The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages, and
2. Object-oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major feature that are required for object based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

1.8 Application of OOP

OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas. The most popular application of object-oriented programming, up to now, has been in the area of user interface design such as window. Hundreds of windowing systems have been developed, using the OOP techniques.

Real-business system are often much more complex and contain many more objects with complicated attributes and method. OOP is useful in these types of application because it can simplify a complex problem. The promising areas of application of OOP include:

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

The object-oriented paradigm sprang from the language, has matured into design, and has recently moved into analysis. It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software system but also its productivity. Object-oriented technology is certainly going to change the way the software engineers think, analyze, design and implement future system.

1.9 Introduction of C++

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's. Stroustrup, an admirer of Simula67 and a strong supporter of C, wanted to combine the best of both the languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. Therefore, C++ is an extension of C with a major addition of the class construct feature of Simula67. Since the class was a major addition to the original C language, Stroustrup initially called the new language 'C with classes'. However, later in 1983, the name was changed to C++. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented version of C.

C+ + is a superset of C. Almost all c programs are also C++ programs. However, there are a few minor differences that will prevent a c program to run under C++ complier. We shall see these differences later as and when they are encountered.

The most important facilities that C++ adds on to C care 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.

1.9.1 Application of C++

C++ is a versatile language for handling very large programs; it is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life applications systems.

- Since C++ allow us to create hierarchy related objects, we can build special object-oriented libraries which can be used later by many programmers.
- While C++ is able to map the real-world problem properly, the C part of C++ gives the language the ability to get closed to the machine-level details.
- C++ programs are easily maintainable and expandable. When a new feature needs to be implemented, it is very easy to add to the existing structure of an object.
- It is expected that C++ will replace C as a general-purpose language in the near future.

1.10 Simple C++ Program

Let us begin with a simple example of a C++ program that prints a string on the screen.

```
                              Printing A String
#include<iostream>
Using namespace std;
int main()
{
cout<<" c++ is better than c \n";
return 0;
}
```

Program 1.10.1

This simple program demonstrates several C++ features.

1.10.1 Program feature

Like C, the C++ program is a collection of function. The above example contain only one function **main().** As usual execution begins at main(). Every C++ program must have a **main()**. C++ is a free form language. With a few exception, the compiler ignore carriage return and white spaces. Like C, the C++ statements terminate with semicolons.

1.10.2 Comments

C++ introduces a new comment symbol // (double slash). Comment start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line, and whatever follows till the end of the line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// some of its features
```

The C comment symbols /*,*/ are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of
   C++ program to illustrate
    some of its features
*/
```

1.10.3 Output operator

The only statement in program 1.10.1 is an output statement. The statement

Cout<<"C++ is better than C.";

Causes the string in quotation marks to be displayed on the screen. This statement introduces two new C++ features, cout and <<. The identifier cout(pronounced as C out) is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices. The operator << is called the insertion or put to operator.

1.10.4 The iostream File

We have used the following #include directive in the program:

#include <iostream>

The #include directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file. The header file **iostream.h** should be included at the beginning of all programs that use input/output statements.

1.10.5 Namespace

Namespace is a new concept introduced by the ANSI C++ standards committee. This defines a scope for the identifiers that are used in a program. For using the identifier defined in the **namespace** scope we must include the using directive, like

Using namespace std;

Here, std is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. **Using** and **namespace** are the new keyword of C++.

1.10.6 Return Type of main()

In C++, main () returns an integer value to the operating system. Therefore, every main () in C++ should end with a return (0) statement; otherwise a warning an error might occur. Since main () returns an integer type for main () is explicitly specified as **int.** Note that the default return type for all function in C++ is **int.** The following main without type and return will run with a warning:

```
main ()
{
  …………..
  ………….
}
```

1.11 More C++ Statements

Let us consider a slightly more complex C++ program. Assume that we should like to read two numbers from the keyboard and display their average on the screen. C++ statements to accomplish this is shown in program 1.11.1

AVERAGE OF TWO NUMBERS

```
        #include<iostream.h> // include header file

        Using namespace std;

        Int main()

        {

                Float number1, number2,sum, average;
                Cin >> number1;        // Read Numbers
                Cin >> number2;        // from keyboard
                Sum = number1 + number2;
                Average = sum/2;
                Cout << ”Sum = “ << sum << “\n”;
                Cout << “Average = “ << average << “\n”;

                Return 0;

        }       //end of example
```

The output would be:

```
Enter two numbers: 6.5  7.5
Sum = 14
Average = 7
```

Program 1.11.1

1.11.1 Variables

The program uses four variables number1, number2, sum and average. They are declared as type float by the statement.

```
        float number1, number2, sum, average;
```

All variable must be declared before they are used in the program.

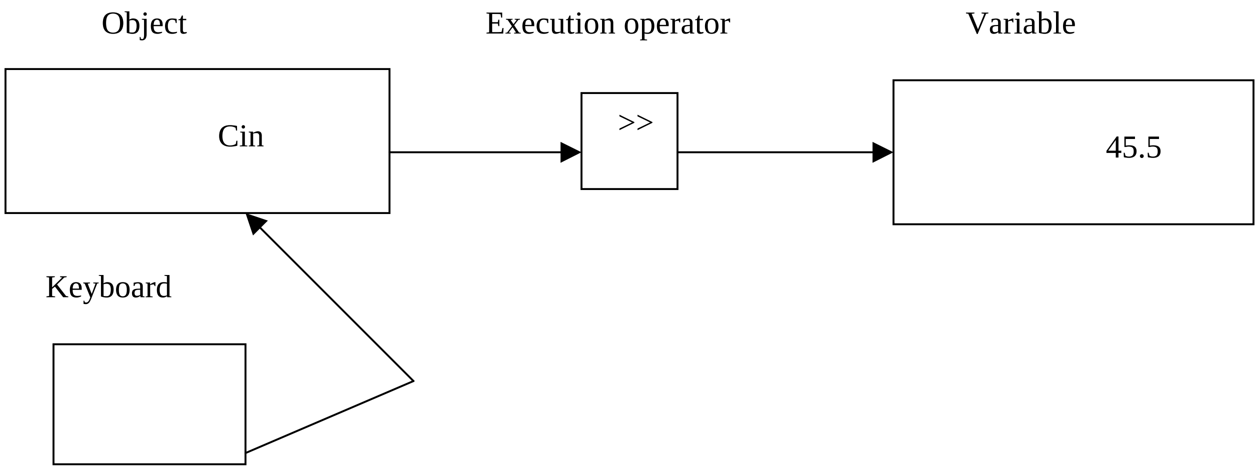
1.11.2 Input Operator

The statement

```
cin >> number1;
```

Is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier cin (pronounced ‘C in’) is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represents the keyboard.

