UNIT II

Control Structures: if, if..else, elseif ladder, nested if, switch, for, while, do..while, break, continue, exit, goto.

Classes and Objects: Specifying a class, defining member functions, C++ program with class, private member functions, arrays within class, memory allocation for objects, static data members, static member functions, arrays of objects, returning objects.

Functions in C++: Main function, function prototyping, call by reference, return by reference, inline functions, default arguments.

2.1 Control Structures

2.1.1 If, if..else elseif ladder, nested if

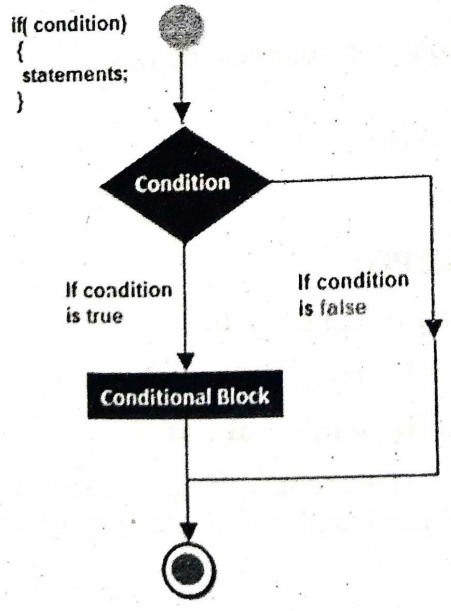
Q1. Write about various control statements used in C++. Explain each with an example. **(Imp.)**

Ans :

If statement

The if statement checks whether the test condition is true or not. If the test condition is true, it executes the code/s inside the body of if statement. But it the test condition is false, it skips the code/s inside the body of if statement.

The if keyword is followed by test condition inside parenthesis ( ). If the test condition is true, the codes inside curly bracket is executed but if test condition is false, the codes inside curly bracket { } is skipped and control of program goes just below the body of if as shown in figure above.



Syntax: If(condition)
{
Statements
.....
}

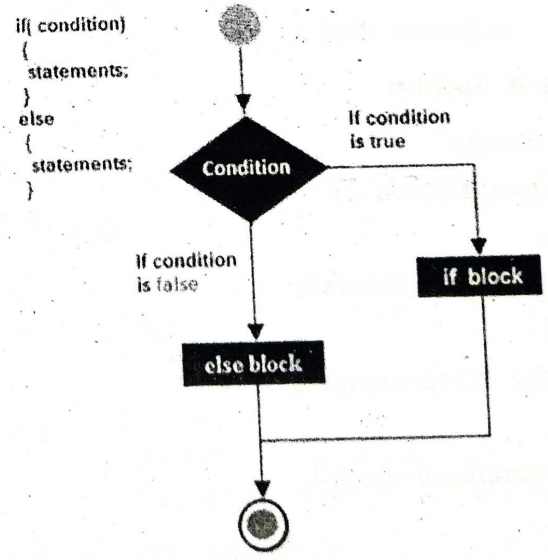
Example: program to find the maximum of two numbers

```
#include< iostream.h>
int main( )
{
 int x,y;
 x=15;
 y=13;
 if (x > y )
 {
  cout << "x is greater than y";
 }
}
```

Output :
x is greater than y

.If – else statement:

In general it can be used to execute one block of statement among two blocks:



Syntax:

```
if( expression )
{
 statement-block1;
}
else
{
 statement-block2;
}
```

If the 'expression' is true, the 'statement-block1' is executed, else 'statement-block1' is skipped and 'statement-block2' is executed.

Example : Program to find greater than of two numbers using if – else

```
void main( )
{
 int x,y;
 x=15;
 y=18;
 if (x > y )
 {
  cout << "x is greater than y";
 }
 else
 {
  cout << "y is greater than x";
 }
}
```

Output :

y is greater than x

. else-if ladder

Syntax:

```
if(expression 1)
{
 statement-block1;
}
else if(expression 2)
{
 statement-block2;
}
else if(expression 3 )
{
 statement-block3;
}
else
 default-statement;
```

The expression is tested from the top(of the ladder) downwards. As soon as the true condition is found, the statement associated with it is executed.

Example : Program to find whether the given number is divisible by 5 or 8

```
void main( )
{
 int a;
 cout << "enter a number";
 cin >> a;
 if( a%5==0 && a%8==0)
 {
  cout << "divisible by both 5 and 8";
 }
 else if( a%8==0 )
 {
  cout << "divisible by 8";
 }
 else if(a%5==0)
 {
  cout << "divisible by 5";
 }
 else
 {
  cout << "divisible by none";
 }
getch();
}
```

OUTPUT

Enter a number : 40

Divisible by both 5 and 8

Points to Remember about if

1. In if statement, a single statement can be included without enclosing it into curly braces { }

2. int a = 5;
3. if(a > 4)
4. cout << "success";

No curly braces are required in the above case, but if we have more than one statement inside if condition, then we must enclose them inside curly braces.

5. == must be used for comparison in the expression of if condition, if you use = the expression will always return true, because it performs assignment not comparison.

6. Other than **0(zero)**, all other values are considered as true.

Nested if...else

Nested if...else are used if there are more than one test expression.

Syntax:

```
if( expression )
{
 if( expression1 )
  {
    statement-block1;
  }
 else
  {
    statement-block2;
  }
}
else
{
 statement-block3;
}
```

if 'expression' is false the 'statement-block3' will be executed, otherwise it continues to perform the test for 'expression 1' . If the 'expression 1' is true the 'statement-block1' is executed otherwise 'statement-block2' is executed.

