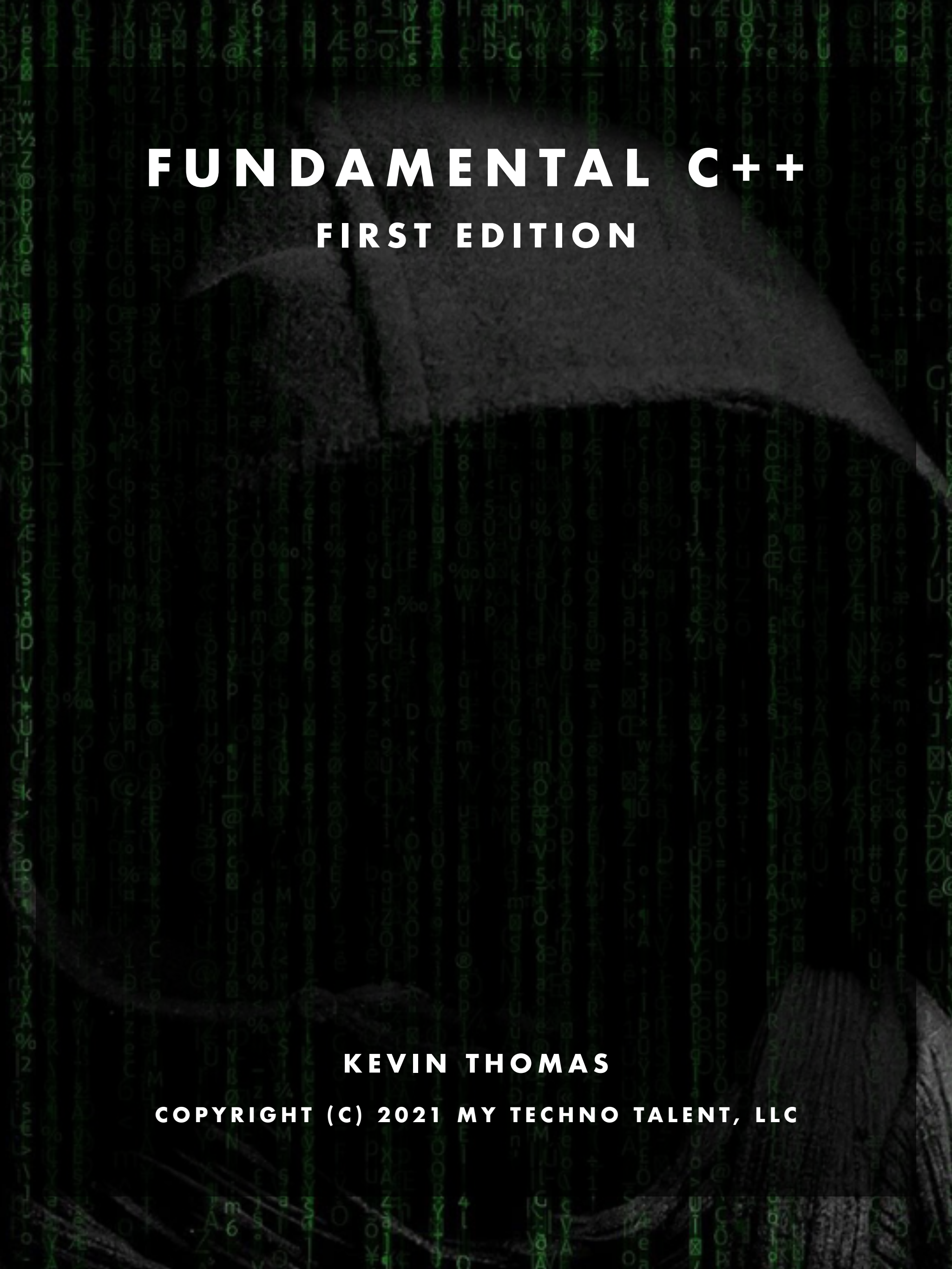

FUNDAMENTAL C++

FIRST EDITION

KEVIN THOMAS

Fundamental C++
First Edition

Kevin Thomas

FORWARD

C++ is one of the most powerful and scalable languages in existence today. C++ is used to create most of our modern web browsers in addition to powerful network security tools like Zeek. C++ also dominates within the gaming community.

We will begin our journey with the basics of C++ and then progress into variables and constants. We will then introduce arrays and modern C++ vectors and dive into the variety of C++ operators. At that point we will discuss proper design and handling of program flow.

We will then cover functions and proper handling of input.

Once we have a grasp on those fundamentals we will introduce the concept of pointers and work our way into OOP classes.

GitHub repo @ https://github.com/mytechnotalent/Fundamental-CPP

CHAPTER 1 - HELLO WORLD

As is tradition in any programming development we will begin with the famous hello world implementation and dive immediately into our first C++ application.

