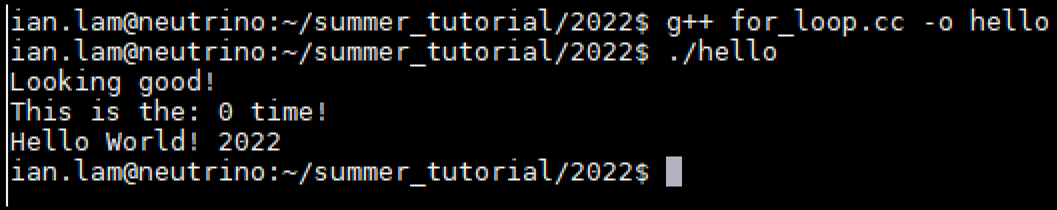
```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==1){
      break;
    }

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

Stops code block execution.
Exits the loop.

Notice that this if statement has no accompanying else.

Note: Code executes sequentially so order matters!

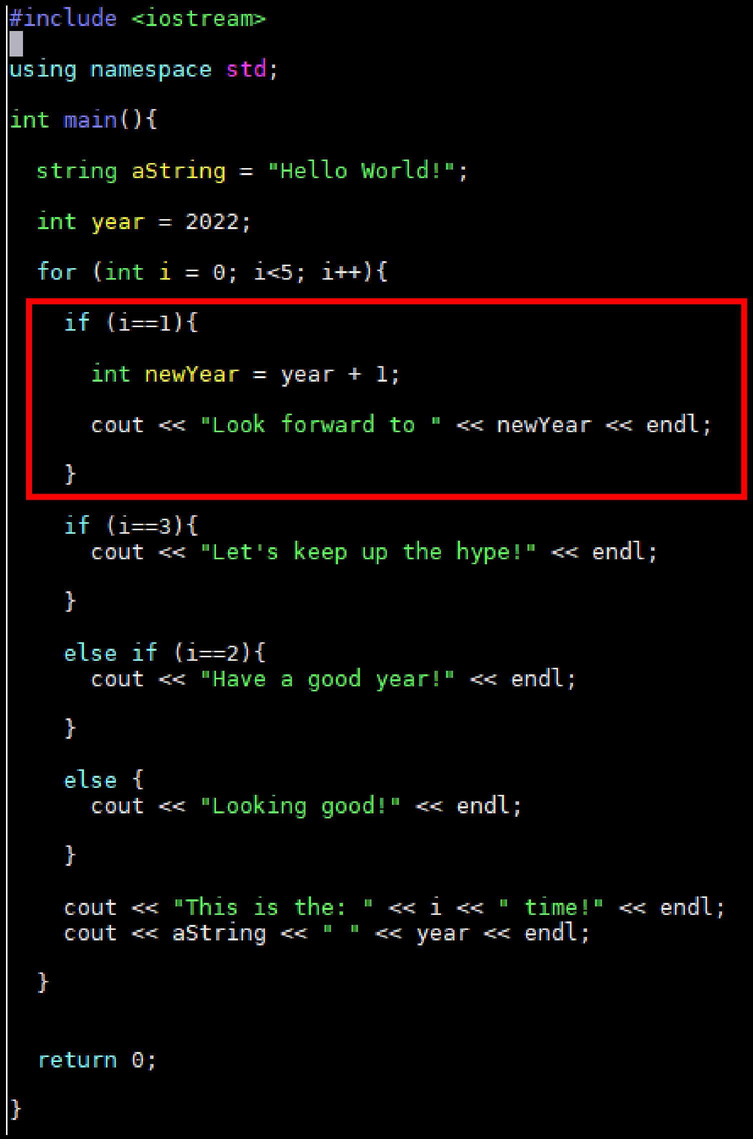
Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Looking good!
This is the: 0 time!
Hello World! 2022
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

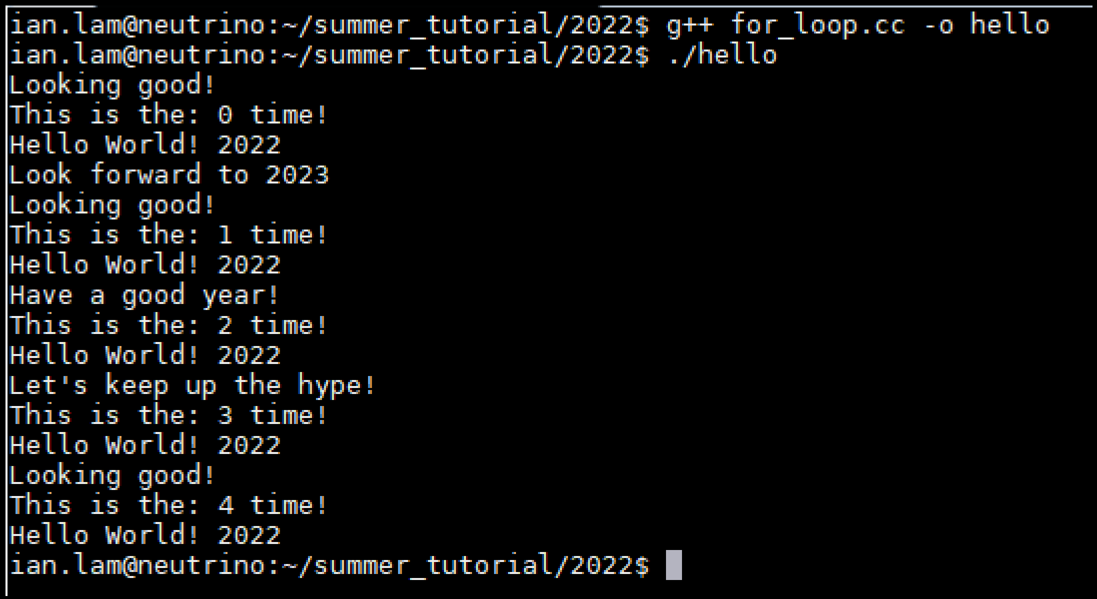
Exercise:
Print an exit message (eg: “Bye bye”) before the break.
Place the break code block further down the code. Is the effect what you expect?

Hello World – variable scope



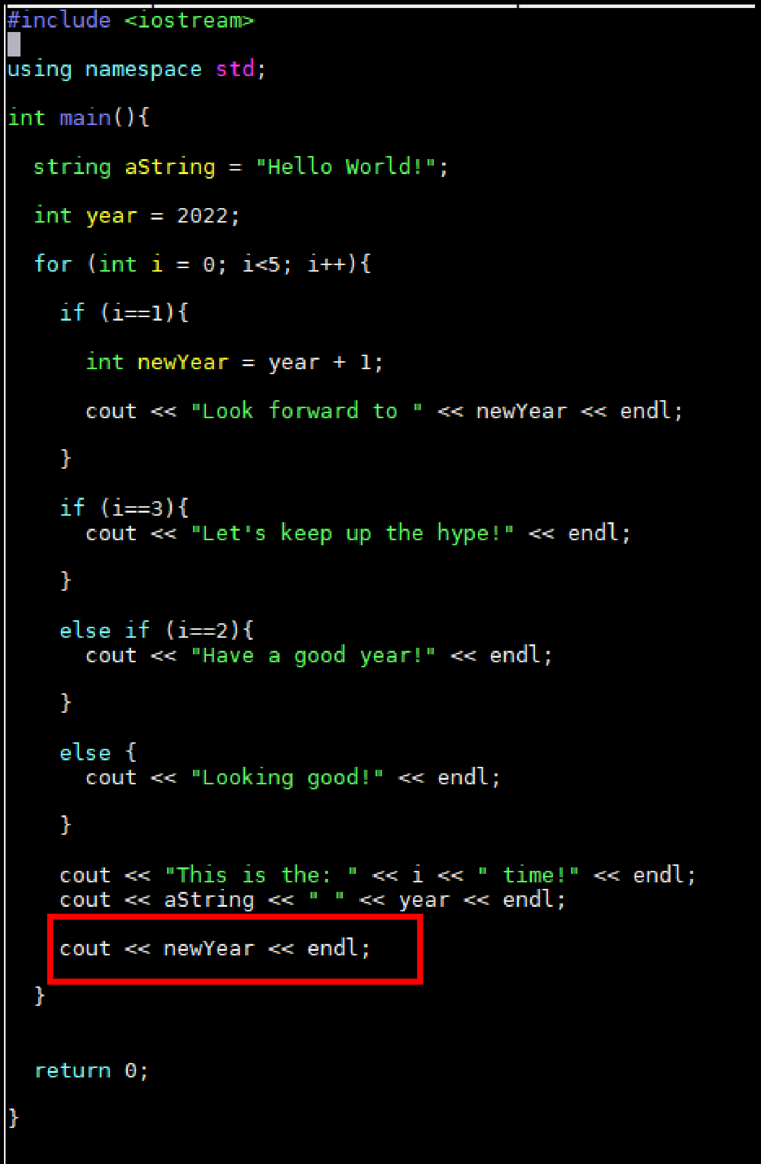
```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==1){