Example : program to find the greatest of three numbers

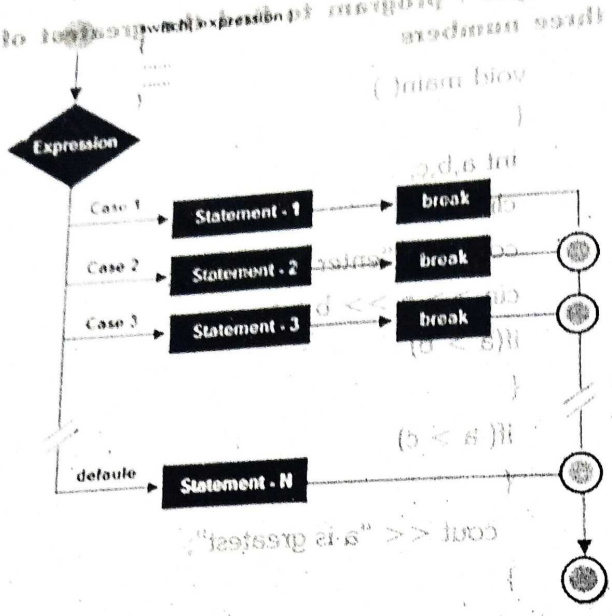
```
void main( )
{
 int a,b,c;
 clrscr();
 cout << "enter 3 number";
 cin >> a >> b >> c;
 if(a > b)
 {
  if( a > c)
  {
    cout << "a is greatest";
  }
  else
  {
    cout << "c is greatest";
  }      }
 else
 {
  if( b> c)
   {
     cout << "b is greatest";
   }
  else
   {
     printf("c is greatest");
   }      }
 getch();
}
```

2.1.2 Switch

Q2. Explain Switch statement in C++ with an example.

Ans : **(Imp.)**

A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.



Rules for apply switch

1. With switch statement use only byte, short, int, char data type.
2. You can use any number of case statements within a switch.
3. Value for a case must be same as the variable in switch .

Limitations of switch

Logical operators cannot be used with switch statement. For instance

Example

case k>=20: //is not allowed

Switch case variables can have only int and char data type. So float da a type is not allowed.

Syntax : switch(ch)

```
{
case1:
statement 1;
break;
case2:
statement 2;
break;
}
```

In this ch can be integer or char and cannot be float or any other data type.

Example of Switch case

```
#include<iostream.h>
#include<conio.h>
void main()
{
int ch;
clrscr();
cout<<"Enter any number (1 to 7)";
cin>>ch;
switch(ch)
{
case 1:
cout<<"Today is Monday";
break;
case 2:
cout<<"Today is Tuesday";
break;
case 3:
cout<<"Today is Wednesday";
break;
case 4:
cout<<"Today is Thursday";
break;
case 5:
cout<<"Today is Friday";
break;
case 6:
cout<<"Today is Saturday";
break;
case 7:
cout<<"Today is Sunday";
break;
default:
cout<<"Only enter value 1 to 7";
}
getch();
}
```

OUTPUT

Enter any number (1 to 7): 5

Today is Friday

***Note:** In switch statement default is optional but when we use this in switch default is executed last whenever all cases are not satisfied the condition.

2.1.3 for, while, do..while

Q3. Explain about various looping statements in C++ with an examples.

Ans : **(Imp.)**

In any programming language, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

A sequence of statement is executed until a specified condition is true. This sequence of statement to be executed is kept inside the curly braces { } known as loop body. After every execution of loop body, condition is checked, and if it is found to be true the loop body is executed again. When condition check comes out to be false, the loop body will not be executed.

There are 3 type of loops in C++ language

1. while loop
2. for loop
3. do-while loop

1. while loop

while loop can be address as an entry control loop. It is completed in 3 steps. In while loop First check the condition if condition is true then control goes inside the loop body other wise goes outside the body. while loop will be repeats in clock wise direction.

- Variable initialization (e.g int x=0;)
- condition(e.g while(x<=10))
- Variable increment or decrement (x++ or x— or x=x+2)

Syntax :

```
variable initialization ;
while (condition)
{
 statements ;
 variable increment or decrement ;
}
```

***Note:** If while loop condition never false then loop become infinite loop.

Example: print the natural numbers below 5

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i;
clrscr();
i=1;
while(i<5)
{
cout<<endl<<i;
i++;
}
getch();
}
```

OUTPUT

1
2
3
4

2. for loop

for loop is used to execute a set of statement repeatedly until a particular condition is satisfied.

For loop contains 3 parts.

- Initialization
- Condition
- Iteration

Syntax: for(initialization; condition ; increment/decrement)

```
{
 statement-block;
}
```

- When we are working with for loop always execution process will start from initialization block.]

- After initialization block control will pass to condition block, if condition is evaluated as true then control will pass to statement block.
- After execution of the statement block control will pass to iteration block, from iteration it will pass back to the condition.
- Always repetitions will happen beginning condition, statement block and iteration only.
- Initialization block will be executed only once when we are entering into the loop first time.
- When we are working with for loop everything is optional but mandatory to place 2 semicolons (; ;)

Example

```
while()          // Error
for( ; ; )       // valid
```

Example : print the natural numbers below 5 using for loop

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i;
clrscr();
for(i=1;i<5;i++)
{
cout<<endl<<i;
}
getch();
}
```

OUTPUT

```
1
2
3
4
```

3. do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of do-while loop. do statement evaluates the body of the loop first and at the end, the condition is checked using while statement. General format of do-while loop is,

Syntax:

```
do
{
 ....
 .....
}
while(condition);
```

When use Do-While Loop

When we need to repeat the statement block atleast 1 time then we use do-while loop.

Example: program to print the natural numbers below 5

```
#include<iostream.h>
#include<conio.h>

void main()
{
int i;
clrscr();
i=1;
do
{
cout<<endl<<i;
i++;
}
while(i<5);
getch();
}
```

OUTPUT

```
1
2
3
4
```


Nested Loop

In Nested loop another loop body.

When we need n number of times use can be design upto 2

Nested for loop

We can also ha loop inside another f

```
for(initialization; con
{
 for(initialization; co
   {
     statement ;
   }
}
```

nested while loop

The syntax of

```
while(condition)
{
  while(condition)
  {
    statement(s);
  }
  statement(s); // y
}
```

nested do...while

The syntax

```
do
{
  statement(s); // y
  do
  {
    statement(s);
  }while( conditio

}while( condition
```

Example

The followin to find the prime

Nested Loop

In Nested loop one loop is place within another loop body.

When we need to repeated loop body itself n number of times use nested loops. Nested loops can be design upto 255 blocks.

