



**INFINITE
RECHARGESM**

HEPHAESTUS HELPS C++ CODING MANUAL



TABLE OF CONTENTS

Introduction	4
Installing C++ Development Tools	6
Making Comments	7
Declaring Libraries	8
Namespaces.....	9
Variables.....	10
Arrays	12
Operators.....	15
If-Statements	19
Switch Statements.....	20
For Loops	22
While Loops	24

Do-While Loops	27
Functions	30
Parameters	33
Classes	35
Objects	39
Constructors	41
Separating Classes	43
Destructors	48
Pointers	51

HEPHAESTUS
ROBOTICS TEAM
— FRC #5390 —

WHAT IS PROGRAMMING?

Programming is the communication of specified instructions to an electrical system to produce a desired outcome. Instructions are delivered to the electrical system by **programming languages**. In FRC (First Robotics Championships), we are only permitted 2 **programming languages**; C++ and Java.

WHAT IS A PROGRAMMING LANGUAGE

Each **programming language** is used for a different task but they all accomplish one main goal, to communicate to an electrical system. In short, all of the information derived from the words we use to write a computer program is turned into **1's** and **0's**, or binary code, for the computer to understand. Every programming language is interpreted by its **syntax** and its **semantics**. The **syntax** is essentially all the "**words**" in a programming language. Each "**word**" conveys a different meaning and can also be used in a variety of ways. Its semantics gives instructions based on all the different possible combinations of **syntax**. Let's start to program!

WHAT IS A COMPILER?

In the last page, we understood that all programming languages are turned into **binary code** for the electrical system to understand. Obviously, this **does not** happen automatically as this is where a **compiler** comes into play. A compiler is basically a translator, that converts our programming languages into binary code for the interpretation of an electrical system. Without the use of a compiler, the electrical system would not understand our programming languages and we would not understand binary code.

WHAT IS USER INTERFACE?

A User Interface (U.I.) is quite simple to understand. When you develop a new technology to be sold commercially, it needs to be both technologically advanced and easy to use. The “easy-to-use” part is where a User Interface (U.I.) can help. Derived from daily objects, a User Interface is made so that anyone can use a particular technology. An example can easily be found on your phone from the buttons that you press, your homescreen that you see everyday and even “the ok-box” seen on messages telling you that your battery is low, all of these are examples of User Interfaces.

Let's set up our own **compiler** and **U.I.** so that we can start programming our robot!

(All of these steps can be found on <https://wpilib.screenstepslive.com/s/currentCS/m/cpp>)

Download the proper installer for Installation: (<https://docs.wpilib.org/en/latest/>)

Since FIRST (For Inspiration and Recognition of Science and Technology) already has a guide, I recommend you go to the link listed. Start from the top link and work your way down, making sure to double-check frequently that you are downloading files needed to program in C++ only.



WHAT ARE COMMENTS?

Comments are parts of the code which are ignored by the computer and act like “notes” solely for the programmer to better comprehend the code displayed. Though comments are not mandatory to write your code, it is a very professional and helpful practice. There are two types of comments that you can use in the C++ language; single-line comments and multi-line comments. This is an example of a single-line comment:

```
// This is what a single line comment looks like in C++
```

Single-line comments are used only to convey small pieces of information, they are very useful and can clear up any misunderstandings.

Multi-line comments, as shown below, are used to convey more information, you could use it to explain what a certain part of your program does. This is an example of a multi-line comment:

```
/*  
  
    This is what a multi-line comment looks  
    like in C++.  
  
*/
```


WHAT IS A LIBRARY?

A library is an addition to your computer program that allows you to access pre-written pieces of code to perform specific functions. The highlighted is an example of a library in a piece of code.

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

int main() {
    cout << "Hello, World!\n";
    return 0;
}
```

Adding a library is made up of 2 parts:

- An “#include” statement
- The name of the library in between “<>”

WHAT IS A NAMESPACE?

An analogy would be best to define the meaning of namespaces. Say in a class you have 2 students named "Mark". To differentiate them, you would want to find out their last names, interests, hobbies. The declaration of a namespaces acts as a differentiation:

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

int main() {

    cout << "Hello, World!\n";
    return 0;
}
```

WHAT HAPPENS WITH NO NAMESPACE?

You'll have to attach "std::" to your code:

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

WHAT ARE VARIABLES?

From math, variables are essentially symbols which are given some sort of value. So whenever you place that symbol in your code, that value is assumed instead of the symbol.

