Principles of Object Oriented Programming **12marks**

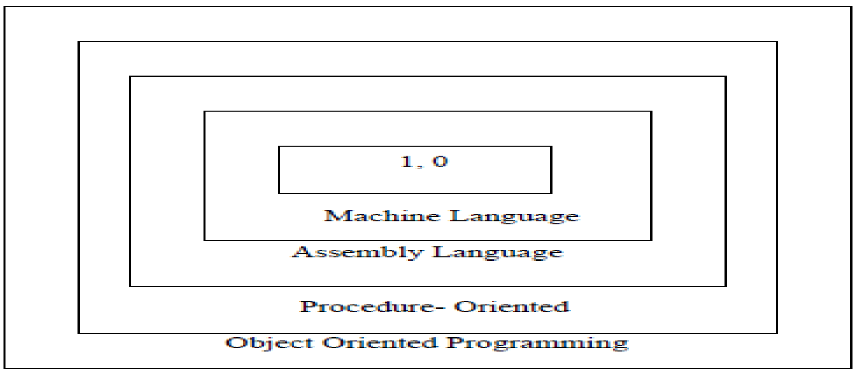
Objectives:

 State OOP's basic Concepts.
 Difference between OOP & POP.
 C++ Programming structure.

1.1 Its need & requirement, Procedure Oriented Programming (POP) verses Object Oriented Programming (OOP), Basic concepts of Object Oriented Programming, Object Oriented Languages, and Applications of OOP.
1.2 Beginning with C++: What is C++? , keywords, variables, constants, basic data types, operators, scope resolution operator, memory management operators, console input/output, structure of C++ program.

1.1 **Its need & requirement**

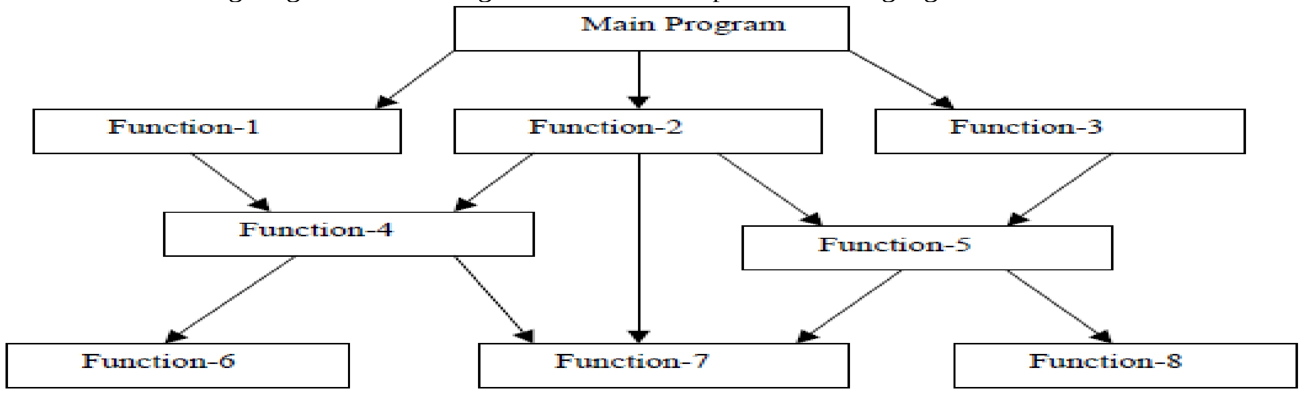
- Developments in software technology continue to be dynamic.
- This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry.



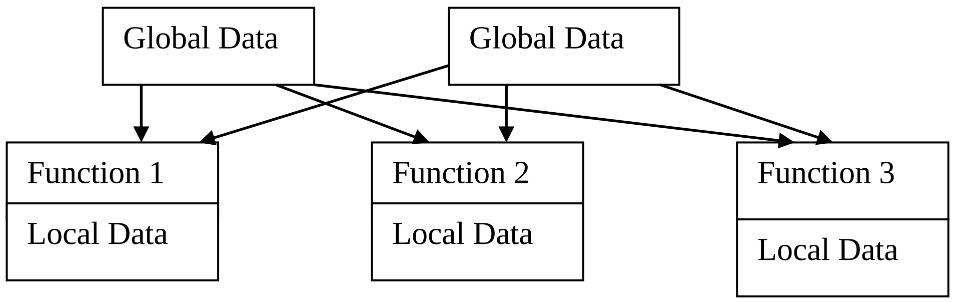
- To build today's complex software it is just not enough to put together a sequence of programming statements and sets of procedures and modules.
- Structured programming (like c) was a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired result in terms of bug-free, easy-to- maintain, and reusable programs.
- *Object Oriented Programming* (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts.
- It is a new way of organizing and developing programs and has ability implement the OOP concepts easily.