Nested for loop

We can also have nested for loop, i.e one for loop inside another for loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
 for(initialization; condition; increment/decrement)
   {
     statement ;
   }
}
```

nested while loop

The syntax of nested while loop is as follows

```
while(condition)
{
  while(condition)
  {
    statement(s);
  }
  statement(s); // you can put more statements.
}
```

nested do...while loop

The syntax of nested do loop is as follows

```
do
{
  statement(s); // you can put more statements.
  do
  {
    statement(s);
  }while( condition );

}while( condition );
```

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100:

```
#include <iostream>
using namespace std;
 int main ()
{
  int i, j;

  for(i=2; i<50; i++) {
    for(j=2; j <= (i/j); j++)
      if(!(i%j)) break; // if factor found, not prime
      if(j > (i/j)) cout << i << " is prime\n";
  }
  return 0;
}
```

This would produce the following result:

OUTPUT

2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime

2.1.4 break, continue

Q4. Explain the use of break and continue statements with an example.

Ans :

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes

true, that is jump out of loop. C language allows jumping from one statement to another within a loop as well as jumping out of the loop.

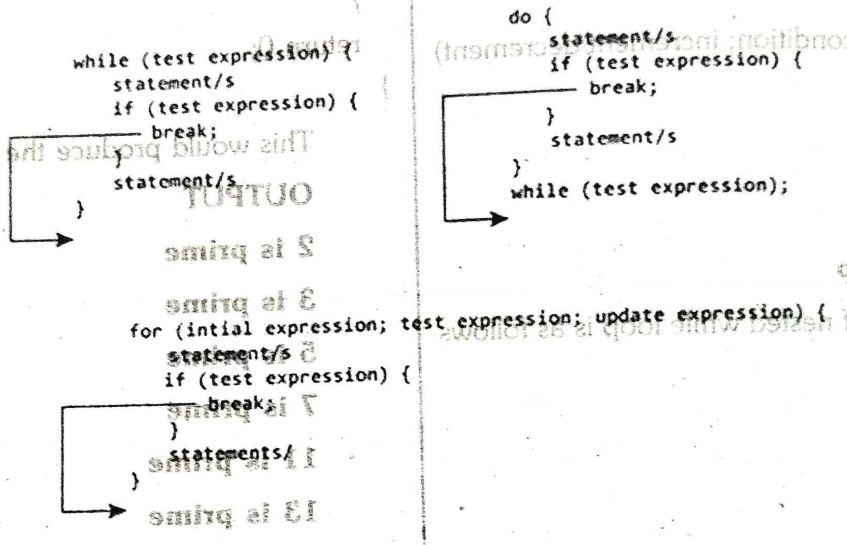
break statement

When break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

Syntax of break : **break;**

In real practice, break statement is almost always used inside the body of conditional statement(if...else) inside the loop.

Working of break statement



Note: The break statement may also be used inside body of else statement

***NOTE : In C programming, break statement is also used with switch...case statement.

Continue Statement

It causes the control to go directly to the test-condition and then continue the loop process. On encountering continue, cursor leave the current cycle of loop, and starts with the next cycle.

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement : **continue;**

How Continue Statement Works?

[* File contains invalid data | In-line.JPG *]

2.1.5 EXIT, GOTO

Q5. What is the use of GOTO statement in C++? Explain with an example.

Ans :

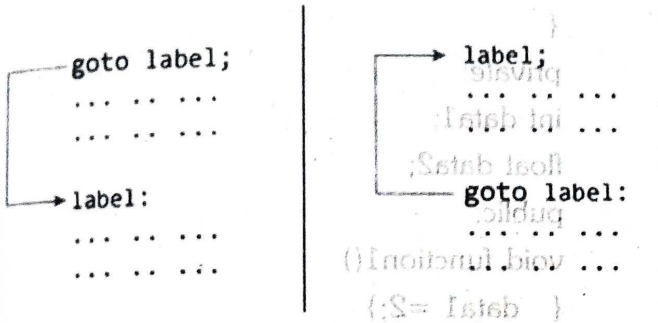
Go to Statement

In C++ programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax of goto Statement : **goto label;**

```
... .. ...
label:
statement;
... .. ...
```

In syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.



Example 1: goto Statement

```
/* C++ program to demonstrate the working of goto statement. */
/* This program calculates the average of numbers entered by user. */
/* If user enters negative number, it ignores that number and
   calculates the average of number entered before it.*/
# include <iostream>
using namespace std;
int main() {
  float num, average, sum = 0.0;
  int i, n;
  cout<<"Maximum number of inputs: ";
  cin>>n;
  for(i=1; i <= n; ++i) {
    cout<<"Enter n"<<i<<": ";
    cin>>num;
    if(num < 0.0) {
      goto jump; /* Control of the program moves to jump; */
    }
    sum += num;
  }
jump:
  average=sum/(i-1);
  cout<<"\nAverage = "<<average;
  return 0;
}
```

Output

```
Maximum number of inputs: 10
Enter n1: 2.3
Enter n2: 5.6
Enter n3: -5.6
Average = 3.95
```

Reason to Avoid goto Statement

The goto statement gives power to jump to any part of program but, makes the logic of the program complex and tangled. In modern programming, goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C++ program with the use of break and continue statements.

2.2 CLASSES AND OBJECTS

2.2.1 Specifying a Class

Q6. What is class ? How to define a class in C++? Explain with an example?

(OR)

Explain about the declaration of class in C++

Ans : **(Imp.)**

A class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from the same description, we can create many objects from a class.

defining a class in C++

A class is defined in C++ using keyword class followed by the name of class.

The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

```
class className
 {
 // some data
 // some functions
 };
```

Example: Class in C++

```
classTest
{
private:
int data1;
float data2;

public:
void function1()
{  data1 =2;}
float function2()
{
        data2 =3.5;
return data2;
}
};
```

Here, we defined a class named Test.

This class has two data members: data1 and data2 and two member functions: function1() and function2().

Q7. What is an object in C++? How to define an Object in C++?

(OR)

Explain how to define an instance of a class.

Ans :

Class is mere a blueprint or a template. No storage is assigned when we define a class. Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.

Each object has different data variables. Objects are initialised using special class functions called Constructors.

Syntax to Define Object in C++

```
className objectVariableName;
```

You can create objects of Test class (defined in above example) as follows:

```
classTest
{
private:
int data1;
float data2;
public:
void function1()
{  data1 =2;}
float function2()
{
        data2 =3.5;
return data2;
}
};
int main()
{
Test o1, o2;
}
```

Here, two objects o1 and o2 of Test class are created.

In the above class Test, data1 and data2 are data members and function1() and function2() are member functions.

2.2.2 Defining Member Functions

Q8. What are Member Functions? Explain about them how to use in class. **(Imp.)**

Ans :

Member functions are the functions, which have their declaration inside the class definition and works on the data members of the class. The definition of member functions can be inside or outside the definition of class.

If the member function is defined inside the class definition it can be defined directly, but if its defined outside the class, then we have to use the scope resolution : : operator along with class name along with function name.

// defining member function

