OBJECT ORIENTED PROGRAMMING USING C++

By:
Collinson C. M. Agbesi

Recommended Textbooks

1. Joyce Farrell *"Object Oriented Programming Using C++"*, 4th Edition, Cengage, 2009

2. Paul Deitel & Harvey Deitel *"How to Program C++"* 8th Edition, Pearson Prentice, 2012

OBJECT ORIENTED PROGRAMMING

- **Object-Oriented Programming (OOP)**: Object oriented programming is a programming paradigm which uses abstraction to create software models based on the real world.
- Object-oriented programming (OOP) refers to a type of computer programming (software design) in which the data type (data structure) and the types of operations (functions) that can be applied to the data structure are combined together.
- OOP is a programming approach in which each object combines data (states/values)and procedures (methods/functions) that act on the data.
- Examples of OOP Programming Languages: C++, C#, Java, PHP, VB etc

OBJECT ORIENTED PROGRAMMING

- **Programming Paradigm**: Is an approach or method of writing computer programs. Some examples; Procedural, Functional, Logic based, Object Oriented etc.

- **Object Oriented Versus Procedural Programming**:

- **Object oriented programming** is a programming paradigm in which the data and functions are combined together into a single unit.
- **Procedural (Traditional) programming** is a programming paradigm in which the data (states) and functions (methods) are separated.

- Object oriented programming (OOP) is based on the hierarchy of classes and their well defined objects. Programmers can create relationships between a class and another class. Objects can inherit characteristics from other objects.

OBJECT ORIENTED PROGRAMMING

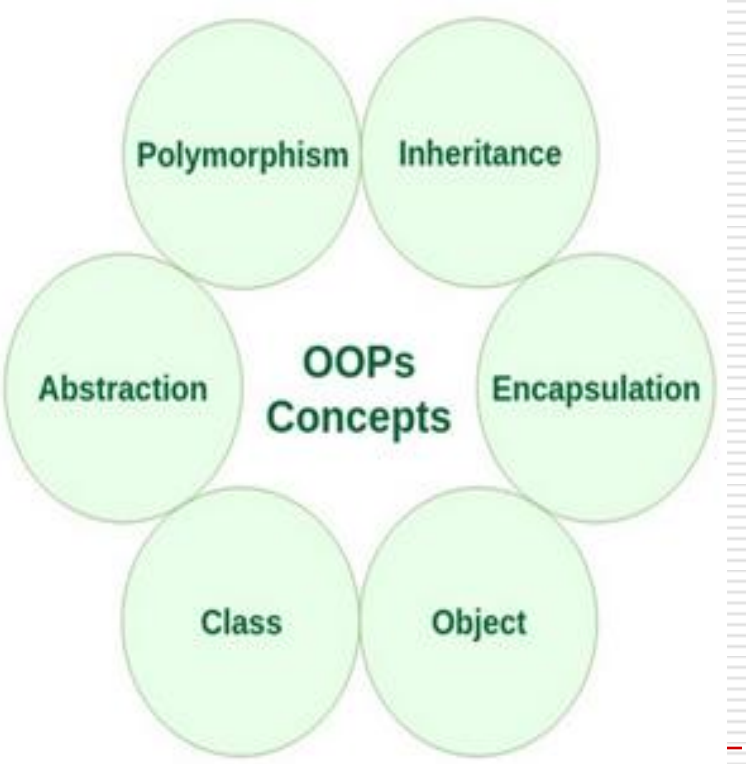
- **Object Oriented Versus Procedural Programming:**

Procedural Programming	Object Oriented Programming
Divided into small parts called functions	Divided into small parts called objects
Uses records in programs	Uses objects in programs
Uses modules in writing programs	Uses classes in writing programs
Uses procedure calls	Uses messages
Uses top down approach	Uses bottom up approach
Less secure	More secure
Adding new data and functions very difficult	Adding new data and functions very easy
Based on unreal world	Based on real world
E.g. C, Pascal, Fortran etc	E.g. C++, C#, Java, PHP, VB etc

OBJECT ORIENTED PROGRAMMING CONCEPTS

- **Object Oriented Programming Concepts:** There are several concepts associated with object oriented programming.

- Some OOP concepts includes;
- Classes
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism



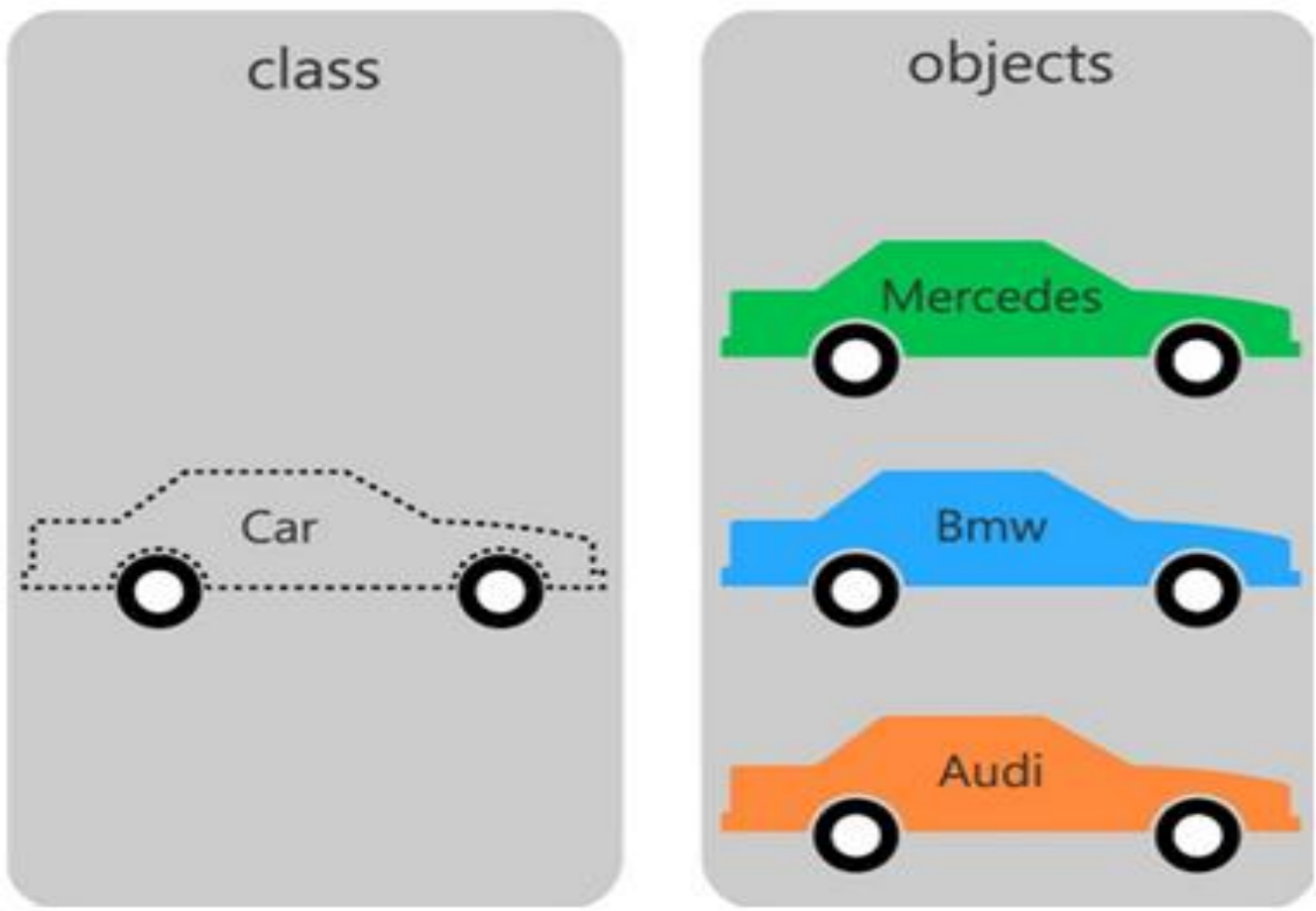
OBJECT ORIENTED PROGRAMMING CONCEPTS

- **Class:** a **class** is a blueprint for creating objects (a data structure) by combining its state (member variables or attributes) and behavior (member functions or methods) into a single unit. A class is a prototype or template from which objects are created. A class can has both data members (states) and functions (methods) combined within itself. An instance of a class is a specific **object** created from that particular **class**.
- **Example**: If a **class is a car**, then the **functions** will be start car, drive car, stop car, open door, close door etc and the **variables** will be name of car, color of car, type of car, weight of car, height of car, length of car etc.

```
Class Car
{
    string carname, carcolor, cartype, carmodel;
    float carheight, carweight, carlength;
};
```

OBJECT ORIENTED PROGRAMMING CONCEPTS

- **Class:**



OBJECT ORIENTED PROGRAMMING CONCEPTS

- **Object:** An object is an instance of a class. Objects are created from classes which define their state and behavior. An object is a "*thing*" that can perform a set of **related** activities. The set of activities the object performs defines the object's behavior (functions) which is specified in the object's parent class.

- An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. Software and real world objects share two characteristics: They all have their *state* (variables) and behavior (functions).
  - **Examples**:
  - Dogs have their states (name, color, breed, weight, height) and behavior (barking, fetching, wagging tail etc).
  - Bicycles have their states (name, type, size, seats, gears) and behaviors (changing gear, applying brakes etc).
- Identifying the state and behavior of objects is a great way to begin thinking in terms of object-oriented programming.