A STRUCTURE OF PROCEDURAL PROGRAMMING:

- Conventional programming languages such as COBOL, FORTRAN, C are commonly known as procedural oriented programming.
- In procedural oriented programming the primary focus is on functions. A number of functions are written to accomplish a specific task.
- The main concentration is done on development of function and very little attention is given to data that are being used by various functions.
- The following diagram shows the general structure of procedural languages.



Drawbacks/Limitations of Procedural language



1) In multi-function program many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own data known as Local Data. When use global data, Global data are more vulnerable to an accidental change by a function.
2) In large program it is very difficult to identify what data is used by which function. As program grows larger it becomes difficult to debug.
3) Another drawback of procedural approach is that it does not model the real world problems very well. This is because the functions are action – oriented and does not correspond to elements of the problem.

Characteristics of Procedural oriented programming:

- Emphasis on doing things (algorithm) rather than data.
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from one function to another.
- Functions transform data from one form to another form.
- Employs Top-Down approach in program design.

Characteristics / Features of Object Oriented Programming.

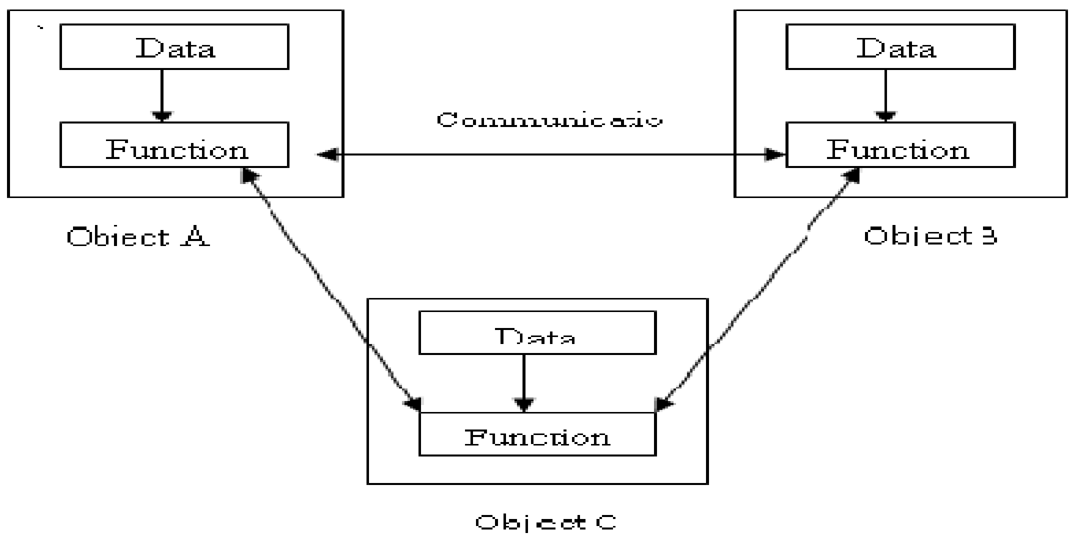
- Emphasis in data rather than procedures.
- Programs are divided into objects.
- Data structures are design such that they characterize object.
- Functions that operate on the data are tied together in the data structure called object.
- Data is hidden and cannot be accessed by external functions.
- Object may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- It follows Bottom-Up approach.

Differentiate Procedural oriented programming and object-oriented programming.

Procedural Oriented Programming	**Object Oriented programming**
It follows top-down approach in program design	It follows bottom-up approach in program design
It emphasis on procedure rather than data	It emphasis on data rather than procedure.
Larger programs are divided into smaller programs known as functions.	Programs are divided into Objects.
Here most of the data is global and data is not hidden and can be accessed by external functions.	Data is hidden and cannot be accesses by external functions.
New data cannot be easily created whenever necessary. It does not model the real-world.	New data and functions can be easily created whenever necessary. It models the real-world very well.
Example is C , Fortran ,COBOL	Example C++ , Java

STRUCTURE OF OBJECT ORIENTED PROGRAMMING:

- OOP treats the data as critical element in the program and does not allow it to flow around the system.
- It ties data more closely to the function that operate on it.
- It protects the data from accidental modification from external functions.
- In OOP the problem is divided in to different number of entities called Object.
- Object contains data and functions that operate on the data.
- The following diagram shows the organization of data and functions in OOP.
- The data of object can be accessed by functions associated with that object.
- However function of one object can communicate with function of other object.