      int newYear = year + 1;

      cout << "Look forward to " << newYear << endl;

    }

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

Output



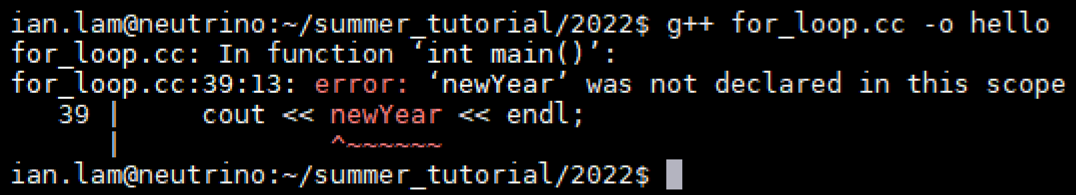
```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Looking good!
This is the: 0 time!
Hello World! 2022
Look forward to 2023
Looking good!
This is the: 1 time!
Hello World! 2022
Have a good year!
This is the: 2 time!
Hello World! 2022
Let's keep up the hype!
This is the: 3 time!
Hello World! 2022
Looking good!
This is the: 4 time!
Hello World! 2022
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

Hello World – variable scope



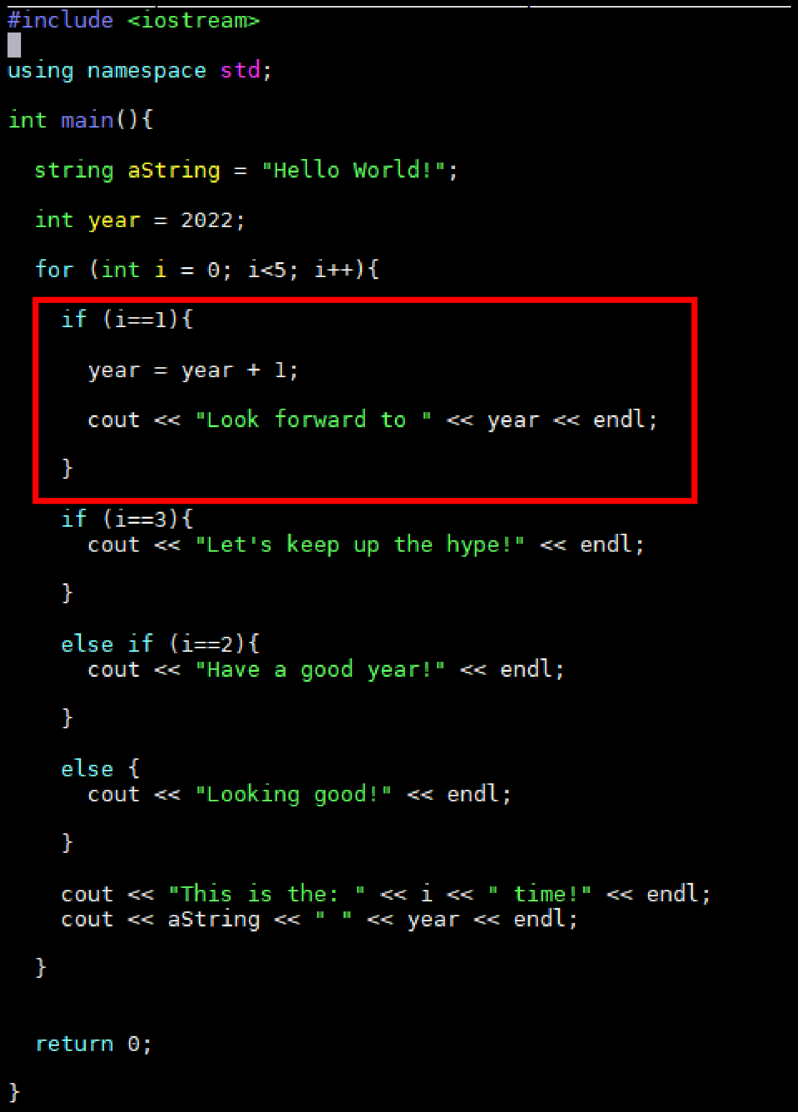
```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==1){

      int newYear = year + 1;

      cout << "Look forward to " << newYear << endl;

    }

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

    cout << newYear << endl;

  }


  return 0;