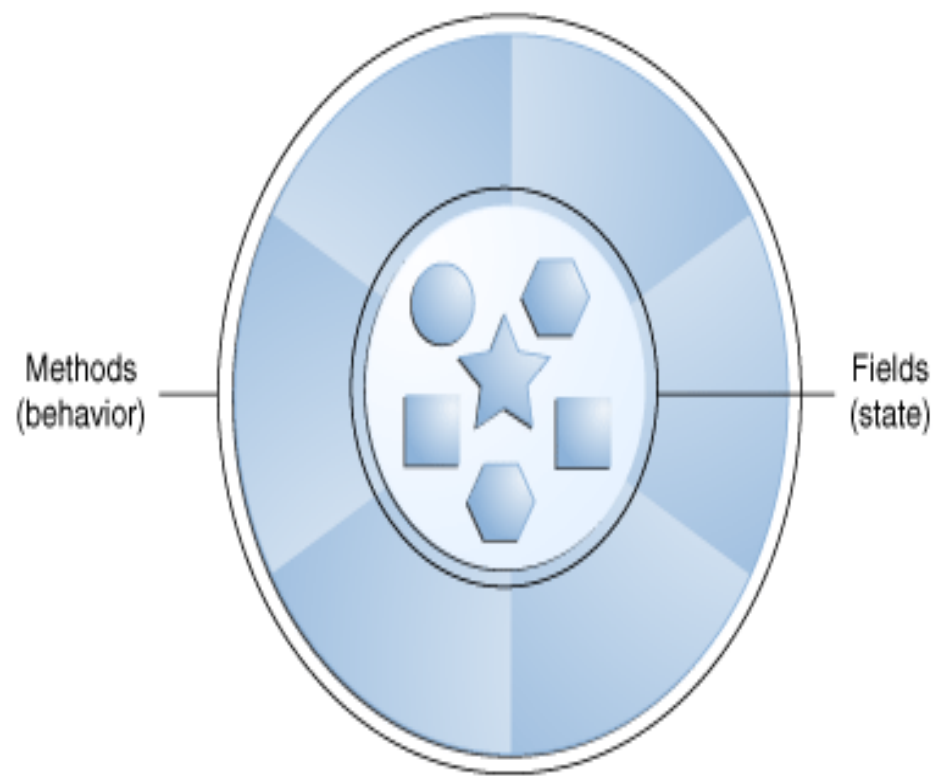
OBJECT ORIENTED PROGRAMMING CONCEPTS

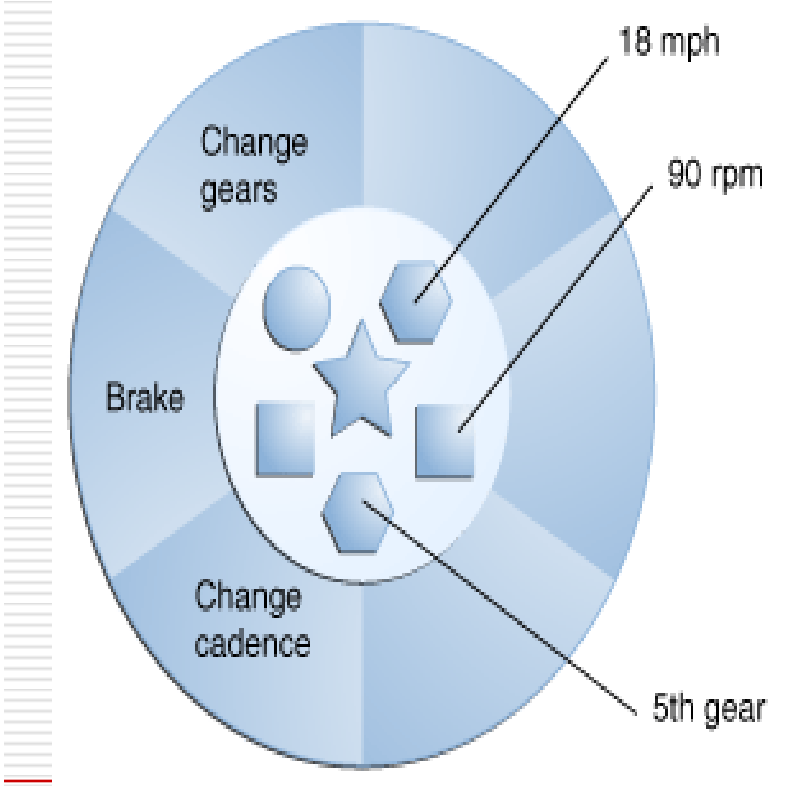
- **Objects**: Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in *fields* (variables) and exposes its behavior through *methods* (functions).
- Fields tell you the states or attributes of objects whiles Methods tell you the behavior or functions of the objects.

- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. An object hides its internal states and requires all interaction to be performed through an object's methods or functions. Software objects send messages to each other by calling their methods. An object sends message to another object by asking another object to invoke it’s method.
- Some bigger objects, may contain other smaller objects. Each object can be viewed as an independent little machine with a distinct role or responsibility.