```
class Cube
{
 public:
 int side;
 int getVolume();    // Declaring function getVolume with no argument and return type int.
};
```

If we define the function inside class then we don't not need to declare it first, we can directly define the function.

// another way of defining member function

```
class Cube
{
 public:
 int side;
 int getVolume()
 {
  return side*side*side;   //returns volume of cube
 }   };
```

But if we plan to define the member function outside the class definition then we must declare the function inside class definition and then define it outside.

// declaring and using member function

```
class Cube
{
 public:
 int side;
 int getVolume();
}
int Cube :: getVolume()      // defined outside class definition
{
 return side*side*side;
}
```

The main() function for both the function definition will be same. Inside main() we will create object of class, and will call the member function using dot . operator.

// calling member function inside main using dot operator

```
int main()
{
 Cube C1;
 C1.side=4;  // setting side value
  cout<< "Volume of cube C1 =" << C1.getVolume();
}
```

Similarly we can define the getter and setter functions to access private data members, inside or outside the class definition.

Q9. Explain, How to access data member and member function in C++?

Ans :

You can access the data members and member functions by using a . (dot) operator. For example,

```
o2.function1();
```

This will call the function1() function inside the Test class for objects o2.

Similarly, the data member can be accessed as:

```
o1.data2 = 5.5;
```

It is important to note that, the private members can be accessed only from inside the class.

So, you can use o2.function1(); from any function or class in the above example. However, the code o1.data2 = 5.5; should always be inside the class Test

Q10. Write about various access controls/ modifiers in C++.

Ans :

Access Control in Classes

Now before studying how to define class and its objects, lets first quickly learn what are access specifiers.

Access specifiers in C++ class defines the access control rules. C++ has 3 new keywords introduced, namely,

1. public
2. private
3. protected

These access specifiers are used to set boundaries for availability of members of class be it data members or member functions

Access specifiers in the program, are followed by a colon. You can use either one, two or all 3 specifiers in the same class to set different boundaries for different class members. They change the boundary for all the declarations that follow them.

Public

Public, means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. Hence there are chances that they might change them. So the key members must not be declared public.

```
class PublicAccess
{
public:          // public access specifier
int x;           // Data Member Declaration
void display();  // Member Function decaration
}
```

Private

Private keyword, means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile time error. By default class variables and member functions are private.

```
class PrivateAccess
{
private:         // private access specifier
int x;           // Data Member Declaration
void display();  // Member Function decaration
}
```

Protected

Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A. We will learn this later.)

```
class ProtectedAccess
{
protected:       // protected access specifier
int x;           // Data Member Declaration
void display();  // Member Function decaration
}
```

Q11. Write about different types of member functions.

Ans :

Types of Member Functions

Following are different types of Member functions,

1. Simple functions
2. Static functions
3. Const functions
4. Inline functions
5. Friend functions

1. Simple Member functions

These are the basic member function, which dont have any special keyword like static etc as prefix. All the general member functions, which are of below given form, are termed as simple and basic member functions.

```
return_type functionName(parameter_list)
{
function body;
}
```

2. Static Member functions

Static is something that holds its position. Static is a keyword which can be used with data members as well as the member functions. We will discuss this in details later. As of now we will discuss its usage with member functions only.

A function is made static by using static keyword with function name. These functions work for the class as whole rather than for a particular object of a class.

It can be called using the object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

Example:

```
class X
{
 public:
static void f(){};
};
int main()
{
 X::f();  // calling member function directly with class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

It doesn't have any "this" keyword which is the reason it cannot access ordinary members. We will study about "this" keyword later.

3. Const Member functions

Const keyword makes variables constant, that means once defined, there values can't be changed.

When used with member function, such member functions can never modify the object or its related data members.

```
//Basic Syntax of const Member Function
void fun() const {}
```

4. Inline functions

All the member functions defined inside the class definition are by default declared as Inline. We will study Inline Functions in details in the next topic.

5. Friend functions

Friend functions are actually not class member function. Friend functions are made to give privateaccess to non-class functions. You can declare a global function as friend, or a member function of other class as friend.

Example:

```
class WithFriend
{
 int i;
 public:
 friend void fun(); // Global function as friend
 };

 void fun()
 {
  WithFriend wf;
  wf.i=10;  // Access to private data member
  cout << wf.i;
 }

 int main()
 {
 fun(); //Can be called directly
 }
```

Hence, friend functions can access private data members by creating object of the class. Similarly we can also make function of other class as friend, or we can also make an entire class as friend class.

```
 class Other
 {
 void fun();
 };
 class WithFriend
 {
 private:
 int i;
 public:
 void getdata();  // Member function of class WithFriend
```

```
friend void Other::fun();  // making function of class Other as friend here
friend class Other;  // making the complete class as friend
};
```

When we make a class as friend, all its member functions automatically become friend functions.

2.2.3 C++ Program with Class

Q12. Write a program to enter student details and display it using classes. (Imp.)

Ans :

```
#include<iostream>
usingnamespace std;
class stud
{
public:
char name[30],clas[10];
int rol,age;
void enter()
{
cout<<"Enter Student Name: ";
cin>>name;
cout<<"Enter Student Age: ";
cin>>age;
cout<<"Enter Student Roll number: ";
cin>>rol;
cout<<"Enter Student Class: ";
cin>>clas;
}
void display()
{
cout<<"\n Age\tName\tR.No.\tClass"; cout<<"\n"<<age<<"\t"<<name<<"\t"<<rol<<"\t"<<clas;
}};
int main()
{
class stud s;
s.enter();
s.display();
cin.get();//use this to wait for a keypress
}
```


2.2.4 Private Membe

Q13. Explain private suitable exampl

Ans :

Private keyword, access the class membe that class. If someone member, they will get default class variables private.

```
class PrivateAccess
{
private:          // priv
int x;           // Dat
void display();  // Me
}
```

Program To Declare And Access It Using

```
# include <iostream
# include <conio.h>
class student
{
    private:
            int
            floa
            voi
        {
            rn=
            fee
        }
    public:
            vo
        {
            re
            co
            co
        }
};
    void n
    {
```

2.2.4 Private Member Functions

Q13. Explain private member functions with suitable example.

(Imp.)