The operator >> is known as extraction or get from operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right fig 1.8. This corresponds to a familiar scanf() operation. Like <<, the operator >> can also be overloaded.



1.8 Input using extraction operator

1.11.3 Cascading of I/O Operators

We have used the insertion operator << repeatedly in the last two statements for printing results.

The statement

```
Cout << “Sum = “ << sum << “\n”;
```

First sends the string “Sum = “ to cout and then sends the value of sum. Finally, it sends the newline character so that the next output will be in the new line. The multiple use of << in one statement is called cascading. When cascading an output operator, we should ensure necessary blank spaces between different items. Using the cascading technique, the last two statements can be combined as follows:

```
Cout << “Sum = “ << sum << “\n”
        << “Average = “ << average << “\n”;
```

This is one statement but provides two line of output. If you want only one line of output, the statement will be:

```
        Cout << “Sum = “ << sum << “,”
              << “Average = “ << average << “\n”;
```

The output will be:

```
        Sum  = 14, average = 7
```

We can also cascade input iperator >> as shown below:

```
Cin >> number1 >> number2;
```

The values are assigned from left to right. That is, if we key in two values, say, 10 and 20, then 10 will be assigned to munber1 and 20 to number2.

1.12 An Example with Class

- One of the major features of C++ is classes. They provide a method of binding together data and functions which operate on them. Like structures in C, classes are user-defined data types.

Program 1.12.1 shows the use of class in a C++ program.

USE OF CLASS

```
#include<iostream.h> // include header file

using namespace std;
class person
{

        char name[30];
        Int age;

        public:
            void getdata(void);
            void display(void);
};
void person :: getdata(void)
{
        cout << “Enter name: “;
        cin >> name;
        cout << “Enter age: “;
        cin >> age;
}
Void person : : display(void)
{
        cout << "\nNameame: " << name;
        cout << "\nAge: " << age;
}

Int main()
{
        person p;
        p.getdata();
        p.display();

        Return 0;

}       //end of example
```

PROGRAM 1.12.1

The output of program is:

```
  Enter Name: Ravinder
  Enter age:30
  Name:Ravinder
  Age: 30
```

The program define **person** as a new data of type class. The class person includes two basic data type items and two function to operate on that data. These functions are called **member function**. The main program uses **person** to declare variables of its type. As pointed out earlier, class variables are known as objects. Here, p is an object of type **person**. Class object are used to invoke the function defined in that class.

1.13 Structure of C++ Program

As it can be seen from program 1.12.1, a typical C++ program would contain four sections as shown in fig. 1.9. This section may be placed in separate code files and then compiled independently or jointly.
It is a common practice to organize a program into three separate files. The class declarations are placed in a header file and the definitions of member functions go into another file. This approach enables the programmer to separate the abstract specification of the interface from the implementation details (member function definition).
Finally, the main program that uses the class is places in a third file which "includes: the previous two files as well as any other file required.

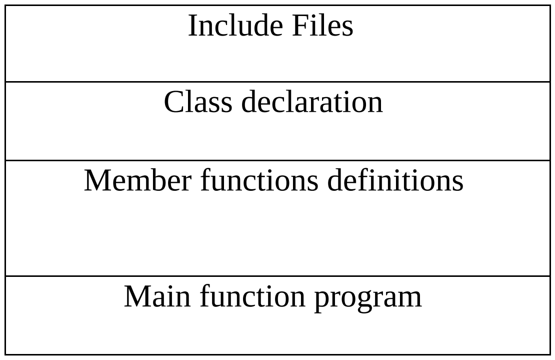
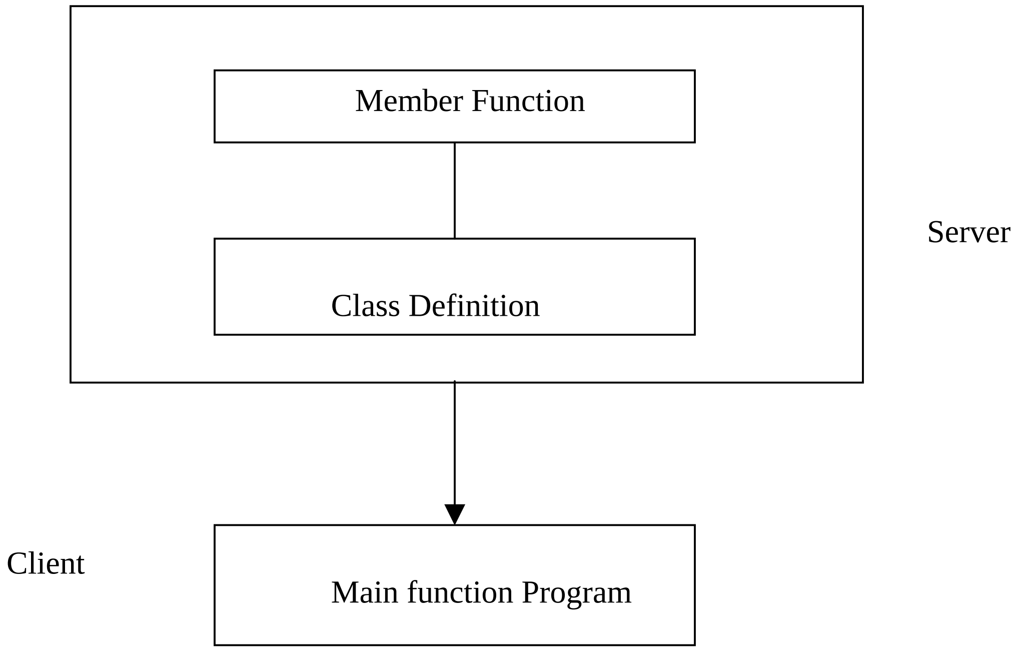


Fig 1.9 Structure of a C++ program

This approach is based on the concept of client-server model as shown in fig. 1.10. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.

Fig. 1.10 *The client-server model*



1.14 Creating the Source File

Like C programs can be created using any text editor. Foe example, on the UNIX, we can use vi or ed text editor for creating using any text editor for creating and editing the source code. On the DOS system, we can use endlin or any other editor available or a word processor system under non-document mode.