OBJECT ORIENTED PROGRAMMING CONCEPTS

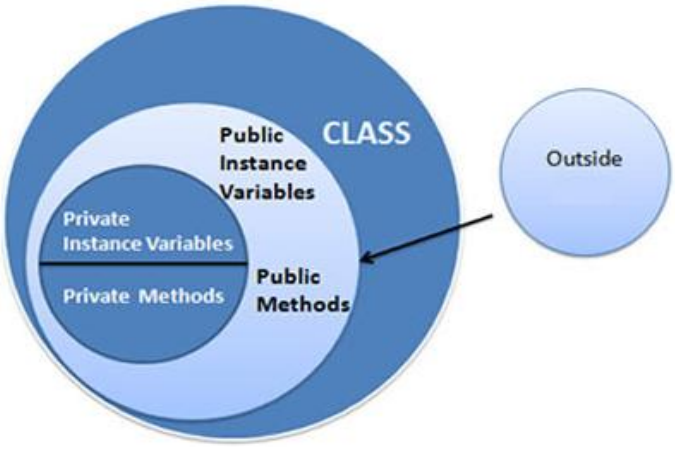
- **Objects**:

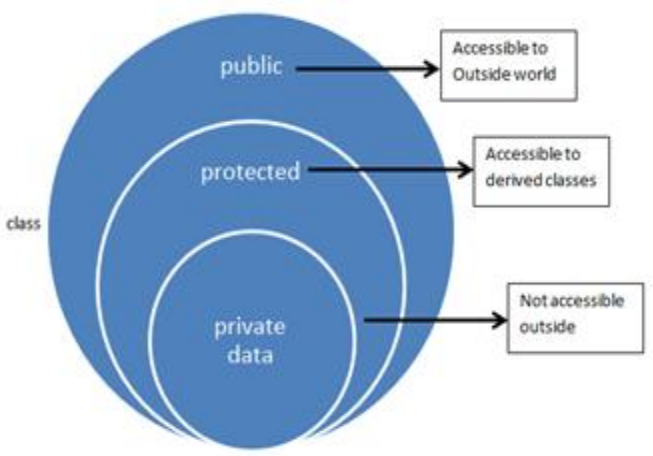




OBJECT ORIENTED PROGRAMMING CONCEPTS

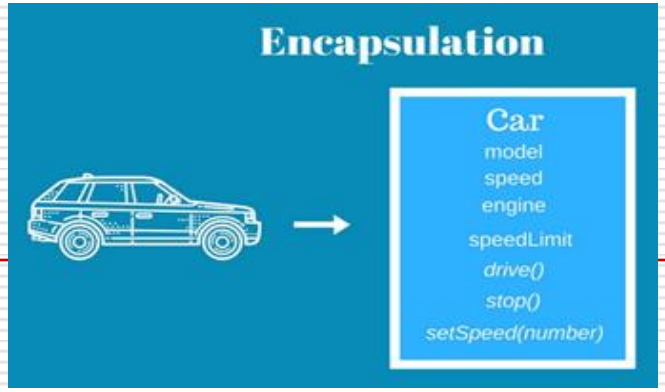
- **Data Abstraction**: Abstraction means displaying only essential information and hiding the details. **Data abstraction** refers to providing only essential information about the object to the outside world and hiding the implementation or background details. A Class can decide which member (data or functions) will be visible to outside world and which should not be visible. The importance of abstraction is derived from its ability to hide irrelevant details of class. The state (variables) can be secured within the **inner part** of the class whiles methods (functions) on the **outer part** act on them.

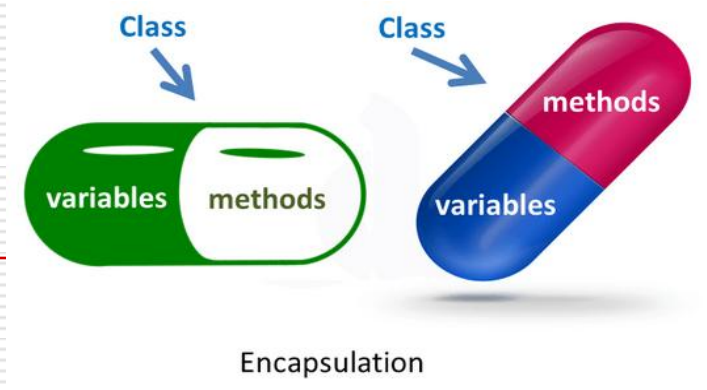




OBJECT ORIENTED PROGRAMMING CONCEPTS

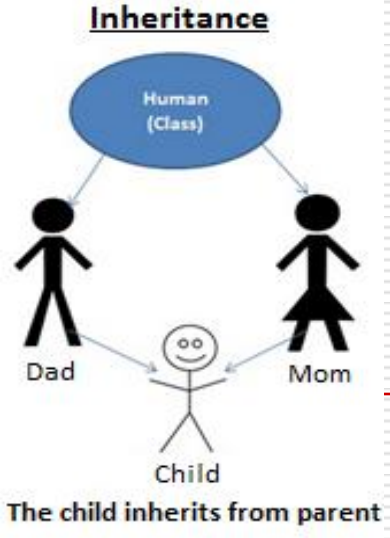
- **Encapsulation (information hiding):** Encapsulation is the bundling of data (states) and methods (functions) that work on that data within a single unit. Encapsulation is the grouping of related data and functions together as one unit. By interacting only with an object's methods, the details of its internal structure are hidden from the outside world.
- **Encapsulation** refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components

