Gearing up for C++

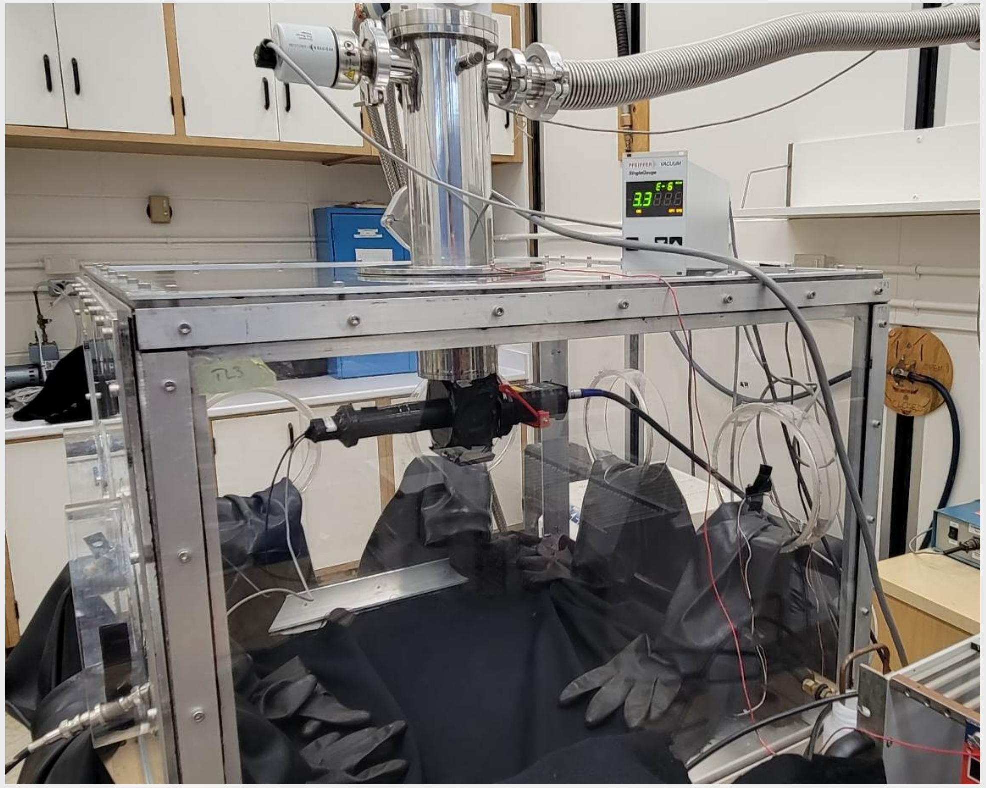
May 10th , 2023

Jonathan Hucker

Queen's UNIVERSITY

Who am I?

- Completed undergrad at Trent University in 2021.
- Currently in final semester of master's at Queen's.
- In my research I investigate photoluminescent and scintillation behaviours of various samples at low temperatures. I'm very passionate about my research so here is some more info:
  - Studied samples include common materials used in the construction of low background rare event search detectors, coatings used in these detectors, and scintillating crystals.
  - Currently writing a report and analyzing data from an experiment involving an inorganic crystal scintillator Cs_2ZrCl_6.
  - While this scintillator exhibits excellent light-yield and energy resolution, its a very slow scintillator, emitting light on the order of 100s of microseconds.
- I enjoy camping, baking, and hockey (go Nucks!)



How did I get here?

- In my undergraduate career, I took two classes on programming. The first was a first year introductory programming class on C# which was a mandatory class to take for physics students. The second was computational physics, a second year physics/comp-sci class focused on applications of python through a physics lens.
- Six months ago, I knew next to nothing about C++. The script used to preform initial analysis of our results was written in C++ and worked okay for fast fluorescent measurements.
- In the fall I worked on a slow scintillating crystal, which required a more careful treatment during data analysis and some modifications to our analysis procedure.
- In principle python could do the job, however the data to be analyzed was 16 raw binary files of 20GB in size each. Preforming initial data analysis on these files was far too slow (~2-3 hours per file). Hence, I was inspired to learn C++ to tackle analyzing these large files.

Goals for this seminar

- Go over the basics:
- Ensure you can compile a simple script.
- Understand the different objects in the language.
- Learn how to find resources for C++ code for when something breaks or you encounter something unfamiliar.
- Collaborate as a team to create a functioning program out of nothing but a blank notepad (or which ever code editor you use)

Rough Outline:

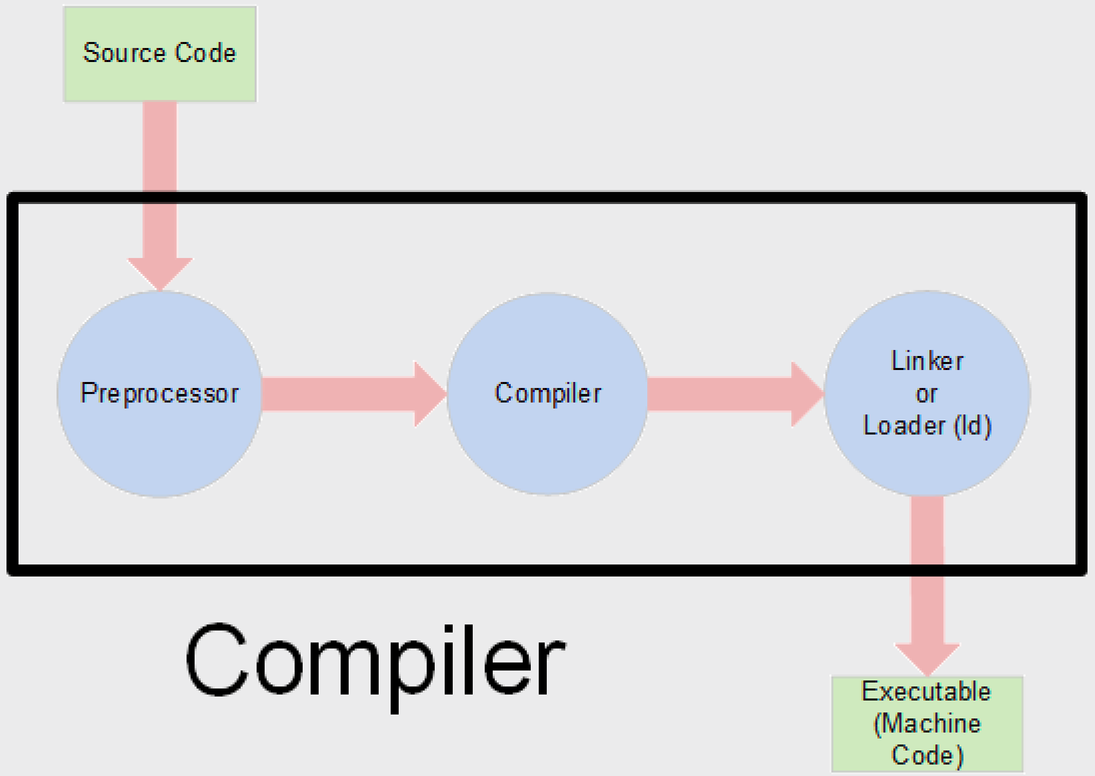
1) The classic: Hello World and the typical framework
2) Data types, escape sequences variables and functions
3) Loops and conditional statements (if and switch)
4) Arrays and Vectors
5) Read and write files
6) Intro to structs
7) Small groups activity (time permitted)

Personal Goals:

1) Provide you the tools and resources needed to continue learning C++
2) Break the ice with coding in C++
3) Collectively learn together to build better code.

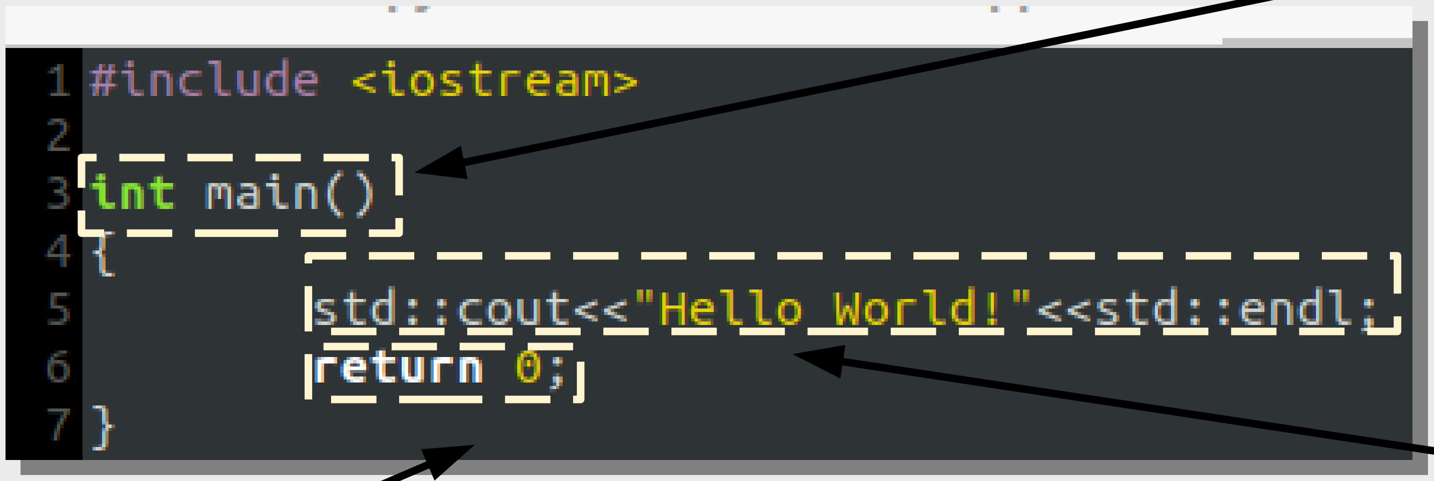
The Basics

- Compiler: the compiler transforms the written C++ code to a version that can be understood by the computer.
- Linker: if you are compiling a code with multiple C++ files, the linker combines these compiled files.
- Executable: a fancy word for application such as google or photoshop. If you were an app developer using C++, this is the final product to be used by the consumers.



Source: https://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/compiler_op.html

The Classic Hello World



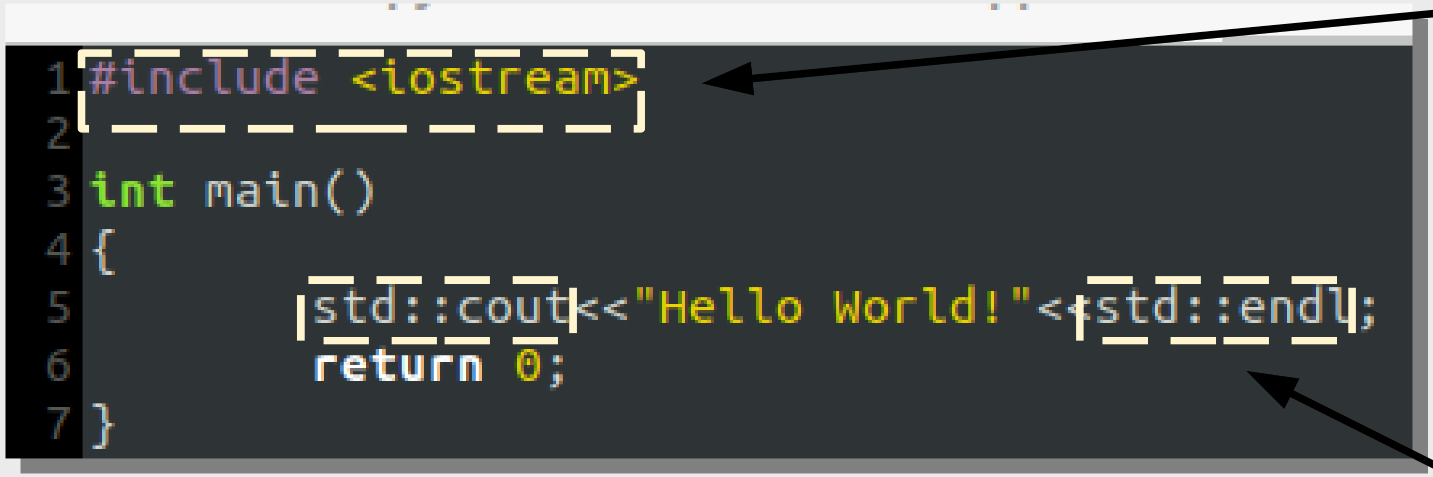
```
#include <iostream>

int main()
{
	std::cout<<"Hello World!"<<std::endl;
	return 0;
}
```

Main function: nothing is input into main and the return of main is an int type

Statement: Something you ask the computer to do before finishing the program

Return statement. Here, return 0 simply implies "the program executes properly".