}
```

Output



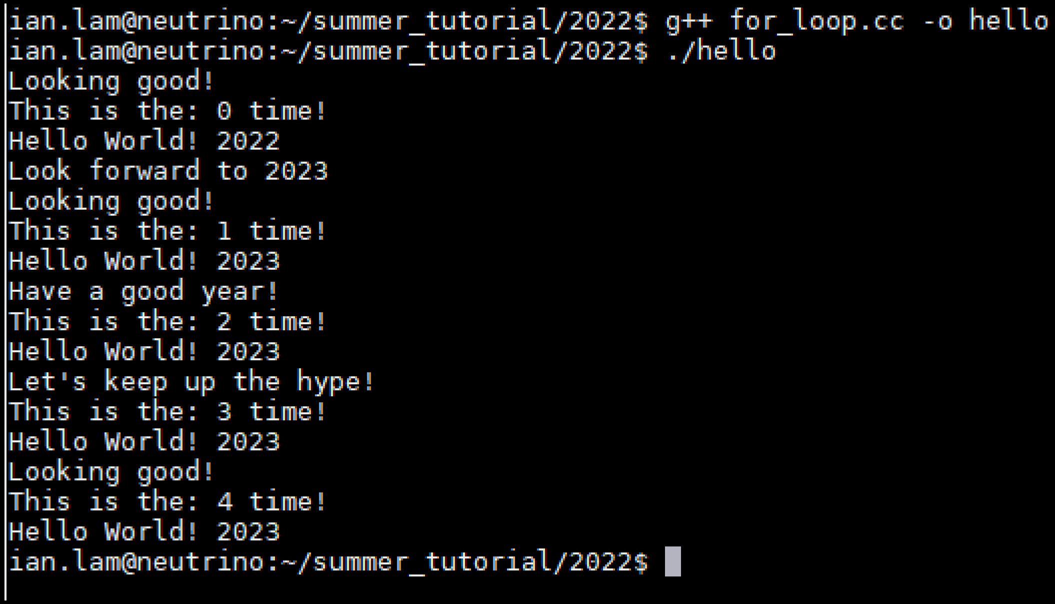
```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
for_loop.cc: In function ‘int main()’:
for_loop.cc:39:13: error: ‘newYear’ was not declared in this scope
   39 |     cout << newYear << endl;
      |             ^~~~~~~
ian.lam@neutrino:~/summer_tutorial/2022$
```

Hello World – variable scope



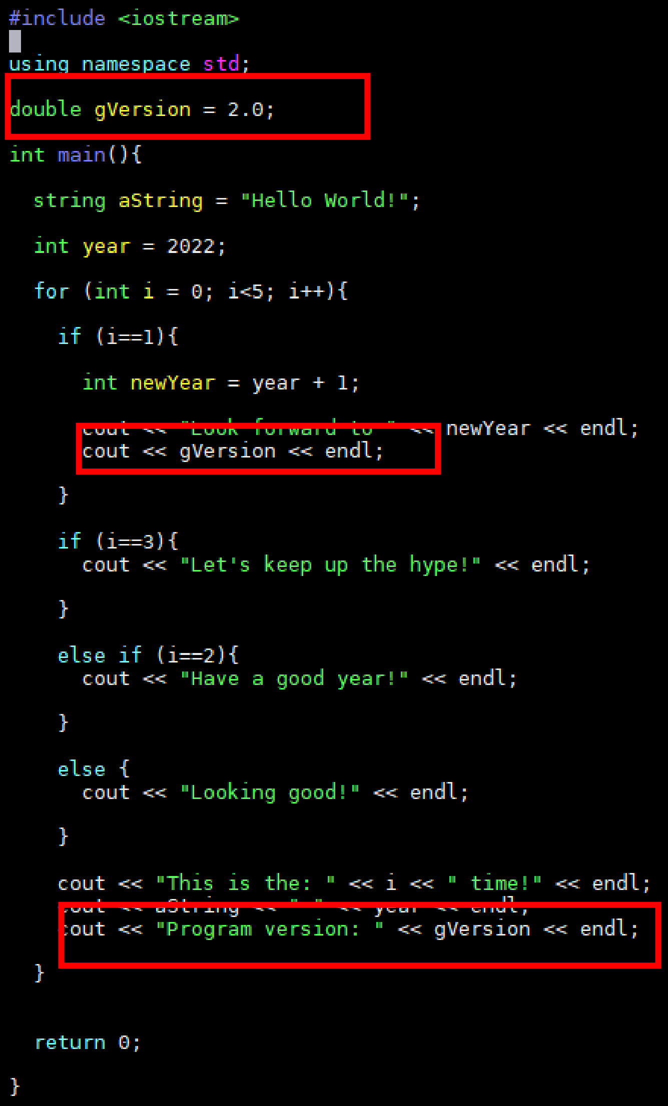
```
#include <iostream>

using namespace std;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==1){

      year = year + 1;

      cout << "Look forward to " << year << endl;

    }
    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;

  }


  return 0;

}
```

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Looking good!
This is the: 0 time!
Hello World! 2022
Look forward to 2023
Looking good!
This is the: 1 time!
Hello World! 2023
Have a good year!
This is the: 2 time!
Hello World! 2023
Let's keep up the hype!
This is the: 3 time!
Hello World! 2023
Looking good!
This is the: 4 time!
Hello World! 2023
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

Hello World – global variable



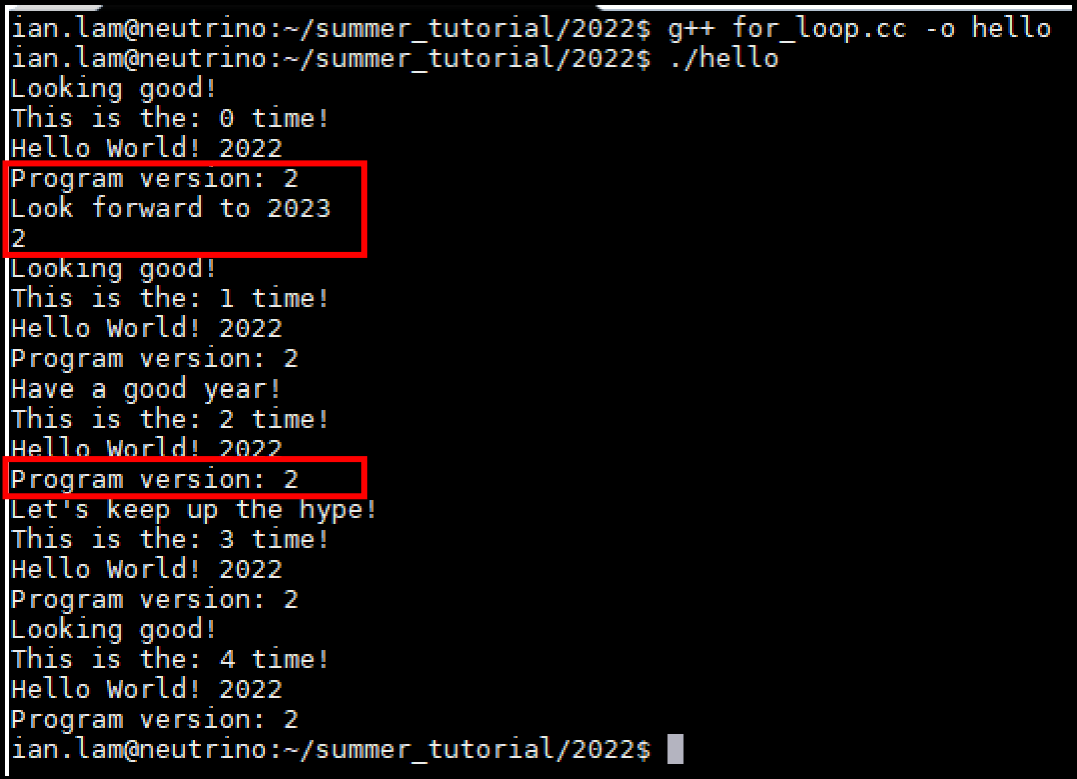
```
#include <iostream>

using namespace std;

double gVersion = 2.0;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    if (i==1){

      int newYear = year + 1;

      cout << "Look forward to " << newYear << endl;
      cout << gVersion << endl;

    }

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;
    cout << "Program version: " << gVersion << endl;

  }


  return 0;

}
```

Global variables, as the name suggests, can be accessed by all functions everywhere in the code.

Usually prefixed with lowercase ‘g’.

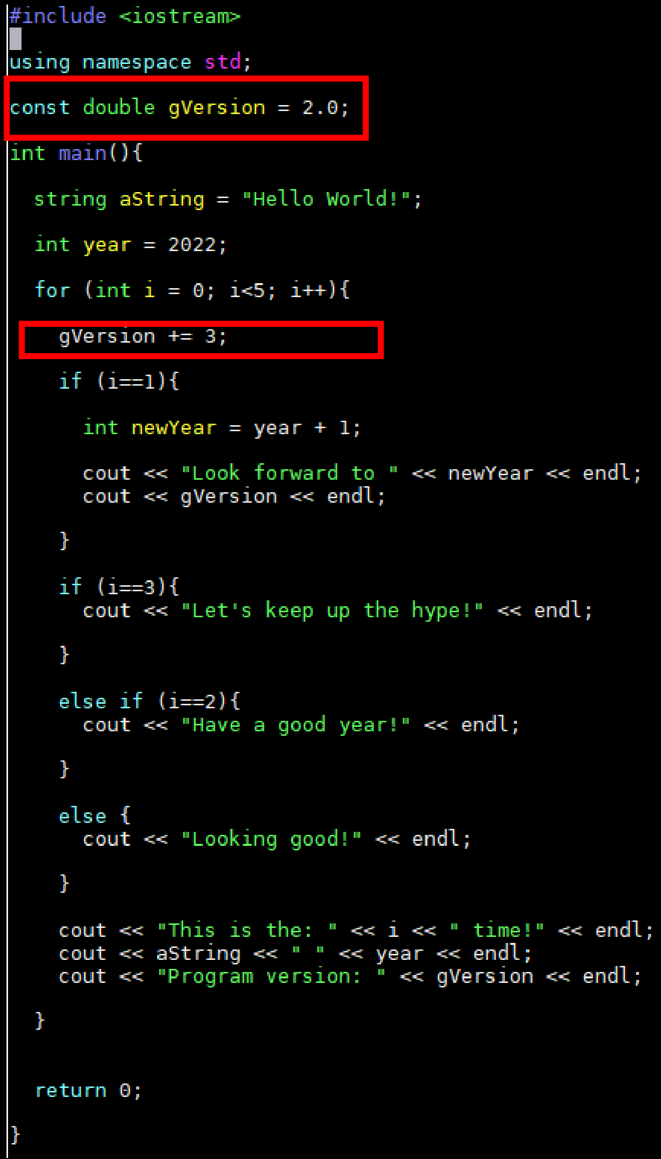
WARNING: Be careful when using global variables. Their global scope means that any modification could break other functions.

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
ian.lam@neutrino:~/summer_tutorial/2022$ ./hello
Looking good!
This is the: 0 time!
Hello World! 2022
Program version: 2
Look forward to 2023
2
Looking good!
This is the: 1 time!
Hello World! 2022
Program version: 2
Have a good year!
This is the: 2 time!
Hello World! 2022
Program version: 2
Let's keep up the hype!
This is the: 3 time!
Hello World! 2022
Program version: 2
Looking good!
This is the: 4 time!
Hello World! 2022
Program version: 2
ian.lam@neutrino:~/summer_tutorial/2022$
```

