RAGHU INSTITUTE OF TECHNOLOGY

Dakamarri (v), Bheemunipatnam (M)
Visakhapatnam Dist, Andhra Pradesh, PIN-531162
(Approved by AICTE, New Delhi, and Affiliated to Jawaharlal Nehru Technological University: Kakinada (AP))

II B. Tech., CSE I - Semester

FACULTY LABORATORY MANUAL

For

Object oriented programming through C++

Prepared by

V Hemanth Kumar, Assistant Professor

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

RAGHU INSTITUTE OF TECHNOLOGY

(Affiliated to JNTU-KAKINADA)
Visakhapatnam-531162

CERTIFICATE

Name of the Laboratory : Object oriented programming through c++

Name of the Faculty : V. Hemanth Kumar

Department : CSE

Program : B.TECH

Year : II

Semester : I

IQAC Members:

Name(s):

Signature(s):

Course Objectives

- To develop skills to design and analyze object oriented program.
- To strengthen the ability to identify and apply the suitable object oriented concept for the given real world problem
- To gain knowledge in practical applications of object oriented concept.

At the end of this lab session

- Competences to design, write, compile, test and execute straightforward programs using a high level language.
- An awareness of the need for a professional approach to design and the importance of good documentation to the finished programs.
- The students will learn to write, compile & execute basic c++ program.
- The student will learn the use of data types & variables, decision control structures: if, nested if etc.
- The student will learn the use loop control structures: do, while, for etc.

CO to PO mapping:

		A	B	C	D	E	F	G	H	I	J	K	L
OOP through C++ lab	Competences to design, write, compile, test and execute straightforward programs using a high level language.		✓										
	An awareness of the need for a professional approach to design and the importance of good documentation to the finished programs.		✓				✓						
	The students will learn to write, compile& execute basic c++ program.		✓										✓
	The student will learn the use of data types & variables, decision control structures: if, nested if etc.		✓						✓				
	The student will learn the use loop control structures: do, while, for etc.		✓						✓				

JNTUK LAB SYLLABUS

1. Write a C++ program illustrating Variable scope.
2. Write a C++ program illustrating Swapping integer values by reference.
3. Write a C++ program illustrating Checking whether the number is even or odd using Ternary operator.
4. Write a C++ program illustrating a program to find the roots of a quadratic equation. Use switch statements to handle different values of the discriminant($b^2 – 4ac$).
5. Write a C++ program illustrating interactive program to multiply 2 variables after checking the compatibility.
6. Write a C++ program illustrating interactive program for computing the roots of a quadratic equation by handling all possible cases. Use streams to perform I/O operations.
7. Write a C++ program illustrating to sort integer numbers.
8. Write a C++ program illustrating factorial using recursion.
9. Write a C++ program illustrating pass by value, pass by reference, pass by address.
10. Write a C++ program illustrating function overloading.
11. Write a C++ program illustrating an interactive program for swapping integer, real, and character type variables without using function overloading .Write the same program by using function overloading features and compare the same with its C counterpart.
12. Write a C++ program illustrating inline functions.
13. Write a C++ program illustrating Friend function.
14. Write a C++ program illustrating Exception Handling.
15. Write a C++ program illustrating Function Template
16. Write a C++ program illustrating Overloading increment, decrement, binary +&<< operator.
17. Write a C++ program illustrating Virtual function.
18. Write a C++ program illustrating an interactive program to process complex numbers. It has to perform addition, subtraction, multiplication and division of complex numbers. Print results in x+iy form. Create a class for the complex number representation.
19. Write a C++ program illustrating user defined string processing functions using pointers (string length, string copy, string concatenation)

20. Write a C++ program illustrating Constructor overloading (Both parameterized and default).
21. Write a C++ program illustrating Copy constructor.
22. Write a C++ program illustrating access data members and member functions using 'THIS' pointer.
23. Write a C++ program illustrating for overloading ++ operator to increment data.
24. C++ program illustrating overloading of new and delete operator.
25. Write a C++ program illustrating Abstract classes.
26. Write a C++ program illustrating inheritance (Multiple, Multilevel, Hybrid)
27. Write a C++ program illustrating Virtual classes & virtual functions
28. Write a C++ program illustrating overloading function template.
29. Write a C++ program illustrating class template

LIST OF PROGRAMS

1. Write a C++ program illustrating Variable scope.
2. Write a C++ program illustrating Swapping integer values by reference.
3. Write a C++ program illustrating Checking whether the number is even or odd using Ternary operator.
4. Write a C++ program illustrating a program to find the roots of a quadratic equation. Use switch statements to handle different values of the discriminant($b^2 – 4ac$).
5. Write a C++ program illustrating to sort integer numbers.
6. Write a C++ program illustrating factorial using recursion.
7. Write a C++ program illustrating pass by value, pass by reference, pass by address.
8. Write a C++ program illustrating function overloading.
9. Write a C++ program illustrating an interactive program for swapping integer, real, and character type variables without using function overloading .Write the same program by using function overloading features and compare the same with its C counterpart.
10. Write a C++ program illustrating inline functions.
11. Write a C++ program illustrating Friend function.
12. Write a C++ program illustrating Exception Handling.
13. Write a C++ program illustrating Function Template
14. Write a C++ program illustrating Overloading increment, decrement, binary +&<< operator.
15. Write a C++ program illustrating Virtual function.
16. Write a C++ program illustrating an interactive program to process complex numbers. It has to perform addition, subtraction, multiplication and division of complex numbers. Print results in x+iy form. Create a class for the complex number representation.
17. Write a C++ program illustrating user defined string processing functions using pointers (string length, string copy, string concatenation)
18. Write a C++ program illustrating Constructor overloading (Both parameterized and default).
19. Write a C++ program illustrating Copy constructor.
20. Write a C++ program illustrating Abstract classes.
21. Write a C++ program illustrating inheritance (Multiple, Multilevel, Hybrid)

22. Write a C++ program illustrating Virtual classes & virtual functions
23. Write a C++ program illustrating class template

Additional programs:

1. Write a c++ program illustrating the concepts of static data members.
2. Write a c++ program to illustrate the concept of manipulators.
3. Write a c++ program to illustrate the concept of pointers.

SCHEDULE/CYCLE CHART

SI NO	PROGRAM NAME/NUMBER	DATE
1	Variable scope	
2	Swapping integers by reference	
3	Checking the number even or odd using Ternary Operator	
4	Roots of a quadratic equation	
5	Sorting integer numbers	
6	Factorial using Recursion	
7	Pass by value, address and reference	
8	Function overloading	
9	Swapping values without/with function overloading	
10	Inline function	
11	Friend function	
12	Exception handling	
13	Function template	
14	Unary & Binary Operator overloading	
15	Virtual function	
16	Class for complex number representation	
17	String processing functions	
18	Constructor overloading	
19	Copy constructor	
20	Overloading Unary operators	
21	Abstract classes	
22	Inheritance	
23	Virtual classes	
24	Class template	

1. VARIABLE SCOPE

Aim:

Write a c++ program illustrating variable scope.

Algorithm:

Step1: Use global variable as glo is 10

Step2: Use local variables as lo=20 , glo=40

Step3: Print valus of variables lo, glo (local variable)

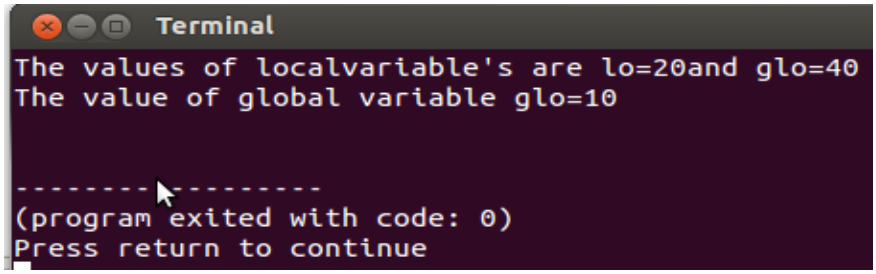
Step4: Print value of glo (global variable)

Step5: stop

Program:

```
#include<iostream>
using namespace std;
int glo=10;
main()
{
int lo=20,glo=40;
cout<<"The values of localvariable's are lo="<<lo<<"and glo="<<glo<<endl;
cout<<"The value of global variable glo="<<::glo<<endl;
}
```

Output:



Viva questions:

1. **What is life time?**

   Ans: Life time is the time period for which a variable exist in the memory.