#include <iostream> ?



```
#include <iostream>

int main()
{
        std::cout<<"Hello World!"<<std::endl;
        return 0;
}
```

Standard input-output stream (iostream). Contains a library of useful objects. Similar to python packages

std::cout and std::endl are objects apart of the iostream library.

std::cout can be thought of as print() in python

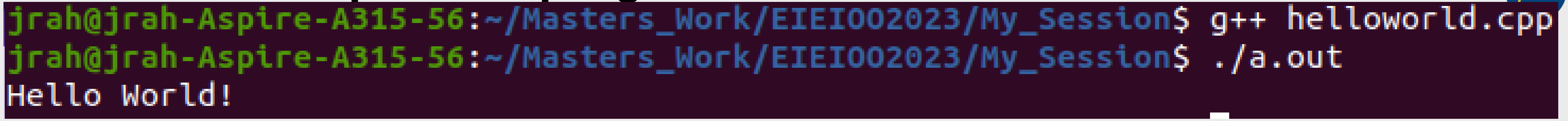
std::endl, short for ‘end line”, creates a new line.

std is referred to as a *namespace*

There are many other useful libraries to include, here are some that I have found useful:

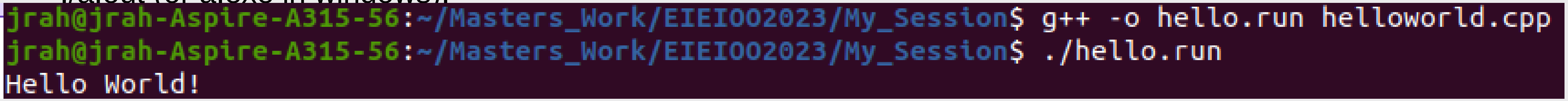
cstdlib: C Standard General Utilities Library
ctime: C Time Library
chrono: Time library (different from ctime)
fstream: input/output file stream
All Standard Libraries
Open source C++ libraries

How do I compile this program?



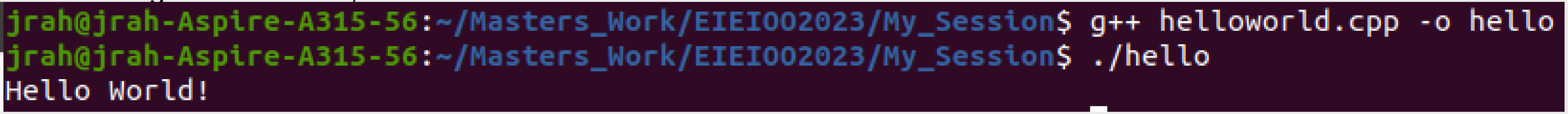
```
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ g++ helloworld.cpp
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ ./a.out
Hello World!
```

- To compile our program from a terminal, we must first navigate to the directory where the program is stored.
- Using the g++ compiler, we can compile the program by typing g++ followed by our file name: *g++ helloword.cpp.*
- While it doesn’t initially appear that anything has happened, an executable file is generated and can be executed by typing ./a.out (or a.exe in windows).



```
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ g++ -o hello.run helloworld.cpp
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ ./hello.run
Hello World!
```

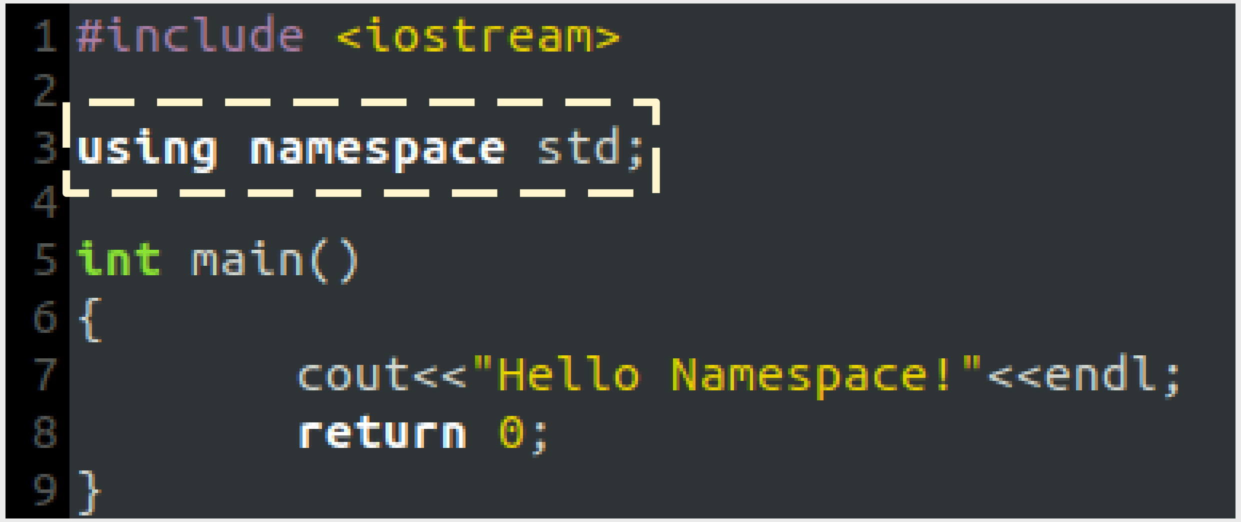
- Rename your executable using the -o flag.
- Other flags exist as well, read about them here.



```
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ g++ helloworld.cpp -o hello
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO02023/My_Session$ ./hello
Hello World!
```

- To compile with a specific version of C++, such as C++ 11, you must add this to the end of your command: -std=c++11

using namespace std?

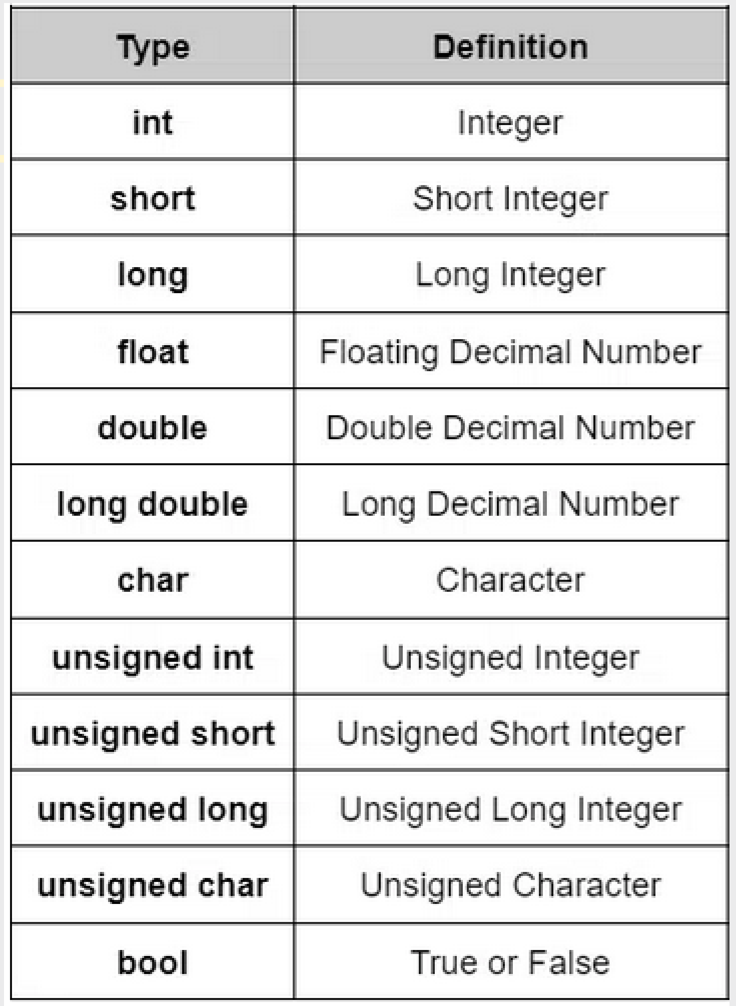


```cpp
#include <iostream>

using namespace std;

int main()
{
		cout<<"Hello Namespace!"<<endl;
		return 0;
}
```

- In this tutorial, you are welcome to use the namespace std, though I will continue to include the std prefix.
- From what I gather, it is prefered/industry standard to typically not use namespaces to eliminate the possibility of confusion.

- Namespaces are a way to organize functions and objects.
- In python, the numpy package package both contain the function *sqrt()*. If you wish to import both *math* and *numpy*, and then call the *sqrt()* function, which function is called?
- Namespaces in C++ alleviate this problem similar to how when calling sqrt, we typically call it as np.sqrt() or math.sqrt() to specify which library the function originates from.

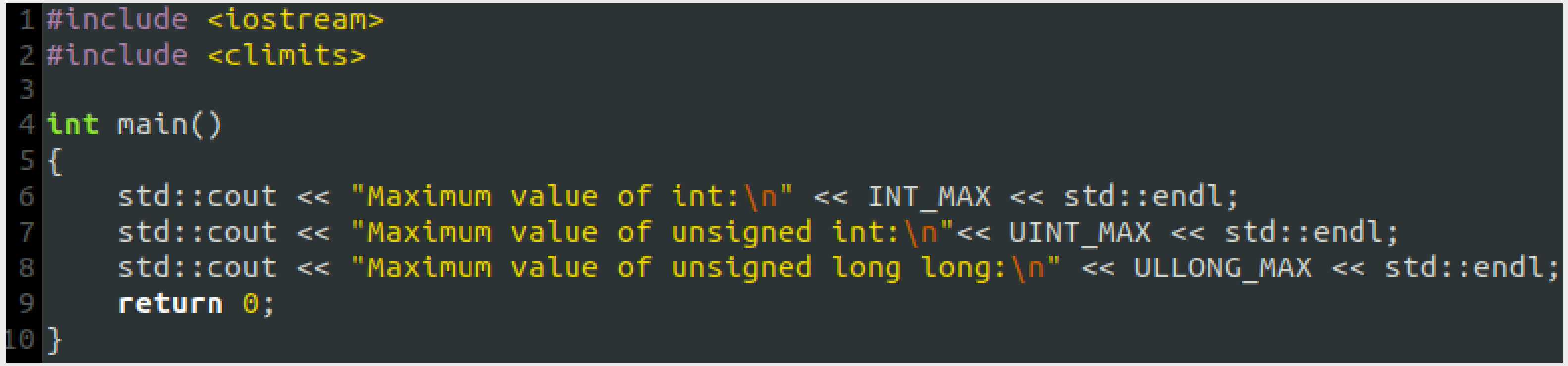
Int, float, long, oh my!



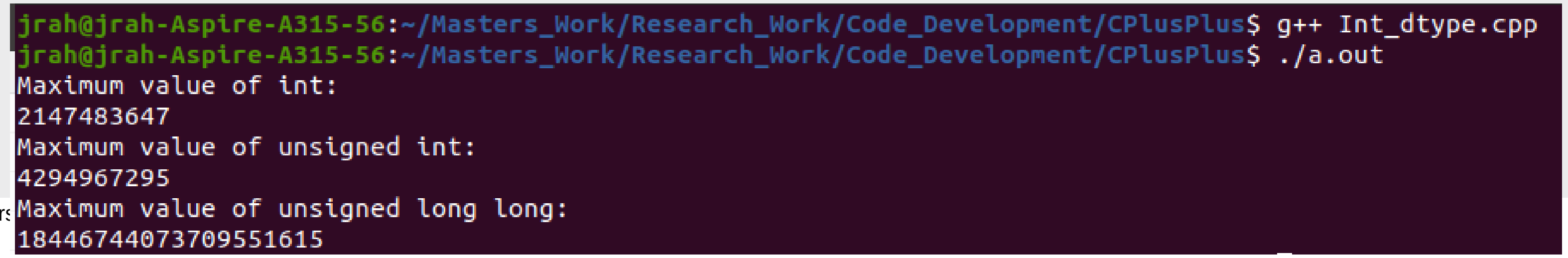
Type	Definition
int	Integer
short	Short Integer
long	Long Integer
float	Floating Decimal Number
double	Double Decimal Number
long double	Long Decimal Number
char	Character
unsigned int	Unsigned Integer
unsigned short	Unsigned Short Integer
unsigned long	Unsigned Long Integer
unsigned char	Unsigned Character
bool	True or False

Source:https://www.codeguru.com/cplusplus/cplusplus-data-types-variables-for-beginners

- Both C++ and python use data types. There are many different data types available, left are some of the most common types. Here are a lot more.
- The string data type, missing from this list, must be included in the header of your script to use it and needs to be prefixed by std::
- More info on the limits of the various data types can be found here. Below is an example.

