Historical Perspective

C++ is considered as one of the most powerful programming language day to date since it has started its glorious journey in the year 1979 at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way. The earliest versions, which were originally referred to as “C with classes,” date back to 1980. As the name C++ implies, C++ was derived from the C programming language: ++ is the increment operator in C. The C++ programming language was created by Bjarne Stroustrup and his team. You can find C++ in the list of top ten most popular programming languages in the world.

Apart from deep understanding of pointers and dynamic memory management techniques system programming game development operating system utilities network programming.

C++ is still holding the dominant position as a different choice of programming language.

C++ is a statically typed, compiled, general-purpose, case-sensitive, free form programming language. That supported procedural, object-oriented, and generic programming language.

As early as 1989 an ANSI Committee (American National Standards Institute) was founded to standardize the C++ programming language. The aim was to have as many compiler vendors and software developers as possible agree on a unified description of the language in order to avoid the confusion caused by a variety of dialects.

In 1998 the ISO (International Organization for Standardization) approved a standard for C++ (ISO/IEC 14882).

Characteristics of C++

C++ is an Object-Oriented Programming Language which has all the features of any object-oriented language – Object, Class, Encapsulation, Inheritance, Polymorphism, Dynamic Binding, and Message Passing.

1) An object is an entity on which we would talk about and would create programs using it. Any entity in real life, such as a table, board, duster, etc., could be an object.
2) The second feature is the class which is a group of objects. Classes consist of all the functions and the variables in a program.
3) Encapsulation internally hides the operation of a function. For, e.g., when we ride a bike, we press the accelerator but doesn't know what's happening behind the scenes or how the engine is working. In layman terms, it wraps the data into a class, and hence only the function is allowed to access the data.
4) Inheritance gives a class the ability to use the features and properties of its parent class. The inherited class could be of type Public, Private, and Protected. Also, new features could be added to the child class as well. The inheritance could be single level, multi-level, multiple, and even hierarchical.
5) Polymorphism is the property in which one entity could have multiple forms, which allow the object to behave differently in different situations. It could be static as well as dynamic.
6) Dynamic Binding would always be at run-time, and according to the requirement or the code, it would call that function which is needed.
7) At run-time, objects could communicate among each other by sending data to and fro with the help of a message passing interface.

Applications

The application of C++ is diversified in various domains because of its flexibility and reliability. Below are some of the few areas where C++ could be used.

- C++ is widely used in the Gaming industry. Various companies hire people with a knowledge of C++ to build interactive games for them.
- One of the other applications of C++ is creating Graphical User Interface, which simplifies the user's interaction with an application.
- In software like Adobe Photoshop or Illustrator, C++ is used as well.

- We can use C++ to create web browsers like Mozilla Firefox and compilers.
- The operating systems are also programmed in C++.
- The medical industry used C++ to build most of their software.
- Few of the other programming languages like Java are built using the C++ language.

Advantages and Disadvantages of C++

Below are the advantages and Disadvantages of C++

Advantages

- C++ is a very efficient language which is fast and reliable.
- C++ has a wide range of usage, and hence learning the language makes it easier to grasp the Object-Oriented Programming Concept.
- C++ makes it easier to learn other programming languages as well.

Disadvantages

- C++ could often be hard to master.
- The error messages in C++ could be extended and often difficult to debug.
- It could be difficult to access the libraries in C++ even.
- The code could be prone to errors as C++ doesn't provide type-checking.

WEEK TWO

C++ Compiler

For the purpose of this course, we will be using Codeblocks compiler to compile our program:. you can download the compiler from the following site: **https://www.codeblocks.org** then install it on your system or get a free version copy from your course lecturer.

BASIC SKELETON OF C++ PROGRAM

Let's vividly look at the following C++ Program and understand every command used in order to give us a clear picture of how our programs look like throughout this book.

```
1. #include <iostream>
2. using namespace std;

3.

4. int main()

5.  { cout << "Hello World!";

6.   return 0;

7. }
```

Code explained

Line 1: `#include <iostream>` is a **header file library** that lets us work with input and output objects, such as **`cout`** (used in line 5). Header files add functionality to C++ programs.

Dynamic memory management

<new> − Low-level memory management utilities. The new-expression is the only way to create an object or an array of objects with dynamic storage duration, that is, with lifetime not restricted to the scope in which it is created.

<memory> − Higher level memory management utilities

<scoped_allocator> (since C++11) − The std::scoped_allocator_adaptor class template is an allocator which can be used with multilevel containers (vector of sets of lists of tuples of maps, etc).