BASIC CONCEPTS OF OOP:

- Object
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

OBJECT:

- "Objects are basic run time entities of an Object Oriented system. "
- It is nothing but an instance of class.
- An Object is encapsulation of data and code that operate on that data.
- The Object may represent a person, a place, a bank or any item that the program has to handle.
- They may also represent the user defined data types such as Complex number, Time or vector.
- Program is analyzed in terms of object and the nature of communication between them.
- When a program is executed, the objects interact with one another by sending messages. For e.g. if "customer" and "account" are two objects, then the customer object may sent a message to "account" object requesting for the bank balance.
  There are two ways of representing the object.

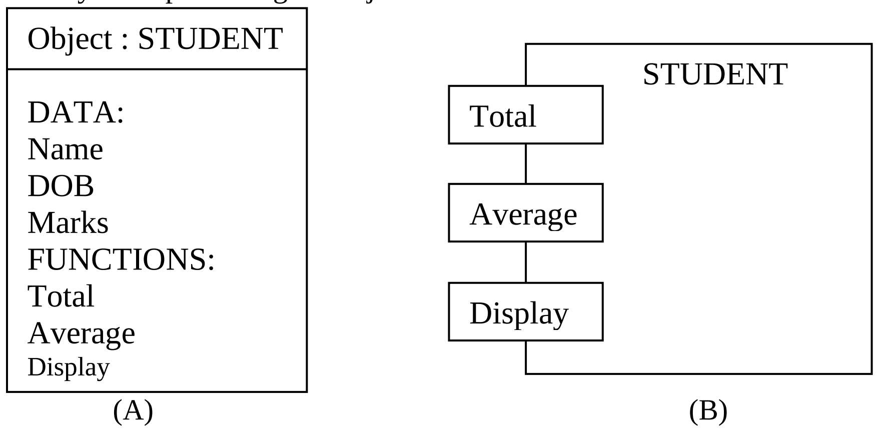


Fig shows two ways of representing the Object

CLASSES:

- The class is a way of binding data and functions associated with data together.
- The entire set of data and the functions can be made user defined data type with help of class. Thus <u>Class is a user defined data type</u>.
- <u>Class act as a template</u>, which is used <u>to create objects</u>. The class contains data and member functions that operate on the data.
- Once the class has been is defined, we can create any number of objects belonging to that class. Thus object is nothing but an instance (variable) of class. In other words <u>class is</u> a collection of similar objects.
- The Class defines what data and what member functions will be included in object of that class.

Example of class:

```
Class employee
{
        char name[40];   // Data member
        int empcode;       // Data member
public :
        void getdata()    // member function
        {
        }
        void showdata()// member function
        {
        }
};
void main()
{
 employee e1,e2; // creating two objects of class employee
}
```

DATA ABSTRACTION:

- Abstraction refers to the act of representing essential features without including the background details or explanation.
- Classes use the concept of abstraction. They encapsulate all essential properties of the objects that are to be created.

DATA ENCAPSULATION:

- "The mechanism of wrapping up of data and the functions associated with the data in to a single unit is called encapsulation." This unit is called object. The object contains data and member functions that operate on the data.
- Only the member functions can access the data.

DATA HIDING:

- In OOP the data and the functions that operate on the data are encapsulated in to a single unit known as object. Thus object contains data and the member functions that operate on the data. Only the member functions can access this data. The data is not accessible to the external functions.
  Only those functions, which are wrapped inside the class (member functions), can aces the data.
- This insulation of data from the external functions is called Data Hiding or Information Hiding.

INHERITANCE:

- "Inheritance is the process by which object of one class acquire the properties of object of another class."
- The concept of inheritance provides the idea of code reusability. This means that we can add additional features to an already existing class without modifying it.
- Once the class has been defined and compiled it can be derived by other classes.
- The old class is called the Base class and the new one is called Derived class.
- The derived class will inherit some or all features of base class; in addition it can have its own additional features.

POLYMORPHISM:

- "Polymorphism means the ability to take more than one forms". An operation may exhibit different behavior in different instances. The behavior depends upon the types of data used in the operation.
- Examples of polymorphism are operator overloading and function overloading.
- C++ allows operators and functions to operate in different ways, depending upon on what they are operating.
- Consider the example of operator overloading. For example if operands are two numbers, the + operator will generate a sum of two numbers. But if operands are strings the + operator generates the third string by concatenation. This mechanism is called operator overloading.
- Similarly functions can be overloaded to perform different task for different situation, depending upon the data types of the arguments.

DYANAMIC BINDING:

- "The term binding refers to the process of linking a function call to the code to be executed in response to that call."
- The linking of code to be executed in response to a particular function call is done at the run time and that is why it is also called Dynamic binding (late binding).
- It is associated with polymorphism and inheritance.