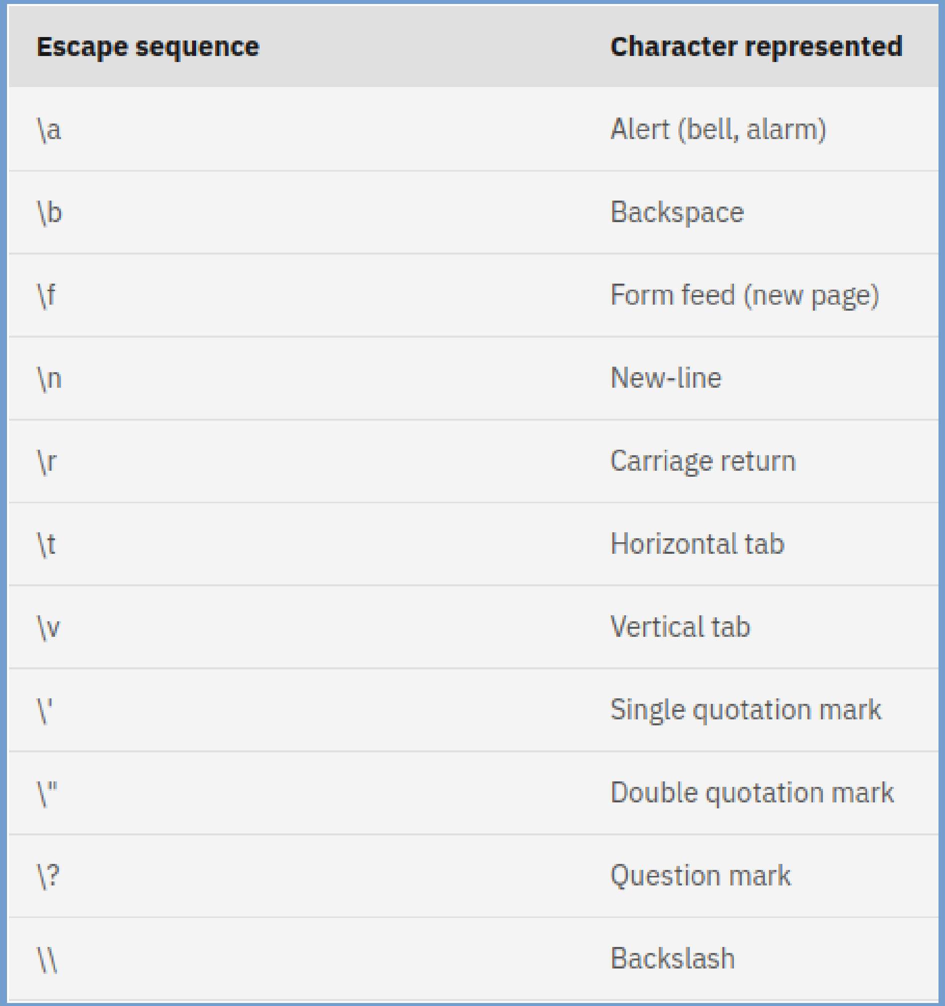


```cpp
#include <iostream>
#include <climits>

int main()
{
    std::cout << "Maximum value of int:\n" << INT_MAX << std::endl;
    std::cout << "Maximum value of unsigned int:\n"<< UINT_MAX << std::endl;
    std::cout << "Maximum value of unsigned long long:\n" << ULLONG_MAX << std::endl;
    return 0;
}
```



```
jrah@jrah-Aspire-A315-56:~/Masters_Work/Research_Work/Code_Development/CPlusPlus$ g++ Int_dtype.cpp
jrah@jrah-Aspire-A315-56:~/Masters_Work/Research_Work/Code_Development/CPlusPlus$ ./a.out
Maximum value of int:
2147483647
Maximum value of unsigned int:
4294967295
Maximum value of unsigned long long:
18446744073709551615
```

\n ? Escape Sequences

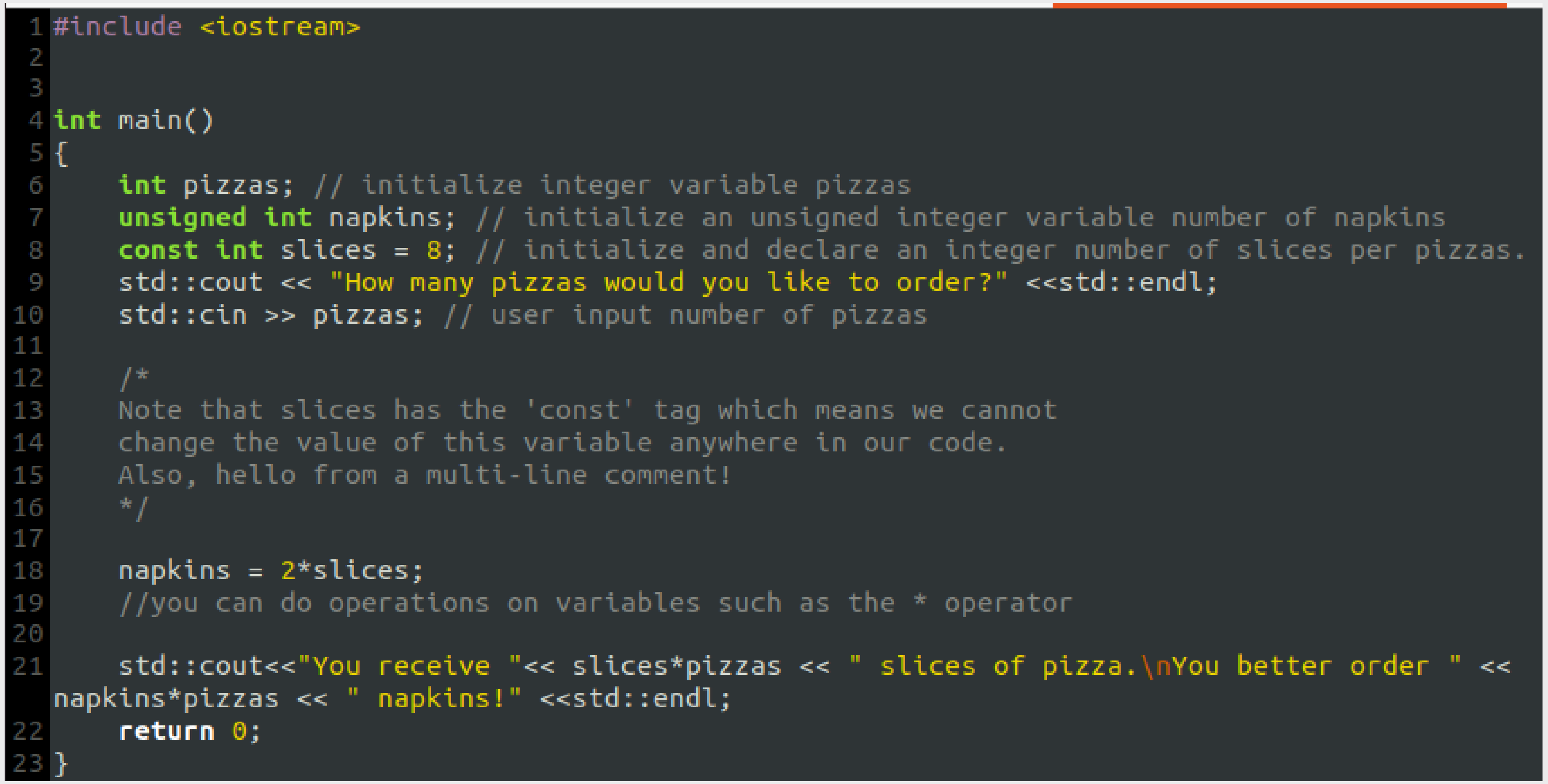


Escape sequence	Character represented
\a	Alert (bell, alarm)
\b	Backspace
\f	Form feed (new page)
\n	New-line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\\	Backslash

- You may have noticed in the previous example that contained in the string of characters was *\n*. You may have also encounter this symbol in python.
- \n is an example of an escape sequence which creates a new line.
- Provided is a table of various escape sequences. More information on escape sequences can be found here.
- The most common that I use include \n, \t, \’ and \”.
- My favourite is \a.

Variable Declaration and Initialization

- There are two steps to stating a variable in C++
- **Declaration**:
  - What is the identifier (name)? What is the data type of this variable?
- **Initialization**:
  - What value should the variable take?

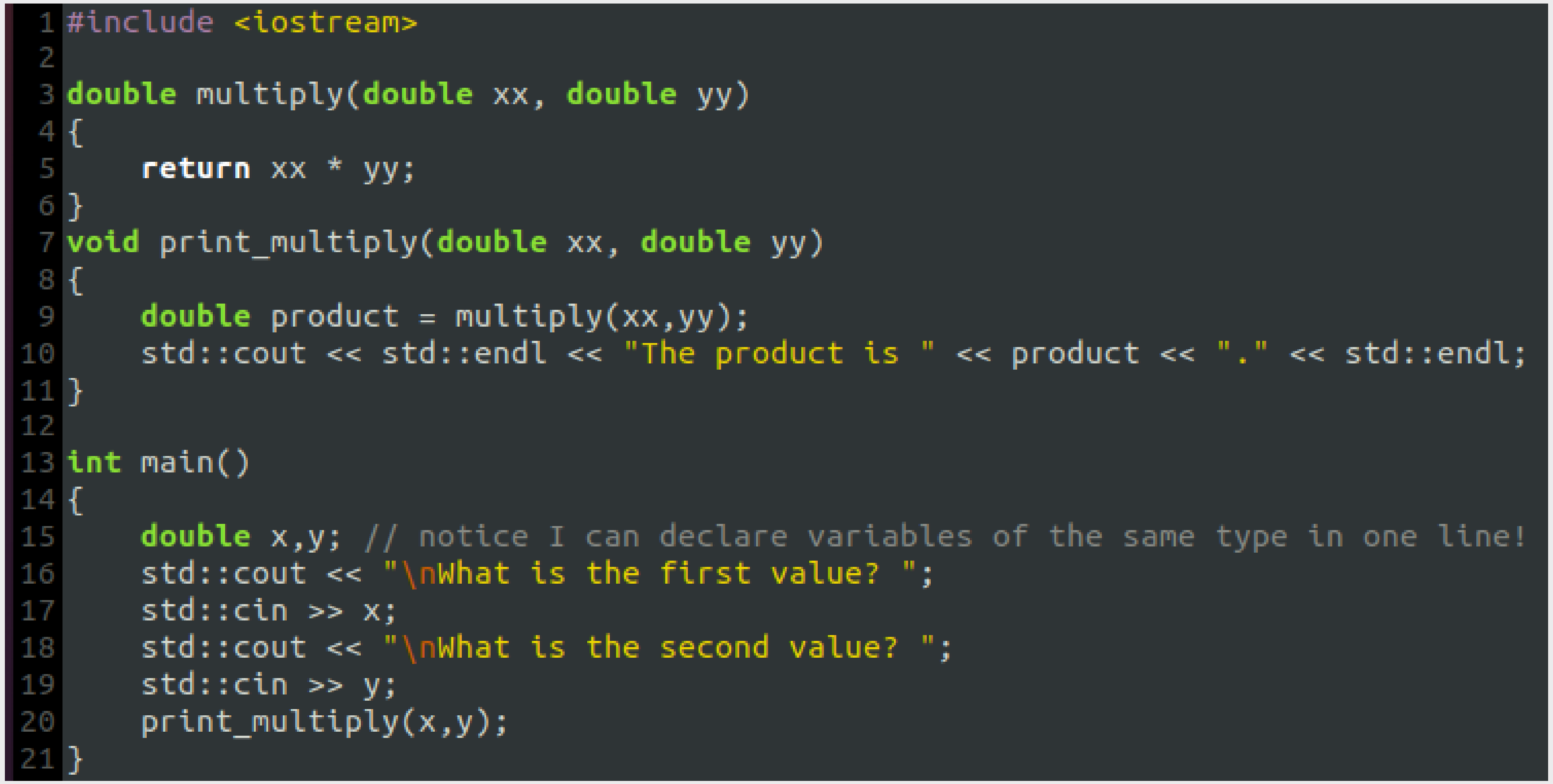


```
#include <iostream>


int main()
{
    int pizzas; // initialize integer variable pizzas
    unsigned int napkins; // initialize an unsigned integer variable number of napkins
    const int slices = 8; // initialize and declare an integer number of slices per pizzas.
    std::cout << "How many pizzas would you like to order?" <<std::endl;
    std::cin >> pizzas; // user input number of pizzas

    /*
    Note that slices has the 'const' tag which means we cannot
    change the value of this variable anywhere in our code.
    Also, hello from a multi-line comment!
    */

    napkins = 2*slices;
    //you can do operations on variables such as the * operator

    std::cout<<"You receive "<< slices*pizzas << " slices of pizza.\nYou better order " <<
napkins*pizzas << " napkins!" <<std::endl;
    return 0;
}
```

Can you identify all the new elements in this code?