2. **What is scope?**

   Ans: Scope is the time in which it can be accessed

3. **What is the scope of a local variable?**

   Ans: the scope of a local variable is the execution time of that particular function where it is declared.

4. **What is the scope of a global variable?**

   Ans: scope of a global variable is same as the execution time of the total program because in each function we can access it.

5. **Which is bigger, scope or lifetime?**

   Ans: scope<=lifetime

6. **What is difference between C and C++?**

   Ans:

   1. C++ is Multi-Paradigm (not pure OOP, supports both procedural and object oriented) while C follows procedural style programming.
   2. In C data security is less, but in C++ you can use modifiers for your class members to make it inaccessible from outside.
   3. C follows top-down approach, but C++ follows a bottom-up approach C++ supports function overloading while C does not support it.
   4. C++ allows use of functions in structures, but C does not permit that.
   5. C++ supports reference variables (two variables can point to same memory location). C does not support this.
   6. C does not have a built in exception handling, C++ directly supports exception handling, which makes life of developer easy.

7. **What are the basics concepts of OOP?**

   Ans: Object and Classes, Data Abstraction and Encapsulation, Polymorphism, Inheritance, Message passing, Dynamic binding

2. SWAPPING INTEGERS BY REFERENCE

Aim:

Write a C++ program illustrating Swapping integer values by reference.

Algorithm:

Step1: Start

Step2: Use a=100, b=200

Step3: Print ' before swapping, values of a, b are ‘

Step4: Call function swap (a, b)

Step5: Print ‘After swapping, values of a, b are’

Step6: stop

Algoritham for function swap(&x,&y)

Step1: Use variable temp.

Step2: temp = x;

Step3: x = y;

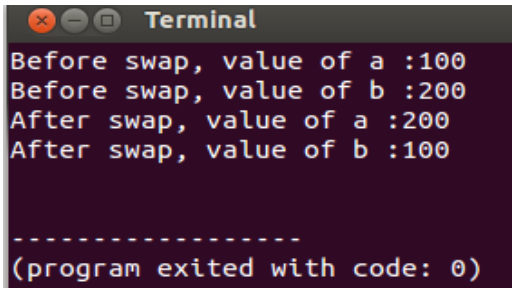
Step4: y = temp;

Program:

```
#include <iostream>
using namespace std;
void swap(int &x, int &y);              // function declaration
int main ()
{                                       // local variable declaration
  int a = 100;
  int b = 200;
  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl;
                        /* calling a function to swap the values using variable reference.*/
  swap(a, b);
  cout << "After swap, value of a :" << a << endl;
```

```
  cout << "After swap, value of b :" << b << endl;
  return 0;
}
                                    // function definition to swap the values.
void swap(int &x, int &y)
{
  int temp;
  temp = x;              /* save the value at address x  in temp */
  x = y;                 /* put y into x */
  y = temp;              /* put temp into y */
}
```

Output:



Viva questions:

1. **What is a reference variable?**

   Ans: a reference variableis an alias (alternative name) of an existing variable,i.e,we can use both the names two access the same variable.

2. **What is the syntax to create a reference?**

   Ans: data_type & reference_name = variable name-value

3. **Give an example.**

   Ans: float total=100;

   float & sum = total; *now both sum and total will access the same value.*/

   cout<<sum<< total; /*both will print same value*/

   sum=sum+10;

```
cout<<total;/*110*/
```

4. **What is the difference between call by value and call by address?**

   Ans: in call by value one copy of the original value is going as the argument to the called function. In call by address the original variable is accessed by pointers from the called function.

5. **What is the difference between call by address and call by reference?**

   Ans: In call by address the original variable is accessed by pointers from the called function. In call by reference the reference of the original variable is created and it is sent to called functions.

6. **What are the differences between pointer and reference?**

   Ans: When a reference is created, it can't reference another object. This can be done with pointers. References cannot be null whereas pointers can be. References cannot be uninitialized and it is not possible to refer directly to a reference object after it is defined.

3. CHECKING THE NUMBER EVEN OR ODD USING TERNARY OPERATOR

Aim:

Write a C++ program to illustrate checking whether the no is even or odd using ternary operator

Algorithm:

Step1: start

Step2: print 'enter a number'

Step3: read n value

Step4: if n%2==0 (use Ternry Operator)

4.1: print the number is even

Else

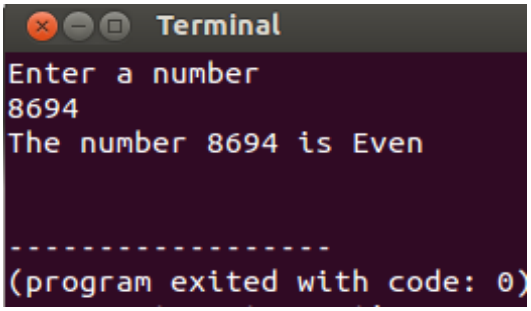
4.2: print the number is odd

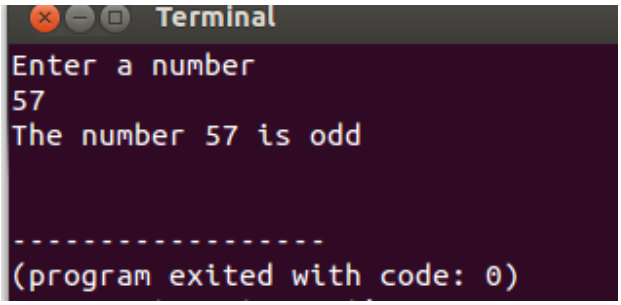
Step5: stop.

Program:

```
#include<iostream>
using namespace std;
int main()
{
 int n;
 cout<<"Enter a number"<<endl;
 cin>>n;
 ((n%2)==0)?cout<<"The number "<<n<<" is Even"<<endl:cout<<"The number "<<n<<" is
odd"<<endl;
 return 0;
}
```

Output:





Viva Questions:

1. **What is a ternary operator?**

   Ans: It is a decision statement.

2. **In which case ternary operator is used?**

   Ans: if no of conditions are more.

3. **What is the syntax of ternary operator?**

   Ans: (condition)?(if true these statement);(if false)

4. **Write an example of ternary operator.**

   Ans: (x<y)?cout<<x is greater):cout <<'y is greater'

5. **Which is easy to implement, if.....else or ternary operator?**

   Ans: programmer dependant.

4. ROOTS OF A QUADRATIC EQUATION

Aim:

Write a C++ program illustrating a program to find the roots of a quadratic equation .Use switch statements to handle different values of the discriminant (b^2 -4*a*c).

.

Algorithm:

Step1: start

Step2: print 'Enter three co-efficients'

Step3: Read the values of a, b, c

Step4: calculate disc=b*b-4*a*c

Step5: if disc>0 then flag=0

Step6: if disc==0 then flag=1

Step7: if disc<0 then flag=2

Step8: if flag = 0

8.1: Calculate x1= (-b + sqrt(disc))/(2*a) & x2=(-b - sqrt(disc))/(2*a);

8.2: Write 'the roots are distinct'

8.3: Print values of x1, x2

Step9: if flag = 1

9.1: Calculate x1=x2=-b/(2*a);

9.2: Write 'the roots are equal'

9.3: print values of x1, x2

Step10: if flag = 2

10.1: Calculate x1=-b/(2*a) & x2=sqrt(fabs(disc))/(2*a);

10.2: Write 'The roots are complex'

10.3: Print 'root1= 'x1+i x2

10.4: Print 'root2= 'x1-i x2

Step11: stop

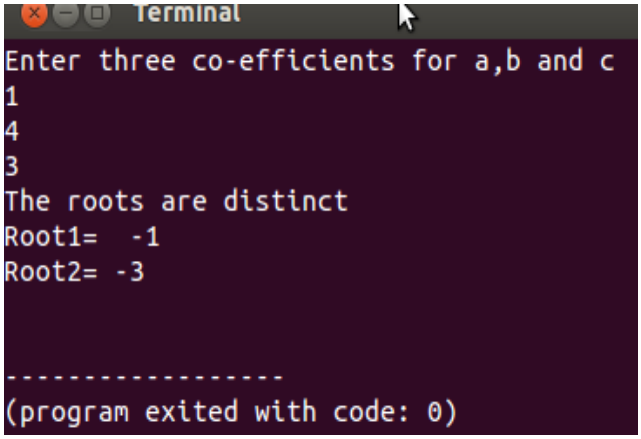
Program:

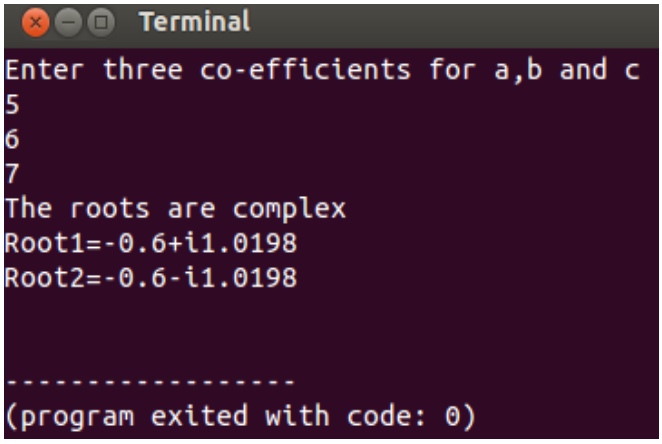
```
#include<cmath>
#include<iostream>
```

```
using namespace std;
int main()
{
	float a,b,c,x1,x2,disc;
	int flag;
	cout<<"Enter three co-efficients for a,b and c"<<endl;
	cin>>a>>b>>c;
	disc=b*b-4*a*c;
	if(disc>0)
		flag=0;
	if(disc==0)
		flag=1;
	if(disc<0)
		flag=2;
	switch(flag)
	{
		case 0:
			x1=(-b+sqrt(disc))/(2*a);
			x2=(-b-sqrt(disc))/(2*a);
			cout<<"The roots are distinct"<< endl;
			cout<<"Root1=  "<<x1<<"\nRoot2=  "<<x2<<endl;
			break;
		case 1:
			x1=x2=-b/(2*a);
			cout<<"The roots are equal";
			cout<<"Root1=  "<<x1<<"\nRoot2=  "<<x2<<endl;
			break;
		case 2:
			x1=-b/(2*a);
			x2=sqrt(fabs(disc))/(2*a);
			cout<<"The roots are complex"<<endl;
			cout<<"Root1="<<x1<<"+i"<<x2<<endl;
```

```
                    cout<<"Root2="<<x1<<"-i"<<x2<<endl;
                    break;
            }
        return 0;
}
```

Output:





Viva questions:

1. **What is the purpose of switch case?**

   Ans: to choose one among many conditions

2. **Give a real time example.**

   Ans : menu driven program.

3. **Is the default statement mandatory?**

   Ans: no

4. **What will happen if break will be removed?**

   Ans: the next executable lines will be executed.

5. **What is the advantage of switch over if else?**

   Ans: if else is a complex programming where as switch is organized in one block.

5. SORTING INTEGER NUMBERS

Aim:

Write a c++ program illustrating to sort integer numbers.

Algorithm:

step1: start.

step2: Take an array X[10]

step3: Print ‘enter array size’

step4: read n value

step5: print 'enter elements in to array'

step6: for i=0 to n-1 insteps of 1 repeat step7

step7: read X[i]

[end for]

step8: call function insertion_sort(X,n)

step9: print 'After sorting, the Array elements are...'

step10: for i=0 to n-1 insteps of 1 repeat step11

step11: print X[i]

[end for]

step12: stop.

Algorithm for function insertion_sort(X,n)

Step1: for i=1 to n-1 insteps of 1 repeat step7

1.1: key=X[i] , pos=i.

1.2: repeat until pos>0 &&X[pos-1]>key

1.2.1:X[pos]=X[pos-1]

1.2.2: pos=pos-1

1.2.3: X[pos]=key;

[end while]

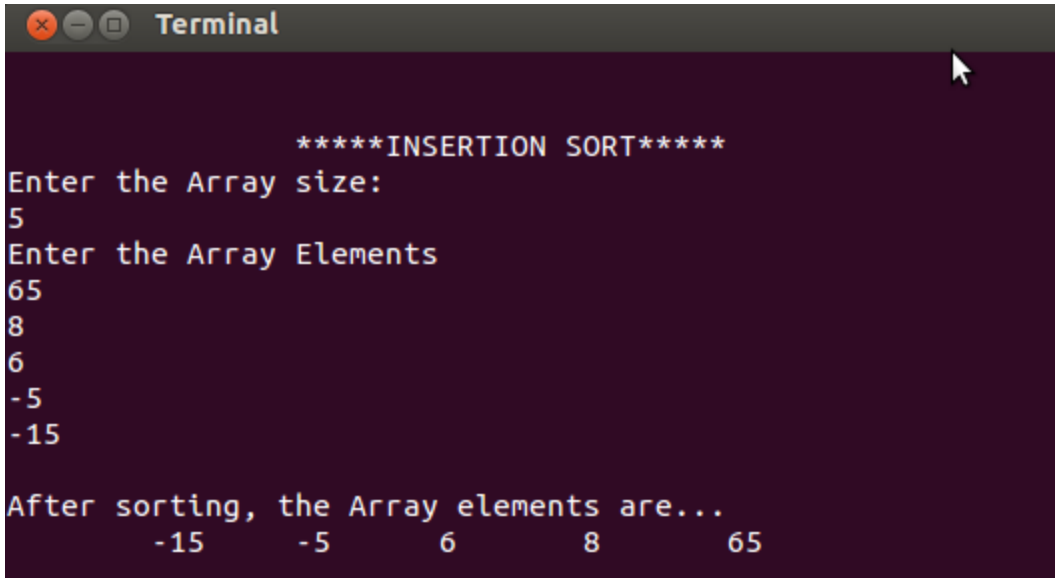
[end for]

Program:

```
#include <cstdlib>
#include <iostream>
#define MAX 20
using namespace std;
void insertion_sort(int X[], int n);
int main()
{
	int array[MAX], n;
	cout<<"\n\n\t\t*****INSERTION SORT*****"<<endl;
	cout<<"Enter the Array size:"<<endl;
	cin>>n;
	cout<<"Enter the Array Elements"<<endl;
	for(int i=0;i<n;i++)
		cin>>array[i];
	insertion_sort(array,n);
	cout<<"\nAfter sorting, the Array elements are...\n";
	for(int i=0;i<n;i++)
		cout<<"\t"<<array[i];
	cout<<endl;
	return 0;
}	//end of main
 void insertion_sort(int X[],int n)
{
	int key,i,pos;
	for(i=1;i<n;i++)
	{
		key=X[i];
		pos=i;
		while((pos>0)&&(X[pos-1]>key))
		{
			X[pos]=X[pos-1];
```

```
                    pos=pos-1;
                    X[pos]=key;
            }       //end of while loop
        }       //end of for loop
}       //end of insertion_sort.
```

Output:



Viva question:

1. **What is sorting?**

   Ans: arranging the data in an ascending to descending order or vice-versa.

2. **How many types of sorting are there?**

   Ans: quick, insertion, selection, merge, bubble etc..

3. **Which sorting is having minimum time complexity?**

   Ans: quick sort(n log n)

4. **What is the time complexity of bubble sort?**

   Ans: $O(n^2)$

5. **Which sorting is having maximum time complexity?**

   Ans: bubble

6. FACTORIAL USING RECURSION

Aim:

Write a C++ program illustrating factorial using recursion.