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

int main() {
    int apple = 5;
    cout << apple;
    return 0;
}
```

Here we can see two new things. "int apple = 5;" and "cout << apple;". Let's break down the first statement, "int" is an integer. Meaning any whole number, negative or positive and including 0. There are 5 types of variables, int, double, float, string, boolean. Floats are numbers with decimal points, doubles are essentially the same thing but allows more decimal points. A string is a variable that holds letters instead of numbers. Boolean variables hold either a 1/0 indicating a true/false value. The "apple" refers to the

name of the variable. The "5" refers to the value that the variable "apple" holds. The ";" is basically ending your statement, similar to a period in english. In the statement "cout << apple;", the "cout" stands for "console out" meaning your printing something to the console. The "<<" is what you use to refer to the next value. "apple" is the name of your variable which has the value of "5". So in your console you should see the number 5:

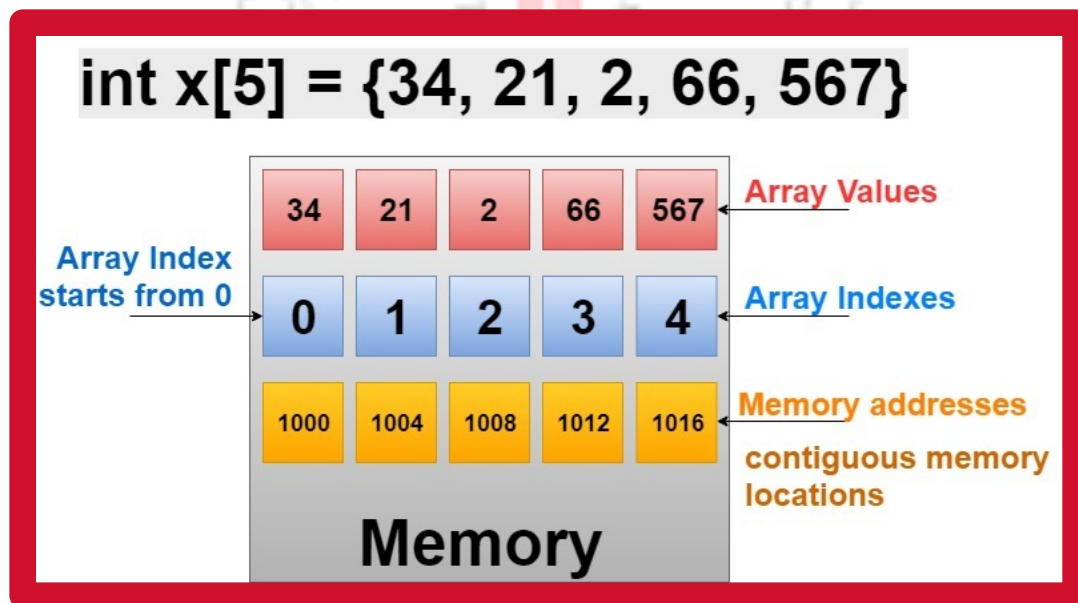
5
Program ended with exit code: 0

The reason is says "Program ended with exit code: 0" is because in my code is used the statement "return 0;". Which basically means, terminate the program.



WHAT ARE ARRAYS?

An array, simply put, is a collection of elements, each identified by one array index or key. The following image will help me explain this concept:



This might look like a lot of random information but don't be intimidated, this concept is very easy to understand. So, let's break down the first statement "int x[5] = {34,21,2,66,567}". The "int" means that all the elements, or values inside the curly brackets, are going to be integers. The "x" refers to the name of this array, but you can name it anything. The "[5]" refers to the amount of values that this array will have, or the number of integers inside the curly brackets. The "{34,21,2,66,567}" refers to the

individual values separated by commas. So for example, the first value of this array is "34", follow by "21", then "2" and so on. Let's look at the chart, is showing what I explained earlier, that the array values are

"34, 21, 2, 66, 567". Now, you might be asking yourself, why do the array indexes count from "0" when you start counting at "1". In programming, the way machines count is that they start from "0". Which is why "34" would not actually be referred to the first values, rather the 0 place value. The value "21" would actually referred to as the first value in that array. The last row, memory adressed do not affect us right now so we will not worry about them. How would this look like in code:

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

int main() {
    int apple[5] = {34, 21, 2, 66, 567};
    return 0;
}
```

Now what would we do if we wanted to print the beginning value of this array:

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

int main() {
    int apple[5] = {34, 21, 2, 66, 567};
    cout<<apple[0]<<endl;
    return 0;
}
```

What is happening in that image? Well, we know what "cout" means, simply "Console out", essentially printing to the console. What does "apple[0]" mean? We first type the name of the array we are interested in, in this case, "apple". Then we specify which value we want, well, we want the beginning value. So would it not be "apple[1]"? No, because the machine begins counting at "0", not "1". So you should type "apple[0]". Which value do you think will be printed to the console?

34

Program ended with exit code: 0

So the value printed was "34" as it was the beginning value of the array "apple".

WHAT ARE OPERATORS?

An operator is a symbol that tells the compiler to perform a mathematical or logical task with the program. They can be divided into two categories: arithmetic operators and relation operators.

WHAT ARE ARITHMETIC OPERATORS?

Arithmetic operators are symbols that tell the compiler to do something mathematical with the program. There are quite a few arithmetic operators: “-,+,*,/,%,++,--”. Each of these operators separated by a comma does something different:

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

int main() {$
    int apple = 5;
    int banana = apple-2;
    int orange = apple+10;
    int kiwi = apple*3;
    int grape = apple/5;
    int mango = apple%2;
    int watermelon = apple++;
    int pineapple = apple--;
    return 0;
}
```