Ans :

Private keyword, means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile time error. By default class variables and member functions are private.

```
class PrivateAccess
{
private:          // private access specifier
int x;           // Data Member Declaration
void display();  // Member Function decaration
}
```

Program To Declare Private Member Function And Access It Using Public Member Function.

```
# include <iostream.h>
# include <conio.h>
class student
{
    private:
            int rn;
            float fees;
            void read()
        {
            rn=12;
            fees=145.10;
        }
 public:
            void show()
        {
            read();
            cout<<"\n Rollno = "<<rn;
            cout<<"\n Fees = "<<fees;
        }
 };
        void main ( )
    {
            clrscr ( );
            student st;
            // st.read ( );  // not accessible
            st.show ( );
            getch();
        }
```

2.2.5 Arrays with in Class

Q14. How to declare arrays with in the class? Explain.

Ans : **(Imp.)**

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

Syntax:

```
class classname
 {
      .....
      ....
      datatype arrayname[size];
      ......
      ...
 };
```

A program to demonstrate the concept of arrays as class members

Example:

```
#include<iostream>
constint size=5;

class student

{

int roll_no;
int marks[size];
public:
voidgetdata();
voidtot_marks();
} ;
```

```
void student :: getdata ()
{
cout<<"\nEnter roll no: ";
Cin>>roll_no;
for(int i=0; i<size; i++)
{
cout<<"Enter marks in subject"<<(i+1)<<": ";
cin>>marks[i] ;
}

void student :: tot_marks() //calculating total marks
{
int total=0;
for(int i=0; i<size; i++)
total+ = marks[i];
cout<<"\n\nTotal marks "<<total;
}

voidmain()
student stu;
stu.getdata() ;
stu.tot_marks() ;
getch();
}
```

Output:

- Enter roll no: 101
  Enter marks in subject 1: 67
  Enter marks in subject 2 : 54
  Enter marks in subject 3 : 68
  Enter marks in subject 4 : 72
  Enter marks in subject 5 : 82
  Total marks = 343

2.2.6 Memory Allocation for Objects

Q15. How to allocate memory for objects in C++? Explain.

Ans : **(Imp.)**

Arrays can be used to store multiple homogenous data but there are serious drawbacks of using arrays. Programmer should allocate the memory of an array when they declare it but most of time, the exact memory needed cannot be determined until runtime. The best thing to do in this situation is to declare the array with maximum possible memory required (declare array with maximum possible size expected) but this wastes memory.

To avoid wastage of memory, you can dynamically allocate the memory required during runtime using new and delete operator.

The new Operator

Syntax : ptr = new float[n];

This expression in the above program returns a pointer to a section of memory just large enough to hold the n number of floating-point data.

The delete Operator

Once the memory is allocated using new operator, it should released to the operating system. If the program uses large amount of memory using new, system may crash because there will be no memory available for operating system. The following expression returns memory to the operating system.

Syntax : delete [] ptr;

The brackets [] indicates that, array is deleted. If you need to delete a single object then, you don't need to use brackets.

Example : delete ptr;

```
// demonstration of new and delete operators
#include<iostream>
usingnamespace std;

classTest{
private:
int n;
float*ptr;
public:

Test(){
    cout <<"Enter total number of students: ";
    cin >> n;

    ptr =newfloat[n];
```

```
    cout <<"Enter GPA of students."<<endl;
for(int i =0; i < n; ++i){
    cout <<"Student"<< i+1<<": ";
    cin >>*(ptr + i);

}
}

~Test(){
delete[] ptr;
}

voidDisplay(){
    cout <<"\nDisplaying GPA of students."<<
    endl;
for(int i =0; i < n; ++i){
    cout <<"Student"<< i+1<<" :"<<*(ptr
    + i)<< endl;

}
}
};
int main(){
Test s;
  s.Display();
return0;
}
```

The output of this program is same as above program. When the object s is created, the constructor is called which allocates the memory for n floating-point data.

When the object is destroyed, that is, object goes out of scope then, destructor is automatically called.

```
  ~Test() {
    delete[] ptr;
  }
```

This destructor executes delete[] ptr; and returns memory to the operating system.

Passing and Returning Object from Function in C++ Programming

In C++ programming, objects can be passed to function in similar way as variables and structures

2.2.7 Static Data Members

Q16. Explain about static data members with an example.

Ans :

It is a variable which is declared with the static keyword, it is also known as class member, thus only single copy of the variable creates for all objects.

Any changes in the static data member through one member function will reflect in all other object's member functions.

Declaration

static data_type member_name;

Defining the static data member

It should be defined outside of the class following this syntax:

data_type class_name :: member_name =value;

If you are calling a static data member within a member function, member function should be declared as static (i.e. a static member function can access the static data members)

Consider the example, here static data member is accessing through the static member function:

```
#include <iostream>
usingnamespace std;

class Demo
{
    private:
        staticint X;

    public:
        staticvoid fun()
        {
            cout <<"Value of X: " << X << endl;
        }
};

//defining
int Demo :: X =10;
```

```
int main()
{
    Demo X;
    X.fun();
    return 0;
}
```

Output

Value of X: 10

2.2.8 Static Member Functions

Q17. Write about static member functions in C++. **(Imp.)**

Ans :

The static member functions are special functions used to access the static data members or other static member functions. A member function is defined using the static keyword. A static member function shares the single copy of the member function to any number of the class' objects. We can access the static member function using the class name or class' objects. If the static member function accesses any non-static data member or non-static member function, it throws an error.

Syntax

```
class_name::function_name (parameter);
```

Here, the class_name is the name of the class.

function_name: The function name is the name of the static member function.

parameter: It defines the name of the pass arguments to the static member function.

Example 2

Let's create another program to access the static member function using the class name in the C++ programming language.

```
#include <iostream>
using namespace std;
class Note
{
// declare a static data member
static int num;
public:
// create static member function
static int func ()
{
return num;
}
};
// initialize the static data member using the class name and the scope resolution operator
int Note :: num = 5;

int main ()
{
// access static member function using the class name and the scope resolution
cout << "The value of the num is: " << Note:: func () << endl
return 0;
}
```

Output

The value of the num is: 5

2.2.9 Arrays of Objects

Q18. What are called arrays of objects? How to declare them ? Explain.

Ans : **(Imp.)**

- Like array of other user-defined data types an array of type class can also be created.
- The array of type class contains the objects of the class as its individual elements.
- Thus, an array of a class type is also known as an array of objects.
- An array of objects is declared in the same way as an array of any built-in data type.