Some systems such as Turboc C++ provide an integrated environment for developing and editing programs

The file name should have a proper file extension to indicate that it is a C++ implementations use extensions such as .c,.C, .cc, .cpp and .cxx. Turboc C++ and Borland C++ use .c for C programs and .cpp(C plus plus) for C++ programs. Zortech C++ system use .cxx while UNIX AT&T version uses .C (capital C) and .cc. The operating system manuals should be consulted to determine the proper file name extension to be used.

1.15 Compiling and Linking

The process of compiling and linking again depends upon the operating system. A few popular systems are discussed in this section.

Unix AT&T C++

This process of implementation of a C++ program under UNIX is similar to that of a C program. We should use the “cc” (uppercase) command to compile the program. Remember, we use lowercase “cc” for compiling C programs. The command

CC example.C

At the UNIX prompt would compile the C++ program source code contained in the file **example.C**. The compiler would produce an object file **example.o** and then automatically link with the library functions to produce an executable file. The default executable filename is **a. out**.

A program spread over multiple files can be compiled as follows:

CC file1.C file2.o

The statement compiles only the file **file1.C** and links it with the previously compiled **file2.o** file. This is useful when only one of the files needs to be modified. The files that are not modified need not be compiled again.

Turbo C++ and Borland C++

Turbo C++ and Borland C++ provide an integrated program development environment under MS DOS. They provide a built-in editor and a menu bar includes options such as File, Edit, Compile and Run.

We can create and save the source files under the **File option,** and edit them under the **Edit option.** We can then compile the program under the **compile** option and execute it under the **Run option.** The **Run option** can be used without compiling the source code.

Summary

- Software technology has evolved through a series of phases during the last five decades.
- POP follows top-down approach where problem is viewed as sequence of task to be performed and functions are written for implementing these tasks.

- POP has two major drawbacks:
- Data can move freely around the program.
- It does not model very well the real-world problems.
- OOP was inventing to overcome the drawbacks of POP. It follows down -up approach.
- In OOP, problem is considered as a collection of objects and objects are instance of classes.
- Data abstraction refers to putting together essential features without including background details.
- Inheritance is the process by which objects of one class acquire properties of objects of another class.
- Polymorphism means one name, multiple forms. It allows us to have more than one function with the same name in a program.
- Dynamic binding means that the code associated with a given procedure is not known until the time of the run time.
- Message passing involves specifying the name of the object, the name of the function and the information to be sent.
- C++ is a superset of C language.
- C++ ads a number of features such as objects, inheritance, function overloading and operator overloading to C.
- C++ supports interactive input and output features and introduces anew comment symbol // that can be used for single line comment.
- Like C programs, execution of all C++ program begins at main() function.

Keywords:

- Assembly Language
- Bottom up Programming
- C++
- Classes
- Data Abstraction
- Data Encapsulation
- Data Hiding
- Data Member
- Dynamic Binding
- Early Binding
- Function overloading
- Functions
- Global Data
- Hierarchical Classification
- Inheritance
- Late Binding
- #include
- Main( )
- Local data
- Machine Language
- Member Function
- Message Passing
- Methods
- Modular Programming
- Multiple Inheritances
- Object Based Programming
- Objective C
- Object Oriented Language
- Object Oriented Programming
- Objects
- Operator Overloading
- Polymorphism
- Procedure Oriented Programming
- Reusability
- Top down Programming
- Extraction Operator

- Cascading
- Namespace
- Class
- Object
- Operator overloading
- Comments
- Output operator
- cout
- edlin
- return ()
- Float
- Get from Operator
- Input operator
- Turbo c++
- iostream
- int
- using
- iostream.h
- windows
- Keyboard

Questions

1. What are the major issues facing the software industry today?
2. What is POP? Discuss its features.
3. Describe how data are shared by functions in procedure-oriented programs?
4. What is OOP? What are the difference between POP and OOP?
5. How are data and functions organized in an object-oriented program?
6. What are the unique advantages of an object-oriented programming paradigm?
7. Distinguish between the following terms:
   (a) Object and classes
   (b) Data abstraction and data encapsulation
   (c) Inheritance and polymorphism
   (d) Dynamic binding and message passing
8. Describe inheritance as applied to OOP.
9. What do you mean by dynamic binding? How it is useful in OOP?
10. What is the use of preprocessor directive #include<iostream>?
11. How does a main () function in c++ differ from main () in c?
12. Describe the major parts of a c++ program.
13. Write a program to read two numbers from the keyboard and display the larger value on the screen.
14. Write a program to input an integer value from keyboard and display on screen "WELL DONE" that many times.

References:

1. Object –Oriented –Programming in C++ by E Balagurusamy.
2. Object –Oriented –Programming with ANSI & Turbo C++ by Ashok N. Kamthane.
3. OO Programming in C++ by Robert Lafore, Galgotia Publications Pvt. Ltd.
4. Mastering C++ By K R Venugopal, Rajkumar Buyya, T Ravishankar.
5. Object Oriented Programming and C++ By R. Rajaram.
6. Object –Oriented –Programming in C++ by Robert Lafore.

Subject: Object Oriented Programming using C++

Paper Code: MCA-302 **Author: Mr. Ganesh Kumar**

Lesson: Function in c++ &Object and classes **Vetter: Dr. Pradeep Bhatia**

Lesson No. : 2

STRUCTURE

4.1 INTRODUCTION

Functions are the building blocks of C++ programs where all the program activity occurs. Function is a collection of declarations and statements.

Need for a Function

Monolethic program (a large single list of instructions) becomes difficult to understand. For this reason functions are used. A function has a clearly defined objective (purpose) and a clearly defined interface with other functions in the program. Reduction in program size is another reason for using functions. The functions code is stored in only one place in memory, even though it may be executed as many times as a user needs.

The following program illustrates the use of a function :

```
//to display general message using function
#include<iostream.h>
include<conio.h>
void main()
   {
     void disp(); //function prototype
     clrscr(); //clears the screen
     disp(); //function call
     getch(); //freeze the monitor
}
```

```
//function definition

void disp()

{

cout<<”Welcome to the GJU of S&T\n”;

cout<<”Programming is nothing but logic implementation”;

}
```

PROGRAM 4.1

In this Unit, we will also discuss Class, as important Data Structure of C++. A Class is the backbone of Object-Oriented Computing. It is an abstract data type. We can declare and define data as well as functions in a class. An object is a replica of the class to the exception that it has its own name. A class is a data type and an object is a variable of that type. Classes and objects are the most important features of C++. The class implements OOP features and ties them together.