In the last image we have established that there is a variable which is an integer whose name is apple and whose value is 5. Now we see different variables which are using the value of the variable named apple. In the variable "banana" we see the value of "apple" being subtracted by 2. The value of "banana" will then be 3. In the variable "orange" we see 10 being added to the value of "apple". The value of "orange" will then be 15. In the variable "kiwi", we see that the value of "apple" being multiplied by 3. In the variable "grape", we see the value of "apple" being divided by 5, which is essentially, 10 being divided by 5, giving us the value of 2. For variable "mango", we will received the value of 0. The reason is that 10 divides perfectly into 5, leaving no remainders, therefore, we get 0. In the variable "watermelon", we see "apple" with "++", which means add 1. So apple would have 1 added to it, giving "watermelon" the value of 6. In the variable "pineapple", we see "apple" and "--" next to it, which means to subtract 1. Leaving "pineapple" with the value of 4.



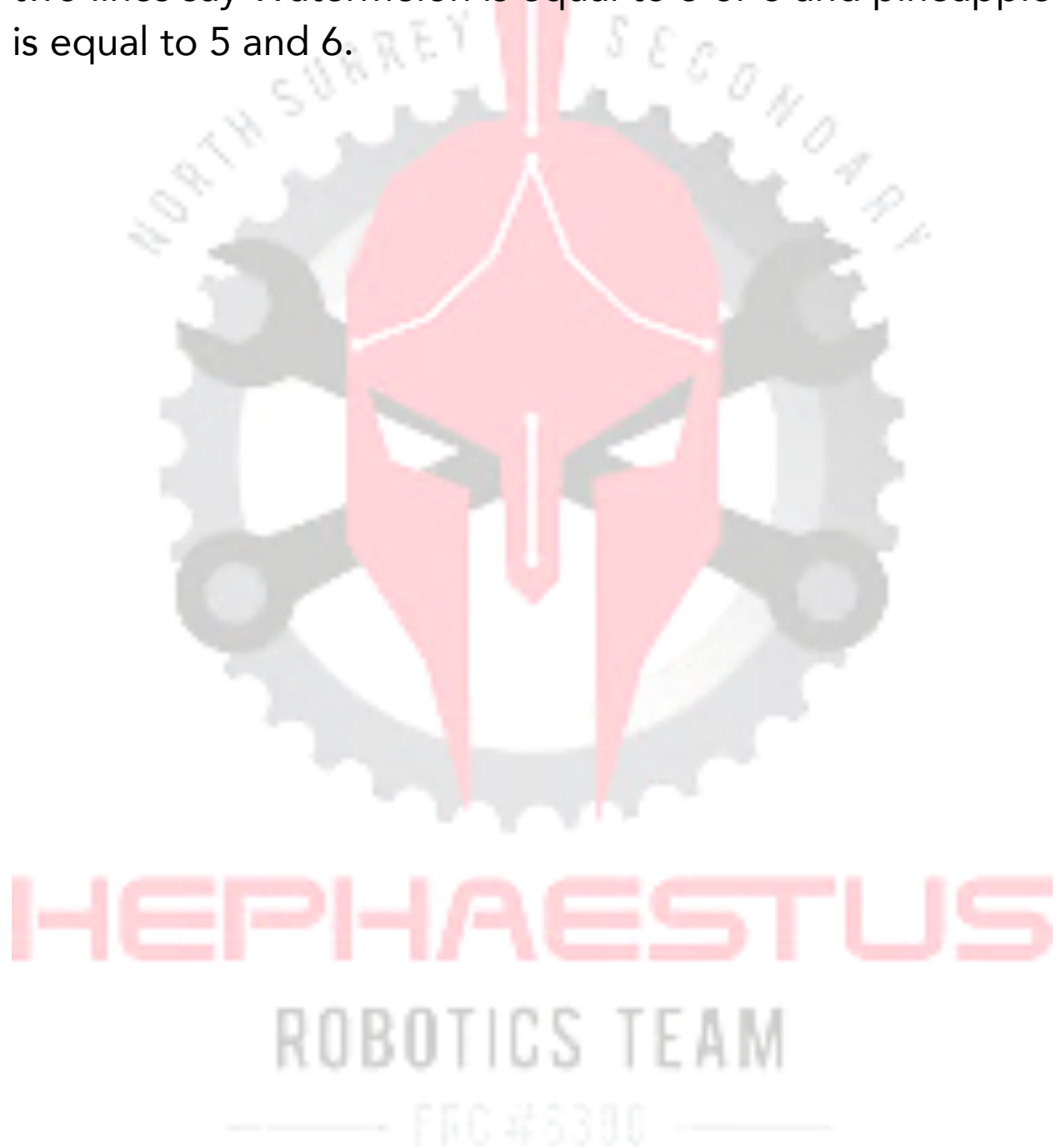
WHAT ARE LOGICAL OPERATORS?