Syntax:

```
class_name array_name [size] ;
```

Example:

```
#include <iostream>

class MyClass {
  int x;
public:
  void setX(int i) { x = i; }
  int getX() { return x; }
};

void main()
{
  MyClass obs[4];
  int i;
  for(i=0; i < 4; i++)
    obs[i].setX(i);
  for(i=0; i < 4; i++)
    cout << "obs[" <<
obs[i].getX() << "\n";
  getch();
}
```

Output:

```
obs[0].getX(): 0
obs[1].getX(): 1
obs[2].getX(): 2
obs[3].getX(): 3
```

2.2.10 Returning

Q19. Write about objects in C

Ans :

An object is a
only allocated wher
when a class is defi

An object can
the return keyword
this is given as follo

Example

```
#include<iostrean
usingnamespace st
classPoint{
  private:
  int x;
  int y;
```

```
public:
 void setX(int i) { x = i; }
 int getX() { return x; }
};

void main()
{
 MyClass obs[4];
 int i;

 for(i=0; i < 4; i++)
   obs[i].setX(i);

 for(i=0; i < 4; i++)
   cout << "obs[" << i << "].getX(): " <<
obs[i].getX() << "\n";

 getch();
}
```

Output:

```
obs[0].getX(): 0
obs[1].getX(): 1
obs[2].getX(): 2
obs[3].getX(): 3
```

2.2.10 Returning Objects

Q19. Write about the concept of returning objects in C++.

Ans : **(Imp.)**

An object is an instance of a class. Memory is only allocated when an object is created and not when a class is defined.

An object can be returned by a function using the return keyword. A program that demonstrates this is given as follows "

Example

```
#include<iostream>
usingnamespace std;
classPoint{
   private:
   int x;
   int y;
   public:
   Point(int x1 =0,int y1 =0){
      x = x1;
      y = y1;
   }
   Point addPoint(Point p){
      Point temp;
      temp.x = x + p.x;
      temp.y = y + p.y;
      return temp;
   }
   void display(){
      cout<<"x = "<< x <<"\n";
      cout<<"y = "<< y <<"\n";
   }
};
int main(){
   Point p1(5,3);
   Point p2(12,6);
   Point p3;
   cout<<"Point 1\n";
   p1.display();
   cout<<"Point 2\n";
   p2.display();
   p3 = p1.addPoint(p2);
   cout<<"The sum of the two points is:\n";
   p3.display();
   return0;
}
```

Output

The output of the above program is as follows.

```
Point 1
x = 5
y = 3
Point 2
x = 12
y = 6
The sum of the two points is:
x = 17
y = 9
```

2.3 Functions in C++

2.3.1 Main Function

Q20. Write about the importance of main function in C++. **(Imp.)**

Ans :

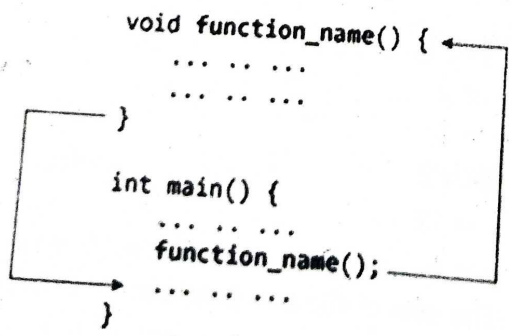
A function is said to be a set of instructions that can perform a specific task. It will be reduce the size of the program by calling and using them at different places in the program.

The advantages are: Reusability, modularity, and overall programming simplicity.

Functions are classified into library functions and user-defined functions.

- **Library Functions:** The functions that are already available in the C++ library are called Library functions. Some important categories of library functions are : Input/ Output functions (iostream.h), Mathematical functions (Math.h), String handling functions (String.h), Console input/output functions (conio.h) and Manipulator functions (iomanip.h).
- **User-Defined Functions:** A function defined by the user is called as user-defined function. The advantage of user-defined function is that the complexity of a program is reduced and the errors can be traced easily.

How user-defined function works in C Programming?



Consider the figure above.

When a program begins running, the system calls the main() function, that is, the system starts executing codes from main() function.

When control of program reaches to function_name() inside main(), the control of program moves to void function_name(), all codes inside void function_name() is executed and control of program moves to code right after function_name() inside main() as shown in figure above.

The Main Function:

In C++, the **main()** returns a value of type **int** to the operating system. C++, therefore, explicitly defines **main()** as matching one of the following prototypes:

```
int main( );
int main( int arg);
```

The function that have a return value should use the **return** statement for termination. The **main( )** function in C++, therefore, defined as follows:

```
int main( )
{ ————————
————————
return 0;
}
```

The return type of function is **int** by default, the keyword **int** in the main( ) header is optional. Most C++ compilers will generate an error or warning if there is no **return** statement.

2.3.2 Function Prototyping

Q21. Write, how to define a function in C++.

Ans : **(Imp.)**

If an user-defined function is defined after main() function, compiler will show error. It is because compiler is unaware of user-defined function, types of argument passed to function and return type.

In C++, function prototype is a declaration of function without function body to give compiler information about user-defined function. Function prototype in above example:

```
int add(int, int);
```

You can see that, there is no body of function in prototype. Also there are only return type of arguments but no arguments. You can also declare function prototype as below but it's not necessary to write arguments.

```
int add(int a, int b);
```

***Note: It is not necessary to define prototype if user-defined function exists before main()function.

Function Definition

The function itself is referred as function definition. Function definition in the above program:

```
/* Function definition */
int add(int a,int b) {      // Function declarator
  int add;
  add = a+b;
  return add;               // Return statement
}
```

When the function is called, control is transferred to first statement of function body. Then, other statements in function body are executed sequentially. When all codes inside function definition is executed, control of program moves to the calling program.

2.3.3 Call By Reference

Q22. Write about passing parameters by calling a function.

Ans : **(Imp.)**

Passing parameters by calling a Function

Functions are called by their names. If the function is without argument, it can be called directly using its name. But for functions with arguments, we have two ways to call them,

1. Call by Value
2. Call by Reference

1. Call by Value

In this calling technique we pass the values of arguments which are stored or copied into the formal parameters of functions. Hence, the original values are unchanged only the parameters inside function changes.

```
void calc(int x);
int main()
{
 int x = 10;
 calc(x);
 printf("%d", x);
}
void calc(int x)
{
 x = x + 10 ;
}
```

Output : 10

In this case the actual variable x is not changed, because we pass argument by value, hence a copy of x is passed, which is changed, and that copied value is destroyed as the function ends(goes out of scope). So the variable x inside main() still has a value 10.