4.2 FUNCTION DEFINITION AND DECLARATION

In C++, a function must be defined prior to it’s use in the program. The function definition contains the code for the function. The function definition for display_message () in program 6.1 is given below the main () function. The general syntax of a function definition in C++ is shown below:

```
Type name_of_the_function (argument list)
     {
       //body of the function
     }
```

Here, the type specifies the type of the value to be returned by the function**.** It may be any valid C++ data type. When no type is given, then the compiler returns an integer value from the function.

Name_of_the_function is a valid C++ identifier (no reserved word allowed) defined by the user and it can be used by other functions for calling this function.

Argument list is a comma separated list of variables of a function through which the function may receive data or send data when called from other function. When no parameters, the argument list is empty as you have already seen in program 6.1. The following function illustrates the concept of function definition :

```
//function definition add()
void add()
{
   int a,b,sum;
   cout<<”Enter two integers”<<endl;
   cin>>a>>b;
   sum=a+b;
   cout<<”\nThe sum of two numbers is “<<sum<<endl;
}
```

The above function add ( ) can also be coded with the help of arguments of parameters as shown below:

```
//function definition add()
void add(int a, int b) //variable names are must in definition
{
  int sum;
   sum=a+b;
   cout<<”\nThe sum of two numbers is “<<sum<<endl;
   }
```

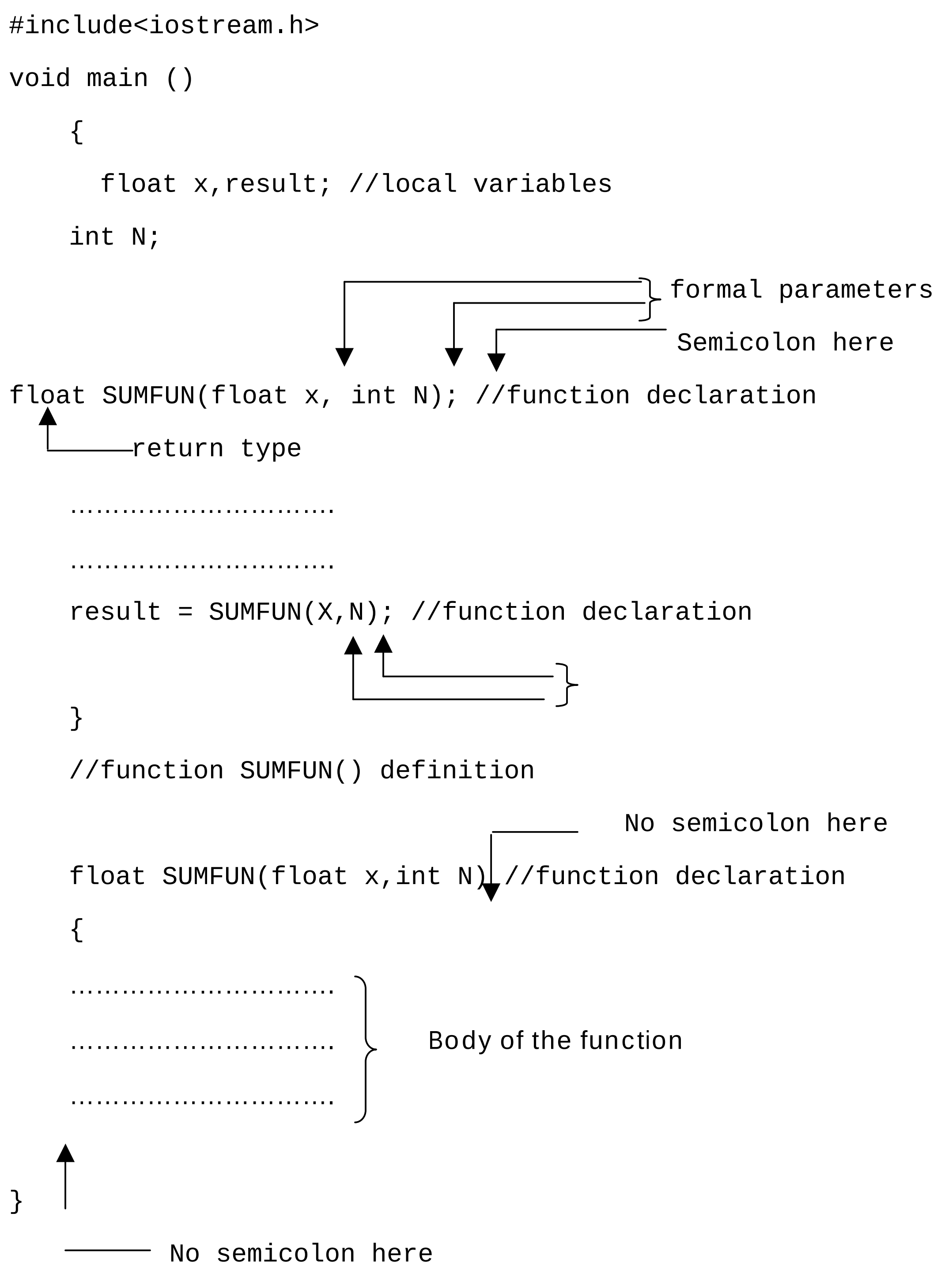
4.3 ARGUMENTS TO A FUNCTION

Arguments(s) of a function is (are) the data that the function receives when called/invoked from another function.

4.3.1 PASSING ARGUMENTS TO A FUNCTION

It is not always necessary for a function to have arguments or parameters. The functions **add ( )** and **divide ( )** in program 6.3 did not contain any arguments. The following example illustrates the concept of passing arguments to function **SUMFUN ( ):**

```
// demonstration of passing arguments to a function
```



```
#include<iostream.h>
void main ()
    {
      float x,result; //local variables
    int N;

float SUMFUN(float x, int N); //function declaration

    ..............................
    ..............................
    result = SUMFUN(X,N); //function declaration

    }
    //function SUMFUN() definition

    float SUMFUN(float x,int N) //function declaration
    {
    ..............................
    ..............................
    ..............................

}
```

4.3.2 DEFAULT ARGUMENTS

C++ allows a function to assign a parameter the default value in case no argument for that parameter is specified in the function call. For example.

```
// demonstrate default arguments function
```

```
#include<iostream.h>
int calc(int U)
{
If (U % 2 = = 0)

   return U+10;
Else
   return U+2
}
Void pattern (char M, int B=2)
{
for (int CNT=0;CNT<B; CNT++)
   cout<calc(CNT) <<M;
   cout<<endl;
}
Void main ()
{
Pattern(‘*’);
Pattern (‘#’,4)’
Pattern (;@;,3);
}
```

4.3.3 CONSTANT ARGUMENTS

A C++ function may have constant arguments(s). These arguments(s) is/are treated as constant(s). These values cannot be modified by the function.