Define and Call Your Own Function

- Similarly to how you declare and initialize a variable, you can declare and initialize your own functions.
- Functions have have arguments or no arguments.
- Functions can have a return or have no return (void).
- Typically a function is defined when you need to run a block of code multiple times.



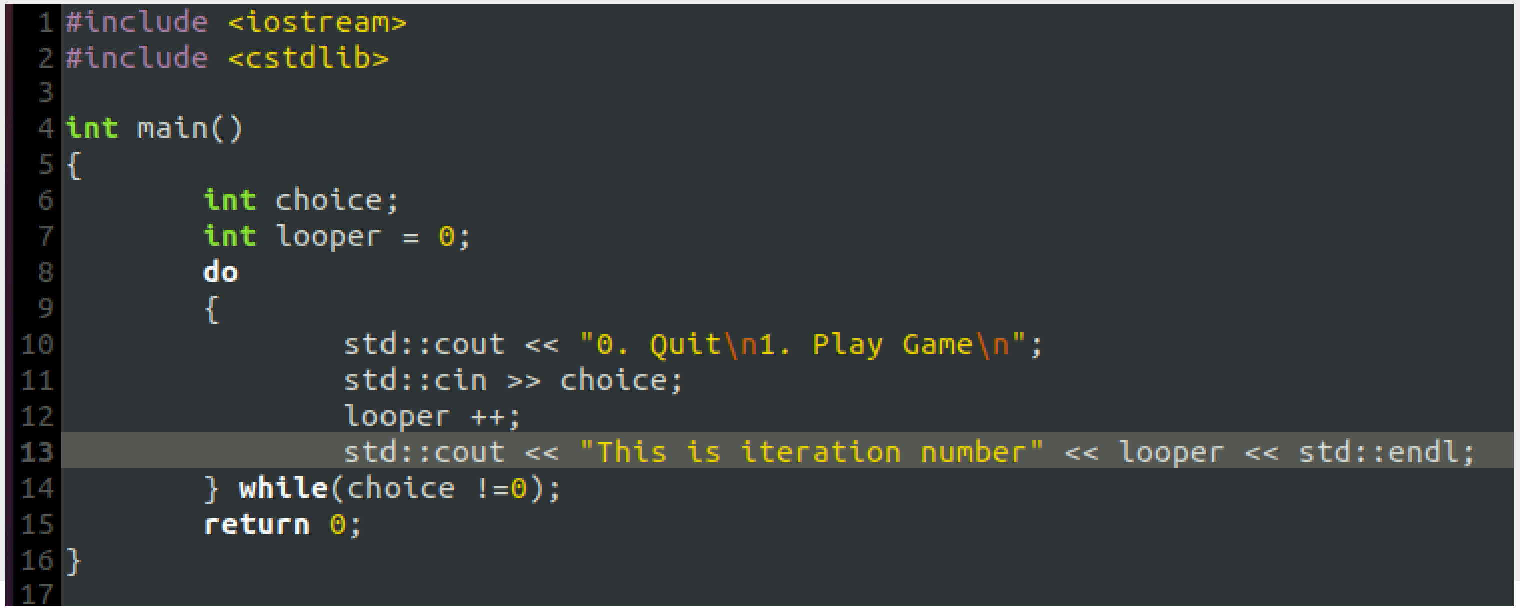
```cpp
#include <iostream>

double multiply(double xx, double yy)
{
    return xx * yy;
}
void print_multiply(double xx, double yy)
{
    double product = multiply(xx,yy);
    std::cout << std::endl << "The product is " << product << "." << std::endl;
}

int main()
{
    double x,y; // notice I can declare variables of the same type in one line!
    std::cout << "\nWhat is the first value? ";
    std::cin >> x;
    std::cout << "\nWhat is the second value? ";
    std::cin >> y;
    print_multiply(x,y);
}
```

Making a game!

- At this stage, we know enough of the absolute basics to start to build a more involved code. Let's create a game!
- The objective of the game will be to guess a randomly generated number, super fun!
- Our game will have a main menu, hints, and high scores!
- We will build this code together, introducing new topics as we go.
- We will first start with making the main menu using a do while loop.

Main Menu: Do While Loop

- Let's start building our main menu. In many games, it is useful to be able to return to a main menu. One way we can set this up is with a do while loop.
- Very straight forward loop: *do* something *while* condition is true
- Notice the semicolon after the while statement.
- Do while loops always iterate at least once.

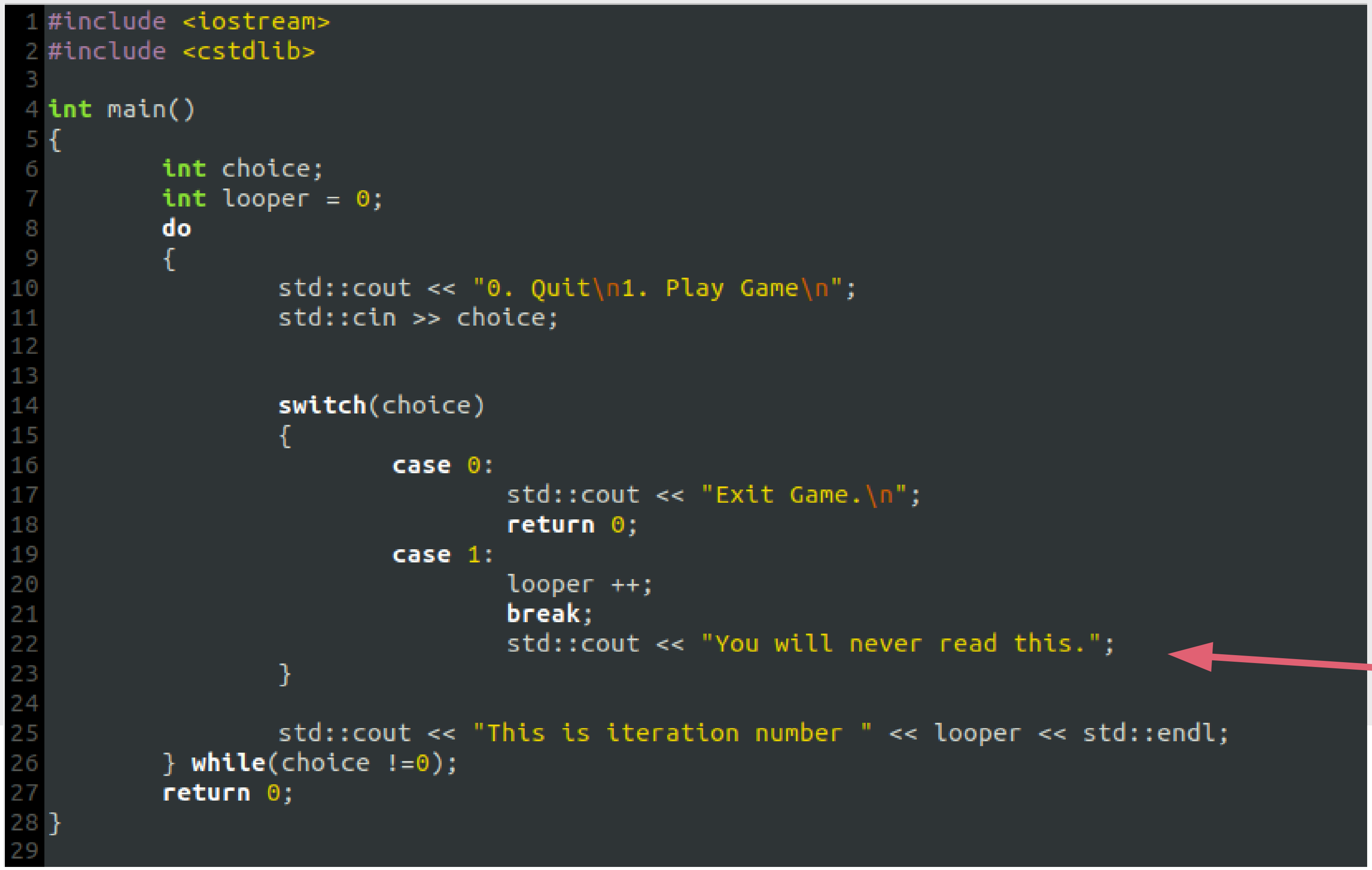


```
#include <iostream>
#include <cstdlib>

int main()
{
        int choice;
        int looper = 0;
        do
        {
                std::cout << "0. Quit\n1. Play Game\n";
                std::cin >> choice;
                looper ++;
                std::cout << "This is iteration number" << looper << std::endl;
        } while(choice !=0);
        return 0;
}
```

New code just dropped!
- Loops:
  - Do
- Operators:
  - ++
  - !=

Main Menu: Conditional Statement: Switch

- A switch is a conditional statement that will execute a block of code if a case is true
- Useful for small number of cases



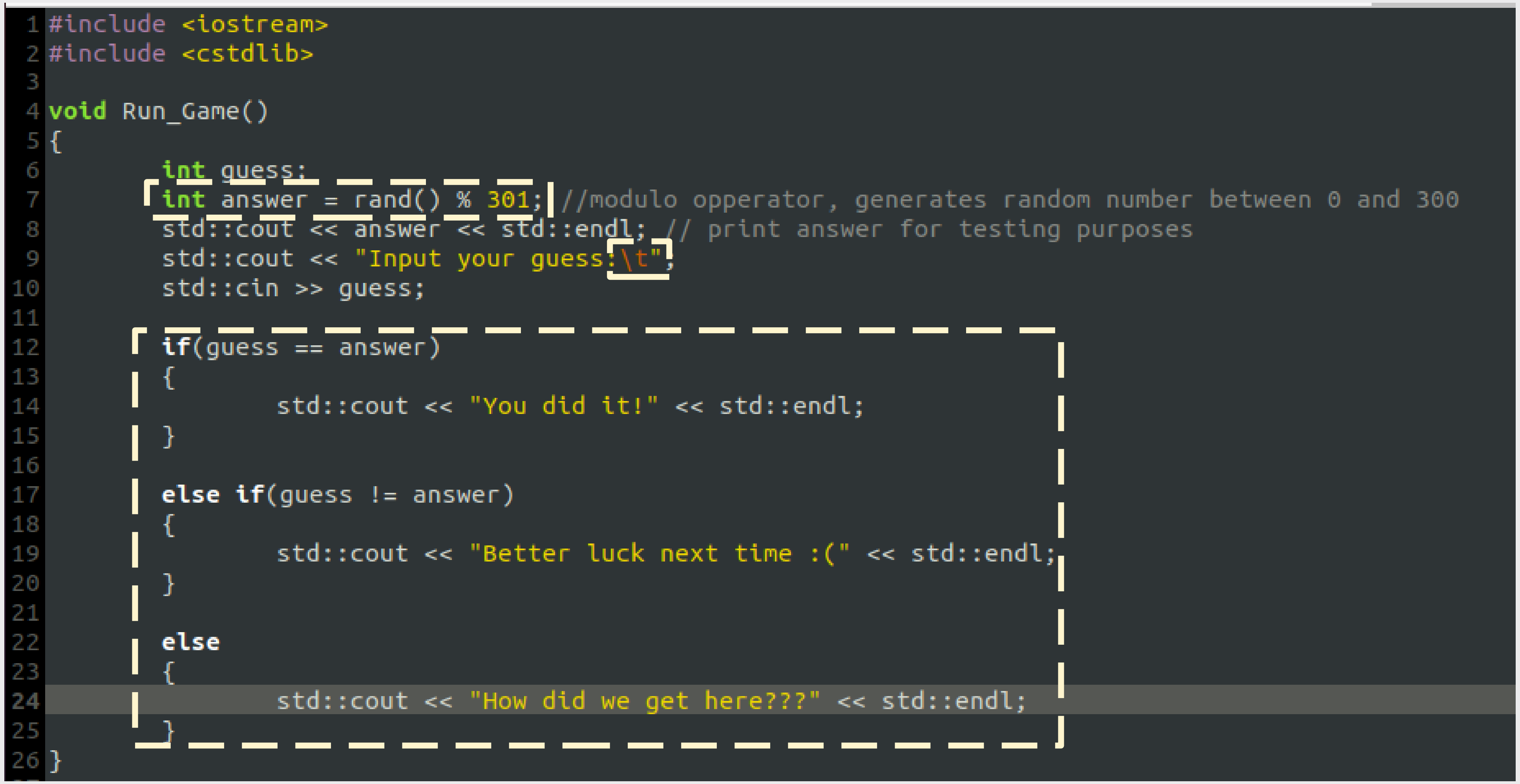
```cpp
#include <iostream>
#include <cstdlib>

int main()
{
	int choice;
	int looper = 0;
	do
	{
		std::cout << "0. Quit\n1. Play Game\n";
		std::cin >> choice;


		switch(choice)
		{
			case 0:
				std::cout << "Exit Game.\n";
				return 0;
			case 1:
				looper ++;
				break;
				std::cout << "You will never read this.";
		}

		std::cout << "This is iteration number " << looper << std::endl;
	} while(choice !=0);
	return 0;
}

```

New code just dropped!
- Conditional:
  - Switch statement
- Break statement
- Return statement...but somewhere else!