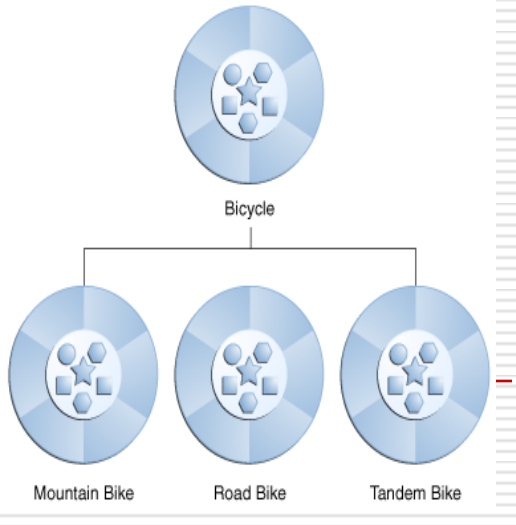




OBJECT ORIENTED PROGRAMMING CONCEPTS

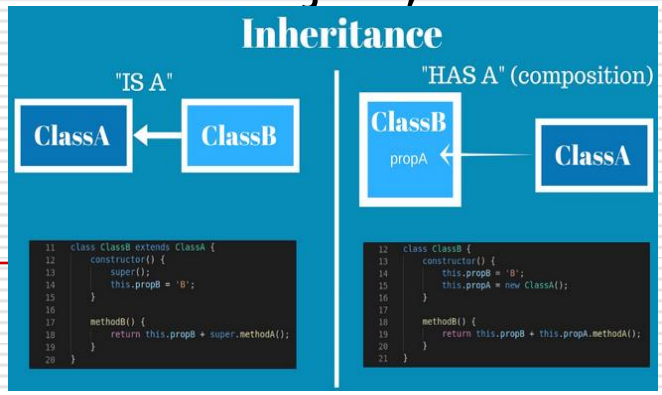
- **Inheritance (Code Reuse):** Inheritance is a mechanism in which one class acquires the property of another class. With inheritance, we can reuse the fields and methods of an existing class. Inheritance provides a way to reuse code of existing class or to establish a subtype from an existing class. If an object (code) already exists, you can use that object (code) in your program. Inheritance means deriving new classes from existing ones such as super class or base or parent class and then forming them into a hierarchy of classes, that is; creating parent-child relationships.





OBJECT ORIENTED PROGRAMMING CONCEPTS

- **Polymorphism:** Polymorphism means having many forms or shapes. It refers to the ability to derive more than one form or shape from the same class or object.
- Polymorphism is the provision of a single interface to entities of different types or the use of a single class to represent multiple types or forms of entities.
- Polymorphisms means the ability to request that the same operations be performed by a wide range of different types of things. Example: If a function expects a Vehicle object, we can easily pass it a Car object, because every car is a vehicle type.



MERITS OF OBJECT ORIENTED PROGRAMMING

- **Modularity:** In OOP objects are self contained and perform its own functions which makes it easy work with multiple objects at the same time without duplicating functions.
- **Modifiability:** It is easy to make changes in the in an OOP program. Changes inside a class does not affect any other part of the program.
- **Extensibility:** Adding new features or responding to changing demands is made easier by modifying some existing ones or introducing new ones.
- **Maintainability:** It is easier to maintain and modify existing code as new objects can be created with small differences to existing ones.
- **Reusability:** In OOP, it is easy for objects to reuse existing codes in different programs.
- **Pluggability and debugging ease**: In OOP, if a particular object turns out to be problematic, you can easily remove it from your application and replace a different object.

DEMERITS OF OBJECT ORIENTED PROGRAMMING

- **Complex Design**: Designing and proper implementation of object oriented programming (OOP) concepts is complex difficult and burdensome. It requires extreme knowledge in order to design and implement OOPs concepts.
- **Many skills required**: An object oriented programmer requires many skills for better programming. Different skills like programming skills, designing skills, logical thinking and problem-solving skills are needed. Not having these skills makes it difficult in OOP.
- **Large size of Programs**: Programs of object-oriented programming are of larger size in comparison with the traditional procedural programming. Due to the larger size of the program, many instructions are needed in order to execute the program. This make the code complex and lengthy.
- **Slow Speed**: Due to large size of the programs its execution speed becomes slow. Many program instructions make the execution of the program slower and affects its overall efficiency.
- **Not suitable for all problems**: There are problems that lend themselves well to functional, logic or procedure based programming styles and applying object oriented programming in these situations will not result in efficient programs.

HISTORY OF C++

- C++ was written by Bjarne Stroustrup at Bell Labs during 1983-1985. C++ is an extension of C. Prior to 1983, Bjarne Stroustrup added features to C and formed what he called "C with Classes". He had combined the Simula's use of classes and object-oriented features with the power and efficiency of C. The term C++ was first used in 1983.