For making the arguments(s) constant to a function, we should use the keyword **const** as given below in the function prototype :

```
Void max(const float x, const float y, const float z);
```

Here, the qualifier **const** informs the compiler that the arguments(s) having **const** should not be modified by the function max (). These are quite useful when call by reference method is used for passing arguments.

4.4 CALLING FUNCTIONS

In C++ programs, functions with arguments can be invoked by :

(a) Value
(b) *Reference*

Call by Value: - In this method the values of the actual parameters (appearing in the function call) are copied into the formal parameters (appearing in the function definition), i.e., the function creates its own copy of argument values and operates on them. The following program illustrates this concept :

```
    //calculation of compound interest using a function

    #include<iostream.h>

    #include<conio.h>

    #include<math.h> //for pow()function

    Void main()

    {

     Float principal, rate, time; //local variables

     Void calculate (float, float, float); //function prototype
clrscr();

     Cout<<"\nEnter the following values:\n";

     Cout<<"\nPrincipal:";

     Cin>>principal;

     Cout<<"\nRate of interest:";

     Cin>>rate;

     Cout<<"\nTime period (in yeaers) :";

     Cin>>time;

     Calculate (principal, rate, time); //function call
```

```
 Getch ();
 }
//function definition calculate()
Void calculate (float p, float r, float t)
{
  Float interest; //local variable
  Interest = p* (pow((1+r/100.0),t))-p;
  Cout<<”\nCompound interest is : “<<interest;
  }
```

Call by Reference: - A reference provides an alias – an alternate name – for the variable, i.e., the same variable’s value can be used by two different names : the original name and the alias name.

In call by reference method, a reference to the actual arguments(s) in the calling program is passed (only variables). So the called function does not create its own copy of original value(s) but works with the original value(s) with different name. Any change in the original data in the called function gets reflected back to the calling function.

It is useful when you want to change the original variables in the calling function by the called function.

```
//Swapping of two numbers using function call by reference
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int num1,num2;
    void swap (int &, int &); //function prototype
    cin>>num1>>num2;
    cout<<”\nBefore swapping:\nNum1: “<<num1;
    cout<<endl<<”num2: “<<num2;
```

```
        swap(num1,num2); //function call
        cout<<"\n\nAfter swapping : \Num1: "<<num1;
        cout<<endl<<"num2: "<<num2;
        getch();
    }
    //function fefinition swap()
    void swap (int & a, int & b)
    {
        Int temp=a;
        a=b;
        b=temp;
    }
```

4.5 INLINE FUNCTIONS

These are the functions designed to speed up program execution. An inline function is expanded (i.e. the function code is replaced when a call to the inline function is made) in the line where it is invoked. You are familiar with the fact that in case of normal functions, the compiler have to jump to another location for the execution of the function and then the control is returned back to the instruction immediately after the function call statement. So execution time taken is more in case of normal functions. There is a memory penalty in the case of an inline function.

The system of inline function is as follows :

```
inline function_header
        {
        _______________________
         body of the function
        _______________________
        }
For example,
//function definition min()
    inline void min (int x, int y)
        cout<< (x < Y? x : y);
    }
```

```
Void main()
{
  int num1, num2;
  cout<<”\Enter the two intergers\n”;
  cin>>num1>>num2;
  min (num1,num2; //function code inserted here
  ------------------
  ------------------
}
```

An inline function definition must be defined before being invoked as shown in the above example. Here min ( ) being inline will not be called during execution, but its code would be inserted into main ( ) as shown and then it would be compiled.

If the size of the inline function is large then heavy memory pentaly makes it not so useful and in that case normal function use is more useful.

The inlining does not work for the following situations :

1. For functions returning values and having a *loop* or a *switch* or a goto statement.
2. For functions that do not return value and having a return statement.
3. For functions having static variable(s).
4. If the inline functions are recursive (i.e. a function defined in terms of itself).

The benefits of inline functions are as follows :

1. Better than a macro.
2. Function call overheads are eliminated.
3. Program becomes more readable.
4. Program executes more efficiently.

4.6 SCOPE RULES OF FUNCTIONS AND VARIABLES

The scope of an identifier is that part of the C++ program in which it is accessible. Generally, users understand that the name of an identifier must be unique. It does not mean that a name can't be reused. We can reuse the name in a program provided that there is some scope by which it can be distinguished between different cases or instances.

In C++ there are four kinds of scope as given below :

1. Local Scope
2. Function Scope
3. File Scope
4. Class Scope

Local Scope:- A block in C++ is enclosed by a pair of curly braces i.e., '{' and '}'. The variables declared within the body of the block are called **local variables** and can be used only within the block. These come into existence when the control enters the block and get destroyed when the control leaves the closing brace. You should note the variable(s) is/are available to all the enclosed blocks within a block.

For example,

```
int x=100;
        { cout<<x<<endl;
            Int x=200;
            {
      cout<<x<<endl;
      int x=300;
      {
      cout<<x<<endl;
      }
      }
      cout<<x<<endl;
}
```

Function Scope : It pertains to the labels declared in a function i.e., a label can be used inside the function in which it is declared. So we can use the same name labels in different functions.

For example,

```
//function definition add1()

      void add1(int x,int y,int z)

      {

            int sum = 0;

            sum = x+y+z;

            cout<<sum;

      }

      //function definition add2()

      coid add2(float x,float y,float z)

      {

      Float sum = 0.0;

      sum = x+y+z;

      cout<<sum;

      }
```

Here the labels x, y, z and sum in two different functions add1 ( ) and add2 ( ) are declared and used locally.

File Scope : If the declaration of an identifier appears outside all functions, it is available to all the functions in the program and its scope becomes file scope. For Example,

```
int x;

      void square (int n)

        {

            cout<<n*n;

        }

      void main ()

         {

            int num;
```

```
                …………...........
                cout<<x<<endl;
                cin>>num;
                squaer(num);
                …………...........
           }
```

Here the declarations of variable **x** and function **square** ( ) are outside all the functions so these can be accessed from any place inside the program. Such variables/functions are called global.

Class Scope : In C++, every class maintains its won associated scope. The class members are said to have local scope within the class. If the name of a variable is reused by a class member, which already has a file scope, then the variable will be hidden inside the class. Member functions also have class scope.

4.7 DEFINITION AND DECLARATION OF A CLASS

A class in C++ combines related data and functions together. It makes a data type which is used for creating objects of this type.

