# UNIT I

**Introduction to OOP:** Procedure oriented programming, object oriented programming, basic concepts of OOP, benefits and applications of OOP, simple C++ program, namespace scope, structure of C++ Program, creating, compiling and linking a file.

**Tokens :** Keywords, identifiers, constants, basic data types, user defined data types, storage classes, derived data types, dynamic initialization of variables, reference variables, operators in C++, scope resolution operator, member dereferencing operators, memory management operators.

## 1.1 Introduction to OOP

### 1.1.1 Procedure Oriented Programming

**Q1. What is procedure oriented/ modular programming. Write about it.**

*Ans :*

Procedure oriented programming also known as modular programming. In this approach instructions organized according to their operations by dividing into small programs or small pieces called sub routine

- The given problem is divided in to a number of sub problems depending upon its functionality.
- The sub problems are called procedures or Methods.
- Any procedure can be called at any point during the program execution.
- The program has global and local variables.
- Global variables can be only be used .
- COBOL and Pascal are called as procedural oriented programming languages.

**Features**

1. Large Programs are divided in to small function or Procedure.
2. Uses Top-Down programming Approach.
3. Data moves freely from one function to another.
4. Most of the functions share common data.
5. Emphasis is given for algorithms.

**Advantages**

(i) **Modularity:** Dividing a program into small pieces 0 also called as divide and conqueror According to their operations into small module

(ii) **Reuseability:** Write code once and use more than once time

(ii) **Readablity :** Easy to read or understand

**Disadvantages**

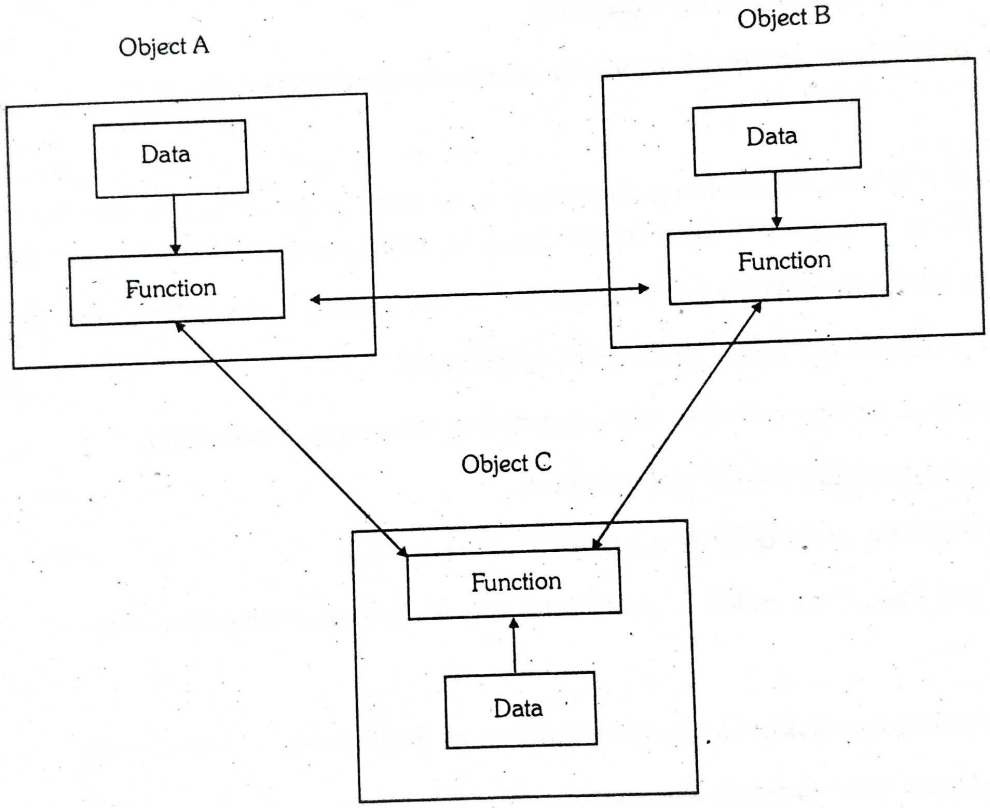
1. Very difficult identify which data is used by which function.
2. Error Correction is difficult.

### 1.1.2 Object Oriented Programming

**Q2. What is object oriented programming? Write about the feature of OOPs.**

*Ans :*

- The Program is divided into number of small units called Object. The data and function are build around these objects.
- The data of the objects can be accessed only by the functions associated with that object.
- The functions of one object can access the functions of other object.




**Features**

1. Emphasis is given on data rather than procedures.
2. Problems are divided into objects.
3. Data structures are designed such that they organize the object.
4. Data and function are tied together.
5. Data hiding is possible.
6. New data and functions can be easily loaded.
7. Object can communicate with each other using functions.
8. Bottom-up programming approach is used.

**Q3. Differentiate between procedure oriented and object oriented programming**

*Ans :*

**Difference between Procedural Programming and Object Oriented Programming**

| S.No. | Procedural Oriented Programming | Procedural Oriented Programming |
|---|---|---|
| 1. | In procedural programming, program is divided into small parts called functions. | In object oriented programming, program is divided into small parts called objects. |
| 2. | Procedural programming follows top down approach. | Object oriented programming follows bottom approach. |
| 3. | There is no access specifier in procedural programming. | Object oriented programming have acces specifiers like private, public, protected etc. |
| 4. | Adding new data and function is not easy | Adding new data and function is easy. |
| 5. | Procedural programming does not have any proper way for hiding data so it is less secure. | Object oriented programming provides datatg so it is more secure. |
| 6. | In procedural programming,overloading is not possible. | Overloading is possible in object oriented programming. |
| 7. | In procedural programming, function is more important than data. | Object oriented programming, data is more important than function. |
| 8. | Procedural programming is based on unreal world. | Object oriented programming is based on real world. |
| 9. | Examples: C. FORTRAN. Pascal. Basic etc. | Examples: C+ + , Java, Python, C# etc. |

### 1.1.3 Basic Concepts of OOP

**Q4. What are the basic concepts of OOPs? Explain them.**

*Ans :*

Object Oriented programming is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

**Features**

1. Object
2. Class
3. Data Hiding and Encapsulation
4. Dynamic Binding
5. Message Passing
6. Inheritance
7. Polymorphism

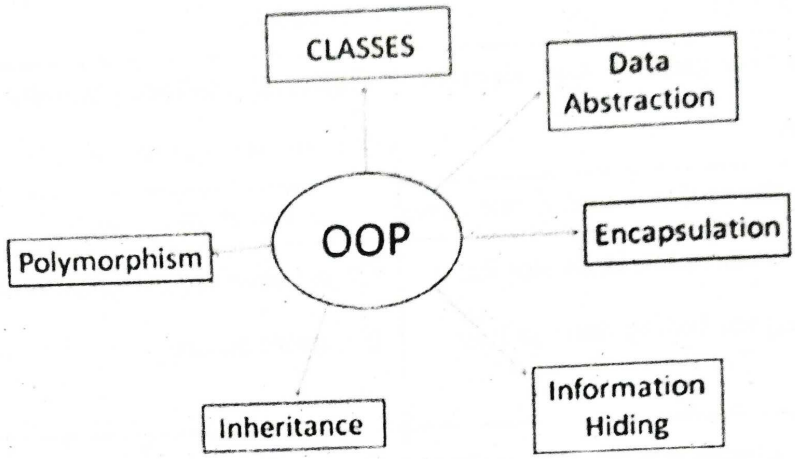



**Fig : Features of OOPs**

**Example :**

Human Beings are living forms, broadly categorized into two types, Male and Female.. Every Human beinghas two legs, two hands, two eyes, one nose, one heart etc. There are body parts that are common for Male and Female, but then there are some specific body parts, present in a Male which are not present in a Female, and some body parts present in Female but not in Males.