HISTORY OF C++

- C++ was developed significantly after its first release. In particular, "ARM C++" added exceptions and templates, and ISO C++ added RTTI, namespaces, and a standard library.
- C++ was designed for the UNIX system environment. With C++ programmers could improve the quality of code they produced and reusable code was easier to write.
- Bjarne Stroustrup had studied in the doctoral program at the Computing Laboratory at Cambridge University prior to joining Bell Labs. Now, Bell Labs no longer has that name since part of Bell Labs became AT&T Labs. The other half became Lucent Bell labs.
- Prior to C++, C was a programming language developed at Bell Labs circa 1969-1973. The UNIX operating system was also being developed at Bell Labs at the same time. C was originally developed for and implemented on the UNIX operating system, on a PDP-11 computer by Dennis Ritchie.

HISTORY OF C++

- He extended the B language by adding types in 1971. He called this NB for New B. Ritchie credited some of his inspiration from theAlgol68 language. Ritchie restructured the language and rewrote the compiler and gave his new language the name "C" in 1972. 90% of UNIX was then written in C.
- The committee that wrote the 1989 ANSI Standard for C had started work on the C Standard project in 1983 after having been established by ANSI in that year. There were quite a number of versions of C at that time and a new Standard was necessary. C is portable, not tied to any particular hardware or operating system. C combines the elements of high-level languages with the functionality of assembly language and has occasionally been referred to as a middle-level computer language. C makes it easy to adapt software for one type of computer to another.
- C was a direct descendant of the language B. The language B was developed by Ken Thompson in 1970 for the new UNIX OS. B was a descendant of the language BCPL designed by Martin Richards, a Cambridge University student visiting MIT.[1]

HISTORY OF C++

- C++ is an "object oriented" programming language created by Bjarne Stroustrup and released in 1985. It implements "data abstraction" using a concept called "classes", along with other features to allow object-oriented programming. Parts of the C++ program are easily reusable and extensible; existing code is easily modifiable without actually having to change the code. C++ adds a concept called "operator overloading" not seen in the earlier OOP languages and it makes the creation of libraries much cleaner.
- C++ maintains aspects of the C programming language, yet has features which simplify memory management. Additionally, some of the features of C++ allow low-level access to memory but also contain high level features.
- C++ could be considered a superset of C. C programs will run in C++ compilers. C uses structured programming concepts and techniques while C++ uses object oriented programming and classes which focus on data. Read about the History of C and also about the History of C++.

HOW TO GET STARTED

- You need the following;
  - A Personal computer with (display, memory, HDD, processor)
  - A Compiler that suits your operating system (OS):
    Compiler is a program that translates the source code of another program written in high level language into a low level machine language for the computer to understand and execute.
- Integrated Development Environments: An IDE is a software development environment or platform that includes an Editor, Compiler and Debugger in an integrated package that is distributed together to facilitate and simplify coding and implementation of programming languages. Some IDEs will require the user to make the integration of the components themselves and others will do it automatically.
- A good IDE is one that permits the programmer to use it to perform tasks and at the same time provide some tools for reading error and debugging and managing the code.
- When selecting an IDE, remember that you are also investing time to become proficient in its use, completeness, stability and portability across different operating systems.

HOW TO GET STARTED

- **Some C++ IDES, Compilers and OS Platforms**: There are several and different integrated development environments (IDE) and platforms that can be used for C++ programming. Some of the IDEs include;

- Linux/Unix OS: Uses the g++ or gcc compilers

- Macintosh/Apple OS: Uses Xcode, NetBeans, Eclipse etc

- Windows OS: Uses DevC++ or Ms Visual C++ or Code::Blocks, NetBeans, Eclipse etc

STRUCTURE OF C++ PROGRAM

```
1  // my first program in C++
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    cout << "Hello World!";
9    return 0;
10 }
```

```
Hello World!
```

- The panel (in light blue) shows the **source code** for our first program. The second one (in light gray) shows the **result/output** of the program once compiled and executed. The grey numbers represent the line numbers and are not part of the program but are shown for informational purposes.

STRUCTURE OF C++ PROGRAM

- The previous program is the typical program that programmers write for the first time, and its output is the displaying on the PC screen the texts "Hello World". It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written.
- **// my first program in C++**
  This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to give short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is about.
- **Note:** C++ supports two comments
  - //My first program in C++ : **Short comments**
  - /*My first program in C++ */ : **Block comments**

STRUCTURE OF C++ PROGRAM

- **#include <iostream>**
  Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include <iostream> tells the preprocessor to include the iostream (input/output stream) file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

- **using namespace std;**
  All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

STRUCTURE OF C++ PROGRAM

- **int main ()**
  This line corresponds to the beginning of the main function. The main function is the point where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. So it is very essential that all C++ programs have a *main* function. The word main is followed in the code by a pair of parentheses ( ). That is because it is a function declaration. In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Often, these parentheses may have a list of parameters within them.
- **{} Curly brackets:**
  The two curly brackets (one in the beginning and one at the end) are used to indicate the beginning and the end of the main function and also called the body of the function. Everything contained within these curly brackets is what the function does when it is called and executed.