Algorithm:

step1: start

step2: print ‘enter a number’

step3: read n value

step4: call function factorial(n) and assign return value to k

step5: Print ' Factorial of ‘ n ‘ is ‘ k

step6: stop

Algorithm for function factorial (n)

step1: if n==0

then return 1.

step2: else

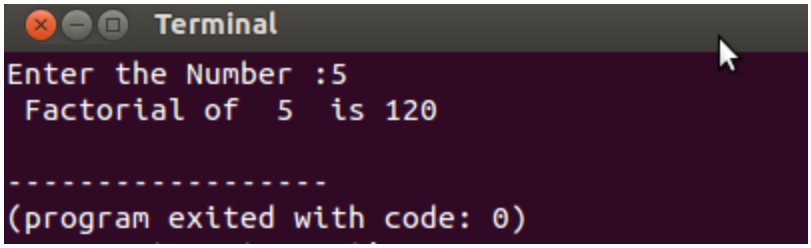
return(n * factorial(n-1))

Program:

```
#include<iostream>
using namespace std;
long factorial(int);            //Function declaration
int main()
{
    int n;
    long int k;                 // Variable Declaration
    cout<<"Enter the Number :";
    cin>>n;                     // Get Input Value
    k=factorial(n);             // Factorial Function Call
    cout<<" Factorial of  ”<<n<<”  is "<<k;
    return 0;
```

```
 }
                                        // Factorial Function using recursion
long int factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}
```

Output:



Viva questions:

1. **What is recursion?**

   Ans: a function calling itself again and again is called recursion.

2. **Give an example?**

   Ans: main(){ hi1(); }

   hi1(){ cout<<"hi" ; hi1(); }

   o/p: infinite times hi.

3. **Can main be recursive?**

   Ans:- yes

4. **Give an example.**

   Ans: main(){cout<<"hi"; main()}

5. **What is factorial of 0 and 1.**

   Ans: 1 for both

7. PASS BY VALUE, ADDRESS, AND REFERENCE

Aim:

Write a C++ program illustrating pass by value, pass by reference, pass by address.

Algorithm:

step1: start

step2: declare variables as num=5, ch

step3: print illustrating pass by value , pass by refarence & pass by address

step4: print enter your choice

step5: read ch value

step6: case 1: Code to be execute if <ch==1>

6.1: print the value of num before function call

6.2: call function passbyvalue(num)

6.3: print the value of num after the function call

step7: case 2: Code to be execute if<ch==2>

7.1: print the value of num and its address value before function call

7.2: call function passbyrefarence(num)

7.3: print the value of num after the function call

Step8: case 3: Code to be execute if<ch==3>

8.1: print the value of num and its address value before function call

8.2: call function passbyaddress(&num)

8.3: print the value of num after the function call

Step9: stop

Algorithm for function Passbyvalue(int num)

Step1: print the value of num in function

Step2: num=num+10

Step3: print the value of num in function

Algorithm for function PassbyReference(int& numRef)

Step1: print the value of numREF in function

Step2: print the address value of numREF in function

Step3: numREF=numREF+10

Step4: print the value of numREF in function

Algorithm for function PassbyAddress(int* ptr)

Step1: print the value of ptr in function

Step2: print the address value of ptr in function

Step3: print the value of variable pointer *ptr in function

Step4: *ptr=*ptr+10

Step5: print the value of variable pointer to by ptr now i.e *ptr in function

Program:

```
/*Program for illustrating Pass by value, Pass by reference, Pass by address          */
#include <iostream>
using namespace std;
void PassbyValue(int num);
void PassbyAddress(int* ptr);
void PassbyReference(int& numRef);
int main()
{
 int ch,num=5;
 cout<<"\n\n\tIllustrating Pass by value, Pass by reference, Pass by address\n\n";
 cout<<"*****Main Menu*****\n";
 cout<<"1.Pass by Value\n2.Pass by Reference\n3.Pass by Address\n";
 cout<<"Enter your choice:";
 cin>>ch;
 switch(ch)
 {
   case 1:
          cout << "In main(), value of num is " << num << endl << endl;
          PassbyValue(num);
          cout << "In main(), value of num is now " << num << endl << endl;
```

```
            break;
    case 2:
            cout << "In main(), num is " << num << endl;
            cout << "In main(), address of num is " << (long int) &num << endl << endl;
            PassbyReference(num);
            cout << "In main(), value of num is now " << num << endl << endl;
            break;
    case 3:
            cout << "In main(), value of num is " << num << endl;
            cout << "In main(), address of num is " << (long int) &num << endl << endl;
            PassbyAddress(&num);
            cout << "In main(), value of num is now " << num << endl << endl;
            break;
  }         //end of switch-case statement
  return 0;
}
void PassbyValue(int num)
{
      cout << "In PassbyValue(), value of num is " << num << endl << endl;
      num += 10;
      cout << "In PassbyValue(), value of num is now " << num << endl << endl;
}
void PassbyAddress(int* ptr)
{
      cout << "In addToInt(), value of ptr is " << (long int) ptr << endl;
      cout << "In addToInt(), address of ptr is " << (long int) &ptr << endl;
      cout << "In addToInt(), value of variable pointed to by ptr is " << *ptr << endl << endl;
      *ptr += 10;
      cout << "In addToInt(), value of variable pointed to by ptr is now " << *ptr << endl << endl;
}
void PassbyReference(int& numRef)
{
```

```
    cout << "In addToInt(), value of numRef is " << numRef << endl;
    cout << "In addToInt(), address of numRef is " << (long int) &numRef << endl;
    numRef += 10;
    cout << "In addToInt(), value of numRef is now " << numRef << endl << endl;
}
```

Output:

```
Terminal

        Illustrating Pass by value, Pass by reference, Pass by address

*****Main Menu*****
1.Pass by Value
2.Pass by Reference
3.Pass by Address
Enter your choice:1
In main(), value of num is 5

In PassbyValue(), value of num is 5

In PassbyValue(), value of num is now 15

In main(), value of num is now 5



------------------
(program exited with code: 0)
```

```
Terminal

        Illustrating Pass by value, Pass by reference, Pass by address

*****Main Menu*****
1.Pass by Value
2.Pass by Reference
3.Pass by Address
Enter your choice:2
In main(), num is 5
In main(), address of num is 3213549788

In addToInt(), value of numRef is 5
In addToInt(), address of numRef is 3213549788
In addToInt(), value of numRef is now 15

In main(), value of num is now 15
```

```
Terminal
        Illustrating Pass by value, Pass by reference, Pass by address

*****Main Menu*****
1.Pass by Value
2.Pass by Reference
3.Pass by Address
Enter your choice:3
In main(), value of num is 5
In main(), address of num is 3220836092

In addToInt(), value of ptr is 3220836092
In addToInt(), address of ptr is 3220836064
In addToInt(), value of variable pointed to by ptr is 5

In addToInt(), value of variable pointed to by ptr is now 15

In main(), value of num is now 15
```

Viva questions:

1. **What is pass by reference?**

   Ans: Pass by reference:

   The callee function receives a set of references which are aliases to variables. If a change is made to the reference variable, the original value (passed by the caller function) will also be

changed. All the references are handled by the pointers. Multiple values modification can be done by passing multiple variables.

2. **What is pass by value?**

   Ans: Pass by value:

   The callee function receives a set of values that are to be received by the parameters. All these copies of values have local scope, i.e., they can be accessed only by the callee function. The simplicity and guarantee of unchanging of values passed are the advantages of pass by value.

3. **What is pass by address?**

   Ans: Pass by address:

   The callee function receives a pointer to the variable. The value of the pointer in the caller function can then be modified. The advantages of this process are that the changes are passed back to the caller function and multiple variables can be changed.

8. FUNCTION OVERLOADING

Aim:

Write a C++ program illustrating function overloading.

Algorithm:

Step1: Declare long add(long, long);

Step2: Declare float add(float, float);

Step3: Call these functions separately in main

Step4: Define these functions separately.