Hello World - const

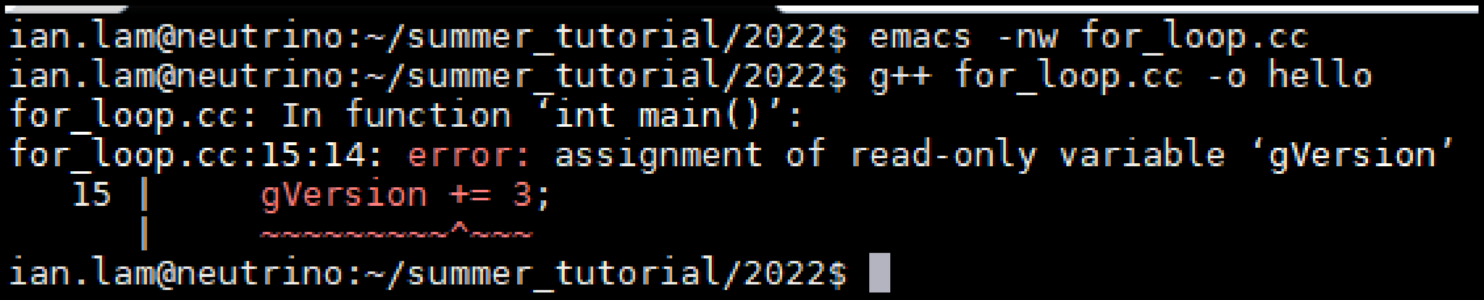


```
#include <iostream>

using namespace std;

const double gVersion = 2.0;

int main(){

  string aString = "Hello World!";

  int year = 2022;

  for (int i = 0; i<5; i++){

    gVersion += 3;

    if (i==1){

      int newYear = year + 1;

      cout << "Look forward to " << newYear << endl;
      cout << gVersion << endl;

    }

    if (i==3){
      cout << "Let's keep up the hype!" << endl;

    }

    else if (i==2){
      cout << "Have a good year!" << endl;

    }

    else {
      cout << "Looking good!" << endl;

    }

    cout << "This is the: " << i << " time!" << endl;
    cout << aString << " " << year << endl;
    cout << "Program version: " << gVersion << endl;

  }


  return 0;

}
```

Prevents modification to variable.

Global variables + const is useful to hardcode physical constants.

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ emacs -nw for_loop.cc
ian.lam@neutrino:~/summer_tutorial/2022$ g++ for_loop.cc -o hello
for_loop.cc: In function ‘int main()’:
for_loop.cc:15:14: error: assignment of read-only variable ‘gVersion’
   15 |     gVersion += 3;
      |     ~~~~~~~~~^~~~
ian.lam@neutrino:~/summer_tutorial/2022$ 
```

Summary

- Have shown the most common methods.
- This is a non-exhaustive list but should be sufficient to get started.

More C++

- We’ve seen various terminologies like ‘classes’ and ‘headers’.
- Probably a key phrase tossed around with regards to C++ is ‘pointers’.
- Don’t worry if you don’t grasp it completely now.

Topic
Classes
Headers
Pointers
ROOT tie-in

Quick link table

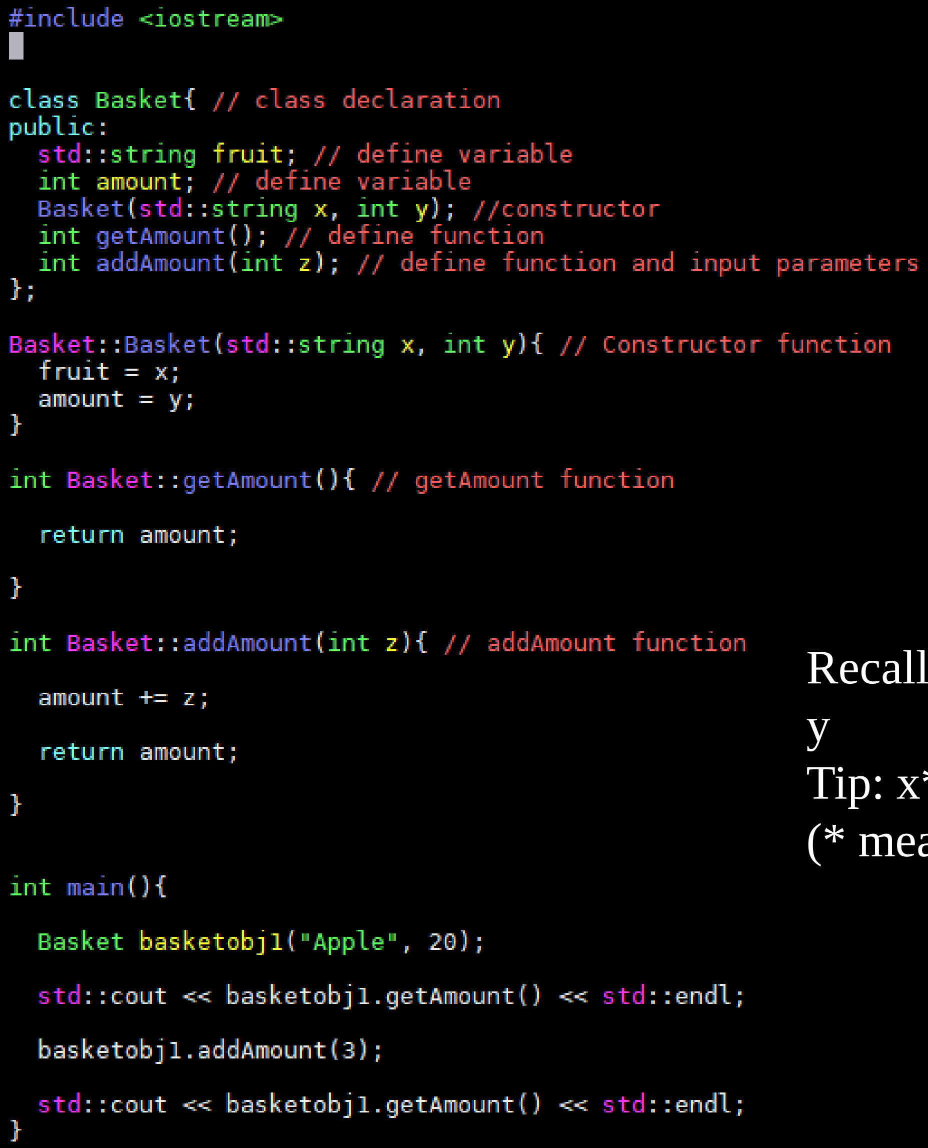
Classes

- As mentioned, C++ is an object-oriented programming language, similar to Python.
- Classes can be thought of as defining a set of properties and behavior/functions an object should have.
- Object is a specific instance created from a class.
    - Depending on the class of the object, properties and functions of the object will differ.

Classes: Baskets

- Say you have a fruit shop with fruits in baskets. Some properties of the baskets could be ‘name of fruit’, ‘amount’, ‘color’.
- Some functions you would like to perform on a basket could be ‘count amount of fruit’, ‘add a fruit’, ‘remove a fruit’.
- You could construct a class called Basket.
- You can then create an instance of a Basket (object), such as 20 apples. A function would be ‘add 3 more green apples to the basket’.
- Actually, lets make this class!

Classes: Baskets



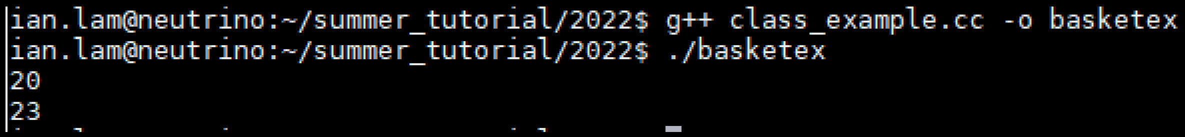
```
#include <iostream>