Logical operators in C++ are symbols that tell the compiler to perform logical commands with the code. The image below will display different types of logical operators.

```
apple == 5
banana != 5
orange > 5
kiwi < 5
grape >= 5
mango <= 5
(watermelon = 5) || (watermelon = 6)
(pineapple = 5) && (pineapple = 6)
```

Let's look at each operator line by line. The first line is saying that the value of apple is 5. In C++, "!" is the symbol for not, therefore, banana is not equal to 5, this is the easiest way to think about the "!" symbol. What actually happens is that the compiler will check "banana" is it is 5, and then return a false value. If "banana" equals any other number, it will return a true value. From math class you may remember ">" and "<". The third line say that orange is greater than 5, the next line is the opposite, kiwi is less than 5. The next two lines are similar to the previous two lines. Grape is greater than or equal to 5, likewise, mango is less than or equal to 5. Next we have two unique

symbols. In C++, "||" is the symbol for "or", "&&" is the symbol for "and". What actually happens in the compiler is that "watermelon" is checked, and a true or false value is returned, the same applies for the "&&" symbol. The last two lines say Watermelon is equal to 5 or 6 and pineapple is equal to 5 and 6.



WHAT ARE IF-STATEMENTS?

If-statements are extremely usefully for performing a function especially, if you want them done after a particular condition. Well how do you set up an if-statement?

```
if (variable)(operator)(value){  
    (then perform this function)  
}
```

We already know what variables, operators and values are. But, how would this look like in a program?

```
int apple = 5;  
  
if(apple == 5)  
    cout << "Apple is 5" << endl;  
}
```

In this snippet of code, the end result would be "Apple is 5". The compiler looks at the first statement which states that apple is 5. Next it will read the condition, and since apple is equal to 5, the compiler will proceed with the next code, to print "Apple is 5".

WHAT ARE SWITCH-STATEMENTS?

Switch statements are essentially a replacement for long if-statement chains. If a variable in your program changes into many other states, then switch statements are vital. The following is an example of a switch statement:

```
int main() {  
    int dice = 5;  
  
    switch (dice) {  
        case 1:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        case 2:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        case 3:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        case 4:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        case 5:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        case 6:  
            cout<<"Dice is equal to "<<dice;  
            break;  
  
        default:  
            cout<<"No proper value for dice";  
            break;  
    }  
}
```

The image above displays how a switch statement works. In the function "Int main", we have a variable of an integer type, named dice whose current value is 5. In the switch statement we see that the value whose value will be switch is "dice". There are six cases for all the values of our variable "dice". In this case, the result of this program will be "Dice is equal to 5" in the compiler. This is because your variable dice is equal to 5, which corresponds to case 5. In this case, the computer is told to print "Dice is equal to " followed by the current value of the variable "dice". You may be thinking what the purpose of the "default" line at the bottom of all the cases, which is required in every switch statement. This is incase the variable "dice" gets a value other than integers from 1-6. Say "dice" is equal to 7, then the compiler would display "No proper value for dice" as the dice value does not correspond to any of the cases.



WHAT IS A FOR-LOOP?

For loops are pieces of code that specializes in repetition. If you want some code to be repeated a certain number of times, you would use a for loop. Let's take a look at for loops in an actual program.

```
#include <iostream>
using namespace std;
int main() {
    for (int Age = 30; Age>0; Age--) {
        cout<<Age<<endl;
    }
    return 0;
}
```