MESSAGE PASSING:

- An object-oriented program consists of a set of objects that communicate with each other.
- The process of programming in the object-oriented language, involves the following steps.

1) Creating classes
2) Creating objects from the classes.
3) Establishing communication among objects.

- Objects communicate with one another by sending and receiving information.
- A message for an object is a request for execution of a procedure.
- The receiving object will invoke a requested method/procedure and generates a desired result.
  For e.g. if "customer" and "account" are two objects, then the customer object may sent a message to "account" object requesting for the bank balance.

OBJECT – ORIENTD LANGUAGES:

The language that is specifically designed to support the object-oriented programming concepts makes it easier to implement them.

The Object Oriented languages can be classified into the following two categories:

1) Object-based programming language.
2) Object-Oriented programming language.

<u>Object – base programming language:</u>

Object based programming primarily supports encapsulation and object identity.

Major features that are supported by the object-based languages are

- Data Encapsulation
- Data Hiding mechanism
- Automatic initialization and clear up of object
- Operator Overloading

Language that supports programming with objects are said to be object based programming language.

'They do not support inheritance and Dynamic binding'.

Ada is a typical object based programming language.

<u>Object – Oriented Programming Language:</u>

Object – Oriented programming languages supports all features of object – based programming. Languages along with two additional features Inheritance and dynamic binding.

Object oriented programming can be characterized by the following statement.

Object – based features + (Inheritance + Dynamic binding)

C++, Java, SmallTalk, Simula, Eiffel, object pascal are typical Object – Oriented Programming languages.

Applications of OOP

Most popular application of OOP is in the area of user interface design such as windows. Hundreds of windowing systems have been developed using OOP techniques.

C++ is suitable for any programming task including

- development of editors, compilers
- Developing data bases
- Developing communication systems and complex real-time applications.
- Simulation and modeling.
- AI and Expert system

Beginning with C++:

What is C++?

- C++ was developed by <u>Bjarne Stroustrup</u> at AT & T Bell's Laboratory, Murray Hill, New Jersey, U.S.A in early eighties.
- A more powerful language that could support object-oriented programming features and still retain the power and elegance of C.
- The class was the major addition to the original 'C' language.
- C++ is an extension of 'C' language.
- C++ combines the power, elegance and flexibility of 'C' and the features of Object-Oriented Programming.
- The idea of name C++ came from the C increment operator ++, there for C++ is incremented version of 'C'.

- C++ adds classes, inheritance, function overloading and operator overloading. These features enable creating of abstract data types, inherit properties from existing data types and support polymorphism, thereby making C++ a truly object-oriented language.

Keywords:

- C++ reserves a set of 63 words for its own use.
- These words are called **keywords**, and each of these keywords has a special meaning with in the C++ language.
- The 15 keywords that are starred (*) were added to the language after its initial release, consequently, some older reference books or material may omit these.

asm auto bool * break case catch char class	const const_cast * continue default delete do double dynamic_cast *	else enum explicit * export * extern false * float for	friend goto if inline int long mutable * namespace *	new operator private protected public register reinterpret_cast * return	short signed sizeof static static_cast * struct switch template	this throw true * try typedef typeid * typename * union	unsigned using * virtual void volatile wchar_t * while

Identifiers

- Simply references to memory locations, which can hold values (data).
- Are formed by combining letters (both upper and lowercase), digits (0–9) and underscore (_).
- Rules for identifier naming are:
  1. The first character of an identifier must be a letter, an underscore (_) also counts as a letter.
  2. The blank or white space character is not permitted in an identifier.
  3. Can be any length. Internal identifier (do not have the external linkage) such as pre-processor macro names at least the first 31 characters are significant, also implementation dependent.
  4. Reserved words/keywords and characters such as main and # also cannot be used.

Variables

- Identifier that value may change during the program execution.
- Every variable stored in the computer's memory has a name, a value and a type.
- All variable in a C / C++ program must be declared before they can be used in the program.
- A variable name in C / C++ is any valid identifier, and must obey the rules mentioned above.
- Initializing a variable means, give a value to the variable, that is the variable's initial value and can be changed later on.
- Variable name are said to be **lvalue** (left value) because they can be used on the left side of an assignment operator.
- Constant are said to be **rvalue** (right value) because they only can be used on the right side of an assignment operator. For example: x = 20;
  x is lvalue, 20 is rvalue.
- Note that lvalue can also be used as rvalue, but not vice versa.