STRUCTURE OF C++ PROGRAM

- **cout << "Hello World!";**
  This line is a C++ statement. A statement is a simple or compound expression that can actually execute. In fact, this statement performs the only action that generates a visible output in our first program. Cout is the name of the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters in this case the **Hello World** into the standard output stream, which usually corresponds to the PC screen.
  C*out* is declared in the *iostream* standard file within the *std* namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.
  Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and it must be included at the end of all statements in all C++ programs. In fact one of the most common syntax errors is to forget to include the **semicolon** after a statement.

STRUCTURE OF C++ PROGRAM

- **return 0;**
  The return statement causes the main function to either give out its output or not. The return may be followed by a return value (in our example it is followed by the value zero) which indicates that it shouldn't give out its output. A return value of zero (*0)* for the *main* function means if the main function executes successfully without any errors during its execution it should end without giving any results back to the program. This is the most usual way to end a C++ console program.

- You may have noticed that not all the lines of this program perform actions when the code is executed. Some lines contain only comments. There were lines with directives for the compiler's preprocessor to import the standard input-output functions. Also there were lines that began the declaration of a the main function and, finally lines with statements, which are included within the body of the main function.

STRUCTURE OF C++ PROGRAM

- The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
1 int main ()
2 {
3   cout << " Hello World!";
4   return 0;
5 }
```

We could have written:

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

- All in just one line and this would have had exactly the same meaning as the previous code.

STRUCTURE OF C++ PROGRAM

- In C++, each statement is separated with an ending semicolon (;), so the separation in different lines of code does not matter. We can write many statements per line or write a single statement that takes many code lines. The separation of code in different lines is only to make it more legible and readable for the human beings to understand.
- Let us add an additional instruction to our first program:

```
1  // my second program in C++
2
3  #include <iostream>
4
5  using namespace std;
6
7  int main ()
8  {
9    cout << "Hello World! ";
10   cout << "I'm a C++ program";
11   return 0;
12 }
```

```
Hello World! I'm a C++ program
```

STRUCTURE OF C++ PROGRAM

- In this case, we performed two insertions using **cout** in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since *main* could have been perfectly valid defined this way:

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

- We were also free to divide the code into more lines if we considered it more convenient:

```
1 int main ()
2 {
3   cout <<
4     "Hello World!";
5   cout
6     << "I'm a C++ program";
7   return 0;
8 }
```

- And the result would again have been exactly the same as in the previous examples.

STRUCTURE OF C++ PROGRAM

- **Preprocessor directives** (beginning with #) are not statements and do not produce any output. A preprocessor directive is meant for the compilers and preprocessor to perform certain actions before a program is executed. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

- **COMMENTS:**
  Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code. C++ supports two ways to insert comments namely;

  - // **Short comments** – discards everything up to end of line
  - /***Block comments***/ - discards everything in between /* */

STRUCTURE OF C++ PROGRAM

- The first of them, known as short comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line.
  We are going to add comments to our second program:

```
1  /* my second program in C++
2     with more comments */
3
4  #include <iostream>
5  using namespace std;
6
7  int main ()
8  {
9    cout << "Hello World! ";     // prints Hello World!
10   cout << "I'm a C++ program"; // prints I'm a C++ program
11   return 0;
12 }
```

```
Hello World! I'm a C++ program
```

- **NOTE:** If you include comments within the source code of your programs without using the comment characters combinations //, /* or */, the compiler will take them as if they were C++ expressions, most likely causing one or several errors within the program when you compile it.

STRUCTURE OF C++ PROGRAM

```
1  //My third program in C++
2
3  #include<iostream>
4  using namespace std;
5
6  int main()
7  {
8      cout<<"Hello world";
9      cout<<"We are programming in C++";
10     cin.get();
11     return 0;
12 }
```

- **cin.get();**
  This is another function call: It is used to accept input from the user. Many compilers will close the console window after running and executing the program. This command prevents the window from closing since the program will wait for an input from the user. The user may enter some value or press any key (enter or escape) to close the program thereby allowing the user to see the program run till the end.

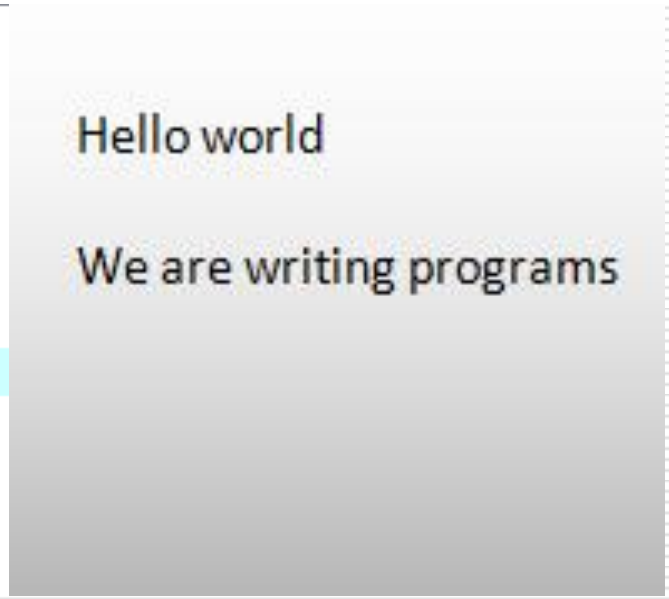
STRUCTURE OF C++ PROGRAM