The for loop may look daunting at first, but the actual logic behind that statement is not hard to understand. Firstly, in the for loop, we start with a variable with the integer type, named "Age" which has a value of 30. This is the initialization part of the statement, in other words, where you give variables a specific value. Secondly, we see "Age" is greater than 0. This part of the for loop is the conditional part, in this case, "Age" is greater than 0, which means that the conditional is true. Otherwise this statement is

abandoned and the code moves on to the next line. Thirdly, we have "Age" with "--" behind it. If we remember from past lessons, this means that "Age" get subtracted by one. Lastly, between the curly brackets, we see a "cout" statement that we have worked with before. In the code snippet above, this statement is just telling the compiler to print out the variable "Age". After the compiler goes through this process once, it will go back to the start of the for loop. In our case, the compiler will ignore the statement where we give "Age" a value of 30 but instead check to see if "Age" is greater than 0. After the for loops runs once, "Age" will have a value of 29, so the conditional will be met and the code will continue running. The value of "Age" will be reduced by one and the new value of "Age" will be printed, "Age" will now be 28. Then the compiler will go back to the start of the for loop. This process will continue until "Age" is equal or less than 0.

```
for (initialization; condition; increment) {  
    statements  
}
```

WHAT IS A WHILE-LOOP?

Let's first take a look to see what a while-loop is made of.

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

int main(){

    while (condition) {
        statements
    }
```

A while-loop is a special command in C++ where the code inside the while-loop will continuously run until the conditional initially set to be true, is false. In the picture above, we see that a while statement is followed by a condition put in brackets, then inside of the curly brackets, there is space for statements. Let's look at an example of a complete while statement.

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

int main(){

    int x = 5;

    while (x > 0) {
        cout<<"X is equal to "<< x <<endl;
        x--;
    }

}
```

Let's analyze the code above. We observe that there is a variable named "x", which is of type integer whose value is 5. In the while statement, we see a conditional that says that the variable "x" is greater than 0. Inside the curly brackets, we see a "cout" statement which will print the following text: "X is equal to ", followed by the value of the variable "x". The last statement inside the curly brackets, is "x--", which from past lessons means that the value of "x" is subtracted by 1. Starting from the top, the compiler reads that the value of "x" is 5. Then the while statement says that while "x" is greater than 0, print out "X is equal to " followed by the value of "x". Then "x" is subtracted by 1, which leaves "x" to have a value of 4. The compiler will then start at the top of the while statement.

Since we know now, that the value of "x" is 4, the while statement will continue and print "X is equal to ", followed by the value of "x" which is 4. Then "x" will have a value of 3. By now, I am sure you are starting to sense a pattern. This loop will continue until the conditional at the top is unsatisfied, in other words, when "x" is either equal to or less than 0. Now, let's take a look at the output of this code.

```
X is equal to 5
X is equal to 4
X is equal to 3
X is equal to 2
X is equal to 1
Program ended with exit code: 0
```

This should be the expected outcome. The variable "x" started at the value of 5 and ended at the value of 1. The variable "x" could not go down to 0, this is because 0 is not larger than 0. Each time, we saw 1 being subtracted from "x", which is why it started at 5, then went down to 4, 3 and all the way until the last time which the condition was satisfied, that "x" is bigger than 0. This final loop was only possible if "x" is equal to 1.

WHAT IS A DO-WHILE LOOP?

A variant of the while-loop, let's take a look at the composition of a do-while loop.

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

int main(){

    do {
        statements
    } while (condition);

}
```

The main purpose of a do-while loop is so that the code inside the curly brackets would be performed at least once, regardless of the condition inside the brackets. There are 3 main steps for how the do-while loop is executed. Firstly, the code replacing the "statements", is performed. Secondly, the expression replacing "condition" is tested. If this code's outcome is true, the do-while loop is repeated and the do-while loop continues

until the expression becomes false. If the expression is false, then the do-while code is terminated.
Let's take a look at a do-while loop in action.

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

int main(){


    int number = 10;

    do {
        cout<<number<<endl;
        number--;
    } while (number>0);

}
```

Inside the "int main()" function, we observe that there is a variable, called "number" of an integer type, with a value of 10. Inside the "do" part of the do-while loop, the value of "number" is print to the console. Then, the value of number is subtracted by one. Finally, in the "while" part of the do-while statement, we see that the value of "number" is indeed more than 0, therefore this loop will continue

until "number" is less than 0. Now, let's take a look at the output of the console.



```
10
9
8
7
6
5
4
3
2
1
Program ended with exit code: 0
```

As expected, the console printed out the initial value of "number" which was 10. The value of "number" was then decrease by 1, which led to the new value of "number" being 9. Since 9 is more than 0, the do-while loop continued until the value of "number" was lower than 0.

HEPHAESTUS
ROBOTICS TEAM
— FRC #5390 —

WHAT ARE FUNCTIONS?

Functions are simply groups of code that you may have similar uses, which you refer to multiple times, using the name of the function. The rule is that you need to put a statement containing the type of function, followed by its name and two brackets. The reason is that C++ works from top to bottom, if you did not have that initial statement, the console would be unaware of that function.