class Basket{ // class declaration
public:
  std::string fruit; // define variable
  int amount; // define variable
  Basket(std::string x, int y); //constructor
  int getAmount(); // define function
  int addAmount(int z); // define function and input parameters
};

Basket::Basket(std::string x, int y){ // Constructor function
  fruit = x;
  amount = y;
}

int Basket::getAmount(){ // getAmount function

  return amount;

}

int Basket::addAmount(int z){ // addAmount function

  amount += z;

  return amount;

}


int main(){

  Basket basketobj1("Apple", 20);

  std::cout << basketobj1.getAmount() << std::endl;

  basketobj1.addAmount(3);

  std::cout << basketobj1.getAmount() << std::endl;
}
```

Constructor: Initializes the object. ‘public’ : *access specifier.* Tells how members of the class can be accessed. ‘public’ means accessible by everyone. ‘private’ means accessible only to members of the class.

Recall: x += y is equivalent to x = x + y
Tip: x*=y is equal to x = x*y
(* means multiply here)

Note: “ // ” is for commenting.

Output



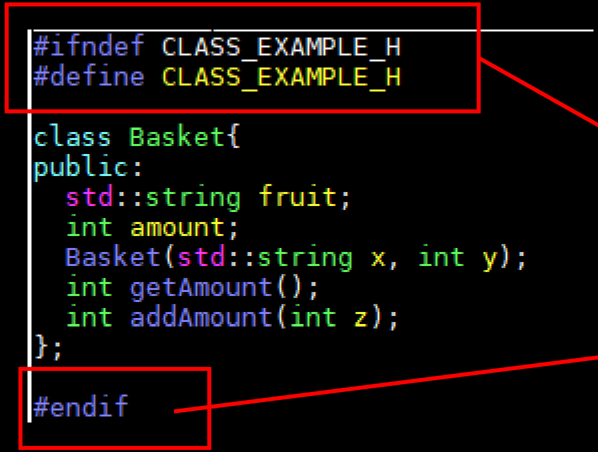
```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ class_example.cc -o basketex
ian.lam@neutrino:~/summer_tutorial/2022$ ./basketex
20
23
```

Exercise:
Change ‘public’ to ‘private’. Did anything change? Why or why not?
Explore the ‘private’ access specifier by creating one and try to access it in various parts of your code.

Headers

- In all our examples so far, the ‘main’ function is in the same file as our classes and function definitions.
- Alright for relatively simple code but can become messy as code grows in length and complexity.
- Also, would be good to re-use code without explicitly copy-pasting.
- Can do this with headers.

basket.h

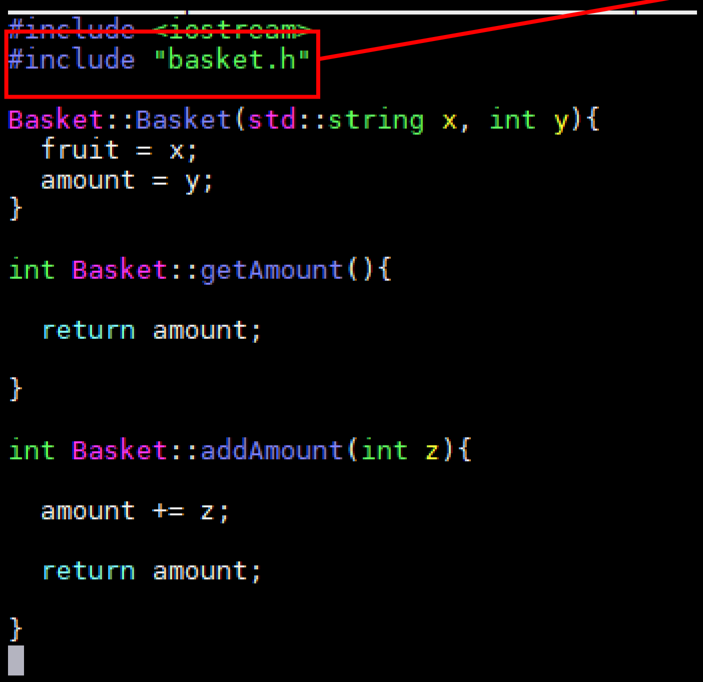


```
#ifndef CLASS_EXAMPLE_H
#define CLASS_EXAMPLE_H

class Basket{
public:
  std::string fruit;
  int amount;
  Basket(std::string x, int y);
  int getAmount();
  int addAmount(int z);
};

#endif
```

This is called the ‘header’ file.

Header guard. Prevents the same header file to be added twice during compile. ~~Not required for compilation but highly recommended, especially for complicated code.~~ Just do it.

basket.cc

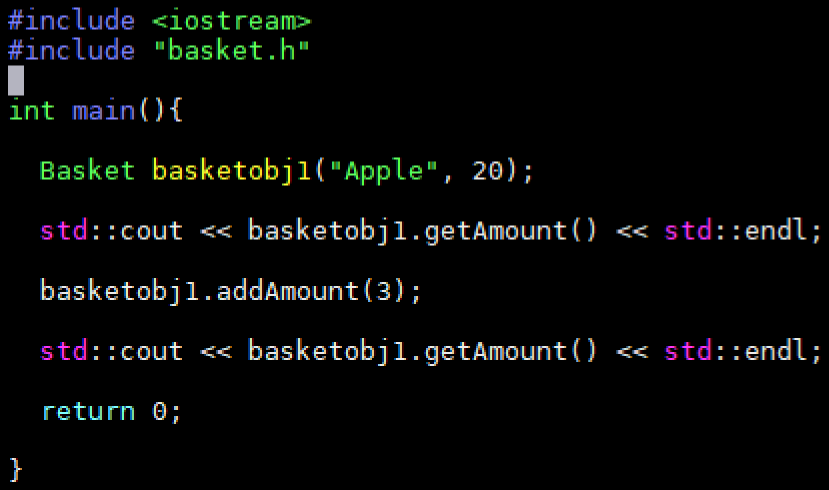


```
#include <iostream>
#include "basket.h"