All Human Beings walk, eat, see, talk, hear etc. Now again, both Male and Female, performs some common functions, but there are some specifics to both, which is not valid for the other. For example : A Female can give birth, while a Male cannot, so this is only for the Female. let's see how all this is related OOPS.

- **Class**

  **Definition :** It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

  **Example :** In the above example, we can take Human Being as a class. A class is a blueprint for any functional entity which defines its properties and its functions. Like Human Being, having body parts, and performing various actions.

- **Objects**

  **Definition :**Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

**Example :** My name is Raju, and I am an instance/object of class Male., which is a kind of human being. We have a physical existence while a class is just a logical definition. We are the objects.

- **Abstraction**

**Definition :** Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes provide methods to the outside world to access & use the data variables, but the variables are hidden from direct access. This can be done access specifiers.

**Example :** Abstraction means, showcasing only the required things to the outside workd while hiding the details. Continuing our example, Human Being's can talk, walk, hear, eat, but the details are hidden from the outside world. We can take our skin as the Abstraction factor in our case, hiding the inside mechanism.

- **Encapsulation**

**Definition :** It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

**Example:** Our Legs are binded to help us walk. Our hands, help us hold things. This binding of the properties to functions is called Encapsulation.

- **Inheritance**

**Definition :** Inheritance is a way to reuse once written code again and again. The class which is inherited is called base calls & the class which inherits is called derived class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

**Example :** Considering Human Being a class, which has properties like hands, legs, eyes etc, and functions like walk, talk, eat, see etc. are the common properties for both Male and Female classes , so male and female inherit these common properties from the class Human Being, hence they can inherit everything from class Human Being using the concept of Inheritance.

- **Polymorphism**

**Definition :** As name suggests, Polymorphism means an ability to assume different forms at different places. In OOP, it is a language's ability to handle objects differently based on their runtime type and use. Polymorphism is briefly described as "one interface, many implementations". Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form..

Polymorphism could be static and dynamic both.

- Overloading is static polymorphism
- Overriding is dynamic polymorphism.

Overloading in simple words means two methods having same method name but takes different input parameters. This called static because, which method to be invoked will be decided at the time of compilation

**This can be expressed in two ways:**

- Operator Overloading
- Function Overloading

Overriding means a derived class is implementing a method of its super class.

- **Exception Handling**

Exception handling is a feature of OOP, to handle unresolved exceptions or errors produced at runtime.

## 1.1.4 Benefits and Applications of OOP

**Q5. What are the benefits of object oriented approach?**

*Ans :*

Object-oriented databases make the promise of reduced maintenance, code reusability, real world modeling, and improved reliability and flexibility. However, these are just promises and in the real world some users find that the object-oriented benefits are not as compelling as they originally believed. For example, what is code reusability?

Some will say that they can reuse much of the object-oriented code that is created for a system, but many say there is no more code reusability in object-oriented systems than in traditional systems. Code reusability is a subjective thing, and depends heavily on how the system is defined. The object-oriented approach does give the ability to reduce some of the major expenses associated with systems, such as maintenance and development of programming code.

Here are some of the benefits of the object-oriented approach:

1. **Reduced Maintenance**

   The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors.

2. **Real-World Modeling**

   Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviors. The model is based on objects, rather than on data and processing.

3. **Improved Reliability and Flexibility**

   Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviors can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.

4. **High Code Reusability**

   When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all superclasses in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviors which are defined to the system.

OOP stands for object oriented programming. It offers many benefits to both the developers and the users. Object-orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The primary advantages are:

**Merits**

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in an implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.
- Polymorphism can be implemented i.e. behavior of functions or operators or objects can be changed depending upon the operations.

**Demerits**

- It requires
- Inadequate
- Inability to
- Compile ti
- Unfamilira

**1.1.5 Simple**

**Q6. Write the it**

*Ans :*

```
/*
This is a
Call this
*/
#includ
usingna
// A C+
intmain
{
    cout <
    return
}
```

**Output :**

```
C++
Expla
Cons
/*
This
Call
*/
```

1. It is igno
2. Cor und

# i

**Demerits**

- It requires more data protection.
- Inadequate for concurrent problems
- Inability to work with existing systems.
- Compile time and run time overhead.
- Unfamiliraity causes training overheads.

### 1.1.5 Simple C++ Program

**Q6. Write the simple C++ Program. Explain it**

*Ans :*

```
/*
This is a simple C++ program.
Call this file Sample.cpp.
*/
#include <iostream>
usingnamespace std;
// A C++ program begins at main().
intmain()
{
cout << "C++ is power programming.";
return 0;
}
```

**Output :**

```
C++ is power programming.
```

Explanation : First C++ Program

Consider the following line in the programs –

```
/*
This is a simple C++ program.
Call this file Sample.cpp.
*/
```

1. It is Comment in C++ Language which is ignored by compiler
2. Comments are written for user understanding.

```
# include <iostream>
```

3. C++ uses different header files like C.
4. Include statement tells compiler to include Standard Input Output Stream inside C++ program.
5. This header is provided to user along with the compiler.

```
int main()
```

6. Each C++ program starts with the main function like C Programming.
7. Return type of the C++ main function is integer, whenever main function returns 0 value to operating system then program termination can be considered as smooth or proper.
8. Whenever some error or exception occurs in the program then main function will return the non zero value to operating system.

```
cout << "C++ is power programming.";
```

9. C++ Insertion operator (<<) is used to display value to the user on the console screen.

```
return 0;
```

10. return statement sends status report to the operating system about program execution whether program execution is proper or illegal.

### 1.1.6 Namespace Scope

**Q7. What is the use of name space scope in C++ ? Explain about it**

*Ans :*

A namespace is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope.

**Defining a Namespace**

A namespace definition begins with the keyword namespace followed by the namespace name as follows -

```
namespace namespace_name {
```

```
    // code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend (::) the namespace name as follows-

```
name::code;  // code could be variable or function.
```

Let us see how namespace scope the entities including variable and functions "

```
#include<iostream>
usingnamespace std;
// first name space
namespace first_space {
void func(){
cout <<"Inside first_space"<< endl;
}
}

// second name space
namespace second_space {
void func(){
cout <<"Inside second_space"<< endl;
}
}
int main (){
// Calls function from first name space.
first_space::func();
// Calls function from second name space.
second_space::func();
return0;
}
```

If we compile and run above code, this would produce the following result -

```
Inside first_space
Inside second_space
The using directive
```

You can also avoid prepending of namespaces with the using namespace directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace. The namespace is thus implied for the following code -

```
#include<iostream>
usingnamespace std;
// first name space
namespace first_space {
void func(){
cout <<"Inside first_space"<< endl;
}
}
// second name space
namespace second_space {
void func(){
cout <<"Inside second_space"<< endl;
}
}
usingnamespace first_space;
int main (){
// This calls function from first name space.
func();
return0;
}
```

If we compile and run above code, this would produce the following result-

```
Inside first_space
```

The 'using' directive can also be used to refer to a particular item within a namespace. For example, if the only part of the std namespace that you intend to use is cout, you can refer to it as follows

```
using std::cout;
```

Subsequent code can refer to cout without prepending the namespace, but other items in the std namespace will still need to be explicit as follows -

```
#include<iostream>
```

```
using std::cout;
int main (){
    cout <<"std::endl is used with std!"<<
std::endl;
return0;
}
```

If we compile and run above code, this would produce the following result-

```
std::endl is used with std!
```

Names introduced in a using directive obey normal scope rules. The name is visible from the point of the using directive to the end of the scope in which the directive is found. Entities with the same name defined in an outer scope are hidden.

**Discontiguous Namespaces**

A namespace can be defined in several parts and so a namespace is made up of the sum of its separately defined parts. The separate parts of a namespace can be spread over multiple files.

So, if one part of the namespace requires a name defined in another file, that name must still be declared. Writing a following namespace definition either defines a new namespace or adds new elements to an existing one "