But we can change this program to modify the original x, by making the function calc() return a value, and storing that value in x.

```
int calc(int x);
int main()
{
 int x = 10;
 x = calc(x);
 printf("%d", x);
}
int calc(int x)
{
 x = x + 10 ;
 return x;
}
```

Output : 20

2. Call by Reference

In this we pass the address of the variable as arguments. In this case the formal parameter can be taken as a reference or a pointer, in both the case they will change the values of the original variable.

```
void calc(int *p);
int main()
{
 int x = 10;
 calc(&x);      // passing address of x as
argument
 printf("%d", x);
}
void calc(int *p)
{
 *p = *p + 10;
}
```

Output : 20

2.3.4 Return By Reference

Q23. Explain, how to return a function by reference in C++.

Ans : **(Imp.)**

In C++ Programming, you can pass values by reference but also you can return a value by reference. To understand this feature, you should have knowledge of global variables. If a variable is defined outside every function, then that variable is called a global variable. Any part of program after global variable declaration can access global variable

Example : Return by Reference

```
#include<iostream>
usingnamespace std;
int n;
int& test();
int main() {
   test()=5;
   cout<<n;
return0;  }
int& test() {
return n;
}
```

Output

5

In program above, the return type of function test() is int&. Hence this function returns by reference. The return statement is return n; but unlike return by value. This statement doesn't return value of n, instead it returns variable n itself.

Then the variable n is assigned to the left side of code test() = 5; and value of n is displayed.

Important Things to Care While Returning by Reference.

```
int& test() {
   int n = 2;
   return n;
}
```

- Ordinary function returns value but this function doesn't. Hence, you can't return constant from this function.

```
int& test() {
   return 2;
    }
```

- You can't return a local variable from this function.

2.3.5 Inline Functions

Q24. What are called as Inline Functions? Explain it with syntax.

Ans : **(Imp.)**

Inline functions are actual functions, which are copied everywhere during compilation, like preprocessor macro, so the overhead of function calling is reduced. All the functions defined inside class definition are by default inline, but you can also make any non-class function inline by using keyword inline with them.

For an inline function, declaration and definition must be done together. For example,

```
inline void fun(int a)
{
  return a++;
}
```

Some Important points about Inline Functions

1. We must keep inline functions small, small inline functions have better efficiency.
2. Inline functions do increase efficiency, but we should not make all the functions inline. Because if we make large functions inline, it may lead to code bloat, and might affect the speed too.
3. Hence, it is adviced to define large functions outside the class definition using scope resolution ::operator, because if we define such functions inside class definition, then they become inline automatically.
4. Inline functions are kept in the Symbol Table by the compiler, and all the call for such functions is taken care at compile time.

Limitations of Inline Functions

1. Large Inline functions cause Cache misses and affect performance negatively.
2. Compilation overhead of copying the function body everywhere in the code on compilation, which is negligible for small programs, but it makes a difference in large code bases.
3. Also, if we require address of the function in program, compiler cannot perform inlining on such functions. Because for providing address to a function, compiler will have to allocate storage to it. But inline functions doesn't get storage, they are kept in Symbol table.

Q25. Write a program to find the multiplication values and the cubic values using inline function.

Ans :

```
#include<iostream.h>
#include<conio.h>
class line
{
  public:
        inline float mul(float x,float y)
    {
                return(x*y);
    }
        inline float cube(float x)
        {
                return(x*x*x);
        }
};

void main()
{
        line obj;
        float val1,val2;
        clrscr();
        cout<<"Enter two values:";
        cin>>val1>>val2;
        cout<<"\nMultiplication value is:"<<obj.mul(val1,val2);
        cout<<"\n\nCube value is :"<<obj.cube(val1)<<"\t"<<obj.cube(val2);
        getch();
}
```

Output:

```
Enter two values: 5  7
Multiplication Value is: 35
Cube Value is: 25 and 343
```

2.3.6 Default Arguments

Q26. What are Default Argumentsin C++. Explain them.

Ans :

In C++ programming, you can provide default values for function parameters. The idea behind default argument is very simple. If a function is called by passing argument/s, those arguments are used by the function. But if all argument/s are not passed while invoking a function then, the default value passed to arguments are used. Default value/s are passed to argument/s in function prototype. Working of default argument is demonstrated in the figure below:

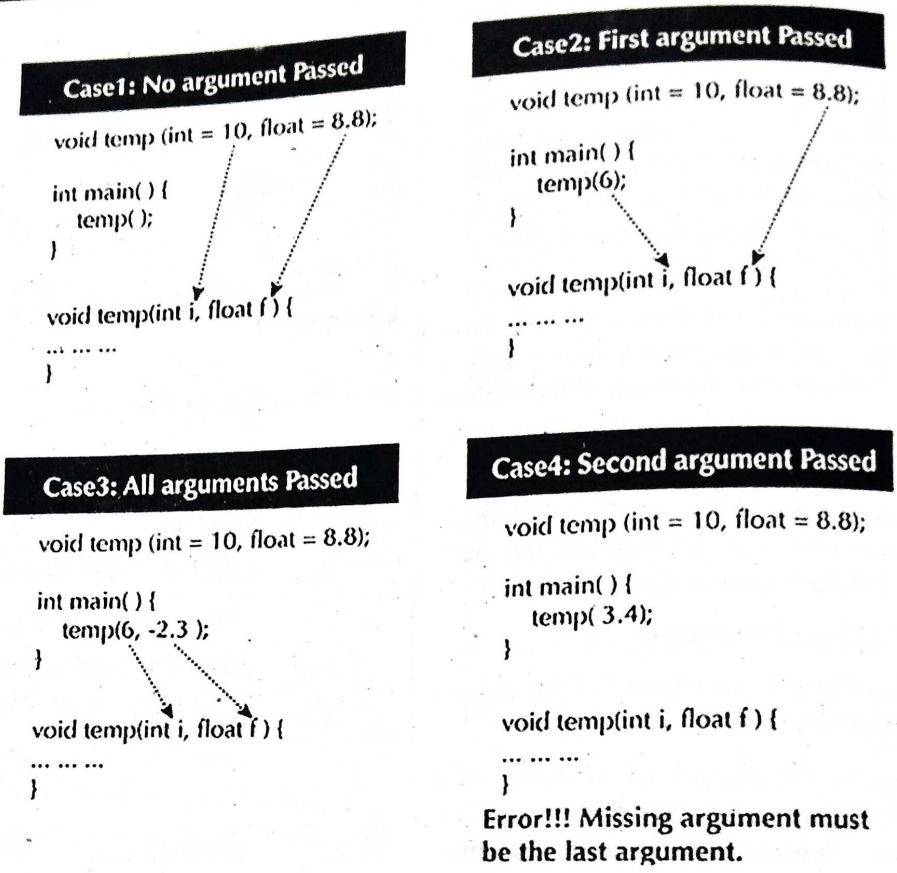


Fig. : Working of Default Argument in C++

```
/* C++ Program to demonstrate working of default argument */
#include<iostream>
usingnamespace std;
void display(char='*',int=1);
int main(){
  cout<<"No argument passed:\n";
  display();
  cout<<"\n\nFirst argument passed:\n";
  display('#');
    cout<<"\n\nBoth argument passed:\n";
  display('$',5);
return0 } void display(char c,int n)
{
for(int i =1; i <=n;++i)
```

```
{
    cout<<c;
}   cout<<endl;
}
```