Basket::Basket(std::string x, int y){
  fruit = x;
  amount = y;
}

int Basket::getAmount(){

  return amount;

}

int Basket::addAmount(int z){

  amount += z;

  return amount;

}
```

Recall that #include is basically copy-pasting the contents of basket.h here.

Necessary. Otherwise, the compiler won't know the definitions of the functions here. Recall slide 17 .

basketsample.cc



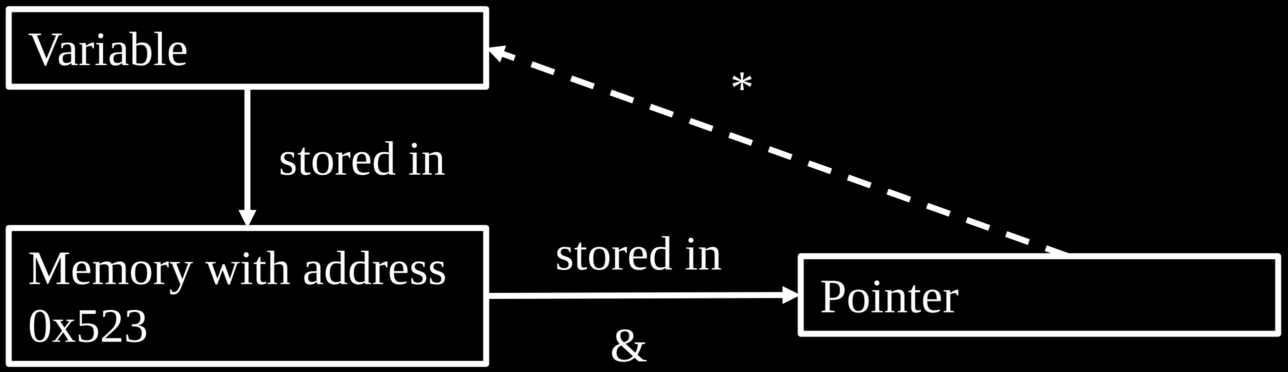
```
#include <iostream>
#include "basket.h"

int main(){

  Basket basketobj1("Apple", 20);

  std::cout << basketobj1.getAmount() << std::endl;

  basketobj1.addAmount(3);

  std::cout << basketobj1.getAmount() << std::endl;

  return 0;

}
```

Exercise:
In the header file and its matching cpp file (basket.h, basket.cc) , there is only one class defined in there. Recall the functions defined in slide 23 (anotherFunc and increasePrice). Incorporate those functions into the basket.h and basket.cc and make sure they are callable.

Pointers

- Variable that stores the memory address of another variable
- Allows modification of a variable directly rather than copying
  - Copying can be expensive memory-wise if it is large. Particle physics data ROOT trees is an example.
  - When passed to a function, variables are copied.
  - Instead, can pass a pointer to the function instead, allowing direct modification of value stored at address.

Pointers (oversimplified)



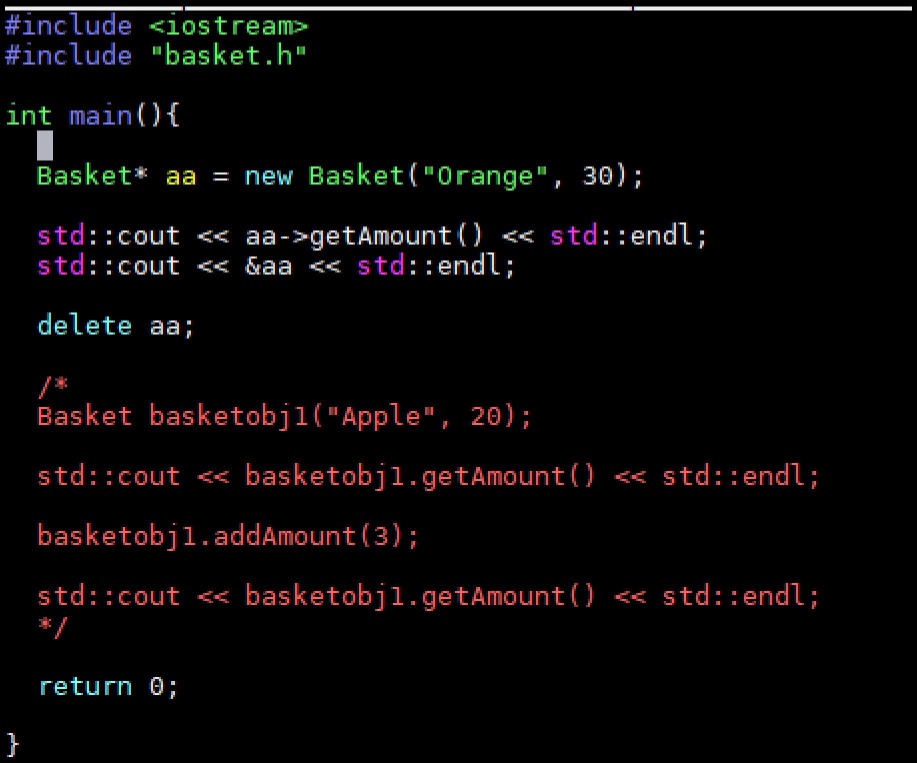
Allows you to modify the variable directly as stored in memory.

Pointers

- Pointers ‘point to’ the variable whose address they store.
- ‘*’ is called the ‘*dereference operator*’. Access the variable the pointer is pointing to directly.
- ‘&’ is called the ‘*address-of operator*’
- ‘*’ is also used when declaring a pointer.
- ‘->’ can be thought of as a dereference operator followed by a dot.

Variable:
```
object.method()
```
Pointers:
```
object->method()
(*object).method()
```

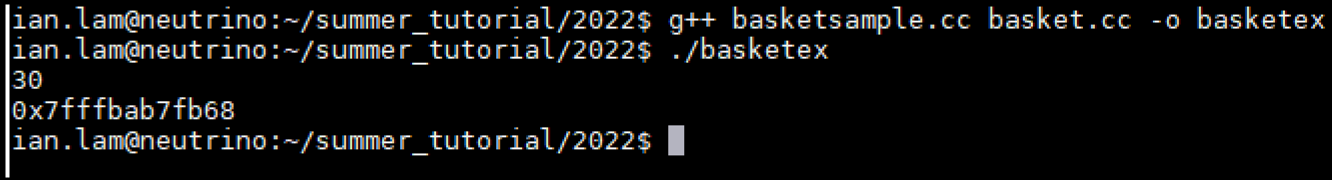
basketsample.cc



```
#include <iostream>
#include "basket.h"

int main(){

  Basket* aa = new Basket("Orange", 30);

  std::cout << aa->getAmount() << std::endl;
  std::cout << &aa << std::endl;

  delete aa;

  /*
  Basket basketobj1("Apple", 20);

  std::cout << basketobj1.getAmount() << std::endl;

  basketobj1.addAmount(3);

  std::cout << basketobj1.getAmount() << std::endl;
  */

  return 0;