Example of the variable declaration

int x, y, z; **short** number_one; **char** Title; **float** commission, yield;

General form:

data_type variable_list;

Declaring and initializing variables examples:

```
1) int m, n = 10;                char * ptr = "TESTING";          float total, rate = 0.5;
   char user_response = 'n';     char color[7] = "green";
```

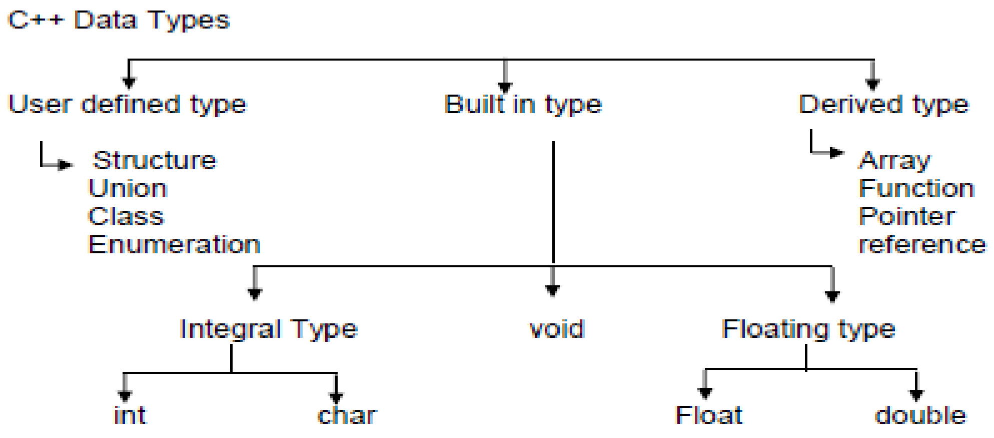
Constants

- Values that do not change during program execution.
- Can be integer, character or floating point type.
- To declare a constant, use keyword const as shown in the following variable declaration example:

const int day_in_week = 7;
const float total_loan = 1100000.35;

Character and String Constants
- A character constant is any character enclosed between two single quotation marks (' and ').
- When several characters are enclosed between two double quotation marks (" and "), it is called a string.
- Examples:
Character constants: '$' '*' ' ' 'z' 'P'
String constants, note that the blank space(s) considered as string: "Name: ", "Type of Fruit", "Day: " ," " .

Basic Data types



1) Built-in-type
i) Integral type . The data types in this type are int and char. The modifiers signed, unsigned, long & short may be applied to character & integer basic data type. The size of int is 2 bytes and char is 1 byte.
2) Void is used for.
i) To specify the return type of a function when it is not returning any value.
ii) To indicate an empty argument list to a function. E.g. - Void function1 (void)
iii) In the declaration of generic pointers. EX- void *gp
A generic pointer can be assigned a pointer value of any basic data type.
Ex . int *ip // this is int pointer
gp = ip //assign int pointer to void.
A void pointer cannot be directly assigned to their type pointers in c++ we need to use cast operator.
```
Ex - Void * ptr1;
      Char * ptr2;
       ptr2 = ptr1;
   is allowed in c but not in c++
       ptr2 = (char *)ptr1
   is the correct statement
```
3) **Floating type:** The data types in this are float & double. The size of the float is 4 byte and double is 8 byte. The modifier long can be applied to double & the size of long double is 10 byte.
2) User-defined type
i) The user-defined data type structure and union are same as that of C.
ii) Classes- Class is a user defined data type which can be used just like any other basic data type once declared. The class variables are known as objects.
3) Enumeration
i) An enumerated data type is another user defined type which provides a way of attaching names to numbers to increase simplicity of the code.
ii) It uses enum keyword which automatically enumerates a list of words by assigning them values 0, 1, 2 .etc.
```
iii) Syntax:-enum shape {
                          circle,
                          square,
                          triangle
                         }
```
Now shape become a new type name & we can declare new variables of this type.
EX . shape oval;
iv) In C++, enumerated data type has its own separate type. Therefore c++ does not permit an int value to be automatically converted to an enum value.
Ex. shape shapes1 = triangle, // is allowed