Output

No argument passed:

*

First argument passed:

#

Both argument passed:

$$$$$

In the above program, At first, display() function is called without passing any arguments. In this case, default() function used both default arguments. Then, the function is called using only first argument. In this case, function does not use first default value passed. Function uses the actual parameter passed as first argument and takes default value(second value in function prototype) as it's second argument. When display() is invoked passing both arguments, default arguments are not used.

*****Note: The missing argument must be the last argument of the list, that is, if you are passing only one argument in the above function, it should be the first argument.**

Short Question and Answers

1. If statement.

Ans :

The if statement checks whether the test condition is true or not. If the test condition is true, it executes the code/s inside the body of if statement. But it the test condition is false, it skips the code/s inside the body of if statement.

The if keyword is followed by test condition inside parenthesis ( ).

2. What is class?

Ans :

A class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from the same description, we can create many objects from a class.

defining a class in C++

A class is defined in C++ using keyword class followed by the name of class.

The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

```
class className
 {
 // some data
 // some functions
 };
```

3. What is an object in C++?

Ans :

Class is mere a blueprint or a template. No storage is assigned when we define a class. Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.

Each object has different data variables. Objects are initialised using special class functions called Constructors.

4. What are Member Functions?

Ans :

Member functions are the functions, which have their declaration inside the class definition and works on the data members of the class. The definition of member functions can be inside or outside the definition of class.

If the member function is defined inside the class definition it can be defined directly, but if its defined outside the class, then we have to use the scope resolution : : operator along with class name along with function name.

5. Private Member Functions.

Ans :

Private keyword, means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile time error. By default class variables and member functions are private.

```
class PrivateAccess
{
```

6. Arrays with in the class.

Ans :

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

Syntax:

```
class classname
{
    .....
```

```
....
datatype arrayname[size];
.....
...
};
```

A program to demonstrate the concept of arrays as class members

7. Memory allocation for objects.

Ans :

Arrays can be used to store multiple homogenous data but there are serious drawbacks of using arrays. Programmer should allocate the memory of an array when they declare it but most of time, the exact memory needed cannot be determined until runtime. The best thing to do in this situation is to declare the array with maximum possible memory required (declare array with maximum possible size expected) but this wastes memory.

To avoid wastage of memory, you can dynamically allocate the memory required during runtime using new and delete operator.

The new Operator

Syntax : ptr = new float[n];

This expression in the above program returns a pointer to a section of memory just large enough to hold the n number of floating-point data.

The delete Operator

Once the memory is allocated using new operator, it should released to the operating system. If the program uses large amount of memory using new, system may crash because there will be no memory available for operating system. The following expression returns memory to the operating system.

Syntax : delete [] ptr;

The brackets [] indicates that, array is deleted. If you need to delete a single object then, you don't need to use brackets.

8. Static member functions.

Ans :

The static member functions are special functions used to access the static data members or other static member functions. A member function is defined using the static keyword. A static member function shares the single copy of the member function to any number of the class' objects. We can access the static member function using the class name or class' objects. If the static member function accesses any non-static data member or non-static member function, it throws an error.

Syntax

class_name::function_name (parameter);

Here, the class_name is the name of the class.

function_name: The function name is the name of the static member function.

parameter: It defines the name of the pass arguments to the static member function.

9. Arrays of objects.

Ans :

- Like array of other user-defined data types, an array of type class can also be created.
- The array of type class contains the objects of the class as its individual elements.
- Thus, an array of a class type is also known as an array of objects.
- An array of objects is declared in the same way as an array of any built-in data type.

Syntax:

class_name array_name [size] ;

Choose the Correct Answers

1. The _____ is an entry-enrolled loop and is used when an action is to be repeated for a predetermined number of times.

(a) while (b) for

(c) do-while (d) switch

2. The _____ is an exit-enrolled loop where the control is transferred back to a particular point in the program.

(a) while (b) for

(c) do-while (d) switch

3. The _____ is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points.

(a) while (b) for

(c) do-while (d) switch

4. Which is used to define the member of a class externally?

(a) : (b) ::

(c) # (d) !!$

5. if you have to make decision based on multiple choices, which of the following is best suited?

a) if (b) if-else

(c) if-else-if (d) All of the above

6. How many objects can present in a single class?

(a) 1 (b) 2

(c) 3 (d) as many as possible

7. Which keyword is used to define the user defined data types?

(a) def (b) union

(c) typedef (d) type

8. Identify the correct syntax for declaring arrays in C++.

(a) array arr[10] (b) array{10}

(c) intarr[10] (d) intarr

9. Which of the following is the default return value of functions in C++? [d]

(a) int (b) char

(c) float (d) void

10. By default how the value are passed in c++? [a]

(a) call by value (b) call by reference

(c) call by pointer (d) call by object

Fill in the Blanks

1. A class in C++ can hold _____ and _____.
2. object in the class is accessed by _____.
3. An _____ represents an instance of a class.
4. the execution of the program starts from _____.
5. _____ function is those which are expanded at each call during the execution of the program to reduce the cost of jumping during execution.
6. The execution of the program is returned to the point from where the function was called and the function from which this function was called is known as _____.
7. The same name function to perform different procedure for different types of parameters or different number of parameters provided to the function..is called as _____.
8. _____ is the storage classes that have global visibility.
9. The _____ will be used when the user value is not given.
10. _____ is used by a pointer object to access members of its class.

ANSWERS

1. data & functions
2. direct member access operator
3. Object
4. main function
5. inline function
6. caller function
7. Function over loading
8. extern
9. default value
10. ->(arrow operator)