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

void myFunction();

int main(){

    myFunction();
    return 0;

}

void myFunction(){

}
```

Let's take a look at using a function in our code.

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

void myFunction();

int main(){
    myFunction();
    return 0;
}

void myFunction(){
    cout<<"Hephaestus 6390 Functions"<<endl;
}
```

In the code above, we observe a "cout" statement inside of a function, whose type is "void" and whose name is "myFunction". This function then has a statement at the top of the code, letting the console know that there will be a function in the rest of the code. The function is then called in the "main" function "int main()". "int main()" is a function created in every C++ program which is where all the code will be run. We should now expect the console to print "Hephaestus 6390 Functions".

```
Hephaestus 6390 Functions  
Program ended with exit code: 0
```

Just as we predicted, the console printed out “Hephaestus 6390 Functions”.



WHAT ARE PARAMETERS?

We know that functions are groups of code that have similar uses, but did you know that you can pass some information through a function? You can pass through information through functions using parameters. Here is how you can pass information through a function using parameters.

```
void Function(parameter 1, parameter 2, parameter 3){  
    // Code that you want to run  
}
```

Let's suppose we want to name different types of juice. Since we already know the second part of their name will be "Juice", we just need to adjust the first part of their name, indicating the type of juice.

```
void Function(string Juice_name){  
    cout<<Juice_name<<" Juice "<<endl;  
}  
  
int main() {  
    Function("Apple");  
    Function("Orange");  
    Function("Mango");  
}
```

The expected output should be the names of the Juice passed through the function. In our example we used "Apple", "Orange" and "Mango". Let's take a look at the output:

```
Apple Juice  
Orange Juice  
Mango Juice  
Program ended with exit code: 0
```

As we expected, we we will see "Apple Juice", "Orange Juice" and "Mango Juice" as our output.



WHAT ARE CLASSES?

In the previous lesson, just as how functions are group of code which have similar uses, classes are groups of functions that have similar uses. Let's look at the composition of a class.

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

class class name {
    instance variables

public:
    member functions

private:
    member functions

protected:
    member functions

};

int main(){

}
```

To declare a class, you must write "class" followed by the name of your choice, then curly brackets. Inside the curly

brackets, it is good practice to declare your variables before the access specifiers, in this case “public”, “private” and “protected”. An access specifier is essentially how someone can access your class. If you set the access specifier to “public”, then the content under “public” can be used anywhere in your code. If the access specifier is “private”, this means the contents under “private” cannot be access anywhere outside of the class. If the access specifier is set to “protected”, the contents under “protected” cannot be accessed anywhere outside your code unless the classes are inherited (You will learn about “Inheritance” later on). Under each of the access specifiers are its content or “member functions”. The class is then completed with a semi-colon. Let’s take a look at a class in action.




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

class myClass {
    int x;
    int y;
    int z;

public:
    void printX(){
        int x = 5;
        cout << x << endl;
    }
private:
    void printY(){
        int y = 10;
        cout << y << endl;
    }
protected:
    void printZ(){
        int z = 15;
        cout << z << endl;
    }
};

int main(){

}
```

In this class named "myClass", I have 3 variables, all of an integer type, named "x","y" and "z". The first access specifier is "public", where I have a function of type "void", named "printX". Inside "printX", I have initialized the variable named "x" of an integer type to a value of 5. Next, I have a "cout" statement, which is supposed to print out the value of "x" to the console. The second access specifier is "private", where I have a function of type "void", named "printY". Inside "printY", I have initialized the variable named "y" of an integer type to a value of 10. Next, I have a "cout" statement, which is supposed to print out the value of "y" to the console. The last access specifier is "protected", where I have a function of type "void", named "printZ". Inside "printZ", I have initialized the variable named "z" of an integer type to a value of 15. Next, I have a "cout" statement, which is supposed to print out the value of "z" to the console.



WHAT ARE OBJECTS?

Simply put, objects are the way you access the contents of your class from the "int main()" function. To make an object, you must first write the name of the class of your choice, followed by the preferred name of your object. In the next line, you can access your functions in your class through the name of your object, followed by the dot operator and then the name of the function.

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

class myClass {
    int x;
    int y;
    int z;

public:
    void printX(){
        int x = 5;
        cout << x << endl;
    }
private:
    void printY(){
        int y = 10;
        cout << y << endl;
    }
protected:
    void printZ(){
        int z = 15;
        cout << z << endl;
    }
};

int main(){
    myClass myObject;
    myObject.printX();
}
```

Since we made an object of class "myClass", whose name is "myObject". When can now access the function "printX()", whose access specifier is "public". Since the objective of the function "printX()" is to print the value of variable "x" to the screen. We should expect the number 5 to be print to the console.