<memory_resource> (since C++17) −The class std::pmr::memory_resource is an abstract interface to an unbounded set of classes encapsulating memory resources.

Numeric limits

<climits> −limits of integral types

<cfloat> − limits of float types

<limits> −standardized way to query properties of arithmetic types

<cstdint> (since C++11) − fixed-size types and limits of other types

<cinttypes> (since C++11) − Provides conversions from C byte and wide strings to std::intmax_t and std::uintmax_t, overloads some math functions for std::intmax_t and provide C style input/output format macros for the types declared in <cstdint>.

Error handling

<exception> − Exception handling utilities

<stdexcept> −Standard exception objects

<cassert> −Conditionally compiled macro that compares its argument to zero

<cerrno> −Macro containing the last error number

<system_error> (since C++11) −std::error_code is a platform-dependent error code. Each std::error_code object holds an error code originating from the operating system or some low-level interface and a pointer to an object of type std::error_category, which corresponds to the said interface.

Strings library

<cctype> − Functions to determine the type contained in character data

<cwctype> − Functions to determine the type contained in wide character data

<cstring> −various narrow character string handling functions

<cwchar> −various wide and multibyte string handling functions

<string> −std::basic_string class template

<cuchar> (since C++11) − C-style Unicode character conversion functions

<string_view> (since C++17) − The class template basic_string_view describes an object that can refer to a constant contiguous sequence of char-like objects with the first element of the sequence at position zero.

Containers library

<array> (since C++11) − std::array container

<vector> − std::vector container

<deque> − std::deque container

<list> − std::list container

<forward_list> (since C++11) − std::forward_list container

<set> − std::set and std::multiset associative containers

<map> − std::map and std::multimap associative containers

<unordered_set> (since C++11) − std::unordered_set and std::unordered_multiset unordered associative containers

<unordered_map> (since C++11) − std::unordered_map and std::unordered_multimap unordered associative containers

<stack> − std::stack container adaptor

<queue> − std::queue and std::priority_queue container adaptors

Algorithms library

<algorithm> − Contains algorithms that operate on containers

<execution> (C++17) −Predefined execution policies for parallel versions of the algorithms

Iterators library

<iterator> − Iterators for the containers

Numerics library

<cmath> − Common mathematics functions

<complex> − Complex number type

<valarray> − Class for representing and manipulating arrays of values

<random> (since C++11) − Random number generators and distributions

<numeric> − Numeric operations on values in containers

<ratio> (since C++11) −Compile-time rational arithmetic

<cfenv> (since C++11) − Floating-point environment access functions

Input/output library

<iosfwd> − forward declarations of all classes in the input/output library

<ios> − std::ios_base class, std::basic_ios class template and several typedefs

<istream> − std::basic_istream class template and several typedefs

<ostream> − std::basic_ostream, std::basic_iostream class templates and several typedefs

<iostream> − several standard stream objects

<fstream> − std::basic_fstream, std::basic_ifstream, std::basic_ofstream class templates and several typedefs

<sstream> − std::basic_stringstream, std::basic_istringstream, std::basic_ostringstream class templates and several typedefs

<syncstream> (since C++20) − std::basic_osyncstream, std::basic_syncbuf, and typedefs

<strstream>(deprecated) − std::strstream, std::istrstream, std::ostrstream

<iomanip> − Helper functions to control the format or input and output

<streambuf> − std::basic_streambuf class template

<cstdio> − C-style input-output functions

Localization library

<locale> − Localization utilities

<clocale> − C localization utilities

<codecvt> (since C++11) (deprecated in C++17) − Unicode conversion facilities

Regular Expressions library

<regex> (since C++11) − Classes, algorithms and iterators to support regular expression processing

Atomic Operations library

<atomic> (since C++11) − Atomic operations library

Thread support library

<thread> (since C++11) − std::thread class and supporting functions

<mutex> (since C++11) − mutual exclusion primitives

<shared_mutex> (since C++14) − shared mutual exclusion primitives

<future> (since C++11) −primitives for asynchronous computations

<condition_variable> (since C++11) − thread waiting conditions

Filesystem library

<filesystem> (since C++17) − std::path class and supporting functions

Line 2: **`using namespace std`** means that we can use names for objects and variables from the standard library.

Don't worry if you don't understand how **`#include <iostream>`** *and* ***`using namespace std`*** *works. Just think of it as something that (almost) always appears in your program.*

Line 3: A blank line. C++ ignores white space.