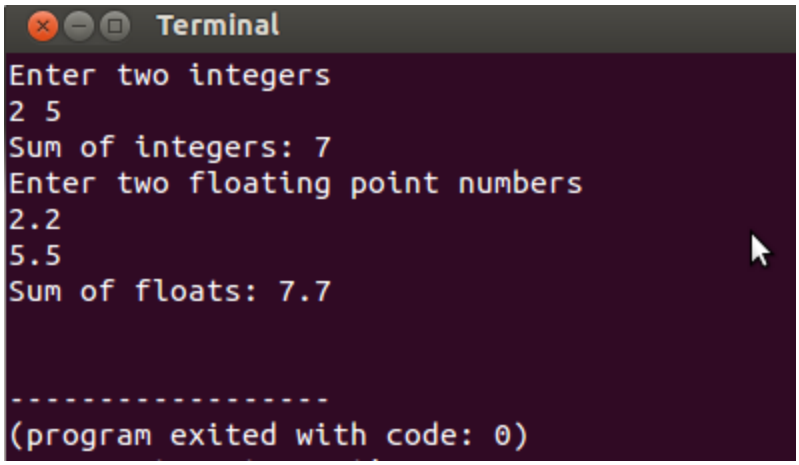
Step5: stop

Program:

```
#include <iostream>
using namespace std;
                    /* Function arguments are of different data type */
long add(long, long);
float add(float, float);
int main()
{
  long a, b, x;
  float c, d, y;
  cout << "Enter two integers\n";
  cin >> a >> b;
  x = add(a, b);
  cout << "Sum of integers: " << x << endl;
  cout << "Enter two floating point numbers\n";
  cin >> c >> d;
  y = add(c, d);
  cout << "Sum of floats: " << y << endl;
  return 0;
}
```

```
long add(long x, long y)
{
  long sum;
  sum = x + y;
  return sum;
}
float add(float x, float y)
{
  float sum;
  sum = x + y;
  return sum;
}
```

Output:



Viva questions:

1. **What is overloading?**

   Ans: In one function name more than one definition

2. **How many types of overloading are there?**

   Ans: Two

3. **What are they?**

   Ans: function overloading and operator overloading

4. **What is function overloading?**

   Ans: one function name more than one definition

5. **How many times a function can be overloaded?**

   Ans: no upper limit.

6. **What is the use of function overloading?**

   Ans: Function overloading is commonly used to create several functions of the same name that perform similar tasks but on different data types.

7. **How a compiler chooses an overloaded function?**

   Ans: While calling an overloaded function, the C++ compiler selects the proper function by examining the number, types and order of the arguments.

9. SWAPPING VALUES WITHOUT/ WITH USING FUNCTION OVERLOADING

Aim:

Write a C++ program illustrating an interactive program for swapping integer, real, and character type variables without using function overloading .Write the same program by using function overloading features and compare the same with its C counterpart.

Algorithm:

/*without using function overloading*/

Step1: start

Step2: Define three different functions, one for swapping integer, one for swapping char and the last one is to swap float nos.

void swap_char(char &x,char &y)

void swap_int(int &x,int &y)

void swap_float(float &x,float &y)

step3: Call these functions one by one in main

step4: Stop

/*with using function overloading*/

Step1: start

Step2: Define one function thrice with different arguments.

void swap(char &x,char &y);

void swap(int &x,int &y);

void swap(float &x,float &y);

step3: Call these functions one by one in main

step4: stop

Program:

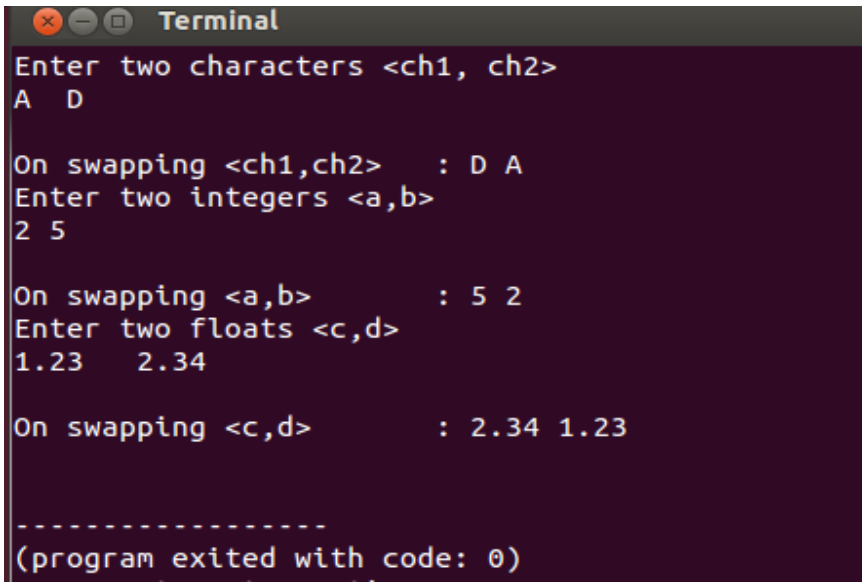
```
//      swapping values without using function overloading
#include<iostream>
using namespace std;
```

```
void swap_char(char &x,char &y)
{
 char temp;
 temp=x;
 x=y;
 y=temp;
}
void swap_int(int &x,int &y)
{
 int temp;
 temp=x;
 x=y;
 y=temp;
}
void swap_float(float &x,float &y)
{
 float temp;
 temp=x;
 x=y;
 y=temp;
}
int main(){
 char ch1,ch2;
 cout<<"Enter two characters <ch1, ch2>";
 cin>>ch1>>ch2;
 swap_char(ch1,ch2);
 cout<<"\nOn swapping <ch1,ch2>\t: "<<ch1<<" "<<ch2<<endl;
 int a,b;
 cout<<"Enter two integers <a,b>";
 cin>>a>>b;
 swap_int(a,b);
 cout<<"\nOn swapping <a,b>\t: "<<a<<" "<<b<<endl;
```

```
 float c,d;
 cout<<"Enter two floats <c,d>";
 cin>>c>>d;
 swap_float(c,d);
 cout<<"\nOn swapping <c,d>\t: "<<c<<" "<<d<<endl;
 return 0;
}
```

Output:

Without function overloading

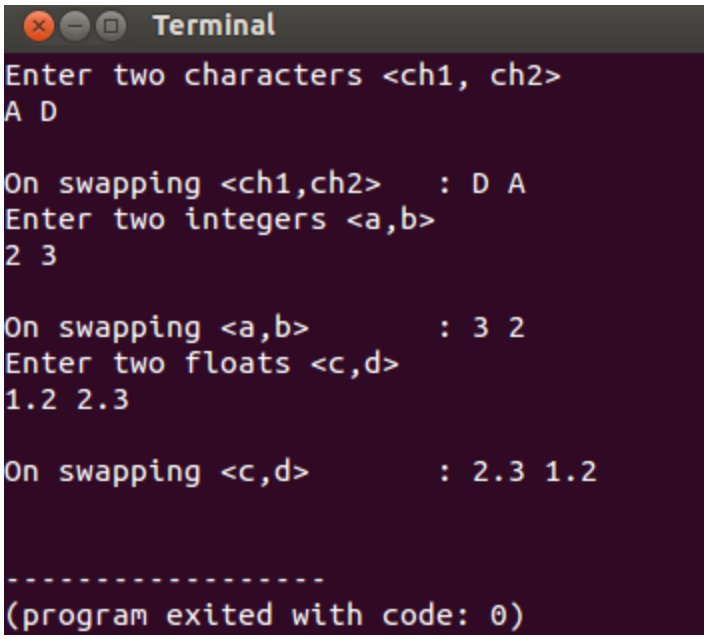


Program:

```
//        swapping values using function overloading
#include<iostream>
using namespace std;
void swap(char &x,char &y)
{
 char temp;
 temp=x;
 x=y;
 y=temp;
}
void swap(int &x,int &y)
{
 int temp;
 temp=x;
 x=y;
 y=temp;
}
void swap(float &x,float &y)
{
 float temp;
 temp=x;
 x=y;
 y=temp;
}
int main(){
 char ch1,ch2;
 cout<<"Enter two characters <ch1, ch2>";
 cin>>ch1>>ch2;
 swap(ch1,ch2);
 cout<<"\nOn swapping <ch1,ch2>\t: "<<ch1<<" "<<ch2<<endl;
 int a,b;
```

```
 cout<<"Enter two integers <a,b>";
 cin>>a>>b;
 swap(a,b);
 cout<<"\nOn swapping <a,b>\t: "<<a<<" "<<b<<endl;
 float c,d;
 cout<<"Enter two floats <c,d>";
 cin>>c>>d;
 swap(c,d);
 cout<<"\nOn swapping <c,d>\t: "<<c<<" "<<d<<endl;
 return 0;
}
```

Output:



Viva question:

1. **What is a function?**

   Ans: block of statement performing a specific task.