Let's begin by creating a folder for our course and our first project folder.

```
mkdir fundamental_c++
cd fundamental_c++
mkdir 0x0001_hello_world
cd 0x0001_hello_world
```

The first thing we want to do is create our Makefile. A Makefile allows us to properly build our applications in a scalable and proper fashion. We will call it **Makefile** and code it as follows.

```
main: main.cpp
    g++ -o main main.cpp -I.
    strip main

clean:
    rm main
```

Our first application will be **main.cpp** within the **0x0001_hello_world folder** as follows.

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;

    return 0;
}
```

Before we get started I want to share the C++ documentation to reference. It is good practice to review the docs as you begin architecting your designs.

https://en.cppreference.com/w/cpp

It is also important to keep up with the home of the C++ standard on the web.

https://isocpp.org

Let's begin by reviewing the *iostream* library.

https://en.cppreference.com/w/cpp/header/iostream

The iostream standard library header is part of the input/output library and handles basic IO within C++.

We use *iostream* so that we can utilize the libraries *cout* and *endl* objects. Let's start by looking at the *cout* docs.

https://en.cppreference.com/w/cpp/io/cout

The *cout* object controls output to a stream buffer which is STDOUT or standard output which in our case is our monitor.

The *endl* object or end line manipulator inserts a new line character into the output sequence and flushes the input buffer.

https://en.cppreference.com/w/cpp/io/manip/endl

Now that we have an understanding of what these two objects are we can review this code. We start with the *#include <iostream>* which the # is referred to as a preprocessor directive as we now have some familiarity with *iostream* from above.

We then see our main entry point or *main* function which is of an integer type. If everything in main completes successfully we return back an integer of *0* to the operating system.

We then see *std::cout* which std references the standard namespace. This design ensures that we do not have naming collisions. In a larger architecture we may have another *cout* which could be part of *foo::cout* such that if it was called there would be no naming collision with *std::cout*.

The *::* is called the scope resolution operator which denotes which namespace an object a part of in our case the standard name space or *std*.

We use the insertion operator << to insert, *"Hello, World",* which is what we call a string literal, into that stream and therefore send it to stdout, our monitor.

We then see another insertion operator << to insert a new line and flush of the input buffer.

The entire line starting with *std::cout* and ending with the semicolon, *;*, is what is referred to as a statement and all statements in C++ must end with a semicolon.

Finally we *return 0* to the operating system such that there was no error and end the program execution.

Let's now run **make** in our terminal.

We happily see, *Hello, World!*, echoed to our terminal.

Project 1

Your assignment will be to create a new folder **0x0001_about_me** and create a **main.cpp** and a **Makefile** and write a program that will tell us about yourself using what you learned above.

CHAPTER 2 - VARIABLES, CONSTANTS, ARRAYS, VECTORS, STATEMENTS, OPERATORS, STRINGS

A variable is simply a way to store data into memory and manipulate it in some way.

Variables are scoped meaning depending where they are will have a different lifecycle. If a variable is defined in main outside of of a class, function or block it will have global scope. If it is within a class, function or block it will have local scope to each respectively.

There are several primitive or built-in variables within C++.

1. Character Types
2. Boolean Type
3. Integer Types
4. Floating-point Types

Characters

Type Name	Size/Precision
char	At least 8 bits.
char16_t	At least 16 bits.
char32_t	At least 32 bits.
wchar_t	Largest avail char set.

Characters are used to represent single chars and the wider types represent wide char sets.

Let's create our second project folder.

```
mkdir 0x0002_var_const_array_vector
cd 0x0002_var_const_array_vector
```

Let's make our **Makefile** as follows.

```
main: main.cpp
    g++ -o main main.cpp -I.
    strip main

clean:
    rm main
```

Let's make our **main.cpp** file.

```
#include <iostream>

int main()
{
  // char
  char x = 'x';
  std::cout << x << std::endl;

  return 0;
}
```

Here we create our char with single quotes. You must use single quotes when creating a char type.

Here we will simply see an *x* print out as expected.

Boolean

Type Name	Size/Precision
Bool	Usually 8 bits, true or false (keyword).

Booleans are great for setting flag conditions and simply represent true or false. Quite simply zero is false and anything non-zero is true.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // bool
  bool isHappy = true;
  std::cout << isHappy << std::endl;

  return 0;
}
```

Here we create our bool type and it will evaluate to either 0 or 1. In our case we are happy so we will print out a *1*.

Integers

Type Name	Size/Precision
signed short int	At least 16 bits.
signed int	At least 16 bits.
signed long int	At least 32 bits.
signed long long int	At least 64 bits.
unsigned short int	At least 16 bits.
unsigned int	At least 16 bits.
unsigned long int	At least 32 bits.
unsigned long long int	At least 64 bits.