Classes represent real world entities that have both data type properties (characteristics) and associated operations (behavior).

The syntax of a class definition is shown below :

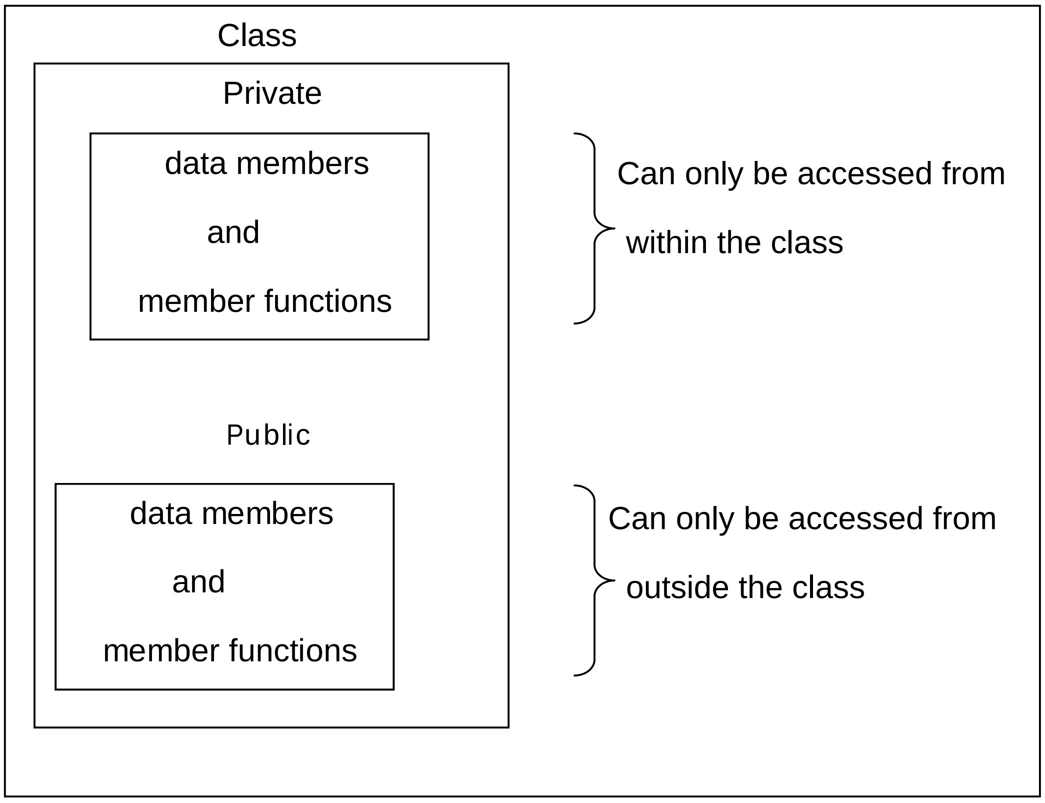
```
Class name_of _class
   {
private   : variable declaration; // data member
           Function declaration; // Member Function (Method)
protected: Variable declaration;
            Function declaration;
public    : variable declaration;
            Function declaration;
};
```

Here, the keyword class specifies that we are using a new data type and is followed by the class name.

The body of the class has two keywords namely :

(i) *private* *(ii) public*

In C++, the keywords **private** and **public** are called access specifiers. The data hiding concept in C++ is achieved by using the keyword **private.** Private data and functions can only be accessed from within the class itself. Public data and functions are accessible outside the class also. This is shown below :


Class
Private
data members
and
member functions
Can only be accessed from within the class
Public
data members
and
member functions
Can only be accessed from outside the class

Data hiding not mean the security technique used for protecting computer databases. The security measure is used to protect unauthorized users from performing any operation (read/write or modify) on the data.

The data declared under **Private** section are hidden and safe from accidental manipulation. Though the user can use the private data but not by accident.

The functions that operate on the data are generally **public** so that they can be accessed from outside the class but this is not a rule that we must follow.

4.8 MEMBER FUNCTION DEFINITION

The class specification can be done in two part :

(i) **Class definition.** It describes both data members and member functions.

(ii) **Class method definitions.** It describes how certain class member functions are coded.

We have already seen the class definition syntax as well as an example.

In C++, the member functions can be coded in two ways :

(a) *Inside class definition*

(b) *Outside class definition using scope resolution operator (::)*

The code of the function is same in both the cases, but the function header is different as explained below :

4.8.1 Inside Class Definition:

When a member function is defined inside a class, we do not require to place a membership label along with the function name. We use only small functions inside the class definition and such functions are known as **inline** functions.

In case of inline function the compiler inserts the code of the body of the function at the place where it is invoked (called) and in doing so the program execution is faster but memory penalty is there.

4.8.2 Outside Class Definition Using Scope Resolution Operator (::) :

In this case the function's full name (qualified_name) is written as shown:

```
Name_of_the_class :: function_name
```

The syntax for a member function definition outside the class definition is :

```
return_type name_of_the_class::function_name (argument list)
  {
    body of function
  }
```

Here the operator::known as scope resolution operator helps in defining the member function outside the class. Earlier the scope resolution operator(::)was ised om situations where a global variable exists with the same name as a local variable and it identifies the global variable.

4.9 DECLARATION OF OBJECTS AS INSTANCES OF A CLASS

The objects of a class are declared after the class definition. One must remember that a class definition does not define any objects of its type, but it defines the properties of a class. For utilizing the defined class, we need variables of the class type. For example,