```
namespace namespace_name {
// code declarations
}
```

Nested Namespaces

Namespaces can be nested where you can define one namespace inside another name space as follows -

```
namespace namespace_name1 {
// code declarations
namespace namespace_name2 {
// code declarations
 }
}
```

You can access members of nested namespace by using resolution operators as follows-

```
// to access members of namespace_name2
using namespace namespace_ name1::
namespace _name2;
// to access members of namespace:name1
using namespace namespace_name1;
```

In the above statements if you are using namespace_name1, then it will make elements of namespace_name2 available in the scope as follows

```
#include<iostream>
usingnamespace std;
// first name space
namespace first_space {
void func(){
cout <<"Inside first_space"<< endl;
}
// second name space
namespace second_space {
void func(){
cout <<"Inside second_space"<< endl;
}
}
}
usingnamespace first_space::second_space;
int main (){
// This calls function from second name space.
func();
return0;
}
```

If we compile and run above code, this would produce the following result -

```
Inside second_space
```

### 1.1.7 Structure of C++ Program

**Q8. Explain about the structure of C++.**

**(OR)**

**Briefly describe about various sections involved in C++ program.**

*Ans :*

**Structure of C++ Program**

Basically C++ program involves the following sections :

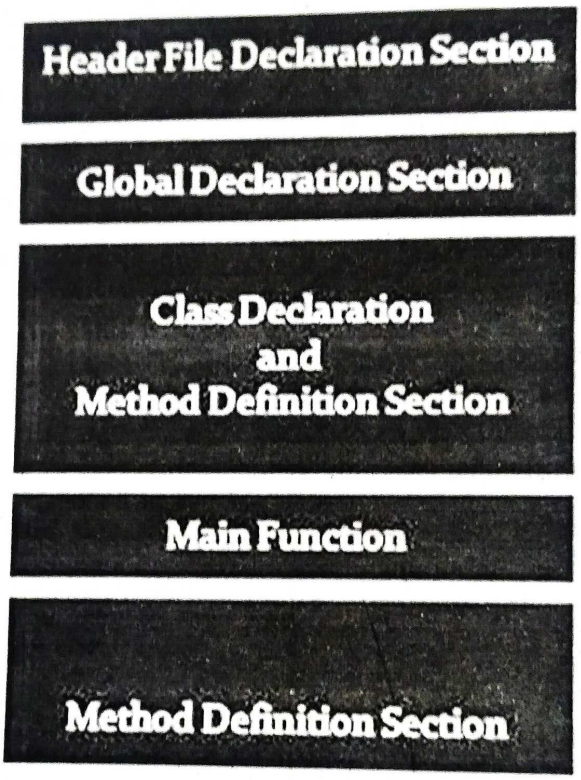



Fig: Structure of C++ Program

**Section 1 : Header File Declaration Section**

Header files used in the program are listed here.

1. Header File provides Prototype declaration for different library functions.
2. We can also include user define header file.
3. Basically all preprocessor directives are written in this section.

**Section 2 : Global Declaration Section**

1. Global Variables are declared here.
2. Global Declaration may include –
   - Declaring Structure
   - Declaring Class
   - Declaring Variable

**Section 3 : Class Declaration Section**

1. Actually this section can be considered as sub section for the global declaration section.
2. Class declaration and all methods of that class are defined here.

**Section 4 : Main Function**

1. Each and every C++ program always starts with main function.
2. This is entry point for all the function. Each and every method is called indirectly through main.
3. We can create class objects in the main.
4. Operating system call this function automatically.

**Section 5 : Method Definition Section**

This is optional section . Generally this method was used in C Programming.

### 1.1.8 Creating, Compiling and Linking a File

**Q9. Explain briefly, how to create a source file in C++.**

*Ans :*

Source code is the fundamental component of a computer program that is created by a programmer. It can be read and easily understood by a human being. When a programmer types a sequence of C language statements into Windows Notepad, for example, and saves the sequence as a text file, the text file is said to contain the source code.

The following steps are involved in creating the source file in C or C++

- The source code file is a text file on disk. It contains instructions for the computer that are written in the C programming language.
- You use a text editor to create the source code file. Most C compilers come with their own text editors. If yours did not, you can use a third-party text editor to do the job. (Some programmers prefer third-party text editors.)
- You can use a word processor to create your source code files. However, save the file as a "plain text" or "DOS text" or "ASCII" or "unformatted" file. (Using a word processor to create a source code file is a lot like using a 747 to drive to work; it's a little too much power for the job at hand.)
- The source code file ends with a C as its filename extension.
- The first part of the source code filename should be the name of the program you want to create.

**Q10. What are the steps involved in Creation and Running of a C++ Program? Explain them.**

*Ans :*

It is the job of programmer to write and test the program. This process is done in four steps. The following are the steps for creating and running programs:

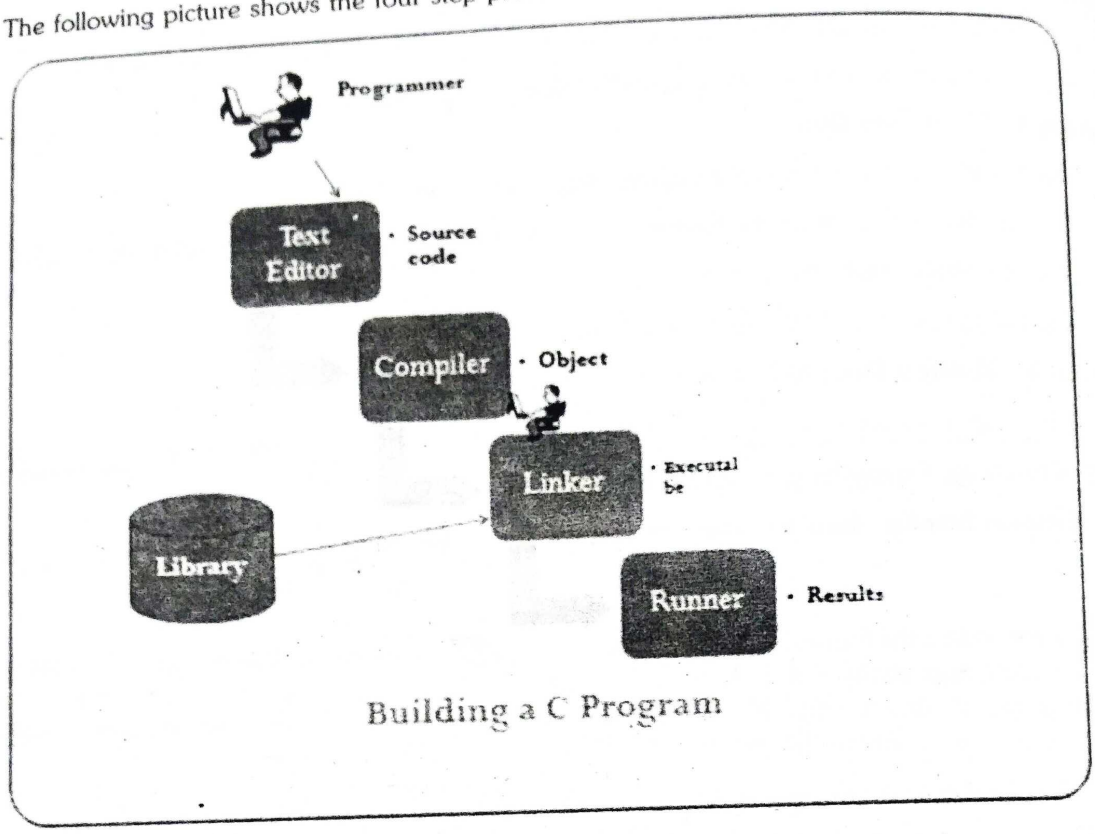
(a) Writing and Editing the Program.

(b) Compiling the Program.

(c) Linking the Program with the required library modules.

(d) Executing the Program

The following picture shows the four step process of creating and running the C program.




Building a C Program

**Writing and Editing the Program.**

To write and Edit the programs we need to use a particular software. The software used to write programs is known as a text editor. A text editor helps us to enter, change and store character data. After the program is completed the program is saved in a file to disk with an extension of .CPP .

This file will be input to the compiler, it is known as source file. Every compiler comes with associated text editor.

**Compiling Programs**

The code in a source file on the disk must be translated in to machine language. This is the job of compiler which translates code in source file stored on disk in to machine language. This translation process is known as compilation.

The entire high level program is converted into the executable machine code file.The Compiler which executes C programs is called as C Compiler.

Example Turbo C, Borland C, GC etc.,

**The C compiler is actually divided into two separate programs:**

- The preprocessor and
- The translator.

The preprocessor reads the source code and prepares it for the compiler. It scans the special instructions known as pre-processor commands. These commands tell the pre-processor to take for special code from libraries, make substitutions in the code.The result of pre-processing is called a translation unit.





After the pre
for compilation, th
of the program
translator reads
resulting object mo
with other prec
program. An obj
language.

This mod
because it does
functions includ

**Linking Progr**

The Lin
program's funct
executable pro

After the
files, another p
an executable
a linker and th
executable is

It links
references to