shape shape1 = 2; // Error in c++
shape shape1 = (shape)2; //ok
v) By default, enumerators are assigned integer values starting with 0, but we can over-ride the default value by assigning some other value.
EX:- enum colour {red, blue, pink = 3}; it will assign red to 0, blue to 1, & pink to 3 or
enum colour {red = 5, blue, green}; it will assign red to 5, blue to 6 & green to 7.
4) Derived Data types
1) Arrays
An array in c++ is similar to that in c, the only difference is the way character arrays are initialized. In c++, the size should be one larger than the number of character in the string where in c, it is exact same as the length of string constant.
Ex - char string1 [3] = “ab”; // in c++
char string1[2] = “ab”; // in c.
2) Functions
Functions in c++ are different than in c there is lots of modification in functions in c++ due to object orientated concepts in c++.
3) Pointers
Pointers are declared & initialized as in c.
Ex- int * ip; // int pointer
ip = &x; //address of x through indirection
c++ adds the concept of constant pointer & pointer to a constant pointer.
char const *p2 = .HELLO.; // constant pointer
typedef Declarations:
You can create a new name for an existing type using **typedef**. Following is the simple syntax to define a new type using typedef: **typedef type newname;**
For example, the following tells the compiler that feet is another name for int: **typedef int feet;**
Now, the following declaration is perfectly legal and creates an integer variable called distance: **feet distance;**

Operators: An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provides the following types of operators:
1) Arithmetic Operators 2) Relational Operators 3) Logical Operators
4) Bitwise Operators 5) Assignment Operators 6) Misc Operators

Arithmetic Operators: There are following arithmetic operators supported by C++ language:
Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Relational Operators: There are following relational operators supported by C++ language
Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.

>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators: There are following logical operators supported by C++ language
Assume variable A holds 1 and variable B holds 0, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators:
Bitwise operators work on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows:
A = 0011 1100
B = 0000 1101

A&B = 0000 1100 A|B = 0011 1101 A^B = 0011 0001 ~A = 1100 0011
The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators:
There are following assignment operators supported by C++ language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

Misc Operators
There are few other operators supported by C++ Language.

Operator	Description
sizeof	sizeof operator returns the size of a variable. For example, sizeof(a), where a is integer, will return 4.
Condition ? X : Y	Conditional operator. If Condition is true ? then it returns value X : otherwise value Y
,	Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
. (dot) and -> (arrow)	Member operators are used to reference individual members of classes, structures, and unions.
Cast	Casting operators convert one data type to another. For example, int(2.2000) would return 2.
&	Pointer operator & returns the address of an variable. For example &a; will give actual address of the variable.
*	Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.

New operators in C++ :

- << **Insertion Operator**
- >> **Extraction Operator**
- **: : Scope Resolution Operator**
- **: : * Pointer-to-member declaration**
- **->* Pointer-to-member Operator**
- **.* Pointer-to-member Operator**
- **delete Memory Release Operator**
- **endl Line Feed Operator**
- **new Memory Allocation Operator**
- **setw Field Width Operator**

Scope resolution operator in c++
C++ is a block structured language. Different program modules are written in various blocks. Same variable name can be used in different blocks. Scope of a variable extends from the point of declaration to the end of the block. A variable declared inside a block is ‘local’ variable. Blocks in C++ are often nested.

e.g.

```
….
{
        int x = 10;
        ….
        {
            int x = 20;
            ….
        }
        ….
}
```

The declaration of the inner block hides the declaration of same variable in outer block. This means, within the inner block, the variable x will refer to the data object declared therein. To access the global version of the variable, C++ provides scope resolution operator.
In the above example, x has a value of 20 but ::x has value 10.
Scope resolution operator (::) is used to

1) define a function outside a class
2) When we want to use a global variable but also has a local variable with same name.

EXAMPLE

```
1 ) #include <iostream>
 char c = 'a';     // global variable
 int main() {
  char c = 'b';   //local variable
   cout << "Local c: " << c << "\n";
  cout << "Global c: " << ::c << "\n";  //using scope resolution operator
   return 0;
}
2) #include <iostream>
int n = 12;   // A global variable
int main() {
   int n = 13;   // A local variable
   cout  << ::n << endl;  // Print the global variable: 12
   cout  << n   << endl;  // Print the local variable: 13
}
```

Memory Management Operators

The concept of arrays has a block of memory reserved. The disadvantage with the concept of arrays is that the programmer must know, while programming, the size of memory to be allocated in addition to the array size remaining constant.
In programming there may be scenarios where programmers may not know the memory needed until run time. In this case, the programmer can opt to reserve as much memory as possible, assigning the maximum memory space needed to tackle this situation. This would result in wastage of unused memory spaces. Memory management operators are used to handle this situation in C++ programming language.
malloc( ) and **calloc( )** functions are used to allocate memory dynamically at run time in c and c++. The function **free( )** to free dynamically the allocated memory.
in c++ The unary operators **new** and **delete** perform the task of allocating and freeing the memory.