2. **What are the three steps to activate a user defined function?**

   Ans: declaration, call, definition

3. **What is an argument?**

   Ans: the value passed from called function to calling function.

4. **What is a return value?**

   Ans: the value returned from called function after execution, to calling function.

5. **How many types of arguments are there?**

   Ans: two types: actual (in function call) and formal(in function definition)

10. INLINE FUNCTION

Aim:

Write a C++ program illustrating inline functions.

Algorithm:

Step1: start

Step2: read a,b

Step3: call an inline function mul() to multiply and return the value of a*b

Step4: call an inline function div() to divide and return the value of a/b

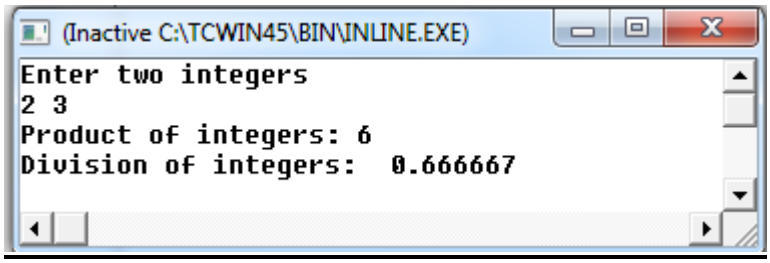
Step5: stop

Program:

```
#include <iostream>
using namespace std;
inline int mul(int x, int y)
{
	return x*y;
}
inline float div(int x, int y)
{
	return (float)x/y;
}
int main()
{
	int a, b, product;
	float division;
	cout << "Enter two integers\n";
	cin >> a >> b;
	product = mul(a, b);
	cout << "Product of integers: " << product << endl;
	division = div(a,b);
```

```
        cout << "Division of integers:  " << division << endl;
        return 0;
}
```

Output:



Viva questions:

1. **What are the tasks done when a function is called?**

   Ans: jumping to the function, saving registers, pushing arguments in stack, execution of code, returning to the calling function.

2. **Why and when a function should be inline?**

   Ans: when the block of code is very small then a lot of time is wasting in the extra tasks like: jumping to the function, saving registers, pushing arguments in stack, returning to the calling function. If we use inline then in place of function call the code will be replaced. So the above overheads can be stopped.

3. **What happens if we use macro in place of inline function?**

   Ans: both have the same functionality. But macro is not a function. So usual error checking is not done during compilation.

4. **What is the syntax to make a function inline.**

   Ans: **inline** return_type function_name(args)

   {

   body of the function

   }

5. **Give an example of inline function?**

   Ans:

```
inline int cube(int a)
{return a*a*a;}
```

11. FRIEND FUNCTION

Aim:

Write a C++ program illustrating Friend function.

Algorithm:

step1: Start

step2: define two classes having a friend function in both

step3: define it outside the classes as a general function accessing the members of both the classes.

step4: Call it in main function.

step5: stop

Program:

```
/*Program for illustrating FRIEND function */
#include <iostream>
using namespace std;
class two;
class one
{
        private:
                int a;
        public:
                void setdata(int x)
                {
                        a=x;
                }
                friend int add(one x,two y);
};
class two
{
        private:
                int b;
```

```
        public:
                void setdata(int y)
                {
                        b=y;
                }
                friend int add(one a, two b);
};
int add(one x,two y)
{
        int c;
        c=x.a+y.b;
        return c;
}
int main()
{
        one a;
        two b;
        a.setdata(5);
        b.setdata(10);
        cout<<"Sum of one and two:"<<add(a,b);
        return 0;
}
```

Output:

```
Terminal
Sum of one and two:15

------------------
(program exited with code: 0)
```

Viva questions:

1. **What is the basic difference between a general member function and a friend function?**

   Ans: a) It is defined outside the class as a general function.

   b) It can't be called with object name because it is not in the scope of any class.

2. **In the function definition can we write the member names directly as in the other member functions?**

   Ans: no. we have to use the object name and dot operator to access a file

3. **Where it should be declared, in public or private?**

   Ans: anywhere we can declare. There is no effect on the code.

   Give an example.

   Ans:

```
class one ;
class two
{       //body of the class
          friend void xyz(one,two);       };
class one{       //body of the class
                 friend void xyz(one,two);           };
void main()
{         …
          xyz(arg......);
          ...        }
```

4. **What is the forward declaration?**

   Ans:in above example the first line is: class one;

   Here class one is not defined. But we use it as an function argument in class two. So just to inform to compiler that "one" is a class name, we do this. This is called forward declaration.

12. EXCEPTION HANDLING

Aim:

Write a C++ program illustrating Exception Handling.

Algorithm:

Step1: Start

Step2: Divide a number by 0 within try block

Step3: Throw the exception

Step4: In catch block display a massage

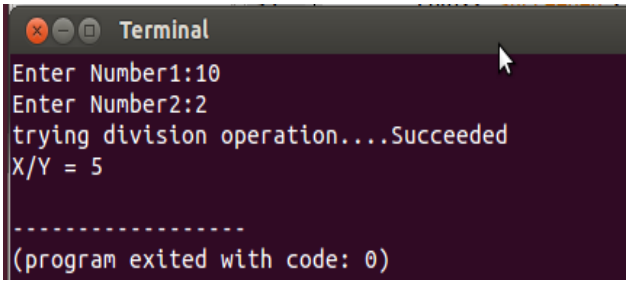
Step5: stop

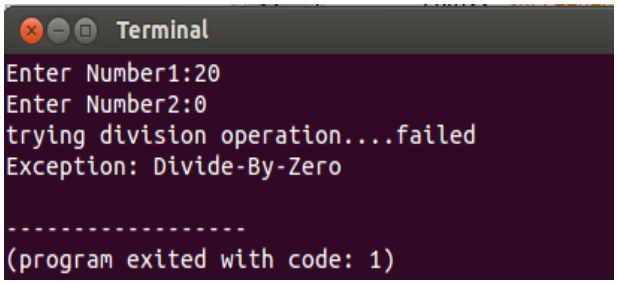
Program:

```
#include<iostream>
using namespace std;
class number
{
        private:
                float num;
        public:
                void read()
                {
                        cin>>num;
                }
                class DIVIDE{};         //Exception Class
                float div(number a)
                {
                        if(a.num==0)                    //check for zero divisor if yes
                                throw DIVIDE();                 //raise exception
                        else
                                return num/a.num;       //computes and returns the result
                }
```

```
};
int main()
{
        number x,y;
        float result;
        cout<<"Enter Number1:";
        x.read();
        cout<<"Enter Number2:";
        y.read();
                        //statements must be enclosed in try block if exception is to be raised
        try
        {
                cout<<"trying division operation....";
                result=x.div(y);
                cout<<"Succeeded"<<endl;
        }
        catch(number::DIVIDE)                //exception handler block
        {
                        //actions taken in response to exception
                cout<<"failed\n";
                cout<<"Exception: Divide-By-Zero";
                return 1;
        }
                //no exception, display result
        cout<<"X/Y = "<<result;
        return 0;
}
```

Output:





Viva questions:

1. **What is an exception?**

   Ans: exception is a run time unusual condition that a program may encounter during execution.

2. **Give some examples.**

   Ans: division by zero, accessing an aray outside its bound, running out of memory or disk space.

3. **What are the key words used in exception handling?**

   Ans: try, through, catch

4. **What is catch(...)**

   ans: when we dont know the type of exception then we use it.

5. **Why multiple catch statements are used?**

   Ans: to catch and handle multiple types of exceptions thrown by a try block.

6. **When should a function throw an exception?**

   Ans: A function should throw an exception when it is not able to fulfill its promise.

13. FUNCTION TEMPLATE

Aim:

Write a C++ program illustrating Function Template

Algorithm:

Step1: start

Step2: create a generic class T using template

Step3: set function arguments as the object reference of T

Step4: in main call the function once for integer, once for character and once for float.