Line 4: Another thing that always appear in a C++ program, is **`int main()`**. This is called a **function**. Any code inside its curly brackets **`{}`** will be executed.

Line 5: **`cout`** (pronounced "see-out") is an **object** used together with the *insertion operator* (**<<**) to output/print text. In our example it will output "Hello World".

Note: Every C++ statement ends with a semicolon `;`.

Note: The body of **`int main()`** could also been written as:
`int main () { cout << "Hello World! "; return 0; }`

Remember: The compiler ignores white spaces. However, multiple lines make the code more readable.

Line 6: **`return 0`** ends the main function.

Line 7: Do not forget to add the closing curly bracket **}** to actually end the main function.

The C++ **Cout** command that display your output on the screen

Example 1.0:

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

int main() {
  cout << "MY COM313 Display";
  return 0;
}
```

The **cout** object, together with the **<<** operator, is used to output values/print text while the **cout** object with the extraction operator (**>>**) is used to receive an input from the keyboard:

Data Types

C++ Data types define the type of data that a variable can hold. It can be user-defined data type like integer, float, double, char, or built-in data type like union, enum, struct or can be a derived data types like functions, pointers, arrays. Data types should be defined before the execution as it informs the compiler the type of data specific variables holds. Integer data type can hold only integer values, it cannot hold the float values or string values.

A data type is to let know the variable, what type of element it is and definitely going to determine the memory allocation of that variable. We are aware that each data type has a different memory allocation. There are three different C++ data types namely; Primitive, Derived and User Defined.

Below are the three different data types in c++ which are explained below:

1. Primitive Data Types

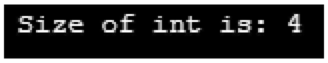
These are pre-defined in c++, also called the built-in data types. We can directly use them to declare the variables.

a. Integer: Usually defined by "int". We can know the size of memory allocated and how the variable is declared as below.

Code:

```
#include <iostream>
using namespace std;
int main()
{
int a{};
cout<< " Size of variable a{} is: " << sizeof(a);
}
```

Output:



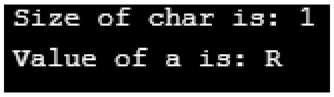
```
Size of int is: 4
```

b. Character: Usually defined by "char". We can know the size of memory allocated and how the variable is declared as below.

Code:

```
#include <iostream>
using namespace std;
int main()
{
char a{'R'};

cout<< " Size of char a{} is: " << sizeof(a)<<endl;
cout<< " Value of a is: " << a;
}
```

Output:



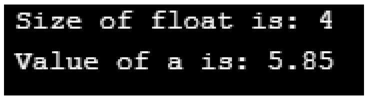
```
Size of char is: 1
Value of a is: R
```

c. Floating Point: Usually defined by “float”. We can know the size of memory allocated and how the variable is declared as below.

Code:

```
#include <iostream>
using namespace std;
int main()
{
auto a{5.89};

cout<< " Size of a{} is: " << sizeof(a)<<endl;
cout<< " Value of a{} is: " << a;
}
```

Output :



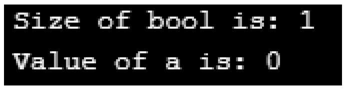
```
Size of float is: 4
Value of a is: 5.85
```

d. Boolean: Usually defined by “bool”. We can know the size of memory allocated and how the variable is declared as below.

Code:

```
#include <iostream>
using namespace std;
int main()
{
bool a{};
cout<< " Size of bool is: " << sizeof(a)<<endl;
cout<< " Value of a is: " << a;
}
```

Output :



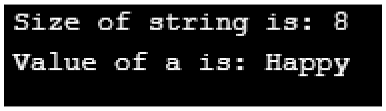
```
Size of bool is: 1
Value of a is: 0
```

e. String: Usually defined by "String". We can know the size of memory allocated and how the variable is declared as below.

Code:

```
#include <iostream>

using namespace std;

int main()

{

string a{"Happy"};

cout<< " Size of string is: " << sizeof(a)<<endl;

cout<< " Value of a is: " << a;

}
```

Output:



```
Size of string is: 8
Value of a is: Happy
```

Here, we also have the concept of signed, unsigned, short and long. So, what are these? These are called the Data type modifiers. These, in fact, decide the actual length of any particular data type.

Signed values give us the numbers of both below and above zero, which is both positive and negative. Whereas the unsigned values contain data which is only positive. And coming to short and long, through the names itself we can clearly interpret that long data modifier has the capacity to store large amounts of values. And in fact, short is the data type must and will hold a minimum of those numbers of values.

C++ Variable Types

A variable is a name which is associated with a value that can be changed. For example when I write int value{38}; here variable name is value which is associated with value 33, int is a data type that represents that this variable can hold integer values. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive

Rules for Assigning Name to a Variable in C++

1. All variable names must begin with a letter of the alphabet or an. underscore (_).
2. After the first initial letter, variable names can also contain letters and numbers.
3. Uppercase characters are distinct from lowercase characters; this is because C++ is a case sensitive language. A variable with name hours is different with HOURS in C++

4. You cannot use a C++ keyword (reserved word) as a variable name.(e.g int, bool, char cannot be used as variable name in C++ but if you want to use one such as the int then you have to add a prefix such as int_score.
5. Variable names in C++ can range from 1 to 255 characters
6. No spaces or special characters are allowed.

Variable types in C++

In C++, there are different **types** of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 258 or -258
- double - stores floating point numbers, with decimals, such as 27.99 or -27.99
- char - holds character value like 'c', 'F', 'B', 'p', 'q' etc.. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

Each variable while declaration must be given a **datatype**, on which the memory assigned to the variable, depends

Syntax for variable declaration: You can declare a variable in C++ using the following syntax:

datatype <Variable name>

Displaying your Variable:

The **cout** object is used together with the **<<** operator to display variables. To combine both text and a variable, separate them with the **<<** operator:
Example:

int my_Score{}
Float School_fees{}
Boolean gender{}

Example 1.2

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

int main() {
  int my_score {55};
  cout << "This display my score "<< my_score;
  return 0;
}
```

Output:

```
This display my score 55
```

Example 1.3 (to declare and display other variable)

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

int main() {
  Char My_Grade{’A’};
  Float Average{38.5};
  bool gender{’F’};
  cout << “My Grade in this course would be ”<< My_Grade;
  cout << “The Average Score of my CA is ”<< Average;
  cout << “The Colleague next to me in the class is ”<<gender;
  return 0;
}
```

Output:

```
My Grade in this course would be A
The Average Score of my CA is 38.5
The Colleague next to me in the class is F
```

To declare more than one variables of the same data type you use a comma to separate them e.g

```
Int x=25,y=33,z=24;
Cout<<x<<” ”<<y<<” ”<<z;
```

C++ Identifiers

We identify all C++ variables with a unique name; which are called identifier, but identifier can be written as a single character such x, r, and y. it is advisable to use descriptive name such as (**age**, **sum**, **Average**, **fees**, **sum**, **total** etc.). This would make your program to give meaning to who is reading or try to modify it in future.

Example 1.4

If you declare an identifier as:

int my_Budeget {50000} // this has a meaning to any one reading it which is referring to a Budget amount; but if you name it as this

int x{50000}// this is correct syntactically but would have no meaning to the reader.

Note: ***The general rules for assigning a name to an identifier is the same the rules for assigning name to a variable in C++.***

C++ CONSTANTS

A constant is an identifier/variable whose value does not change or overwritten in the process of executing a program in C++. A valid constant name can start with a letter or an underscore. Numbers are allowed, but they cannot be used at the beginning of the constant name. For example, the constant name 1First_Name is not properly written. Instead, you might use something like First_Name1 or First1_Name.

Example 1.5

```
const int Emp_Tax {305};  // Emp_Tax will always be 305
Emp_Tax {25};  // error: assignment of read-only variable ' Emp_Tax '
```

```
if you properly code the program as follows:

#include <iostream>
using namespace std;

int main() {
  const int Emp_Tax{305};
  Emp_Tax{25};
  cout << Emp_Tax;
  return 0;
}

Output:
error: assignment of read-only variable 'Emp_Tax'
```

EXERCISE:

1. Write a C++ Program that would declare the a variables and it's equivalent data type for the following supplied information:

1) ₦38,400
2) 25 x 10^5
3) True or False
4) Hello Dud!
5) 3.46

2. Fill in the missing parts to create three variables of the same type, using a **comma-separated list**:

```
[      ] x1 =4[      ] x2 = 4[      ] X3 = 50;
cout << x1 + x2 + x3;
```

4. Take two integer inputs from user. First calculate the sum of two then product of two. Finally, print the sum and product of both obtained results.

5. Write a C++ program to print an int, a double and a char on screen.

6. Take value of length and breath of a rectangle from user as float. Find its area and print it on screen after type casting it to int.

6. Write a program to assign a value of 100.235 to a double variable and then convert it to int.

7. **What is** difference between Declaration and Definition of a variable.