addresses. Ea
other object
in libraries oth
to tell the lin

The lin
told it to use

C pro
defined fun

Exam

Their
attached to

**Executing**

Once
for execut
operating
the progra

Gett
of an Op
Loader lo
it in to me

In a
reads data

After the preprocessor has prepared the code for compilation, the translator does the conversion of the program into machine language. The translator reads the translation unit and writes resulting object module to a file that can be combined with other precompiled units to form the final program. An object module is the code in machine language.

This module is not ready for execution because it does not have the required C and other functions included.

**Linking Programs**

The Linker assembles all functions, the program's functions and system's functions into one executable program.

After the compiler has created all the object files, another program is called to bundle them into an executable program file. That program is called a linker and the process of bundling them into the executable is called linking.

It links all the object files by replacing the references to undefined symbols with the correct addresses. Each of these symbols can be defined in other object files or in libraries. If they are defined in libraries other than the standard library, you need to tell the linker about them.

The linker looks at all the object files you have told it to use.

C programs are made up of many pre-defined functions.

Example: printf ( ) , scanf(), main()... etc

Their code exists in the library. They must be attached to our program.

**Executing Programs**

Once our program has been linked, it is ready for execution . To execute a program, we use operating system command, such as run to load the program in to main memory and execute it.

Getting program in to memory is the function of an Operating System programs called loader. Loader locates the executable program and reads it in to memory.

In a typical program execution, the program reads data for processing, either from user or from file. After the program processes the data, it prepares output. Data output can be to user's monitor or to a file. When program has executed, Operating System removes the program from memory.

## 1.2 Tokens

**Q11. Explain briefly about various tokens in C++**

***Ans :***

Each individual word and punctuation is referred to as a token in C++. Tokens are the smallest building block or smallest unit of a C++ program.

**These following tokens are available in C++:**

- Identifiers
- Keywords
- Constants
- Operators
- Strings

**Identifiers**

Identifiers are names given to different entries such as variables, structures, and functions. Also, identifier names should have to be unique because these entities are used in the execution of the program.

**Keywords**

Keywords is reserved words which have fixed meaning and its meaning cannot be changed. The meaning and working of these keywords are already known to the compiler. C++ has more numbers of keyword than C and those extra ones have special working capabilities.

**Operators**

C++ operator is a symbol that is used to perform mathematical or logical manipulations.

**Constants**

Constants are lik a variable, except that their value never changes during execution once defined.

**Strings**

Strings are objects that signify sequences of characters.

### 1.2.1 Keywords

**Q12. Write a short note on C++ keywords.**

*Ans :*

**Keywords**

These are the words used for special purposes or predefined tasks. These words should be writ in small letters or lowercase letters.

**List of the keyword used in c++ are :**

| | | | |
|---|---|---|---|
| Asm | Else | Operator | Template |
| Auto | Enum | Private | This |
| Break | Extern | Protected | Throw |
| Case | Float | Public | Try |
| Catch | For | Register | Typedef |
| Char | Friend | Return | Union |
| Class | Goto | Short | Unsigned |
| Const | If | Signed | Virtual |
| Continue | Inline | Sizeof | Void |
| Default | Int | Static | Volatile |
| Delete | Long | Struct | While |
| Double | New | Switch | - |

**Keywords cannot be used for the:**

1. Declaring Variable Name
2. Declaring Class Name
3. Declaring Function Name
4. Declaring Object Name





### 1.2.2 Iden

**Q13. Expla**

*Ans :*

**Identifier**

Vario called as Id

1.
2.
3.
4.

**The rules**

1. C+
2. The whil
3. Onl ide
4. Oth
5. Key

➢ So

So

### 1.2.3 C

**Q14. Ex**

*Ans :*

Co
Co
C

**Declari**

T
S
E

### 1.2.2 Identifiers

**Q13. Explain various identifiers used in C++**

*Ans :*

**Identifier**

Various data items with symbolic names in C++ is called as Identifiers. Following data items are called as Identifier in C++ –

1. Names of functions
2. Names of arrays
3. Names of variables
4. Names of classes

**The rules of naming identifiers in C++ :**

1. C++ is case-sensitive so that Uppercase Letters and Lower Case letters are different
2. The name of identifier cannot begin with a digit. However, Underscore can be used as first character while declaring the identifier.
3. Only alphabetic characters, digits and underscore (_) are permitted in C++ language for declaring identifier.
4. Other special characters are not allowed for naming a variable / identifier
5. Keywords cannot be used as Identifier.

➢ Some valid examples:- sum, a1, a2, _1, _a, average, a_b, x123y...

Some invalid examples:- 1a, a-b, float

### 1.2.3 Constants

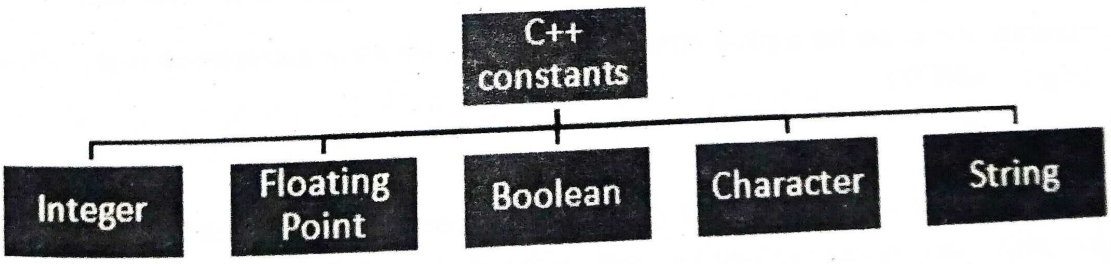
**Q14. Explain about Constants / Literals in C++**

*Ans :*

Constants are the items whose value cannot be changed during the program execution.

Constants are also called as Literals.

C++ supports five types of constants




**Declaring Constants**

To declare a constant we need to use a keyword 'const'.

Syntax: const type constant_name;

Example: int const SIDE = 50;

**Integer literals:**

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

**Following are other examples of various types of Integer literals:**

```
30      // int
30u     // unsigned int
30l     // long
30ul    // unsigned long
```

**Floating-point literals:**

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

Here are some examples of floating-point literals:

```
3.14159      // Legal
314159E-5L   // Legal
```

**Boolean literals:**

There are two Boolean literals and they are part of standard C++ keywords:

- A value of true representing true.
- A value of false representing false.

You should not consider the value of true equal to 1 and value of false equal to 0.

**Character literals:**

Character literals are enclosed in single quotes. If the literal begins with L (uppercase only), it is a wide character literal (e.g., L'x') and should be stored in wchar_t type of variable . Otherwise, it is a narrow character literal (e.g., 'x') and can be stored in a simple variable of char type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

**String literals**

String literals are enclosed in double quotes. A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separate them using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, world"

### 1.2.4 Basic Data Types

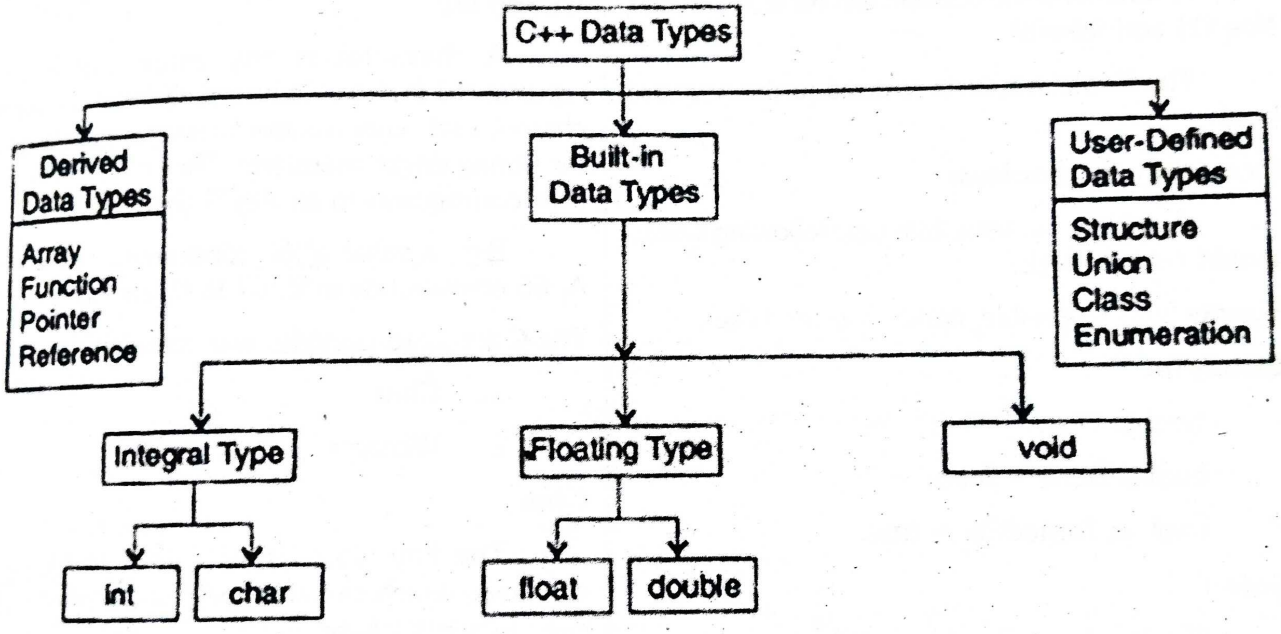
**Q15. Explain about the C++ basic /primitive data types.**

*Ans :*

A data type is a set of values and a set of predefined operations on those values. Every programming language deals with some data. For example to print some message or to solve some mathematical expression some data is necessary. C++ is very rich in data types.

We can broadly divide all data type in c++ into three categories.

- Primitive / Built in data types
- Derived Data types
- User defined Data types




**Primitive data types:**

The primitive data types in c language are the inbuilt data types. Programmers can use these data types when creating variables in their programs.

For example, a programmer may create a variable called "rollno" and define it as a integer data type. The variable will then store data as a integer values.

The primitive data types can be classified into three general categories.

- Void
- Integral
- Floating –point

**Void Type**

The void type is denoted by the keyword void, it means no values and no operations. Void can be mostly used to designate that a function has no parameters and no return value and also define a pointer to generic data.

For example- void msg (void); -

For example - int average(void);-

**Integral Type**

Integral data type cannot contain a fractional part, they are whole numbers.

C language has three integral types.

1. Boolean
2. Character
3. Integer

**Boolean**

A variable of the boolean can have two values: true (1) and false(0).

The Boolean type is referred to the keyword bool.

**Declaring the boolean**

To declare boolean data type following syntax should be followed:

**Syntax :** bool variable_name = true / false ;

**Example:**

```
bool is Pass;
bool is Done = false;
bool is TurnedOn = true;
```

**Note :**

The header stdbool.h must be used to use Boolean data type. Let us look at the following example

**Example:**

```
# include <stdbool.h>
int main()
{
bool gender[2] = {true, false};
return 0;
}
```

**Example**

```
//Program to print Boolean constants
#include <stdbool.h>
int main()
{
// local declaration
bool a = true;
bool b= false;
cout<< "The Boolean values are :%d %d\n", a,b;
}
```

It gives the following output:

The Boolean values are : 1 0

**Character**

A character is any value that can be represented in the computer's alphabet, known as character set. char is a special integer type designed for storing single characters. The integer value of a char corresponds to an ASCII character.

E.g., a value of 65 corresponds to the letter A, 66 corresponds to B, 67 to C, and so on.

The C standard provides two character types.

1. Char
2. Wchar_t

**Char**

The key word char is used to represent character, which can store only one character. The size of char is 1 byte.

**Declaring the char**

To declare char data type following syntax should be followed:

**Syntax:** char variable_name = 'value' ;

**Example :** char c1= 'a'; (The value stored in ' ' represent the value in ascii)

Here, c1 is a variable of type character which is storing a character 'a'.

For example:

For, 'a', value=97

For, 'A', value=65

For, '2', value=49 (if enclosed in ' ' even the integer value is also converted into ASCII value)

For, '&', value=33

Example:

```
int main()
{
    char a, b, c;
    a='E';
    b='I';
    c='O';
    cout << "value of a= %c," a;
    cout << "value of b= %c," b;
    cout << "value of c= %c," c;
    return 0;
}
```

**The following program will give the output :**

Value of a =E

Value of b =I

Value of c =O

**Wchar_t**

can hold a wide character. It is also required for declaring or referencing wide characters and wide strings.

wchar.h is a header file in the C standard library to represent wide characters.

***** NOTE :**

It is beyond the scope of introductory programming.

**IntegerType**

Integer data type is used to declare a variable that can store numbers without a decimal. They can be in various sizes and sign, depending on whether they will take up the negative numbers or not. Type qualifiers are used to specify the sizes ans signs of the variable.

**Declaring the integer**

To declare integer data type following syntax should be followed:

**Syntax**

[qualifier] int variable_name; /* qualifier is optional*/

Keyword int is used for declaring the variable with integer type. The type of qualifier is used to represent size and sign qualifiers.

C++ supports four different type of integer data types:

- byte
- Short
- Int
- Long

| S.No. | Data type | Size (in bytes) | Range |
|---|---|---|---|
| 1 | byte | 1 | +127 to -128 |
| 2 | short | 2 | + 32767 to -32767 |
| 3 | int | 4 | +2147483647 to -2147483647 |
| 4 | long | 8 | +9223372036854775807 to -9223372036854775808 |

**Floating Point Data types**

C++ supports three floating point types. Real, imaginary and complex. Floating point types are always signed.

**Real**

Real type can hold the values with fractions. C supports three different sizes of real type, float, double and long double

Here, the details of the real type with storage sizes and value ranges and their precision.

| Type | Storage size | Value range | Precision |
|---|---|---|---|
| Float | 4 byte | 1.2E-38 to 3.4E+38 | 6 decimal places |
| Double | 8 byte | 2.3E-308 to 1.7E+308 | 15 decimal places |
| long double | 10 byte | 3.4E-4932 to 1.1E+4932 | 19 decimal places |

**Declaring the real :** to declare real data type following syntax should be followed:

**Syntax :** type variable = value

**For example:**

```
float f1;
double f2;
```

Here, both f1 and f2 are floating type variables.

In C++, floating values can be represented in exponential form as well.

For example: float f3=43.345e2

**Example:**

```
main()
{
    float age;
    age = 10.5;
```

```
cout<< "The boy is over %f years old.\n", age;
}
```

It gives the following output :

The boy is over 10.5 years old

**Imaginary**

An imaginary number is the real number multiplied by square root of -1. Like real , an imaginary type can be three different sizes, float imaginary, double imaginary and long double imaginary.

**Complex**

A complex number is a combination of both real and imaginary number. It can be in three sizes. float complex, double complex and long double complex.

**Note**

Complex data types use the header file complex.h

**Declaring the complex**

To declare integer data type following syntax should be followed:

**Syntax :** type variable = value;

**Example :** double complex a;

### 1.2.5 User Defined Data Types

**Q16. What are the various types of user defined data types in C++?**

*Ans :*

**User-Defined Data Types**

Various user-defined data types provided by C++ are structures, unions, enumerations and classes.

**Structure, Union and Class**

Structure and union are the significant features of C language. Structure and union provide a way to group similar or dissimilar data types referred to by a single name. However, C++ has extended the concept of structure and union by incorporating some new features in these data types to support object-oriented programming.

**Enumeration**

An enumeration is a set of named integer constants that specify all the permissible values that can be assigned to enumeration variables. These set of permissible values are known as enumerators. For example, consider this statement.