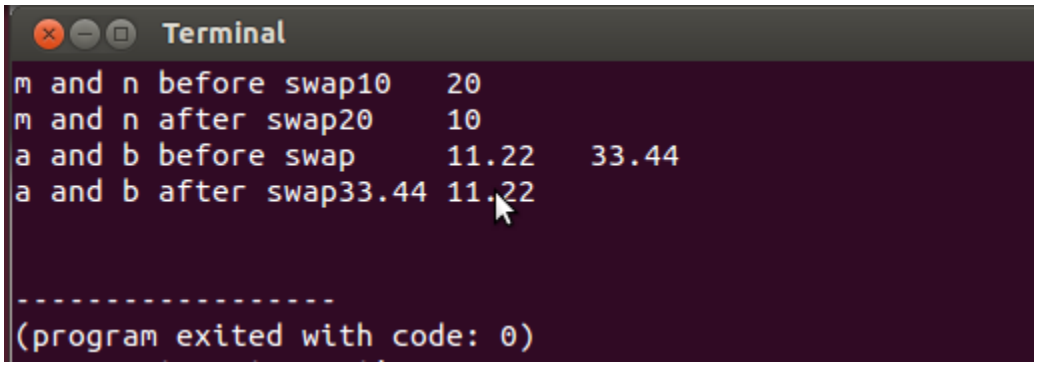
Step5: stop

Program:

```
#include<iostream>
using namespace std;
template <class T>
void swap_value(T &x,T &y)
{
	T t;
	t=x;
	x=y;
	y=t;
}
int main()
{
	int m=10,n=20;
	cout<<"m and n before swap"<<m<<"\t"<<n<<endl;
	swap_value(m,n);
	cout<<"m and n after swap"<<m<<"\t"<<n<<endl;
	float a=11.22, b=33.44;
	cout<<"a and b before swap\t"<<a<<"\t"<<b<<endl;
	swap_value(a,b);
```

```
    cout<<"a and b after swap"<<a<<"\t"<<b<<endl;
    return 0;
}
```

Output:



Viva questions:

1. **What is template?**

   Ans: It is a pre-defined class to create generic type object.

2. **What is generic data type?**

   Ans: object/variable of this data type can be assigned by any value from integer, character, float or double.

3. **What is the advantage of creating objects of generic data type?**

   Ans: objects of generic data type can be assigned by any value of any data type. Basically it is more useful when we pass argument of different data types to the same function. Instead of function overloading we can use it.

4. **What is the syntax to create a generic data type object?**

   Ans: template< class new_name>

   ex: template<class T>

   In the above example T is the generic data type.

5. **What is a function template?**

   Ans: a function whose arguments are of generic data type are called as function template

14. UNARY & BINARY OPERATOR OVERLOADING

Aim:

Write a C++ program illustrating Overloading increment, decrement binary+&<< operator.

Algorithm:

/*algorithm ++*/

Step1: start

Step2: increament all members by one

Step3: stop

/*algorithm --*/

Step4: start

Step5: decreament all members by one

Step7: stop

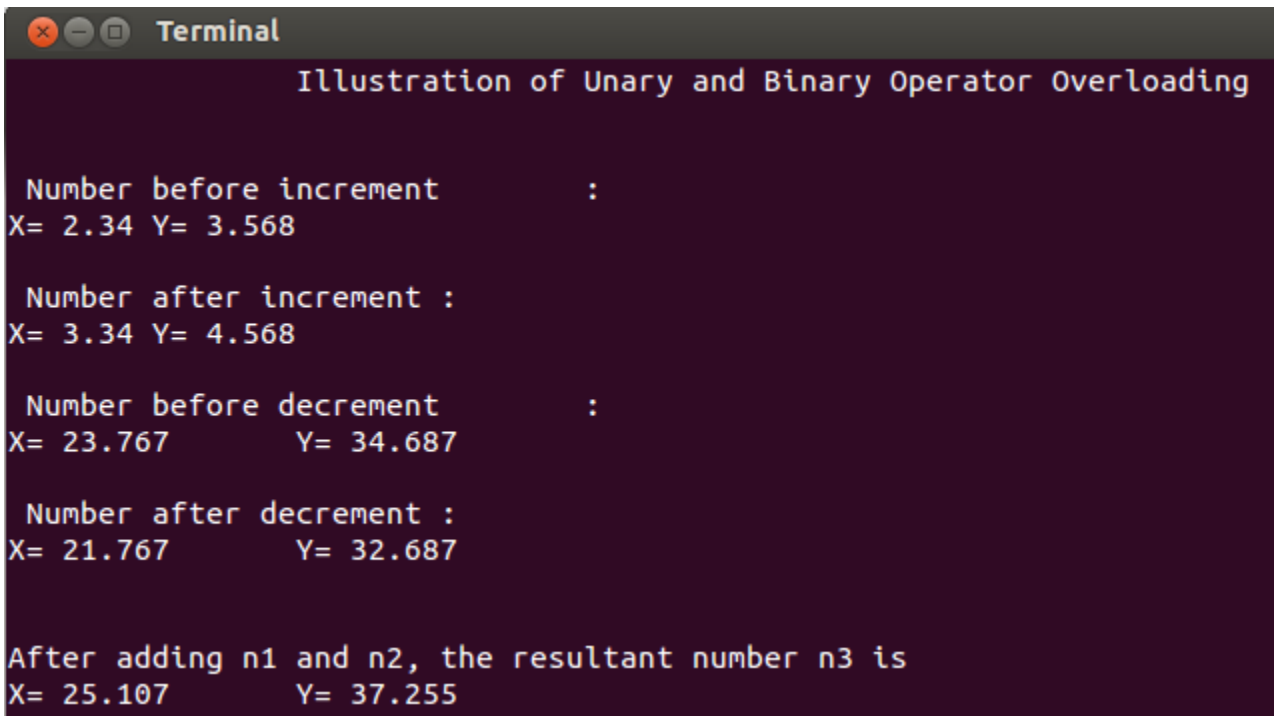
Program:

```
/*      Write a C++ program illustrating Overloading increment, decrement, binary+ operator.    */
#include<iostream>
using namespace std;
class number
{
        private:
                float x;
                float y;
        public:
                number()
                {       }
                number(float a, float b)
                {
                        x=a,y=b;
                }
                void operator ++(int)           //Post-increment
```

```
        {
                x++;
                y++;
        }
        void operator --()                      //Pre-decrement
        {
                --x;
                --y;
        }
        number operator+(number D)
        {
                number temp;
                temp.x=x+D.x;
                temp.y=y+D.y;
                return temp;
        }
        void show()
        {
                cout<<"\nX= "<<x;
                cout<<"\tY= "<<y<<endl;
        }
};
int main()
{
        number n1(2.34,3.568);
        number n2(23.767,34.687);
        number n3;
        cout<<"\t\tIllustration of Unary and Binary Operator Overloading\n\n";
        cout<<"\n Number before increment\t:";
        n1.show();
        n1++;
        cout<<"\n Number after increment\t:";
```

```
        n1.show();
        cout<<"\n Number before decrement\t:";
        n2.show();
        --n2;
        cout<<"\n Number after decrement\t:";
        --n2;
        n2.show();
        n3=n1+n2;
        cout<<"\n\nAfter adding n1 and n2, the resultant number n3 is ";
        n3.show();
        return 0;
}
```

Output:



Viva questions:

1. **What is operator overloading?**

   Ans: when one operator is overloaded to do some more task then it's usual task then it is called operator overloading.

2. **What is the syntax?**

   Ans:

```
return_type class_name::operator@(op-arg.list)
{
//function body
}
```

   where @ can be replaced by operators.

3. **Which operators cannot be overloaded?**

   Ans: ., *, ::, sizeof, ?:

4. **Can the gramatical rules of an operator changed using operator overlaoding?**

   Ans: no,it can't be. Number of operands, precedance, associetivity cant be changed.

5. **Is it mandatory to make it member function?**

   Ans: no. we have two options. Either we will make it member function or friend function.

15. VIRTUAL FUNCTION

Aim:

Write a C++ program illustrating Virtual function.

Algorithm:

Step 1: Start

Step 2: Declare the base class base.

Step 3: Declare and define the virtual function show().

Step 4: Declare and define the function display().

Step 5: Create the derived class from the base class.

Step 6: Declare and define the functions display() and show().

Step 7: Create the base class object and pointer variable.

Step 8: Call the functions display() and show() using the base class object and pointer.