Integers are used to represent whole numbers and have both signed and unsigned versions.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // int
  int y = 42;
  std::cout << y << std::endl;

  return 0;
}
```

Here we create our int type and it will print out a *42*.

Floating-Point

Type Name	Size/Precision	Range
float	7 decimal digits.	1.2 * 10 ^ -38 — 3.4 * 10 ^ 38
double	No less than float / 15 decimal digits.	2.2 * 10 ^ -308 — 1.8 * 10 ^ 308
long double	No less than double / 19 decimal digits.	3.3 * 10 ^ -4932 — 1.2 * 10 ^ 4932

Floating-point numbers represent non-integers and have a mantissa and exponent where you will find scientific notation denoted.

Precision is measured by the number of digits in the mantissa and precision and size are completely compiler dependent.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  std::cout.precision(17);
  double zz = 42.111111111111111;
  std::cout << zz << std::endl;

  return 0;
}
```

Here we create our int type and it will print out a *42.111111111111114* as we see we are losing precision in the last digit.

Constants

Constants are like variables as they have names and take up memory and are usually typed however there value can't change once declared.

There are several types of constants. We will cover three here.

1. Literal Constants - 42;
2. Declared Constants - const int MAGIC_NUMBER = 42;
3. Defined Constants - #define MAGIC_NUMBER 42; // not used in modern C++

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // constants
  const int MAGIC_NUMBER = 42;
  std::cout << MAGIC_NUMBER << std::endl;

  return 0;
}
```

Here we create our declared constant and it will print out a *42*.

Arrays

An array is a compound data type and all elements are of the same type.

Arrays are of fixed size and store all values in memory contiguously. The first element is at index 0 and the last element is at index size - 1.

There are NO BOUNDS CHECKING in arrays.

Arrays are not good practice in modern C++ as we will use vectors as they offer bounds checking and dynamic sizing in addition to significantly greater flexibility and design.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // arrays
  int favorite_numbers[2] = {42, 7};
  std::cout << &favorite_numbers << std::endl;
  std::cout << favorite_numbers[0] << std::endl;
  std::cout << favorite_numbers[1] << std::endl;

  return 0;
}
```

Here we create our array and see that the first value printed is a memory address which is in the stack. This is because an array is nothing more than a pointer to memory where the name of the array simply holds the first starting address of the array in memory.

The 1st element or [0] holds *42* and the 2nd element [1] holds *7*.

Vectors

Vectors are part of the C++ Standard Template Library or STL and as stated are more flexible than a traditional array and have functions such as find, sort, reverse and many more.

Vectors are part of the standard namespace and you begin by including the #include <vector> to init.

Let's update our **main.cpp** file.

```
#include <iostream>
#include <vector>

int main()
{
  // vectors
  std::vector<int> favourite_numbers;  // always init to 0, no need to init
  favourite_numbers.push_back(42);
  favourite_numbers.push_back(7);
  std::cout << &favourite_numbers << std::endl;
  std::cout << favourite_numbers.at(0) << std::endl;
  std::cout << favourite_numbers.at(1) << std::endl;
  std::cout << favourite_numbers.size() << std::endl;
  int da_fav_num = 42;
  std::vector<int>::iterator it;
  it = std::find(favourite_numbers.begin(), favourite_numbers.end(),
da_fav_num);
  if (it != favourite_numbers.end())
  {
    std::cout << da_fav_num << " found at position: ";
    std::cout << it - favourite_numbers.begin() << std::endl;
  }
  else
    std::cout << da_fav_num << " not found...";

  return 0;
}
```

Here we create our *favourite_numbers* vector and add two values into it which are *42* and *7*. We then print out the memory address which is the address on the stack of the beginning of our vector.

If you compare the value of *favourite_number*, which due to ASLR, address space layout randomization, will be different on each run, to the same run of the array, *favorite_number*, you will clearly see that the stack grows downward.

We finally use the find function to search for *da_fav_num* and see that it is actually found and at position *0*.

There is SO MUCH MORE you can do with vectors but as this is a fundamentals course I will keep it brief so that we can continue our C++ development however I encourage you to review the docs and take the time to really use and scale vectors in your upcoming projects.

Statements

Statements are a complete line of code that performs an action and is usually terminated with a semi-colon.

There are many types of statements. Here are just a few.

1. Declaration - char x;
2. Assignment - favorite_number = 42;
3. Expression - 42 + 7;
4. Assignment - x = 42 + 7;
5. If = if (favorite_number == 42) …

Operators

Operators build up expressions and are unary, binary or ternary.