```
enum country {US, UN, India, China};
// declaring an
//  enum type
```

In this statement, an enumeration data-type country (country is a tag name) , consisting of enumerators US, UN and so on, is declared. Note that these enumerators represent integer values, so any arithmetic operation can be performed on them.

By default, the first enumerator in the enumeration data type is assigned the value zero. Hence, the value of US is 0, value of UN is 1 and so on.

We can assign values explicitly to the enumerators as shown here.

```
enum country {US, UN=3, India, china} ;
```

In this declaration, the value of US is O by default, the value of UN is 3, India is 4 and soon.

Once an enum type is declared, its variables can be declared using this statement.

```
country countryl, country 2;
```

### 1.2.6 Storage Classes

**Q17. Write about various types of storage classes used in C++.**

*Ans :*

Storage classes are used to specify the lifetime and scope of variables. How storage is allocated for variables and How variable is treated by complier depends on these storage classes.

There are following storage classes, which can be used in a C++ Program

- auto
- register
- static
- extern

**1. The auto Storage Class**

The auto storage class is the default storage class for all local variables.

```
{
int mount;
autoint month;
}
```

The example above defines two variables with the same storage class, auto can only be used within functions, i.e., local variables.

**2. The register Storage Class**

The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
registerint miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

**3. The static Storage Class**

The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C++, when static is used on a class data member, it causes only one copy of that member to be shared by all objects of its class.

```
// demonstration of static storage class
#include<iostream>
// Function declaration
void func(void);
staticint count =10;/* Global variable */
main()
{
while(count—)
{ func(); }
return0;
}
// Function definition
void func(void)
{
staticint i =5;// local static variable
i++;
std::cout <<"i is "<< i ;
std::cout <<" and count is "<< count <<
std::endl;
}
```

When the above code is compiled and executed, it produces the following result:

```
i is6and count is9
i is7and count is8
i is8and count is7
i is9and count is6
i is10and count is5
i is11and count is4
i is12and count is3
i is13and count is2
i is14and count is1
i is15and count is0
```

**4. The extern Storage Class**

The extern storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern'





the variabl
has been
When you
other files
function.
file.
The exter
global var
// demons
// first file
# include
int count
externvoi
main()
{
count =5
write_exte
}
// second
# include
externint
void write
{
std::cout
}
Here, ext

**1.2.7 Derive**

**Q18. What are**

*Ans :*

Data type
various derived

- Array An
the eleme
element is
of an elem
- Function
task. In C
parts of a

the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will be used in other files also, then extern will be used in another file to give reference of defined variable or function. Just for understanding extern is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

```
// demonstration of extern storage class
// first file main.cpp
# include<iostream>
int count ;
externvoid write_extern();
main()
{
count =5;
write_extern();
}
// second file support.cpp
# include<iostream>
externint count;
void write_extern(void)
{
std::cout <<"Count is "<< count << std::endl;
}
```

Here, extern keyword is being used to declare count in another file.

### 1.2.7 Derived Data Types

**Q18. What are derived data types? How many types of derived data types used in C++**

*Ans :*

Data types that are derived from the built-in data types are known as derived data types. The various derived data types provided by C++ are arrays, junctions, references and pointers.

- Array An array is a set of elements of the same data type that are referred to by the same name. All the elements in an array are stored at contiguous (one after another) memory locations and each element is accessed by a unique index or subscript value. The subscript value indicates the position of an element in an array.
- Function A function is a self-contained program segment that carries out a specific well-defined task. In C++, every program contains one or more functions which can be invoked from other parts of a program, if required.

➢ Pointer A pointer is a variable that can store the memory address of another variable. Pointers allow to use the memory dynamically. That is, with the help of pointers, memory can be allocated or de-allocated to the variables at run-time, thus, making a program more efficient.

### 1.2.8 Dynamic Initialization of Variables

**Q19. What is known as dynamic initialization of variables?**

*Ans :*

The process of initializing variable at the time of its declaration at run time is known as dynamic initialization of variable.

Thus in dynamic initialization of variable a variable is assigned value at run time at the time of its declaration.

```
int main()
{
    int a;
    printf("Enter Value of a");
    scanf("%d",&a);
    int cube = a * a * a;
    printf("Cube is=%d\n",cube);
    return 0;
}
```

In above example variable cube is initialized at run time using expression a * a * a at the time of its declaration.

### 1.2.9 Reference Variables

**Q20. What is a reference variable in C++? Explain briefly.**

*Ans :*

Reference variable is an alternate name of already existing variable. It cannot be changed to refer another variable and should be initialized at the time of declaration and cannot be NULL. The operator '&' is used to declare reference variable.

The following is the syntax of reference variable.

```
datatype variable_name; // variable declaration
datatype& refer_var = variable_name; // reference variable
```

Here,

datatype " The datatype of variable like int, char, float etc.

variable_name " This is the name of variable given by user.

refer_var " The name of reference variable.

The following is an example of reference variable.

**Example**