- **new** to create an object
- **delete** to destroy an object

A data object created inside a block with **new**, will remain in existence until it is explicitly destroyed by using **delete**. Thus the life time of an object is directly under our control and is unrelated to the block structure of the program.
New operator:
The new operator in C++ is used for dynamic storage allocation. This operator can be used to create object of any type.
The general syntax of new operator in C++ is as follows:

Pointer-variable = new datatype;

In the above statement, new is a keyword and the pointer variable is a variable of type datatype.
For example:

```
int *a=new int;
```

In the above example, the new operator allocates sufficient memory to hold the object of datatype int and returns a pointer to its starting point. The pointer variable a holds the address of memory space allocated.
Dynamic variables are never initialized by the compiler. Therefore, the programmer should make it a practice to first assign them a value.
The assignment can be made in either of the two ways:

```
int *a = new int;
*a = 20;
            or
 int *a = new int(20);
```

delete operator:
The delete operator in C++ is used for releasing memory space when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory.
The general syntax of delete operator in C++ is as follows:

delete pointer-variable;

In the above example, delete is a keyword and the pointer variable is the pointer that points to the objects already created in the new operator.
Some of the important points the programmer must note while using memory management operators are described below:

- The programmer must take care not to free or delete a pointer variable that has already been deleted.
- Overloading of new and delete operator is possible (to be discussed in detail in later section on overloading).
- We know that sizeof operator is used for computing the size of the object. Using memory management operator, the size of the object is automatically computed.
- The programmer must take care not to free or delete pointer variables that have not been allocated using a new operator.
- Null pointer is returned by the new operator when there is insufficient memory available for allocation.

Example: to understand the concept of new and delete memory management operator in C++:

```
#include
void main()
{
//Allocates using new operator memory space in memory for storing a integer data type
int *a= new a;
*a=100;
cout << " The Output is:a="<<a;
//Memory Released using delete operator
delete a;
}
```

The output of the above program is
The Output is:a=100
In the above program, the statement: **int *a= new a;**
Holds memory space in memory for storing a integer datatype. The statement: ***a=100**
This denotes that the value present in address location pointed by the pointer variable a is 100 and this value of a is printed in the output statement giving the output shown in the example above. The memory allocated by the new operator for storing the integer variable pointed by a is released using the delete operator as:
delete a;
console input/output

CONSOLE INPUT/OUTPUT

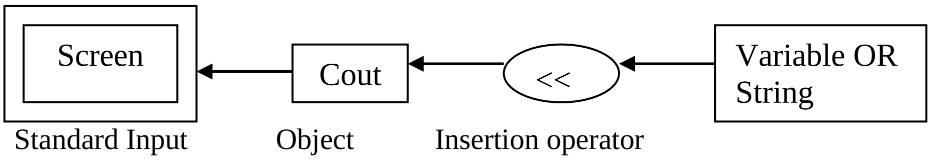
Displaying output using Insertion Operator (<<)

- << is called Insertion or PutTo operator.
- It is similar to printf () in c programming.
- It inserts the contents of variable on the R.H.S. side to the object specified on L.H.S.

```
Syntax: Cout << variable-name;        example: cout<<sum ;
        Cout << “string” ;            example : cout << “Hello” ;
```

- The identifier cout is a predefined object that represents the standard output stream.
- Here the standard output stream is Screen.

The statement, cout << “C++ is an Object-Oriented Language.”; causes the specified string to be displayed on the screen.



Extraction Operator (>>)

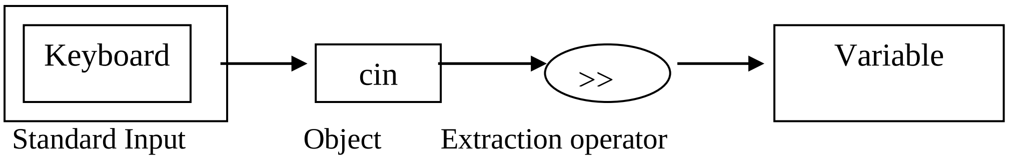
- >> is called Extraction or Getfrom operator.
- It is similar to scanf() in c programming.
- It extracts the value from the keyboard and assigns it to the variable on its R.H.S.

Syntax: cin << variable-name;

Example: cin >> num1 ;

The above statement causes the program to wait until the user type a number. The types number is then places in a variable num1.

- The identifier cin is a predefined object that represents the standard input stream.
- Here the standard input stream is keyboard.



FIRST C++ PROGRAM.

```
# include <iostream.h>   // include header file
void main()
{
cout << “C++ is an Object-Oriented Language.”;
}
```

In C++ we include <iostream.h> header file.

This directive causes the preprocessor to add the contents of the iostream file to the program. It contains declarations for the identifiers cout and << operator as well as declarations for cin and >> operator.

Comments in C++

Comments are non-executable statements. The compiler ignores the comments and do not execute it.