Occurs after break statement

Play Game Function: if statements

- Our game is pretty boring, let's add to it.
- We want to generate a random number and have the user guess that number.
- Let's accomplish this using another function.
- We will use an *if* statement to check if the user guesses the correct number.

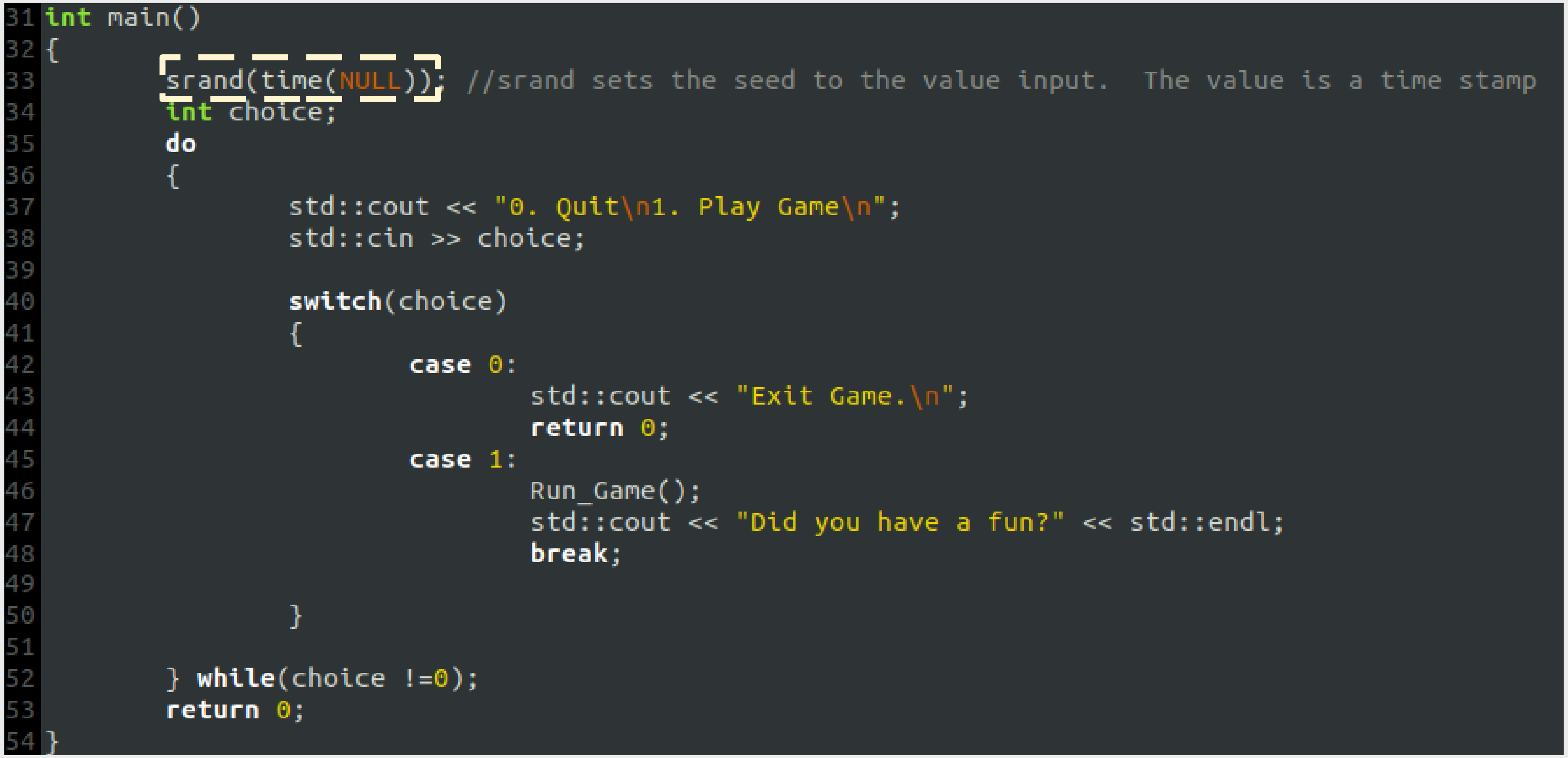


```cpp
#include <iostream>
#include <cstdlib>

void Run_Game()
{
	int guess;
	int answer = rand() % 301; //modulo opperator, generates random number between 0 and 300
	std::cout << answer << std::endl; // print answer for testing purposes
	std::cout << "Input your guess:\t";
	std::cin >> guess;

	if(guess == answer)
	{
		std::cout << "You did it!" << std::endl;
	}

	else if(guess != answer)
	{
		std::cout << "Better luck next time :(" << std::endl;
	}

	else
	{
		std::cout << "How did we get here???" << std::endl;
	}
}
```

Pssst: don't forget to call the function!

srand

- Do you notice that each subsequent time you play the game, it gets easier?
- The random number generated is *technically* not random.
- You need to specify the *seed* to start the sequence of random numbers.
- We can do this in main with *srand.*
- **Need to #include <ctime>**

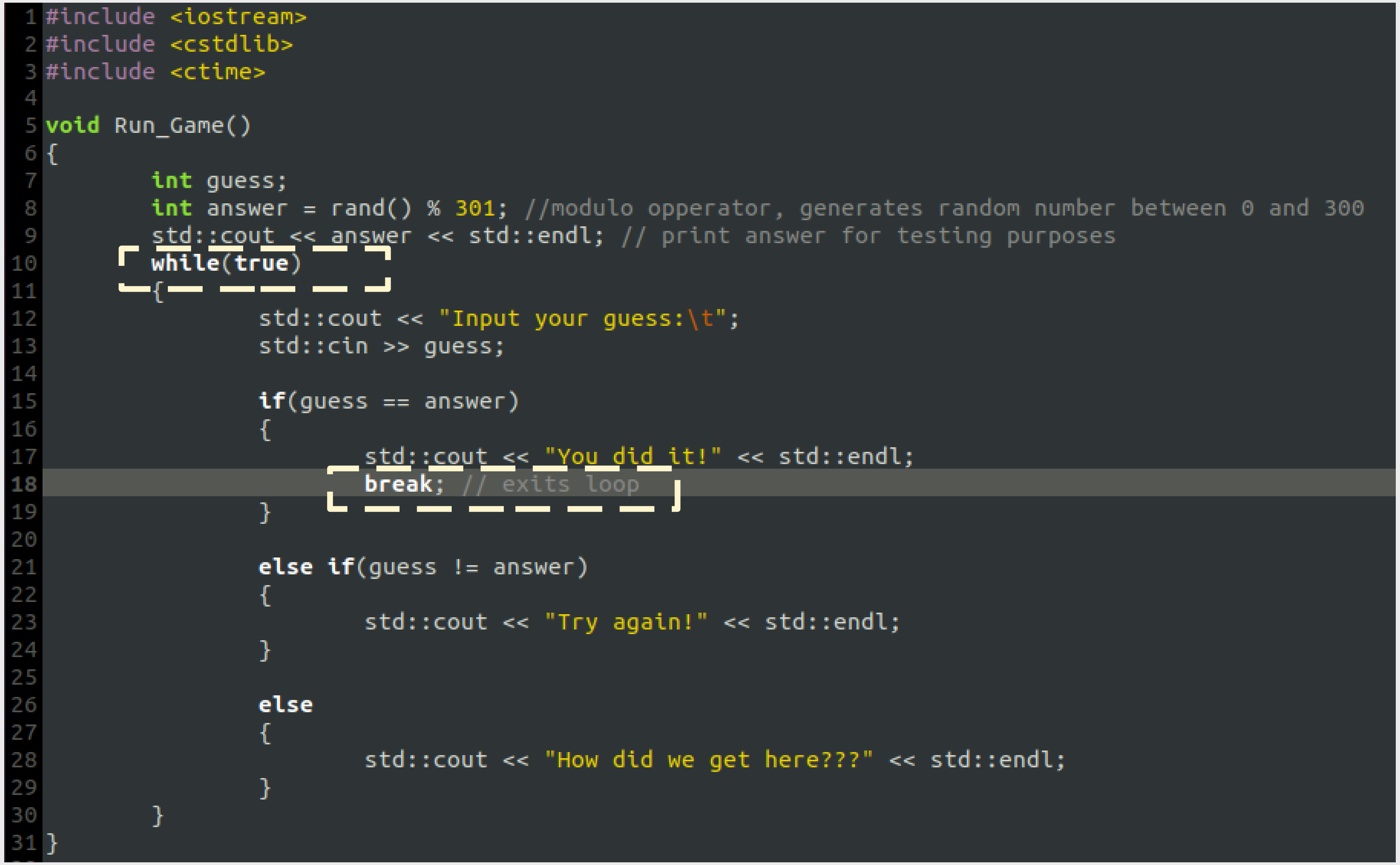


```
int main()
{
        srand(time(NULL)); //srand sets the seed to the value input.  The value is a time stamp
        int choice;
        do
        {
                std::cout << "0. Quit\n1. Play Game\n";
                std::cin >> choice;

                switch(choice)
                {
                        case 0:
                                std::cout << "Exit Game.\n";
                                return 0;
                        case 1:
                                Run_Game();
                                std::cout << "Did you have a fun?" << std::endl;
                                break;

                }

        } while(choice !=0);
        return 0;
}
```

Bonus point: any guesses as to what time(NULL) represents? Hint: try printing this value.

The while loop

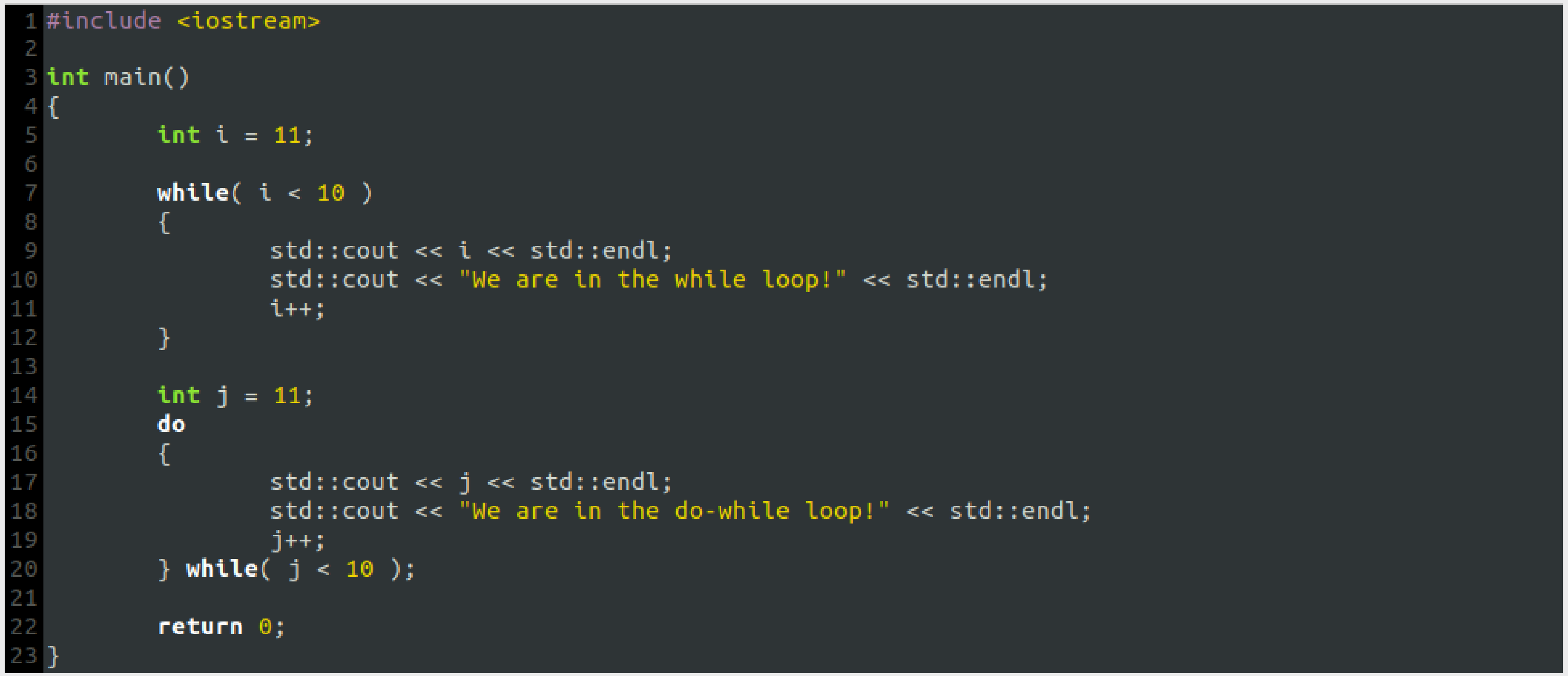
- Let's allow our user to play the game until they guess the correct integer.
- To do this, let's use a while loop.
- The while loop will run a block of code so long as the condition is true.
- Don't forget to ensure that the loop can end (unless you want an infinite loop!)