```
# include<iostream>
usingnamespace std;
int main(){
int a =8;
int& b = a;
cout <<"The variable a : "<< a;
cout <<"\nThe reference variable r : "<< b;
return0;
}
```

**Output**

The variable a : 8

The reference variable r : 8

In the above program, a variable of integer type is declared and initialized with a value.

int a = 8;

The variable b is declared which is referring variable a.

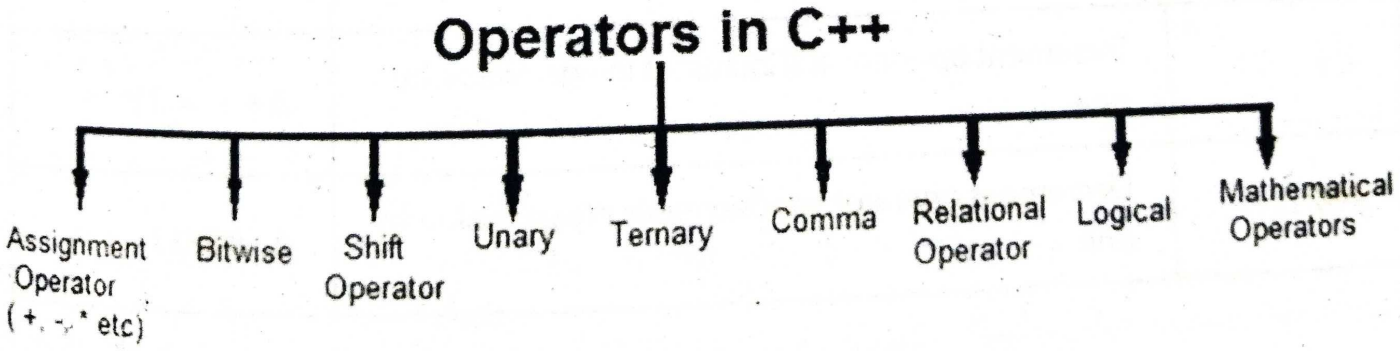
int & b = a;

### 1.2.10 Operators in C++

**Q21. Explain about various operators used in C++.**

***Ans :***

Operators are special type of functions, that takes one or more arguments and produces a new value. For example : addition (+), substraction (-), multiplication (*) etc, are all operators. Operators are used to perform various operations on variables and constants.




An operator is a symbol which tells compiler to take an action on operands and yield a value. The value on which operator operates is called as operands. C++ supports wide verity of operators. Supported operators are listed below –

| Operator | Explanation |
|---|---|
| Arithmetic Operators | Used for arithmetic operations |
| Relational Operators | Used for specifying relation between two operands |
| Logical Operators | Used for identifying the truth value of the expression |
| Bitwise Operators | Used for shifting the bits |
| Assignment Operators | Used for assigning the value to the variable |
| Misc Operators | - |

**Arithmetic Operators :**

Consider that we have A = 20 and B = 10

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands or variables | A + B = 30 |
| - | Subtracts second operand from the first | A – B = 10 |
| * | Multiplies both operands | A * B = 200 |
| / | Divides numerator by denominator | A / B = 2 |
| % | After dividing the numberator by denominator remainder will be returned after division | A % B = 0 |
| ++ | Increment operator will increases integer value by one | A++ = 21 |
| #8211; - | Decrement operator will decreases integer value by one | A- #8211; = 19 |

**Relational Operators :**

Consider that A = 40 and B = 20

| Symbol | Meaning | Example |
|---|---|---|
| > | Greater than | A > B returns true |
| < | Less than | A < B returns false |
| >= | Greater than equal to | A >= B returns false |
| <= | Less than equal to | A <= B returns false |
| == | Equal to | A == B returns false |
| != | Not equal to | A != B returns true |

| **Logical Operators** | | |
|---|---|---|
| Consider that | | A = 0 and B = 0 |
| Operator | Description | Example |
| Logical AND (&&) | If both the operands are non-zero then only condition be comes true | (A && B) is false. |
| Logical OR (\|\|) | If both the operands are zero then only condition becomes false | (A \|\| B) is true. |
| Logical NOT (!) | It will reverses the state of its operand i.e true will become false | (!A) is true |

**Bitwise Operators**

There are used to change individual bits into a number. They work with only integral data types like char,int and long and not with floating point values.

- Bitwise AND operators &
- Bitwise OR operator |
- And bitwise XOR operator ^
- And, bitwise NOT operator ~

They can be used as shorthand notation too, & = , |= , ^= , ~= etc.

**Shift Operators**

Shift Operators are used to shift Bits of any variable. It is of three types,

1. Left Shift Operator <<
2. Right Shift Operator >>
3. Unsigned Right Shift Operator >>>

**Unary Operators**

These are the operators which work on only one operand. There are many unary operators, but increment++ and decrement — operators are most used.

**Other Unary Operators**

Address of &, dereference *, new and delete, bitwise not ~, logical not !, unary minus - and unary plus +.

**Ternary Operator**

The ternary if-else ? : is an operator which has three operands.

```
int a = 10;
a > 5 ? cout << "true" : cout << "false"
```

**Comma Operator**

This is used to separate variable names and to separate expressions. In case of expressions, the value of last expression is produced and used.

**Example :**

```
int a,b,c; // variables declaration using comma operator
a=b++, c++; // a = c++ will be done.
```

**Q22. What is Operator Precedency? Explain it briefly.**

*Ans :*

In order to properly evaluate an expression such as 4 + 2 * 3, we must understand both what the operators do, and the correct order to apply them. The order in which operators are evaluated in a compound expression is called operator precedence. Using normal mathematical precedence rules (which state that multiplication is resolved before addition), we know that the above expression should evaluate as 4 + (2 * 3) to produce the value 10.

In C++, all operators are assigned a level of precedence. Those with the highest precedence are evaluated first.

Thus, 4 + 2 * 3 evaluates as 4 + (2 * 3) because multiplication has a higher level of precedence than addition.

If two operators with the same precedence level are adjacent to each other in an expression, the associativity rules tell the compiler whether to evaluate the operators from left to right or from right to left. For example, in the expression 3 * 4 / 2, the multiplication and division operators are both precedence level 5. Level 5 has an associativity of left to right, so the expression is resolved from left to right: (3 * 4) / 2 = 6.

- Precedence level 1 is the highest precedence level, and level 17 is the lowest. Operators with a higher precedence level get evaluated first.
- L->R means left to right associativity.
- R->L means right to left associativity.

| Operators | Read from |
|---|---|
| Unary operators: ! ~ ++ -- + - * & (typecast) sizeof | Left to right |
| * / % | Right to left |
| + - | Left to right |
| << > | Left to right |
| < <= > >= | Left to right |
| == != | Left to right |
| & | Left to right |
| ^ | Left to right |
| \| | Left to right |
| && | Left to right |
| \|\| | Left to right |
| ?: | Right to left |
| = += -= *= /= %= &= ^= \|= <<= >= | Right to left |
| , | Left to right |

### 1.2.11 Scope Resolution Operator

**Q23. What is scope resolution operator? Explain it.**

*Ans :*

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

For example, when the global and local variable or function has the same name in a program, and when we call the variable, by default it only accesses the inner or local variable without calling the global variable. In this way, it hides the global variable or function. To overcome this situation, we use the scope resolution operator to fetch a program's hidden variable or function.

**Uses of the scope resolution Operator**

1. It is used to access the hidden variables or member functions of a program.
2. It defines the member function outside of the class using the scope resolution.
3. It is used to access the static variable and static function of a class.
4. The scope resolution operator is used to override function in the Inheritance.

Program to access the hidden value using the scope resolution (::) operator

**Program1.cpp**

```
#include <iostream>
using namespace std;
// declare global variable
int num = 50;
int main ()
{
// declare local variable
int num = 100;
// print the value of the variables
cout << " The value of the local
variable num: " << num;

// use scope resolution operator (::) to
access the global variable
cout << "\n The value of the global
variable num: " << ::num;
return 0;
}
```

**Output**

The value of the local variable num: 100

The value of the global variable num: 50

### 1.2.12 Member Dereferencing Operators

**Q24. What is known as dereferencing operator? Write about it**

*Ans :*

A dereference operator, also known as an indirection operator, operates on a pointer variable.

It returns the location value, or l-value in memory pointed to by the variable's value. In the C programming language, the dereference operator is denoted with an asterisk ( * ).

The pointer related operator & and * are called as referening and dereferencing operators. The referencing operator (&) is a unary operator and it returns the address of its operand variable.

The dereferencing operator (*) is a unary operator that returns the value present at the specified address.

Consider the below program to understand use of member referencing and dereferencing operator.