For single line comments use //

For multi-line comments use /* and */

Return statement in main()

- main() in C++ should end with return(0) statement otherwise warning/error might occur.
- Or another option is write void main()

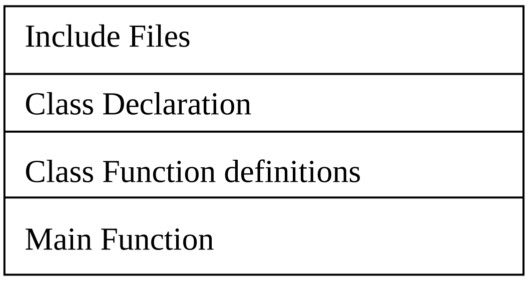
```
main()             // Default return data type is int
{
 ---
return(0);
}
```

- Generally main() function does not return any value and that is why its return data type is mentioned as void.
- If you do not write void main() then the compiler considers int as a return data type.
- If the funtion’s return data type is int , it must return a value.
- So to avoid this warning, place return(0) statement as a last statement of your program.

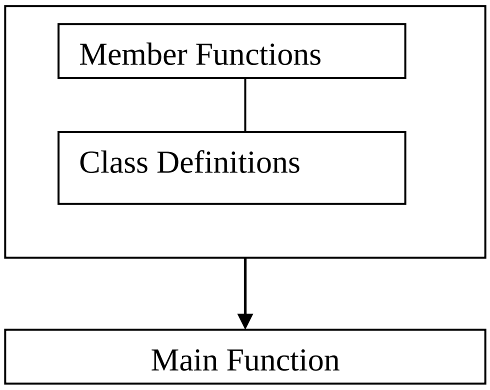
Write a program in c++ to add two numbers.

```
main()
{
 int num1 , num2 , sum;
cout<< “ Enter two numbers : “;
cin >> num1;
cin >> num2;
sum = num1 + num2;
cout << “Total = “ << sum;
return (0);
}
```

STRUCTURE OF C++ PROGRAM:



- The class declaration is placed in header files and definitions of member functions are placed in another filed.
- The main program that used the class is placed in the third file which includes the previous two files as well as any other files required.
- This approach is based on the concept of Client – Server model.
- The figure below shows the Client – Server Model.



BENEFITS OF OOP:

1. OOP supports Inheritance through which we can eliminate redundant code and extend the use of existing classes.
2. The principle of Data Hiding helps the programmer to build secure programs.
3. We can build programs from the standard working module that communicate with one another
4. This leads to saving of development time and gives higher productivity.
5. It is possible to have multiple instances of an object to co-exist without any interference.
6. The new data types can be easily created.
7. It is easy to partition the work in Objects rather than in functions.
8. Object – oriented systems can be easily upgraded from small to large systems.
9. Software complexity can be easily managed.

```
#include < iostream.h>
#include <conio.h>
void main( )
{
char str [10];
clrscr( );
int count=0;
cout<<”Enter String”;
cin>>str;
cout<<str;
for(int i=0; i<10;i++)
{
if(str[i]==’a’ ||str[i]==’e’||str[i]==’i’||str[i]==’o ’||str[i]==’u’)
{
count++;
}
}
cout<<count;
getch( ) ;
}

#include<iostream.h>
#include<conio.h>
void main()
{
int n,i,f=0,s=1,t;
clrscr();
cout<<"Enter the number of terms of Fibonacci series you want"<< endl;
cin >> n;
cout<<"First " << n << " terms of Fibonacci series are:- "<< endl;
for(i=0;i<n;i++)
{
if(i<=1)
{
t=i;
}
else
{
t=f+s;
f=s;
s=t;
}
cout<<t<<" ";
}
getch();
}


#include<iostream.h>
#include<conio.h>
{
private:
int hra, da, basic, sal;
public:
salary(int t, int b, int d=100, int h=200)
{
ta=t;
ba=b;
da=d;
hra=h;
}
void display()
{
sal=ta+da+hra+ba;
cout<<"salary="<<sal;
}
};
void main()
{
salary s(200,300);
s.display();
getch();
}
```

```
#include<iostream.h>
#include<conio.h>
float volume(float l)
{
return(l*l*l);
}
float volume(float r,float h)
{
return(3.14*r*r*h);
}
void main()
{
float length,radius,height,vol;
clrscr();
cout<<"\n Enter length of cube : ";
cin>>length;
vol=volume(length);
cout<<"\n Volume of Cube : "<<vol;
cout<<"\n\n Enter Radius of cylinder : ";
cin>>radius;
cout<<"\n Enter height of cylinder : ";
cin>>height;
vol=volume(radius,height);
cout<<"\n Volume of Cylinder : "<<vol;
getch();
}
```