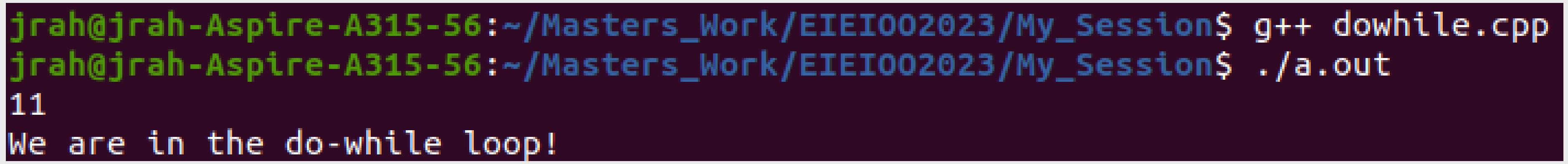
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

void Run_Game()
{
        int guess;
        int answer = rand() % 301; //modulo opperator, generates random number between 0 and 300
        std::cout << answer << std::endl; // print answer for testing purposes
        while(true)
        {
                std::cout << "Input your guess:\t";
                std::cin >> guess;

                if(guess == answer)
                {
                        std::cout << "You did it!" << std::endl;
                        break; // exits loop
                }

                else if(guess != answer)
                {
                        std::cout << "Try again!" << std::endl;
                }

                else
                {
                        std::cout << "How did we get here???" << std::endl;
                }
        }
}
```

Wait...the do while and while loops look the same?

- Whoa there, not so fast! There is a distinct difference.
- A do-while loop will execute the code block before checking the condition whereas a while loop checks the condition first before compiling.
- Therefore a do-while loop will always execute *at least* once.



```cpp
#include <iostream>

int main()
{
	int i = 11;

	while( i < 10 )
	{
		std::cout << i << std::endl;
		std::cout << "We are in the while loop!" << std::endl;
		i++;
	}

	int j = 11;
	do
	{
		std::cout << j << std::endl;
		std::cout << "We are in the do-while loop!" << std::endl;
		j++;
	} while( j < 10 );

	return 0;
}
```



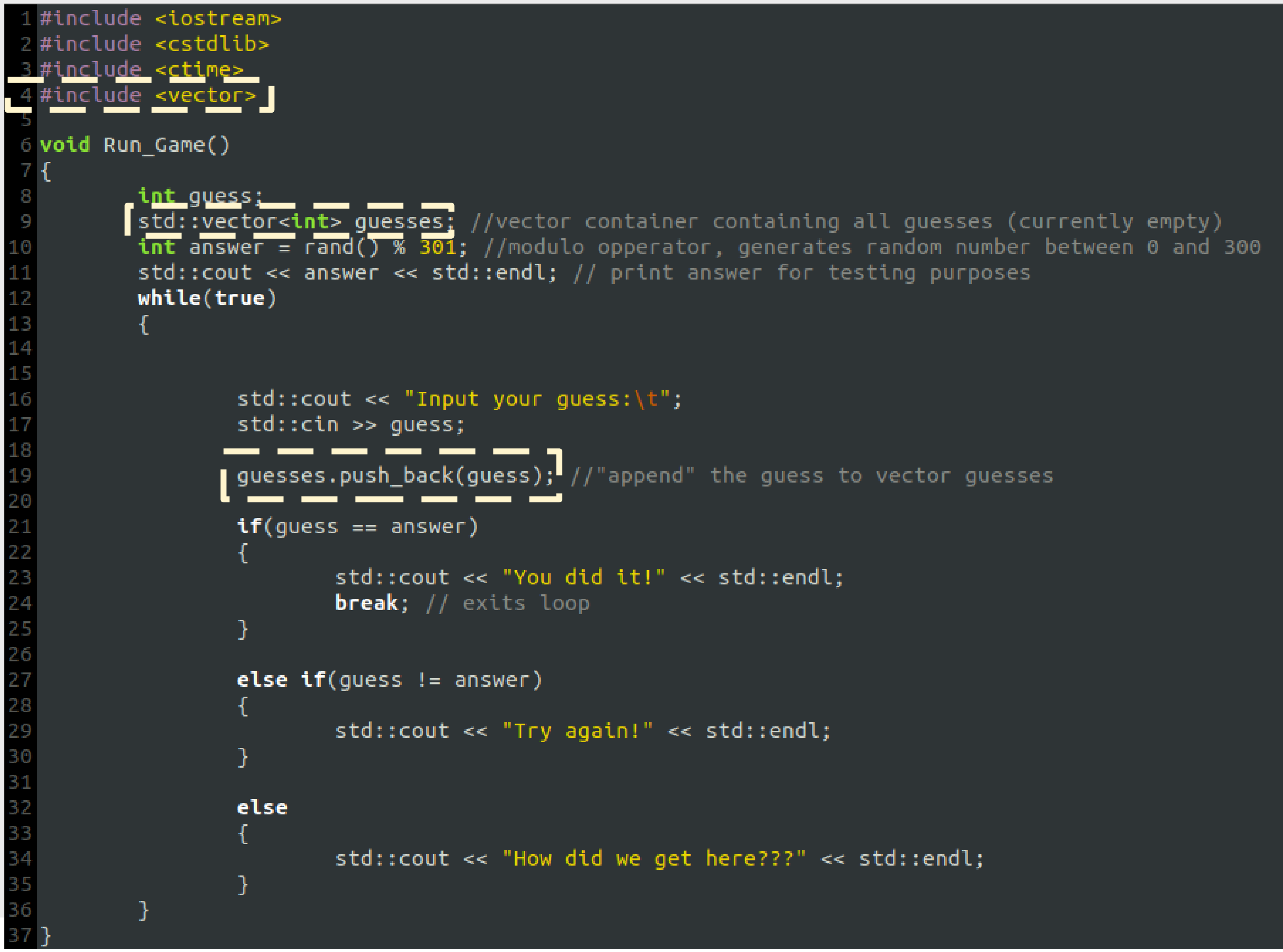
```
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO2023/My_Session$ g++ dowhile.cpp
jrah@jrah-Aspire-A315-56:~/Masters_Work/EIEIO2023/My_Session$ ./a.out
11
We are in the do-while loop!
```

Arrays and vectors

- It will be useful to have an idea of what values already been guessed. Let’s introduce a method to store multiple values.
- There are C style arrays and C++ vectors (technically called vector containers)
- Arrays have the following syntax: *int myArray[100];*
  - **Int**: specifies the data type allocated to each element in the array
  - **myArray**: the identifier of the variable
  - **[100]**: denotes the size of the array (indexing starts at 0)
  - Arrays are **statically** sized, you cannot change their size once created.
- Vectors have the following syntax: *std::vector<int> myVector;*
  - **std::vector**: the object from the standard library (must #include <vector>)
  - **<int>** specifies the data type for each element in the vector
  - **myVector**: identifier of the vector
  - Vectors are **dynamically** sized, you can change their size and do not need to specify their size upon declaration.

Store guess as a vector

- First we must include the vector class (yes vector containers are technically classes, more on this later).
- Next we must declare our vector and give it a useful identifier.
  - Recall that vectors are dynamic, you can change their size.
- To fill our vector, we will call the function .push_back() from our vector guesses and input our guess.
  - Python equivalent would be .append() to a list.
  - .append() is referred to as a method whereas .push_back() is a function.

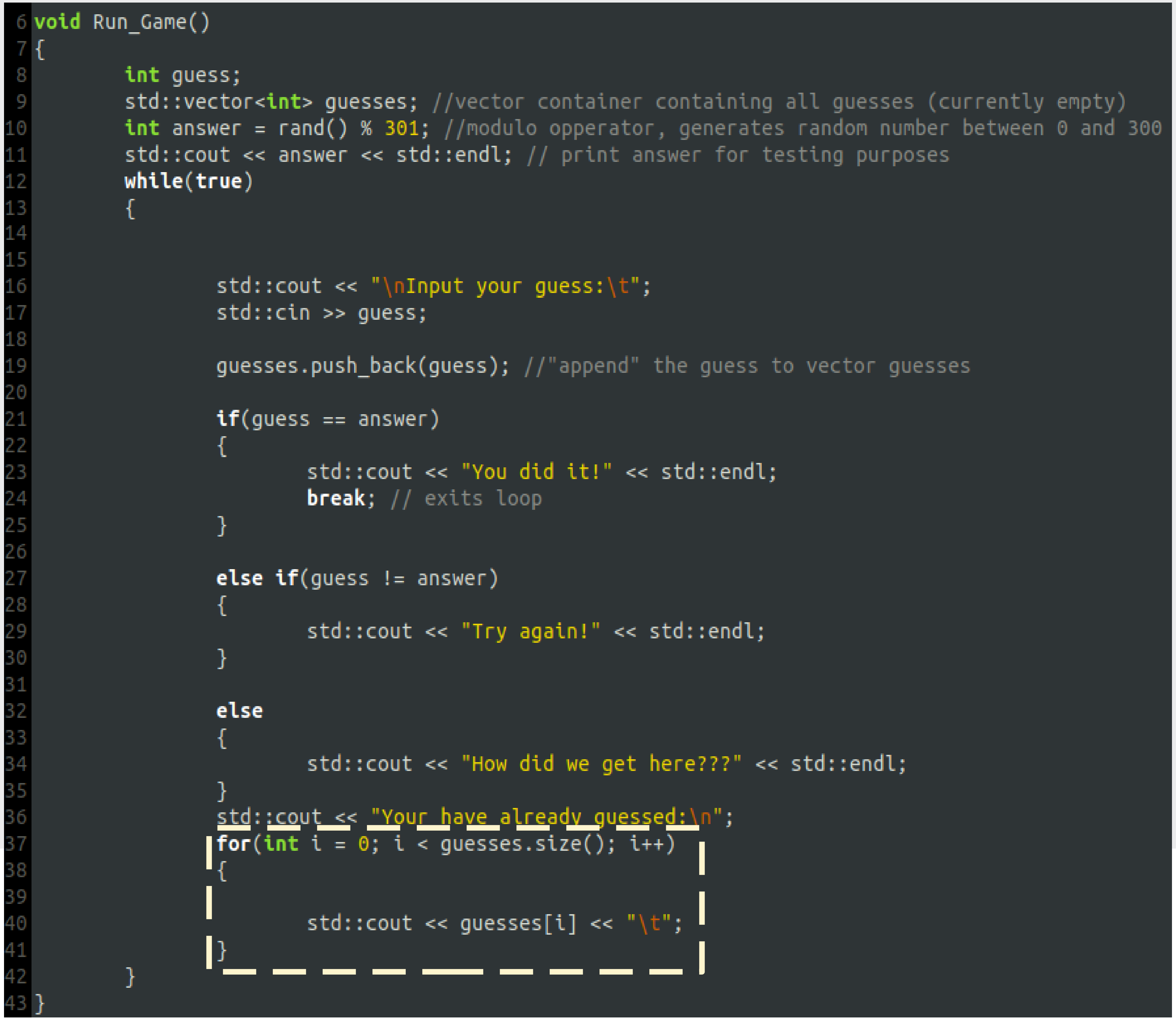