1. Assignment - x = 42;
2. Arithmetic - 42 + 7;
3. Increment - x++;
4. Decrement - x—;
5. Relational - x >= 42;
6. Logical - &

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // operators
  int result = 0;
  result++;
  std::cout << result << std::endl;

  return 0;
}
```

Here we show a simple example of using an operator that will print out *1*.

Strings

C++ strings are different from C-style strings which build up char* into strings. To use strings you must use #include <string> and they are in the standard namespace.

C++ strings are contiguous in memory and are dynamic in size and work with input and output streams.

C++ strings have useful member functions as well and are generally safer to work with than C-style strings.

Let's update our **main.cpp** file.

```
#include <iostream>
#include <string>

int main()
{
  // strings
  std::string secret_number = "forty-two";
  std::cout << secret_number << std::endl;
  secret_number.at(4) = 'e';
  std::cout << secret_number << std::endl;
  std::cout << secret_number.length() << std::endl;
  std::cout << (secret_number == "forte-two") << std::endl;
  std::cout << secret_number.substr(3, 8) << std::endl;
  std::cout << secret_number.find("te-two") << std::endl;
  std::cout << secret_number.erase(0, 3) << std::endl;
  secret_number.clear();
  std::cout << secret_number << std::endl;

  return 0;
}
```

Here we create a *secret_number* string and assign it, *"forty-two"*. We then change the value at index 4 to *'e'*. We then call the length function to get our string length. We then use a comparison operator to see if *secret_number* and *"forte-two"* are equivalent which returns a *1* as they are. We then print out a slice of our string which returns *"te-two"*. We then call the find function to see if we can find the presence of chars in our string as it returns back the index of the first char if found which is *3*. We then change our string and permanently remove the 0 to 3 elements and the string becomes *"te-two"*. Finally we clear the string and we find an empty string printed.

Project 2

Your assignment will be to create a new folder **0x0002_my_fav_nums** and create a **main.cpp** and a **Makefile** and write a program that creates a vector of int where you create elements, manipulate them with the built-in vector functions and display each step using what you learned above.

CHAPTER 3 - PROGRAM FLOW

In this chapter we are going to cover program flow.

Let's create our third project folder.

```
mkdir 0x0003_program_flow
cd 0x0003_program_flow
```

Let's make our **Makefile** as follows. Notice we target C++ 11 here.

```
main: main.cpp
    g++ -std=c++11 -o main main.cpp -I.
    strip main

clean:
    rm main
```

If

With the if statement we either evaluate to one of two conditions.

1. If the expression is true, execute the block.

2. If the expression is false, skip the block.

Let's make our **main.cpp** file.

```
#include <iostream>

int main()
{
  // if
  int x = 10;
  if (x >= 10)
  {
    std::cout << "is >=" << std::endl;
  }
  else if (x < 10)
  {
    std::cout << "is <" << std::endl;
  }
  else
  {
    std::cout << "not a number" << std::endl;
  }

  return 0;
}
```

Here we evaluate to >= as only the first statement is true.

Switch

With the switch statement you can use a variable to create a menu-based selection.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // switch
  char menu_item = 'a';
  switch(menu_item)
  {
    case 'a':
      std::cout << "You chose menu a..." << std::endl;
      break;
    case 'b':
      std::cout << "You chose menu b..." << std::endl;
      break;
    default:
      std::cout << "Invalid Selection!" << std::endl;
      break;
  }

  return 0;
}
```

Here we evaluate to, *"You chose menu a…"*, as this would be normally the result of properly sanitized input which we will cover in a later lesson.

For

With the for loop we have three elements.

1. Initialization
2. Condition
3. Increment

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // for
  for (int i=1; i<=10; ++i)
    std::cout << i << std::endl;

  return 0;
}
```

Here we evaluate to printing 1-10 on the screen.

For (Range)

With the for loop we can also have ranges.

Let's update our **main.cpp** file.

```
#include <iostream>
#include <vector>

int main()
{
  // for (range)
  std::vector<double> scores;
  scores.push_back(100.1);
  scores.push_back(20.5);
  scores.push_back(50.25);
  double average_score = 0;
  double total_scores = 0;
  for (auto score: scores)
    total_scores += score;
  if (scores.size() != 0)
    average_score = total_scores / scores.size();
  std::cout << "Average Score: " << average_score << std::endl;

  return 0;
}
```

Here we evaluate the average score to *56.95*.

While

With the while loop we start with a conditional and if it fails the block will not execute or if a condition in the block nullifies the initial conditional it will terminate.

Let's update our **main.cpp** file.

```
#include <iostream>

int main()
{
  // while
  int num = 42;
  while (num <= 77)
  {
    std::cout << num << std::endl;
    ++num;
  }

  return 0;
}
```

Here we print out the numbers from *42* to *77*.

Project 3

Your assignment will be to create a new folder **0x0003_my_fav_nums** and create a **main.cpp** and a **Makefile** and write a program that creates a vector of int where you create elements, manipulate them with the built-in vector functions and display each step using what you learned above this time adding in conditionals.