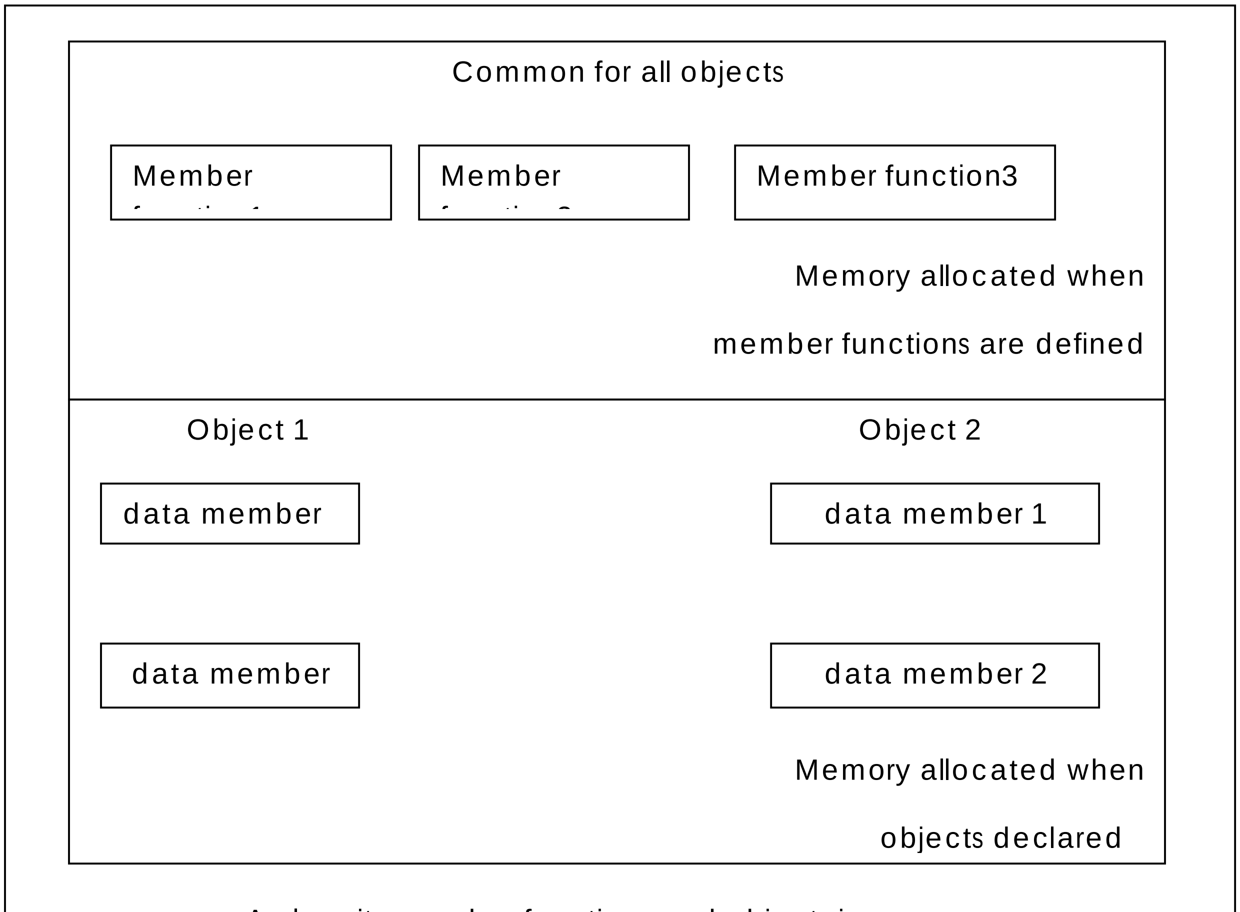
```
Largest ob1,ob2; //object declaration
```

will create two objects **ob1** and **ob2 of largest** class type. As mentioned earlier, in C++ the variables of a class are known as objects. These are declared like a simple variable i.e., like fundamental data types.

In C++, all the member functions of a class are created and stored when the class is defined and this memory space can be accessed by all the objects related to that class.

Memory space is allocated separately to each object for their data members. Member variables store different values for different objects of a class.

The figure shows this concept



A class, its member functions and objects in memory.

4.10 ACCESSING MEMBERS FROM OBJECT(S)

After defining a class and creating a class variable i.e., object we can access the data members and member functions of the class. Because the data members and member functions are parts of the class, we must access these using the variables we created. For functions are parts of the class, we must access these using the variable we created. For Example,

```
Class student
 {
   private:
      char reg_no[10];
`     char name[30];
        int age;
      char address[25];
public :
      void init_data()
      {
      - - - - - //body of function
      - - - - -
      }
      void display_data()
      }
};
student ob; //class variable (object) created
- - - - -
- - - - -
Ob.init_data(); //Access the member function
ob.display_data(); //Access the member function
-  - - - -
- - - - -
```

Here, **the data members can be accessed in the member functions** as these have **private** scope, and the member functions can be accessed outside the class i.e., before or after the main() function.

4.11 STATIC CLASS MEMBERS

Data members and member functions of a class in C++, may be qualified as static. We can have static data members and static member function in a class.

4.11.1 **Static Data Member:** It is generally used to store value common to the whole class. The **static** data member differs from an ordinary data member in the following ways :

(i) Only a single copy of the static data member is used by all the objects.

(ii) It can be used within the class but its lifetime is the whole program.

For making a data member static, we require :

(a) Declare it within the class.

(b) Define it outside the class.

For example

```
Class student
  {
    Static int count; //declaration within class
    -----------------
    -----------------
    -----------------
};
```

The static data member is defined outside the class as :

```
int student :: count; //definition outside class
```

The definition outside the class is a must.

We can also initialize the static data member at the time of its definition as:

```
int student :: count = 0;
```

If we define three objects as : sudent obj1, obj2, obj3;

4.11.2 **Static Member Function:** A static member function can access only the static members of a class. We can do so by putting the keyword static before the name of the function while declaring it for example,

```
Class student
           {
             Static int count;
             -----------------
             public :
                 -----------------
                 -----------------
           static void showcount (void) //static member function
                 {
                   Cout<<”count=”<<count<<”\n”;
```

```
    }
};
int student ::count=0;
```

Here we have put the keyword static before the name of the function shwocount ().

In C++, a static member function fifers from the other member functions in the following ways:

(i) Only static members (functions or variables) of the same class can be accessed by a static member function.
(ii) It is called by using the **name of the class** rather than an object as given below:

```
                Name_of_the_class :: function_name
    For example,
                student::showcount();
```

4.12 FRIEND CLASSES

In C++ , a class can be made a friend to another class. For example,

```
class TWO; // forward declaration of the class TWO
class ONE
  {
  ..........................
  ................
public:
  ................
  ................
friend class TWO; // class TWO declared as friend of class ONE
};
```

Now from class TWO , all the member of class ONE can be accessed.

4.13 Summary

In this Unit, we have discussed the concept of function in c++, its declaration and definition. we have also discussed the concept of class, its declaration and definition. It also explained the ways for creating objects, accessing the data members of the class. We have seen the way to pass objects as arguments to the functions with call by value and call by reference.

4.14 Keywords

Inline Functions:- An inline function is expanded in the line where it is invoked.

Member Function:- Private means that they can be accessed only by the functions within the class.

Classes:- When you create the definition of a class you are defining the attributes and behavior of a new type.

Objects:- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).

4.15 Review Questions

Q. 1. what is a function ? How will you define a function in C++ ?

Q. 2. How are the argument data types specified for a C++ function? Explain with Suitable example.

Q. 3. What types of functions are available in C++ ? Explain.

Q. 4. What is recursion? While writing any recursive function what thing(s) must be taken care of ?

Q. 5. What is inline function? When will you make a function inline and why ?

Q.6. What is a class? How objects of a class are created ?