```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>

void Run_Game()
{
        int guess;
        std::vector<int> guesses; //vector container containing all guesses (currently empty)
        int answer = rand() % 301; //modulo opperator, generates random number between 0 and 300
        std::cout << answer << std::endl; // print answer for testing purposes
        while(true)
        {


                std::cout << "Input your guess:\t";
                std::cin >> guess;

                guesses.push_back(guess); //"append" the guess to vector guesses

                if(guess == answer)
                {
                        std::cout << "You did it!" << std::endl;
                        break; // exits loop
                }

                else if(guess != answer)
                {
                        std::cout << "Try again!" << std::endl;
                }

                else
                {
                        std::cout << "How did we get here???" << std::endl;
                }
        }
}
```

How could we instead use an array?

For loop

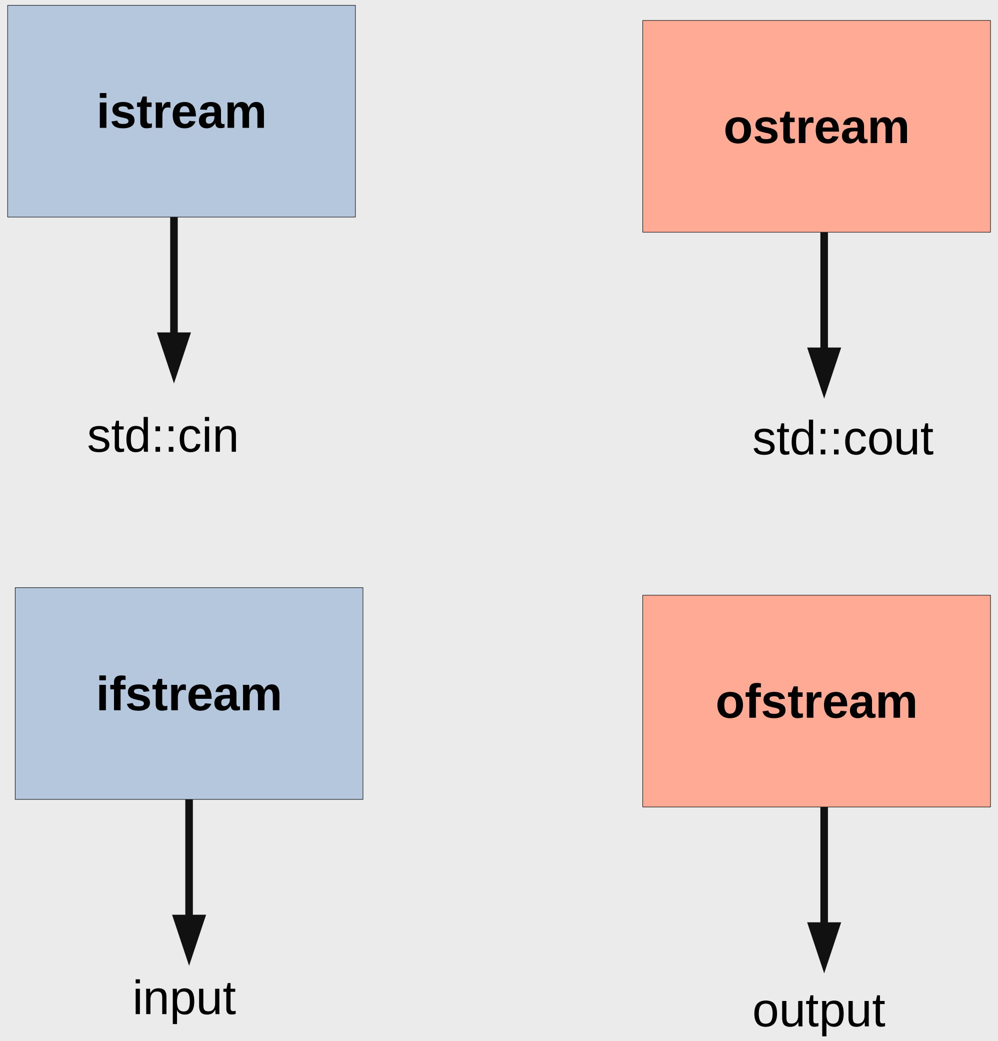
- To inform the user of guesses they have already tried, we can print each element of our vector using a for loop.
- Syntax is as follows:
  - for(init; condition; increment)
  - for(int i = 0; i < 10; i++) {std::cout << i << std::endl;}
- Let's use a for loop to iterate over elements of our vector (call a specific element of a vector with square brackets [ ])
- Your increment does not need to be ++, (eg., i--, i = i + 20, i *=)



```
void Run_Game()
{
        int guess;
        std::vector<int> guesses; //vector container containing all guesses (currently empty)
        int answer = rand() % 301; //modulo opperator, generates random number between 0 and 300
        std::cout << answer << std::endl; // print answer for testing purposes
        while(true)
        {


                std::cout << "\nInput your guess:\t";
                std::cin >> guess;

                guesses.push_back(guess); //"append" the guess to vector guesses

                if(guess == answer)
                {
                        std::cout << "You did it!" << std::endl;
                        break; // exits loop
                }

                else if(guess != answer)
                {
                        std::cout << "Try again!" << std::endl;
                }

                else
                {
                        std::cout << "How did we get here???" << std::endl;
                }
                std::cout << "Your have already guessed:\n";
                for(int i = 0; i < guesses.size(); i++)
                {

                        std::cout << guesses[i] << "\t";
                }
        }
}
```

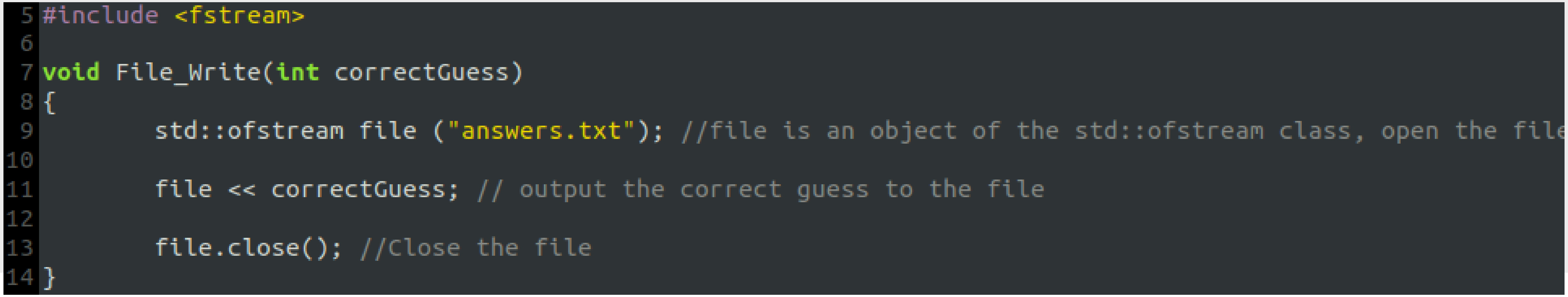
Intro to fstream

- The iostream library allowed for user interaction through a terminal input and outputs
- The ifstream will allow for inputs and outputs from a file.
- Imagine you have an experiment with multiple parameters that are allowed to change such as pressure, temperature, voltage, etc...
- If these parameters need to be input into your C++ script, it is inefficient to *hard code* variables every time these variables change.
- Values stored in an excelsheet or text file can be read by C++, allowing for quick modification of our input parameters.
- ifstream and ofstream are combined into one library called ifstream (more information **here**).



fstream: write to textfile

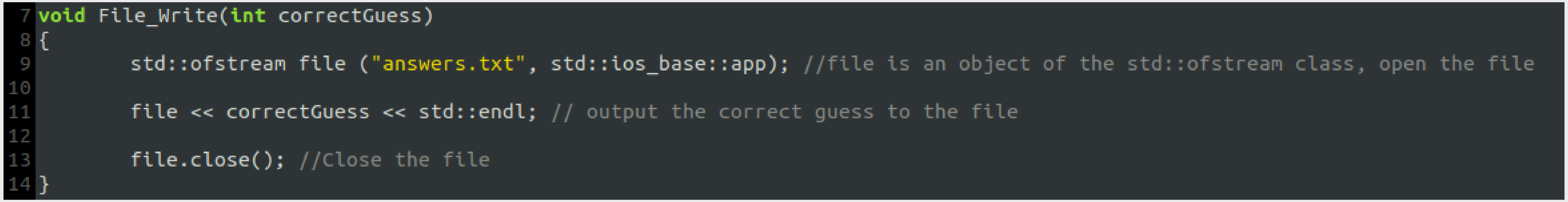
- Let's write a function that outputs our correct guess to a text file.
- We will need to include fstream. The return of our function will be void and will need one argument, the correct guess.
- To generate a file type object, we need to declare it from its class and initialize it with a file name.
- From here, use file as you would with cout.
- Although not strictly necessary, closing the file is good practice.
- Once you build this function, call it in Play_Game() when the user guesses the correct guess. Does your text file generate? What happens if you run the code multiple times?



```
#include <fstream>

void File_Write(int correctGuess)
{
        std::ofstream file ("answers.txt"); //file is an object of the std::ofstream class, open the file

        file << correctGuess; // output the correct guess to the file

        file.close(); //Close the file
}
```

fstream: write to textfile: append

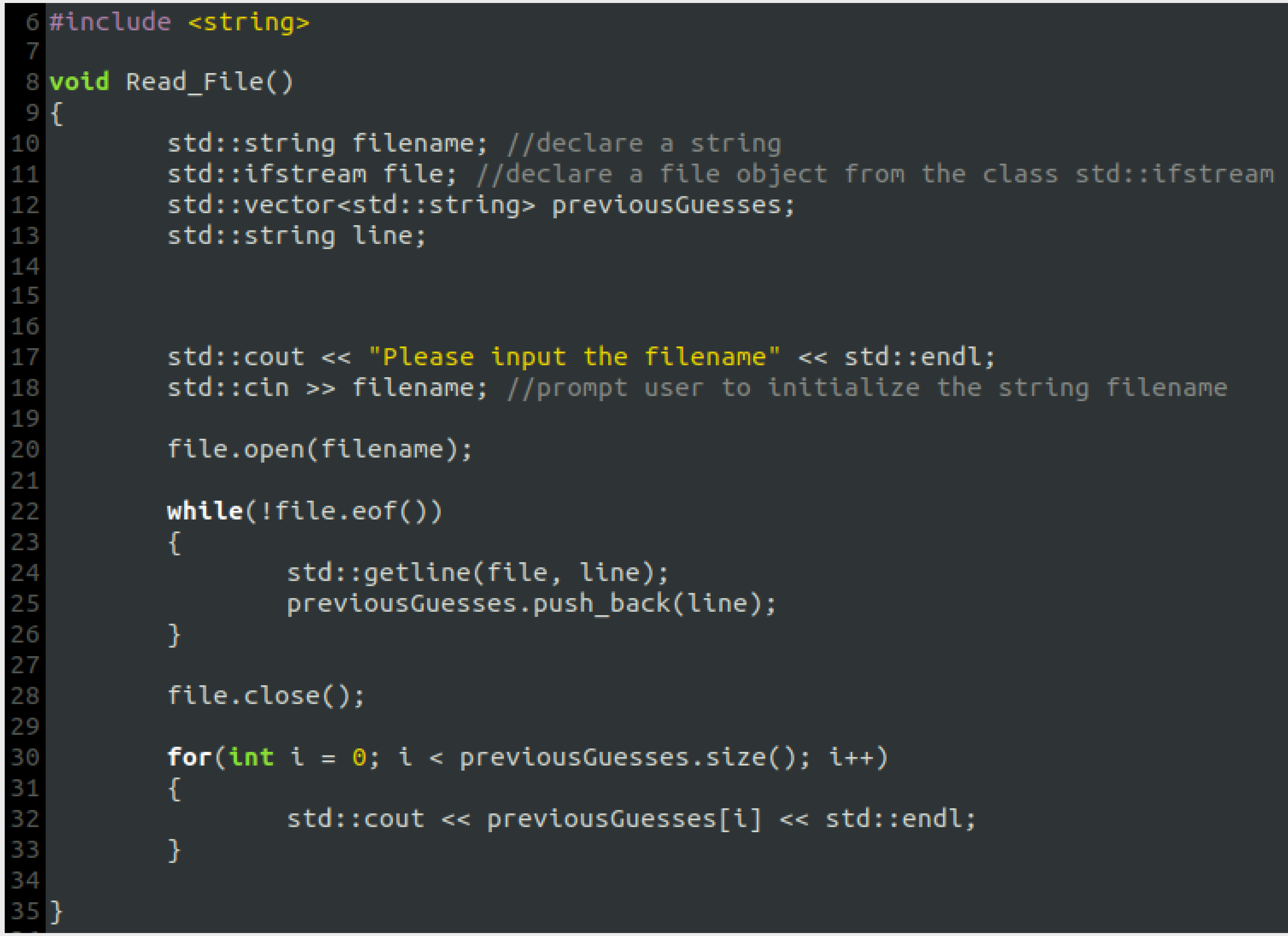
- You may have noticed that each time your run your program, the file is overwritten.
- We can prevent this std::ios_base::app
- There are other opening modes to consider. std::ios_base::trunc will delete the contents of the file upon opening.
- Here is a reference for other opening modes.