}
```

Note: Example of bulk commenting.

Output



```
ian.lam@neutrino:~/summer_tutorial/2022$ g++ basketsample.cc basket.cc -o basketex
ian.lam@neutrino:~/summer_tutorial/2022$ ./basketex
30
0x7fffbab7fb68
ian.lam@neutrino:~/summer_tutorial/2022$
```

Pointers - Note

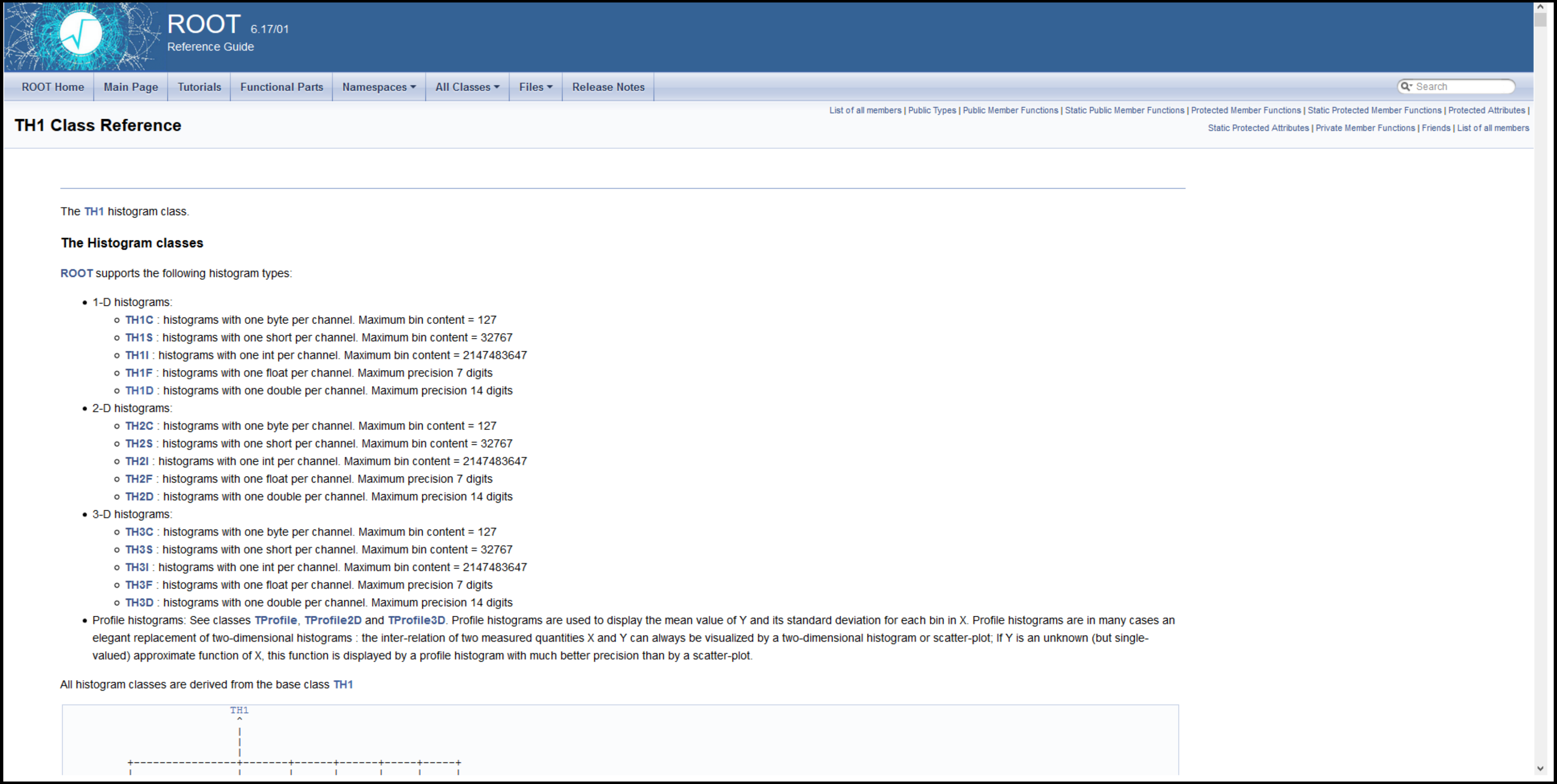
- If you use pointers, you **HAVE** to delete them after you are done. (Notice the delete in slide 56)
- Not so important in simple code like this but especially important if you are pushing pointers through loops.
- If you forget to delete a pointer, and you create a new object with the same pointer name, you'll get a memory leak. NOT GOOD.
- Deletion frees up memory. If you are running a program and you notice that your memory consumption is high and climbing, you probably have a memory leak that could be due to undeleted pointers.
- Order of deletion also matters. 'LIFO' : Last In First Out.

Pointers - Note

- Notice I kept pointers till the end. It goes to show that you can do a lot without having to use pointers.
- However, ROOT depends quite a lot on pointers, so you should get comfy.

ROOT tie-in

- With all this in mind, you should be able to look up various ROOT classes and use them effectively.
- Example: type ‘root cern th1’ in Google.
- TH1 : 1D histogram class for ROOT
- TH1D : 1D histogram of the type ‘double’.
- TH1F : 1D histogram of the type ‘float’, etc.


ROOT 6.17/01
Reference Guide
ROOT Home | Main Page | Tutorials | Functional Parts | Namespaces | All Classes | Files | Release Notes
Search
List of all members | Public Types | Public Member Functions | Static Public Member Functions | Protected Member Functions | Static Protected Member Functions | Protected Attributes | Static Protected Attributes | Private Member Functions | Friends | List of all members

TH1 Class Reference

The TH1 histogram class.

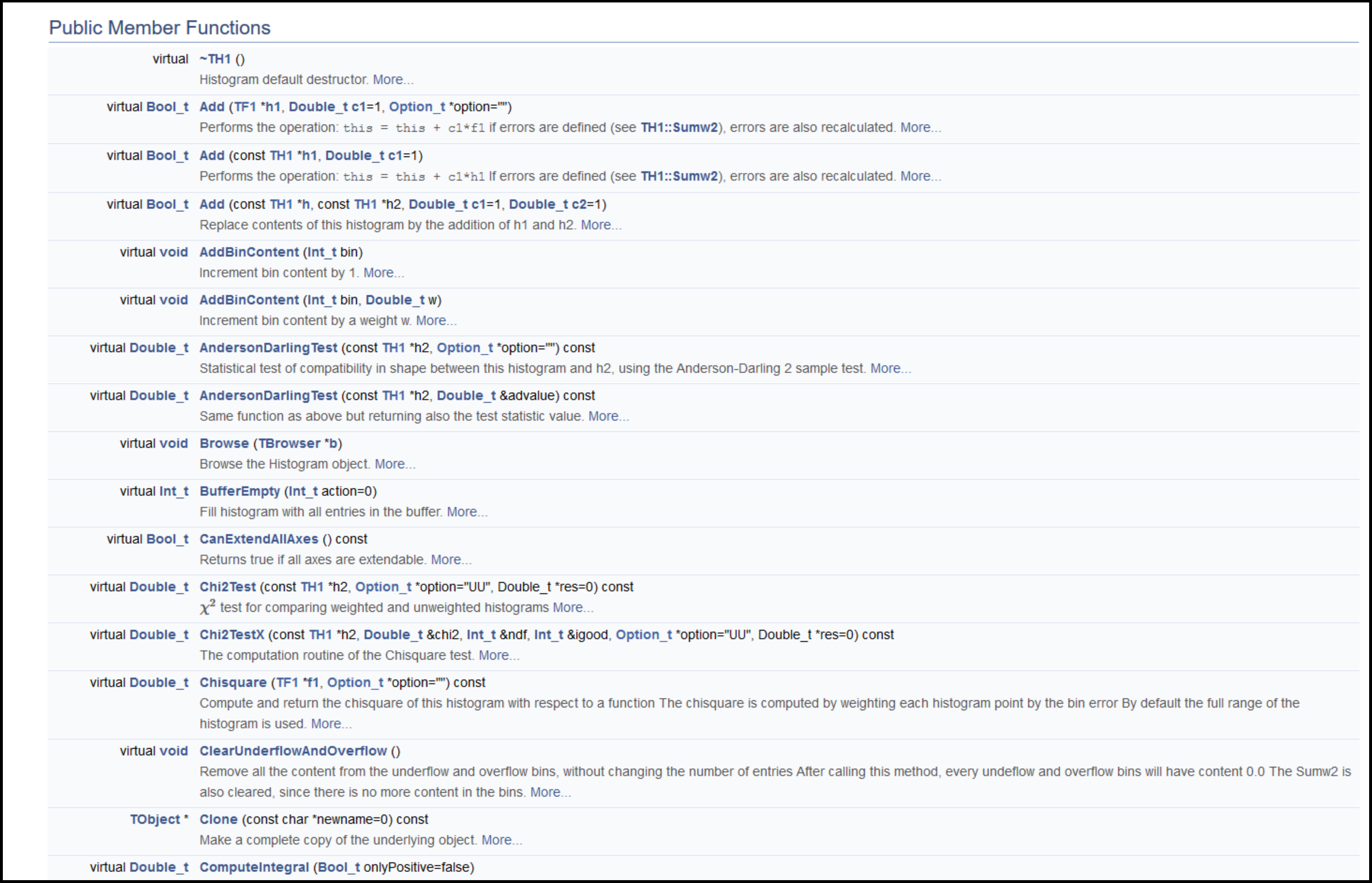
The Histogram classes

ROOT supports the following histogram types:

- 1-D histograms:
  - TH1C : histograms with one byte per channel. Maximum bin content = 127
  - TH1S : histograms with one short per channel. Maximum bin content = 32767
  - TH1I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH1F : histograms with one float per channel. Maximum precision 7 digits
  - TH1D : histograms with one double per channel. Maximum precision 14 digits
- 2-D histograms:
  - TH2C : histograms with one byte per channel. Maximum bin content = 127
  - TH2S : histograms with one short per channel. Maximum bin content = 32767
  - TH2I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH2F : histograms with one float per channel. Maximum precision 7 digits
  - TH2D : histograms with one double per channel. Maximum precision 14 digits
- 3-D histograms:
  - TH3C : histograms with one byte per channel. Maximum bin content = 127
  - TH3S : histograms with one short per channel. Maximum bin content = 32767
  - TH3I : histograms with one int per channel. Maximum bin content = 2147483647
  - TH3F : histograms with one float per channel. Maximum precision 7 digits
  - TH3D : histograms with one double per channel. Maximum precision 14 digits
- Profile histograms: See classes TProfile, TProfile2D and TProfile3D. Profile histograms are used to display the mean value of Y and its standard deviation for each bin in X. Profile histograms are in many cases an elegant replacement of two-dimensional histograms : the inter-relation of two measured quantities X and Y can always be visualized by a two-dimensional histogram or scatter-plot; If Y is an unknown (but single-valued) approximate function of X, this function is displayed by a profile histogram with much better precision than by a scatter-plot.

All histogram classes are derived from the base class TH1

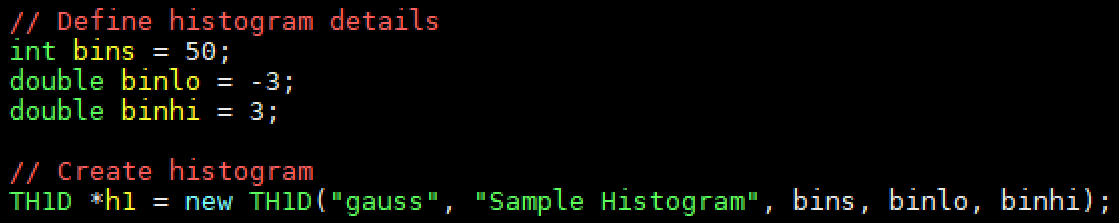
```
                          TH1
                           ^
                           |
                           |
                           |
      +-----------------+------------+------------+-------------+----------+----------+
      |                 |            |            |             |          |          |
```



Public Member Functions

virtual	~TH1 () Histogram default destructor. More...
virtual Bool_t	Add (TF1 *h1, Double_t c1=1, Option_t *option="") Performs the operation: `this = this + c1*f1` if errors are defined (see TH1::Sumw2), errors are also recalculated. More...
virtual Bool_t	Add (const TH1 *h1, Double_t c1=1) Performs the operation: `this = this + c1*h1` If errors are defined (see TH1::Sumw2), errors are also recalculated. More...
virtual Bool_t	Add (const TH1 *h, const TH1 *h2, Double_t c1=1, Double_t c2=1) Replace contents of this histogram by the addition of h1 and h2. More...
virtual void	AddBinContent (Int_t bin) Increment bin content by 1. More...
virtual void	AddBinContent (Int_t bin, Double_t w) Increment bin content by a weight w. More...
virtual Double_t	AndersonDarlingTest (const TH1 *h2, Option_t *option="") const Statistical test of compatibility in shape between this histogram and h2, using the Anderson-Darling 2 sample test. More...
virtual Double_t	AndersonDarlingTest (const TH1 *h2, Double_t &advalue) const Same function as above but returning also the test statistic value. More...
virtual void	Browse (TBrowser *b) Browse the Histogram object. More...
virtual Int_t	BufferEmpty (Int_t action=0) Fill histogram with all entries in the buffer. More...
virtual Bool_t	CanExtendAllAxes () const Returns true if all axes are extendable. More...
virtual Double_t	Chi2Test (const TH1 *h2, Option_t *option="UU", Double_t *res=0) const χ^2 test for comparing weighted and unweighted histograms More...
virtual Double_t	Chi2TestX (const TH1 *h2, Double_t &chi2, Int_t &ndf, Int_t &igood, Option_t *option="UU", Double_t *res=0) const The computation routine of the Chisquare test. More...
virtual Double_t	Chisquare (TF1 *f1, Option_t *option="") const Compute and return the chisquare of this histogram with respect to a function The chisquare is computed by weighting each histogram point by the bin error By default the full range of the histogram is used. More...
virtual void	ClearUnderflowAndOverflow () Remove all the content from the underflow and overflow bins, without changing the number of entries After calling this method, every undeflow and overflow bins will have content 0.0 The Sumw2 is also cleared, since there is no more content in the bins. More...
TObject *	Clone (const char *newname=0) const Make a complete copy of the underlying object. More...
virtual Double_t	ComputeIntegral (Bool_t onlyPositive=false)

ROOT tie-in