```
5  
Program ended with exit code: 0
```

Just as we suspected, the number 5 was print to the console.



WHAT ARE CONSTRUCTORS?

Constructors are functions that get called as soon as an object corresponding to a particular class is made which automatically executes code. Before, we would have to use an object followed by a dot operator and then the name of the function. Instead, you can make a constructor and as soon as the object is made, the constructor executes its contents.

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

class myClass {
    int x;
    int y;
    int z;
public:
    myClass(){
        cout<<"Code that constructor will automatically print"<<endl;
    }
    void printX(){
        int x = 5;
        cout << x << endl;
    }
private:
    void printY(){
        int y = 10;
        cout << y << endl;
    }
protected:
    void printZ(){
        int z = 15;
        cout << z << endl;
    }
};

int main(){
    myClass myObject;
    myObject.printX();
}
```

Under the access specifier "public", you may see a function with the same name as the class, that is the constructor for this class. The constructor always has the same name as its class. Inside the class, there is a "cout" statement to print text to the console. We should now expect the text for the constructor to print before the value of variable "x" from function "printx()". Let's take a look at the output of the console.

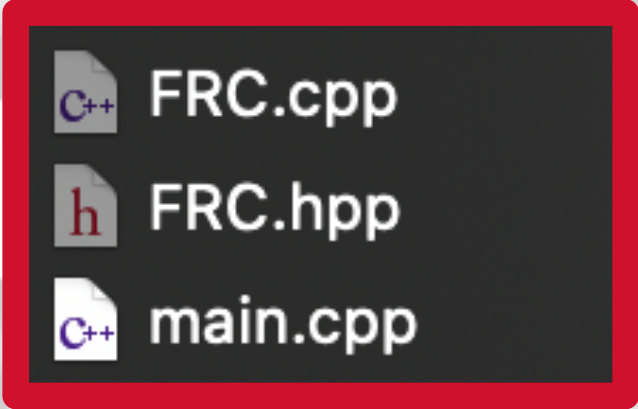
```
Code that constructor will automatically print  
5  
Program ended with exit code: 0
```

Just as we thought, the text from the constructor was printed before the value of variable "x" from public function "printX()".



WHAT IS SEPARATING FILES?

When programming, especially for an upcoming FRC competition, it is good practice to separate your class into different files. Programmers do this to organize their files so that they do not have to continuously keep scrolling to find the code they need to update or fix. When you make a new C++ file into your program, the endings of the files should look something like this. The only difference is that the ".hpp" file should instead be ".h".



The image shows a red-bordered box containing three file icons and their names. The first icon is a C++ file icon (a document with 'C++' in the top left corner) next to the text 'FRC.cpp'. The second icon is a header file icon (a document with 'h' in the top left corner) next to the text 'FRC.hpp'. The third icon is a C++ file icon (a document with 'C++' in the top left corner) next to the text 'main.cpp'.

The main file where I wrote my code was the "main.cpp" files, you may have that file named something different. The files which I want you to focus on are the "FRC.cpp" file and the "FRC.hpp". As I mentioned, the "FRC.hpp" should just be "FRC.h" for you. Anyways, when I create a new C++ file, the new files that were added were the "FRC.cpp" and the "FRC.hpp". The "FRC.hpp" is referred to as a header file and the "FRC.cpp" is referred to as the C++ source file. The header file is where we are going to

be putting all of the titles of the functions and classes and also where we will be declaring our variables. The C++ source file is where we are actually going to be building all the code itself. When we click on the "FRC.hpp" file or "FRC.h" file we should see something resembling the following image.

```
#ifndef FRC_hpp
#define FRC_hpp

class FRC{
    public:
        FRC();
    protected:
    private:
};

#endif /* FRC_hpp */
```

The main text to focus on is the FRC class, since we have covered class you should know what all this text means, if not, please refer back to the lesson where we covered classes. Now, if we click on the "FRC.cpp" file, we should see something resembling the following image.

```
#include "FRC.hpp"
#include <iostream>

using namespace std;

FRC::FRC()
{

}

}
```

Again, the main text to focus on is the text underneath "using namespace std;". This might look familiar to you as it looks like a constructor but you may be confused by the "::". That is known as the "Binary Scope Resolution Operator", its function simply is to show that the constructor "FRC()" comes from the class named "FRC". If the "::" was missing, the console would not know which class the constructor "FRC()" is related to.

HEPHAESTUS
ROBOTICS TEAM
— FRC #5390 —

Now, to use the code in our "FRC.hpp" and "FRC.cpp" we need to include the header of "FRC.hpp", here is what it will look like.

```
#include <iostream>
#include "FRC.hpp"

using namespace std;
```

In addition to the "#include <iostream>", we include the "FRC.hpp" file by typing "#include "FRC.hpp"". Now, we are ready to start using this class with separated files. In the constructor named "FRC()" in our "FRC.cpp" file, let's create a "cout" statement to test if this is working.

```
#include "FRC.hpp"
#include <iostream>

using namespace std;

FRC::FRC()
{
    cout<<"FRC SEASON 2020"<<endl;
}
```

If you recall, constructors will run their code only once the object for the same class has been created. Let's create an object for the class "FRC" in our "main.cpp" file.

```
#include <iostream>
#include "FRC.hpp"

int main(){
    FRC FRCobject;
}
```

Let's now build the program and run it to see if the code inside the constructor has been read by the console.

```
FRC SEASON 2020
Program ended with exit code: 0
```

Now we know that the code inside the constructor has been read and run.