```cpp
void File_Write(int correctGuess)
{
        std::ofstream file ("answers.txt", std::ios_base::app); //file is an object of the std::ofstream class, open the file

        file << correctGuess << std::endl; // output the correct guess to the file

        file.close(); //Close the file
}
```

fstream: read from textfile

- Reading multiple values from a file requires a bit more effort. Here is one way to do it:
- This code snippet prompts the user for a filename and stores this information in a string called filename.
- file is an variable of the ifstream class. We then open the file with file.open().
- We will use a vector of strings to save the data
- The while loop will continue executing until the end of the file. Recall that while loops execute until the condition is false.
- std::getline will store the value of the ith iteration of the loop in string line.
- We then close the file and then print the previous guesses. Don't forget to call the function!

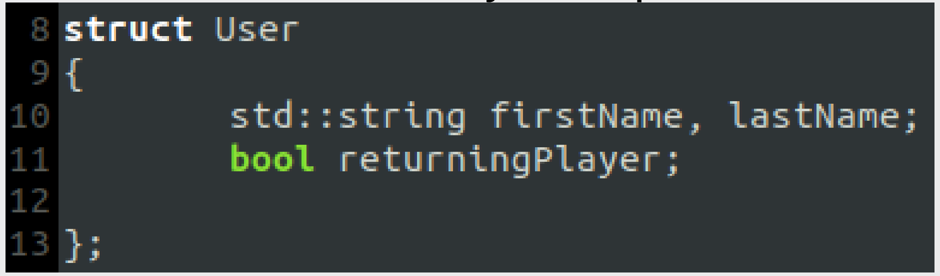


```
#include <string>

void Read_File()
{
        std::string filename; //declare a string
        std::ifstream file; //declare a file object from the class std::ifstream
        std::vector<std::string> previousGuesses;
        std::string line;



        std::cout << "Please input the filename" << std::endl;
        std::cin >> filename; //prompt user to initialize the string filename

        file.open(filename);

        while(!file.eof())
        {
                std::getline(file, line);
                previousGuesses.push_back(line);
        }

        file.close();

        for(int i = 0; i < previousGuesses.size(); i++)
        {
                std::cout << previousGuesses[i] << std::endl;
        }

}
```

How could we improve/safeguard the code?

Classes and Structs

- Recall that our vectors had some functions associated with them: push_back(), size(). Vectors are a class, meaning any object created from them will have the same associated functions.
- You can create your own classes with their own properties/functions.
- Instead of creating a class, we will create a struct (very similar idea, difference is how we call them).
- Structs are useful if you require a custom data type to meet a specific purpose.

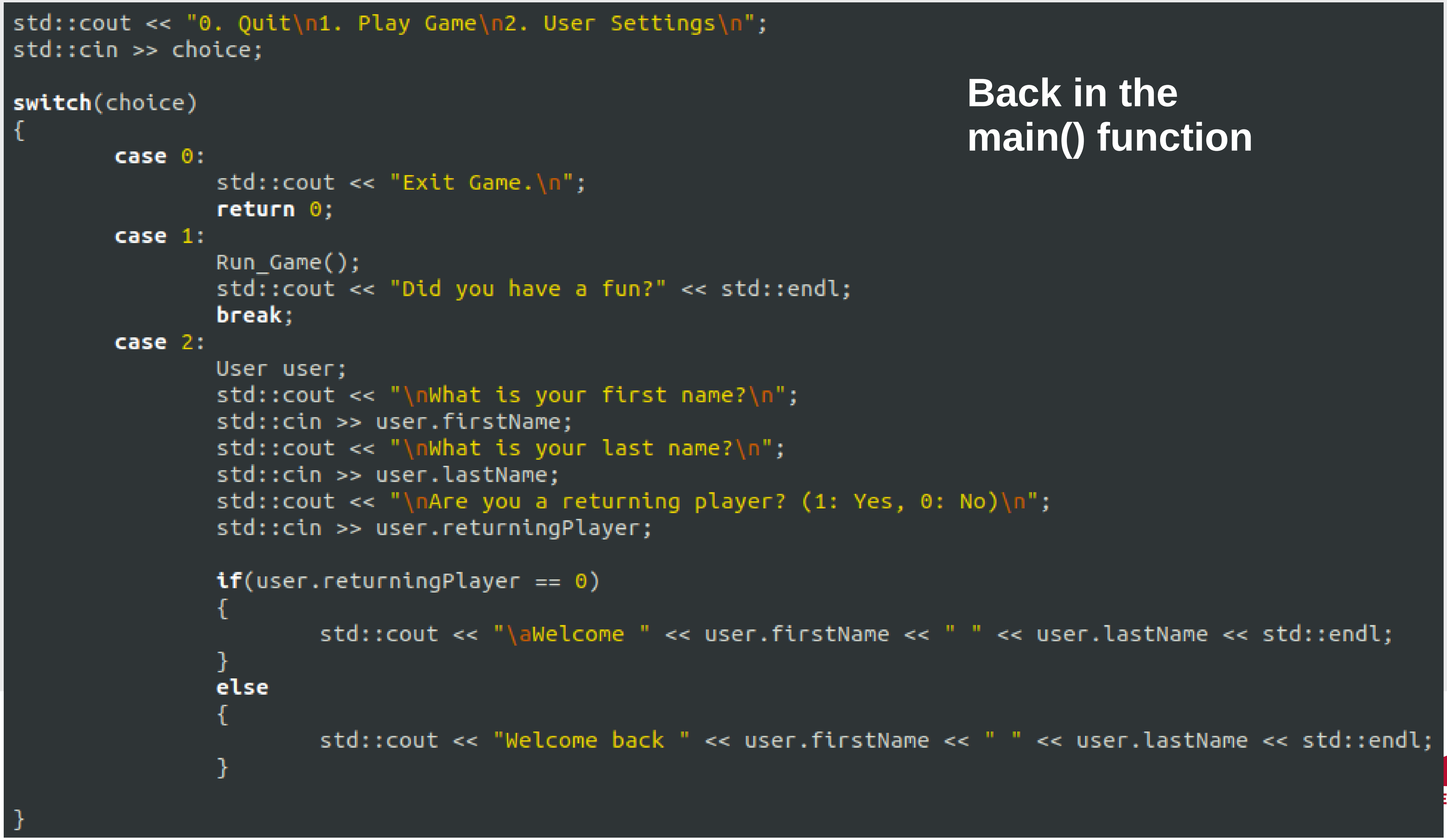


```
struct User
{
        std::string firstName, lastName;
        bool returningPlayer;

};
```

NOTE: SEMICOLON AFTER
BRACE

Using our struct

- Once we declare a variable as a custom struct, we can call the different attributes of our struct. See the following.

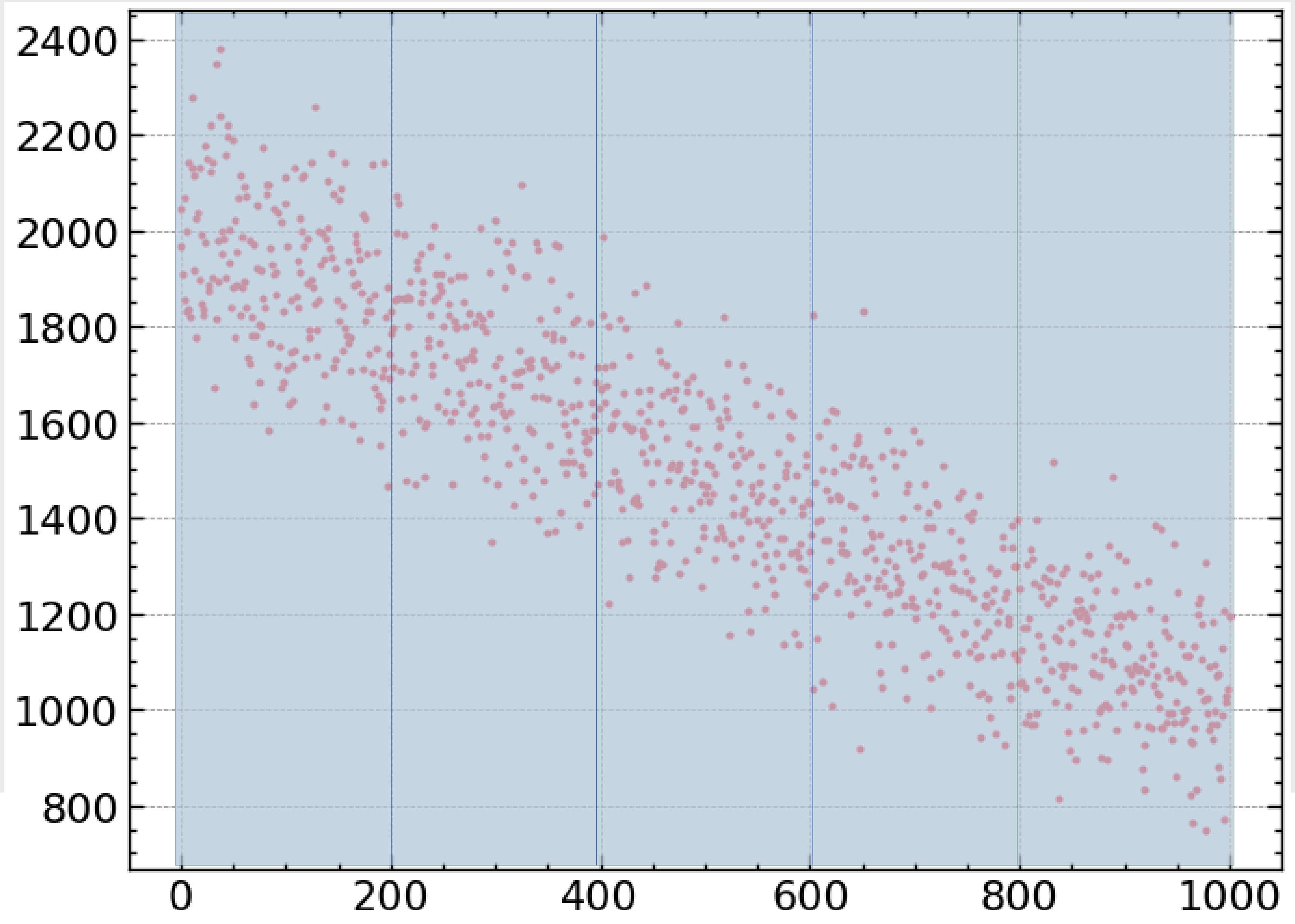


Back in the main() function

```
std::cout << "0. Quit\n1. Play Game\n2. User Settings\n";
std::cin >> choice;

switch(choice)
{
        case 0:
                std::cout << "Exit Game.\n";
                return 0;
        case 1:
                Run_Game();
                std::cout << "Did you have a fun?" << std::endl;
                break;
        case 2:
                User user;
                std::cout << "\nWhat is your first name?\n";
                std::cin >> user.firstName;
                std::cout << "\nWhat is your last name?\n";
                std::cin >> user.lastName;
                std::cout << "\nAre you a returning player? (1: Yes, 0: No)\n";
                std::cin >> user.returningPlayer;

                if(user.returningPlayer == 0)
                {
                        std::cout << "\aWelcome " << user.firstName << " " << user.lastName << std::endl;
                }
                else
                {
                        std::cout << "Welcome back " << user.firstName << " " << user.lastName << std::endl;
                }

}
```

Time to collaborate!

- Download the text file titled dataFile.txt
- Write a script that accomplishes the following tasks:
  - Create a function that reads in the data.
  - Create a function that calculates the average of the entries 0 to 199, then 200 to 399 entries, and so on until entry 999. There will be 5 averages. Subtract the average from each corresponding 200 entries (e.g., subtract the first average from the first 200 entries, subtract the second average from the second 200 entries...).
  - Create a function that outputs the modified data to a text file.
  - Call these three functions from the main function
- Afterwards, create a short python script that visualizes the data.



- Calculate the average within each window.
- Subtract this value from each data point in the corresponding window.
- Return these values to a textfile.
- Visualize your results in python.

What's next?

- There are many topics to continue to explore in C++. Here is a **small** list of some places to go next:
  - Classes and Structs
  - Passing by reference and pointers
  - Constructors and Destructors
  - Additional C++ libraries
  - Header file + implementation file + main file -> makefile
  - Make your own C++ scripts!
  - Create your own namespaces
  - Root!?!
  - Inheritance and polymorphism???

Queen's UNIVERSITY