```
#include<iostream.h>
void main()
{
int n= 15;
int *ptr;
ptr=&n; // referencing operator
cout<<"\nAddress of n ="<<&n;
cout<<"\nValue in variable ptr ="<< ptr;
cout<<"\nValue of n ="<<n;
cout<<"\nValue using dereferencing
operator="<<*ptr;
}
```

**Output :**

Address of n=0x8fa5fff4

Value in variable ptr=0x8fa5fff4

Value of n=15

Value using dereferencing operator=15

### 1.2.13 Memory Management Operators

**Q25. Write about the memory management operators used in C++.**

**(OR)**

**Write about the usage of new and delete operators in C++.**

*Ans :*

C++ has two types of memory management operators:

- new
- delete

These two memory management operators are used for allocating and releasing memory blocks in an efficient and convenient way.

**New operator**

It is used for dynamic storage allowance. The new operator allows you to create objects of any type.

The general syntax of this new operator under C++ is:

pointer variable = new datatype;

**Delete operator**

The C++ delete operator is used to release memory space when the object is no longer required. Once a new operator is used, it is effective to use the corresponding deletion operator to free up memory.

The general syntax for the delete operator in C++ is:

delete pointer variable;

**Example1:**

```
#include <iostream>
using namespace std;
int main()
{
// Allocates memory space using new operator for storing a integer datatype
int *a= new int;
*a=100;
cout << " The Output is:a= " << *a;
//Memory Released using delete operator
delete a;
return 0;
}
```

# Short Question and Answers

**1. What is procedure oriented/ modular programming.**

*Ans :*

Procedure oriented programming also known as modular programming. In this approach instructions organized according to their operations by dividing into small programs or small pieces called sub routine

- The given problem is divided in to a number of sub problems depending upon its functionality.
- The sub problems are called procedures or Methods.
- Any procedure can be called at any point during the program execution.
- The program has global and local variables.
- Global variables can be only be used .
- COBOL and Pascal are called as procedural oriented programming languages.

**2. What is object oriented programming?**

*Ans :*

- The Program is divided into number of small units called Object. The data and function are build around these objects.
- The data of the objects can be accessed only by the functions associated with that object.
- The functions of one object can access the functions of other object.

**3. Feature of OOPs.**

*Ans :*

1. Emphasis is given on data rather than procedures.
2. Problems are divided into objects.
3. Data structures are designed such that they organize the object.
4. Data and function are tied together.
5. Data hiding is possible.
6. New data and functions can be easily loaded.
7. Object can communicate with each other using functions.
8. Bottom-up programming approach is used.

**4. What are the basic concepts of OOPs?**

*Ans :*

Object Oriented programming is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

**5. Benefits of object oriented approach.**

*Ans :*

**1. Reduced Maintenance**

The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors.

**2. Real-World Modeling**

Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviors. The model is

based on objects, rather than on data and processing.

3. **Improved Reliability and Flexibility**

Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviors can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviors may be inherited from super-classes, and novel behaviors may be added without effecting existing systems functions.

4. **High Code Reusability**

When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all superclasses in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviors which are defined to the system.

## 6. Various tokens in C++.

*Ans :*

**Identifiers**

Identifiers are names given to different entries such as variables, structures, and functions. Also, identifier names should have to be unique because these entities are used in the execution of the program.

**Keywords**

Keywords is reserved words which have fixed meaning and its meaning cannot be changed. The meaning and working of these keywords are already known to the compiler. C++ has more numbers of keyword than C and those extra ones have special working capabilities.

**Operators**

C++ operator is a symbol that is used to perform mathematical or logical manipulations.

**Constants**

Constants are lik a variable, except that their value never changes during execution once defined.

**Strings**

Strings are objects that signify sequences of characters.

## 7. Explain various identifiers used in C++

*Ans :*

**Identifier**

Various data items with symbolic names in C++ is called as Identifiers. Following data items are called as Identifier in C++ –

1. Names of functions
2. Names of arrays
3. Names of variables
4. Names of classes

**The rules of naming identifiers in C++ :**

1. C++ is case-sensitive so that Uppercase Letters and Lower Case letters are different
2. The name of identifier cannot begin with a digit. However, Underscore can be used as first character while declaring the identifier.
3. Only alphabetic characters, digits and underscore (_) are permitted in C++ language for declaring identifier.
4. Other special characters are not allowed for naming a variable / identifier
5. Keywords cannot be used as Identifier.

➢ Some valid examples:- sum, a1, a2, _1, _a, average, a_b, x123y...

Some invalid examples:- 1a, a-b, float

## 8. What are derived data types?

*Ans :*

Data types that are derived from the built-in data types are known as derived data types. The various derived data types provided by C++ are arrays, junctions, references and pointers.

- Array An array is a set of elements of the same data type that are referred to by the same name. All the elements in an array are stored at contiguous (one after another) memory locations and each element is accessed by a unique index or subscript value. The subscript value indicates the position of an element in an array.
- Function A function is a self-contained program segment that carries out a specific well-defined task. In C++, every program contains one or more functions which can be invoked from other parts of a program, if required.

**9. What is Operator Precedency?**

*Ans :*

In order to properly evaluate an expression such as 4 + 2 * 3, we must understand both what the operators do, and the correct order to apply them. The order in which operators are evaluated in a compound expression is called operator precedence. Using normal mathematical precedence rules (which state that multiplication is resolved before addition), we know that the above expression should evaluate as 4 + (2 * 3) to produce the value 10.

In C++, all operators are assigned a level of precedence. Those with the highest precedence are evaluated first.

Thus, 4 + 2 * 3 evaluates as 4 + (2 * 3) because multiplication has a higher level of precedence than addition.

If two operators with the same precedence level are adjacent to each other in an expression, the associativity rules tell the compiler whether to evaluate the operators from left to right or from right to left. For example, in the expression 3 * 4 / 2, the multiplication and division operators are both precedence level 5. Level 5 has an associativity of left to right, so the expression is resolved from left to right: (3 * 4) / 2 = 6.

**10. Scope Resolution Operator.**

*Ans :*

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

For example, when the global and local variable or function has the same name in a program, and when we call the variable, by default it only accesses the inner or local variable without calling the global variable. In this way, it hides the global variable or function. To overcome this situation, we use the scope resolution operator to fetch a program's hidden variable or function.

**11. Dereferencing operator.**

*Ans :*

A dereference operator, also known as an indirection operator, operates on a pointer variable.

It returns the location value, or l-value in memory pointed to by the variable's value. In the C programming language, the dereference operator is denoted with an asterisk ( * ).

The pointer related operator & and * are called as referening and dereferencing operators.The referencing operator (&) is a unary operator and it returns the address of its operand variable.

The dereferencing operator (*) is a unary operator that returns the value present at the specified address.

# Choose the Correct Answers

1. Which concept allows you to reuse the written code? [ c ]
   (a) Encapsulation (b) Abstraction
   (c) Inheritance (d) Polymorphism

2. Which of the following is the correct way of declaring a pointer to a constant in C++? [ c ]
   (a) int * const pointer1=&p; (b) const * int pointer1=&p;
   (c) intconst * pointer1=&p; (d) int pointer1 * const=&p;

3. Which of the following is the address operator? [ c ]
   (a) @ (b) #
   (c) & (d) %

4. Which of the following is the scope resolution operator? [ c ]
   (a) . (b) *
   (c) :: (d) ~

5. Which of the following will not return a value? [ b ]
   (a) null (b) void
   (c) empty (d) free

6. The operator used for dereferencing or indirection is _____. [ a ]
   (a) * (b) &
   (c) -> (d) ->>

7. Identify the logical AND operator. [ b ]
   (a) || (b) &&
   (c) & (d) !

8. Which of the following can be considered as the members that can be inherited but not accessible in any class? [ c ]
   (a) Public (b) Protected
   (c) Private (d) Both A and C

9. Which of the following statements is correct about the class? [ a ]
   (a0 An object is an instance of its class
   (b) A class is an instance of its object
   (c) An object is the instance of the data type of that class
   (d) Both A and C

10. Which of the following is an abstract data type? [ c ]
    (a) int (b) float
    (c) class (d) string

# Fill in the Blanks

1. Wrapping data and its related functionality into a single entity is known as _____.
2. The size of an object or a type can be determined using which operator?
3. A reference variable must be initialized at the time of _____.
4. The _____ data type was used to specify the return type of a function when it is not returning any value.
5. _____ used in C++ provides an alias (alternative name) for a previously defined variable.
6. The _____ operator automatically returns the correct pointer type, so that there is no need to use a type cast.
7. The operator _____ is known as a compound assignment or short-hand assignment operator.
8. The _____ are just like variables except that their values cannot be changed.
9. _____ produce results of type bool which takes a value true or false.
10. C++ permits initialization of the variables at run time which is referred to as _____ initialization.

## Answers

1. Encapsulation
2. sizeof
3. declaration
4. void
5. alias
6. new
7. +=
8. named constant
9. Relational expressions
10. dynamic