```
// Define histogram details
int bins = 50;
double binlo = -3;
double binhi = 3;

// Create histogram
TH1D *h1 = new TH1D("gauss", "Sample Histogram", bins, binlo, binhi);
```

Creating a pointer to a TH1D class and initializing.

ROOT tie-in

```
// Define inputs for generating random values
int points = 5000;
double mu = 0;
double sigma = 1;


// Define histogram details
int bins = 50;
double binlo = -3;
double binhi = 3;

// Create histogram
TH1D *h1 = new TH1D("gauss", "Sample Histogram", bins, binlo, binhi);

// Create rand object to generate random number
TRandom3 rand;

// Initialize rand object. TRandom3 is the number generator. The 0 argument means use new seed everytime
// This guarantees new random numbers every time the code is run.
// Ref: https://root.cern.ch/doc/master/classTRandom.html

rand = TRandom3(0);

// Generate random numbers and fill into histogram
for (int i=0; i<points; i++){
  double temp = rand.Gaus(mu,sigma);
  h1->Fill(temp);
}
```

Fill that histogram with 5000 random values drawn from a Gaussian distribution.

Exercise:
Get the above code to compile and run. Before running the code, what do you expect your histogram to look like? Change the value of the mu and sigma variable. Does your histogram change as expected? To compile with ROOT, you probably need something like this: g++ myfile.cc -o roothist `root-config --cflags --libs` (notice: use ` and not ‘)

ROOT tie-in

Exercise:
Rewrite the code in the previous slide to not use pointers.

Thank you!

If you have any questions, send me an email or ask in Discord (#eieioo-cplusplus), and I’ll do my best to answer!

Email: ian dot lam at queensu dot ca