Q. 7. What is the significance of scope resolution operator (::) ?

Q. 8. Define data members , member function, private and public members with example.

Q. 9. Define a class student with the following specifications:

Adm_no	integer
Sname	20 characters
Eng, math, science	float (marks in three subjects)
Total	float
Ctotal()	a function to calculate eng + math + science marks

Public member functions of class student

Takedata() function to accept values for adm_no , sname, marks in eng, math, science and invoke ctotal() to calculate total.

Showdata() function to display all the data members on the screen.

Q.10. Define a string data type with the following functionality:

- A constructor having no parameters,
- Constructors which initialize strings as follows:
  - A constructor that creates a string of specific size
  - Constructor that initializes using a pointer string
  - A copy constructor
- Define the destructor for the class
- It has overloaded operators. (This part of question will be taken up in the later units).
- There is operation for finding length of the string.

4.16 Further Readings

1. Rambagh J. , " Object Oriented Modeling and Design" , Prentice Hall of India , New Delhi.
2. E. Balagrusamy, "Object Oriented Programming with C++", Tata McGraw Hill.

Subject: Object Oriented Programming using C++

Paper Code: MCA-302 **Author: Mr. Ganesh Kumar**

Lesson: Constructors and Destructors,Operator Overloading and Type Conversions **Vetter: Dr. Pradeep Bhatia**

Lesson No. : 3

STRUCTURE

5.1 INTRODUCTION

A **constructor** (having the same name as that of the class) is a member function which is automatically used to initialize the objects of the class type with legal initial values. Destructors are the functions that are complimentary to constructors. These are used to de-initialize objects when they are destroyed. A destructor is called when an object of the class goes out of scope, or when the memory space used by it is de allocated with the help of **delete** operator.

Operator overloading is one of the most exciting features of C++. It is helpful in enhancement of the power of extensibility of C++ language. Operator overloading redefines the C++ language. User defined data types are made to behave like built-in data types in C++. Operators +, *. <=, += etc. can be given additional meanings when applied on user defined data types using operator overloading. The mechanism of providing such an additional meaning to an operator is known as operator overloading in C++.

5.2 Declaration and Definition of a Constructor:-

It is defined like other member functions of the class, i.e., either inside the class definition or outside the class definition.

For example, the following program illustrates the concept of a constructor :

```
//To demonstrate a constructor
#include <iostram.h>
#include <conio.h>
Class rectangle
{
   private :
        float length, breadth;
   public:
    rectangle ()//constructor definition
        {
        //displayed whenever an object is created
        cout<<"I am in the constructor";
        length-10.0;
```

```
            breadth=20.5;
        }
        float area()
        {
          return (length*breadth);
        }
    };
    void main()
    {
    clrscr();
    rectangle rect; //object declared
    cout<<"\nThe area of the rectangle with default parameters
    is:"<<rect.area()<<"sq.units\n";
    getch();
    }
```

5.3 Type Of Constructor

There are different type of constructors in C++.

5.3.1 Overloaded Constructors

Besides performing the role of member data initialization, constructors are no different from other functions. This included overloading also. In fact, it is very common to find overloaded constructors. For example, consider the following program with overloaded constructors for the figure class :

```
//Illustration of overloaded constructors
//construct a class for storage of dimensions of circles.
//triangle and rectangle and calculate their area
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<string.h> //for strcpy()
```

```
        Class figure
        {
        Private:
           Float radius, side1,side2,side3; //data members
           Char shape[10];
        Public:
           figure(float r) //constructor for circle
                {
        radius=r;
        strcpy (shape, “circle”);
        }
        figure (float s1,float s2) //constructor for rectangle
        strcpy
        {
                Side1=s1;
                Side2=s2;
                Side3=radius=0.0; //has no significance in rectangle
strcpy(shape,”rectangle”);
        }
        Figure (float s1, floats2, float s3) //constructor for triangle
        {
           side1=s1;
           side2=s2;
           side3=s3;
           radius=0.0;
           strcpy(shape,”triangle”);
           }
        void area() //calculate area
        {
```

```
        float ar,s;
        if(radius==0.0)
           {
                 if (side3==0.0)
                 ar=side1*side2;
           else
                 ar=3.14*radius*radius;
        cout<<”\n\nArea of the “<<shape<<”is :”<<ar<<”sq.units\n”;
        }
        };
        Void main()
        {
        Clrscr();
        Figure circle(10.0); //objrct initialized using constructor
Figure rectangle(15.0,20.6);//objrct initialized using onstructor
Figure Triangle(3.0, 4.0, 5.0); //objrct initialized using constructor
        Rectangle.area();
        Triangle.area();
        Getch();//freeze the monitror
        }
```

5.3.2 Copy Constructor

It is of the form classname (classname &) and used for the initialization of an object form another object of same type. For example,

```
        Class fun
         {
           Float x,y;
        Public:
        Fun (floata,float b)//constructor
        {
           x = a;
           y = b;
        }
Fun (fun &f) //copy constructor
        {cout<<"\ncopy constructor at work\n";
        X = f.x;
        Y = f.y;
        }
        Void display (void)
        {
        {
        Cout<<""<<y<<end1;
        }
};
```

Here we have two constructors, one copy constructor for copying data value of a fun object to another and other one a parameterized constructor for assignment of initial values given.

5.3.3 Dynamic Initialization of Objects

In C++, the class objects can be initialized at run time (dynamically). We have the flexibility of providing initial values at execution time. The following program illustrates this concept:

```
        //Illustration of dynamic initialization of objects
        #include <iostream.h>
        #include <conio.h>
        Class employee
        {
        Int empl_no;
        Float salary;
```

```
Public:

Employee() //default constructor

{}

Employee(int empno,float s)//constructor with arguments

{

Empl_no=empno;

Salary=s;

}

Employee (employee &emp)//copy constructor

{

Cout<<”\ncopy constructor working\n”;

Empl_no=emp.empl_no;

Salary=emp.salary;

}

Void display (void)

{

Cout<<”\nEmp.No:”<<empl_no<<”salary:”<<salary<<end1;

}

};

Void main()

{

int eno;

float sal;

clrscr();

cout<<”Enter the employee number and salary\n”;

cin>>eno>>sal;

employee obj1(eno,sal);//dynamic initialization of object

cout<<”\nEnter the employee number and salary\n”;

cin>eno>>sal;
```