WHAT ARE DECONSTRUCTORS?

Deconstructors are very similar to constructors, the only difference is that deconstructors are activated when all the objects are being deleted due to the end of a C++ program. Let's find out how to make a deconstructor in our class. Deconstructors are written the same way as constructors expect there is a "~" before the declaration. First, in our "FRC.hpp" file, let's declare a deconstructor.

```
#ifndef FRC_hpp
#define FRC_hpp

class FRC{
    public:
        FRC();
        ~FRC();
    protected:
    private:
};

#endif /* FRC_hpp */
```

You may notice underneath "FRC();", I have written the exact same text underneath with the exception of the "~", which signifies a deconstructor.

Now let's move over to our "FRC.cpp" file where we have to write out the destructor and type what we would like the destructor to do once it has been activated. For the sake of this demonstration, we want the destructor to print out "FRC INFINITE RECHARGE", this is what we would do.

```
#include "FRC.hpp"
#include <iostream>

using namespace std;

FRC::FRC()
{
    cout<<"FRC SEASON 2020"<<endl;
}

FRC::~~FRC(){
    cout<<"FRC INFINITE RECHARGE"<<endl;
}
```

Now we have written what that we have a destructor by the name of "FRC" which comes from class "FRC", which prints out "FRC INFINITE RECHARGE" when it has been

activated. We should now expect "FRC SEASON 2020" followed by "FRC INFINITE RECHARGE" on the next line.

```
FRC SEASON 2020  
FRC INFINITE RECHARGE  
Program ended with exit code: 0
```

Just as we expected, we got "FRC SEASON 2020" followed by "FRC INFINITE RECHARGE" on the next line.



WHAT ARE POINTERS?

Pointers are variable that contain the memory address of another variable, whose value usually looks like a mix of letters and numbers, which is called a hexadecimal. Let's find out how to create a pointer!

First, we need to create a variable because we need to store the memory address of an item. We will create a variable whose name is "points", whose type is integer and whose value is 7. Now to find the memory address of this "points", we need to create a "cout" statement, but, instead of writing just the name of the variable in the "cout" statement, we need to put a "&" before the name of the variable, which should look something like the following picture.

```
#include <iostream>

using namespace std;
int main() {

    int point = 7;
    cout<<&point<<endl;

    return 0;
}
```

The output should be a "hexadecimal" like I mentioned in the first paragraph of this lesson. In other words, the output of the console should be a mix of random letters and numbers.

```
0x7ffeefbfff4b8  
Program ended with exit code: 0
```

Now that we have a reference to compare our pointer to, let's create the actual pointer. To do so, we have to first type the type we want the pointer to be, in this case, we want the "int" type. After you finish specifying the type of pointer we need, type a "*" followed by the name of your pointer. Then, set that equal to the memory address of the variable as we discussed last paragraph. To see if we successfully made a pointer correctly, make a "cout" statement to print out the value of the pointer. This time, when typing the name of the pointer into the "cout" statement, it is not necessary to include a "*" sign as C++ will now automatically recognize the name of your pointer. After following all of these instructions, you should have something similar to the following code.

```
#include <iostream>

using namespace std;
int main() {

    int points = 7;
    cout<<&points<<endl;

    int *pointsPointer = &points;
    cout<<pointsPointer<<endl;

    return 0;
}
```

Now, when you run this program, you should have 2 lines of output, with the exact same “hexadecimal” on each line.

```
0x7ffeebf4b8
0x7ffeebf4b8
Program ended with exit code: 0
```

We can now see that we have successfully made a pointer.

This concludes the C++ handbook for getting started with programming in C++. Thank for reading, we hope this has helped your team. If this book has helped you please shout out our team or if your team has any further questions contact the following social medias:

Instagram: @frc6390

Twitter: @NSS_Hephaestus

Email: robotics.hephaestus@gmail.com

Website: <http://www.hephaestus6390.com/>

Written & Arranged by Rohan Jagtap

Collaborators: Siddarth Sudhir, Mohammed Awwad

Sources:

https://www.w3schools.com/cpp/cpp_intro.asp

<http://www.cplusplus.com/doc/tutorial/functions/>

<https://www.programiz.com/cpp-programming/function>