```
1   //My fourth program in C++
2
3   #include<iostream>
4   using namespace std;
5
6   int main()
7   {
8       cout<<"Hello world";
9       cout<<"We are programming in C++";
10      system("pause");
11      return 0;
12  }
```

- **system ("pause");**
  This line of code prevents the console windows from closing until you press any key. It displays a message "Press any key to continue…" after printing/executing your **cout** statement. Hence it allows the user to see the program run till the end.

STRUCTURE OF C++ PROGRAM

```
1  //My fifth program in C++
2
3  #include<iostream>
4  using namespace std;
5
6  int main()
7  {
8      cout<<"\nHello world";
9      cout<<"\nWe are writing programs";
10     cin.get();
11     return 0;
12 }
```



- **\n - Newline**
  This code is used to create a New line. It helps you to put each text(s) or string on a new line. Using **\n (backlash with n)** together means the compiler will put each string or text on a separate line. The strings or texts will not be joined together.

STRUCTURE OF C++ PROGRAM

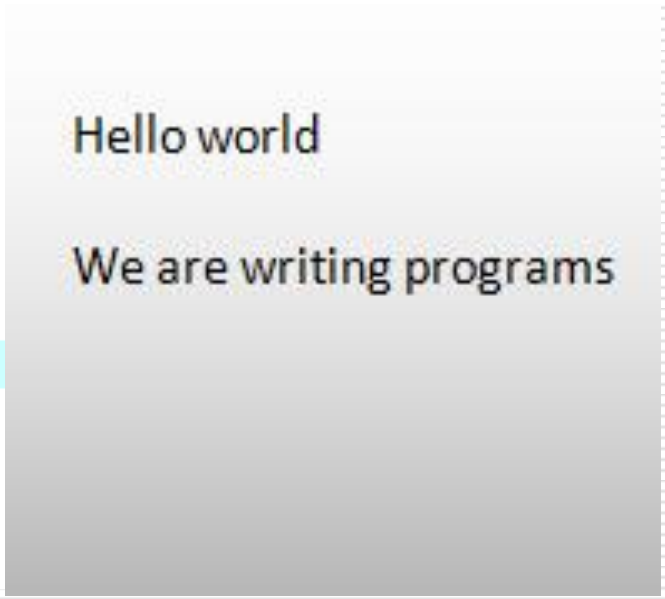
```
1  //My sixth program in C++
2
3  #include<iostream>
4  using namespace std;
5
6  int main()
7  {
8      cout<<"\nHello world";
9      cout<<"\nWe are writing programs";
10     cout<<"\nAt Sunford University College";
11     cout<<"\nProgramming is really fun";
12     cin.get();
13     return 0;
14 }
```

- **\n - Newline**

  This code is used to create a New line. It helps you to put each text(s) or string on a new line. Using **\n (backlash with n)** together means the compiler will put each string or text on a separate line. The strings or texts will not be joined together.

STRUCTURE OF C++ PROGRAM

```
1   //My seventh program in C++
2
3   #include<iostream>
4   using namespace std;
5
6   int main()
7   {
8       cout<<"Hello world"<<endl;
9       cout<<"We are writing programs"<<endl;
10      cin.get();
11      return 0;
12  }
```



- **endl - End of Line**
  This code is used to end a line. It indicates the end of a line hence whatever you type or input again will move to the next line. It is used to put each text(s) or string you type on a separate line. Using the command **endl (end of line)** means the compiler will put each string or text on a separate line. The strings or texts will not be joined together.

STRUCTURE OF C++ PROGRAM

```
1  //My eighth program in C++
2
3  #include<iostream>
4  using namespace std;
5
6  int main()
7  {
8      cout<<"Hello world"<<endl;
9      cout<<"We are writing programs"<<endl;
10     cout<<"At Sunford University College"<<endl;
11     cout<<"Programming is really fun"<<endl;
12     cin.get();
13     return 0;
14 }
```

```
Hello world
We are writing programs
At Sunford University College
Programming is really fun
```

- **endl - End of Line**
  This code is used to end a line. It indicates the end of a line hence whatever you type or input again will move to the next line. It is used to put each text(s) or string you type on a separate line. Using the command **endl (end of line)** means the compiler will put each string or text on a separate line. The strings or texts will not be joined together.

THE END

- T o b e c o n t i n u e d……