Step 9: Create the derived class object and call the functions display() and show() using the derived class object and pointer.

Step 10: Stop

Program:

```
#include<iostream>
using namespace std;
class base
{
   public:
     virtual void show()
     {
             cout<<"\n  Base class show:";
     }
     void display()
     {
            cout<<"\n  Base class display:" ;
     }
};
```

```
class derived : public base
{
  public:
    void display()
    {
          cout<<"\n  Derived class display:";
    }
    void show()
    {
          cout<<"\n  Derived class show:";
    }
};
 int main()
{
  base obj1;
  base *p;
  cout<<"\n\t P points to base:\n"  ;
  p=&obj1;
  p->display();
  p->show();
  cout<<"\n\n\t P points to derived:\n";
  derived obj2;
  p=&obj2;
  p->display();
  p->show();
  return 0;
}
```

Output:

```
Terminal

        P points to base:

Base class display
Base class show

        P points to derived:

Base class display
Derived class show
```

Viva question:

1. **When virtual function is required?**

   Ans: if same member function is existing in both base and derived class then while executing compiler executes according to the type of pointer(base class pointer or derived class pointer). If we make the base class function as virtual function then the rather than the type of pointer, it will give priority to the object to which(base or derived) it points.

2. **Can a virtual function be static?**

   Ans: no.

3. **Can a virtual function be friend of another class?**

   Ans: yes

4. **Can we make constructors and destructors virtual?**

   Ans: we can't have virtual constructors. But we can have virtual destructors.

5. **Where pure virtual function is used?**

   Ans: if we need to make a base class as abstract class, then we have to use pure virtual class in that base class.

16. CLASS FOR COMPLEX NUMBER REPRESENTATION

Aim:

Write a C++ program illustrating an interactive program to process complex numbers. It has to perform addition, subtraction, multiplication, and division of complex numbers. Print results in x+iy form. Create a class for the complex number representation.

Algorithm:

Step1: create objects of class complex c1,c2,c3,c4,c5,c6

Step2: call getdata() for c1 and c2.

Step3: call add_complex(c1,c2) for c3

Step4: display c1, c2, c3

Step5: call sub_complex(c1,c2) for c4

Step6: display c1, c2, c4

Step7: call mul_complex(c1,c2) for c5

Step8: display c1, c2, c5

Step9: call div_complex(c1,c2) for c6

Step10: display c1, c2, c6

Step11:stop

Program:

```
#include<iostream>
using namespace std;
class complex
{
        float real;
        float imag;
    public:
        void getdata()
        {
        cout<<"Enter real and imaginary parts:\n";
        cin>>real>>imag;
```

```
}
void setdata(float r,float i)
{
real=r;
imag=i;
}
void display()
{
cout<<"\tReal= "<<real;
cout<<"\tImag= "<<imag<<endl;
}
void add_complex(complex c1, complex c2)
{
real=c1.real+c2.real;
imag=c1.imag+c2.imag;
}
void sub_complex(complex c1, complex c2)
{
real=c1.real-c2.real;
imag=c1.imag-c2.imag;
}
void mul_complex(complex c1, complex c2)
{
real=c1.real*c2.real-c1.imag*c2.imag;
imag=c1.real*c2.imag+c2.real*c1.imag;
}
void div_complex(complex c1, complex c2)
{
float temp,re,im;
temp=c2.real*c2.real+c2.imag*c2.imag;
re=c1.real*c2.real+c1.imag*c2.imag;
im=c1.real*c2.imag-c2.real*c1.imag;
```

```
                real=re/temp;
                imag=im/temp;
                }
};
int main()
{
        complex c1,c2,c3,c4;
        c1.getdata();
        c2.getdata();
        c3.add_complex(c1,c2);
        c1.display();
        c2.display();
        cout<<"\nAddition of two complex numbers is\n";
        c3.display();
        c4.sub_complex(c1,c2);
        cout<<"\nSubtraction of two complex numbers is\n";
        c4.display();
        complex c5;
        c5.mul_complex(c1,c2);
        cout<<"\nMultiplication of two complex numbers is\n";
        c5.display();
        complex c6;
        c6.div_complex(c1,c2);
        cout<<"\nDivision of two complex numbers is\n";
        c6.display();
        return 0;
}
```

Output:

```
Terminal
Enter real and imaginary parts:
2.1
3.5
Enter real and imaginary parts:
2.2
1.6
        Real= 2.1       Imag= 3.5
        Real= 2.2       Imag= 1.6

Addition of two complex numbers is
        Real= 4.3       Imag= 5.1

Subtraction of two complex numbers is
        Real= -0.1      Imag= 1.9

Multiplication of two complex numbers is
        Real= -0.98     Imag= 11.06

Division of two complex numbers is
        Real= 1.38108   Imag= -0.586487
```

Viva Questions:

1. **What is a class?**

   Ans: Class defines a datatype, it's type definition of category of thing(s). But a class actually does not define the data; it just specifies the structure of data.

2. **What is an Object/Instance?**

   Ans: Object is the instance of a class, which is concrete.

3. **What do you mean by C++ access specifiers?**

   Ans: Access specifiers are used to define how the members (functions and variables) can be accessed outside the class.

4. **How many classifiers are there in a class?**

   Ans: There are three access specifiers defined which are **public, private, and protected**

   - Public – Here the data members and functions are accessible outside the class.

   - Protected - Data members and functions are available to derived classes only.

   - Private - Data members and functions are not accessible outside the class.

5. **Differentiate between class and structure.**

   Ans:

   - The members of structures are public while those of a class are private.
   - Classes provide data hiding while structures don't.
   - Class bind both data as well as member functions while structures contain only data

6. **Class members are by default?**

   Ans: Private

7. **Explain the of scope resolution operator**

   Ans: A scope resolution operator (::) is used to define the member functions of a class outside the class. A scope resolution operator is required when a data member is redefined by a derived class or an overridden method of the derived class wants to call the base class version of the same method.

8. **What is the value of i?**

   Ans: i=sqrt(-1)

9. **Can we write the above program without using class?**

   Ans: yes.

10. **What is the formula to multiply two complex nos?**

    real=c1.real*c2.real-c1.imag*c2.imag;

    imag=c1.real*c2.imag+c2.real*c1.imag;

11. **What is the formula to divide two complex nos?**

    temp=c2.real*c2.real+c2.imag*c2.imag;

    re=c1.real*c2.real+c1.imag*c2.imag;

    im=c1.real*c2.imag-c2.real*c1.imag;

    real=re/temp;

    imag=im/temp;

12. **What are different concepts that we can implement for this program**

    Ans: operator overloading, class, friend function, template

17. STRING PROCESSING FUNCTIONS

Aim:

Write a C++ program illustrating user defined string processing functions using pointers (string length, string copy String concatenation)

Algorithm:

Step1: start

Step2: declare user defined string manipulating functions

```
int string_len(char *str);
void string_copy(char *s2,char *s1);
void string_concat(char *s1,char *s2);
int string_compare(char *s1,char *s2);
```

step3: define the said functions with argument as pointer to the string

step4: call these in main

step5: stop

Algorithm for string copy

Step1: declare two strings str1, str2

Step2: enter str1

Step3: read one character from str1

Step4: write it in str2

Step5: repeat step 3 and 4 till '\0'

Step6: display str2

Step7: stop

Algorithm for string concatenation

Step1: start

Step2: read two strings str1 str2

Step3: move the pointer of first string to the end of the string

Step4: read one character from str1

Step5: write it in str2

Step6: repeat step 4 and 5 till '\0'

Step7: display str1

Step8: stop

Algorithm for string comparison

Step1: start

Step2: enter two strings str1 and str2

Step3: read one character from each

Step4: compare. If equal, continue else display the difference

Step5: repeat step 3 and 4 till a difference or till one '\0' which is earlier

Step6: stop

Program:

```
#include<iostream>
using namespace std;
int string_len(char *str);
void string_copy(char *s2,char *s1);
void string_concat(char *s1,char *s2);
int string_compare(char *s1,char *s2);
int main()
{
 char temp[100],*s1,*s2,*s3,*s4;
 int len1,len2,flag=0;
 cout<<"Enter String1\t:";
 cin>>temp;
 len1=string_len(temp);
 s1=new char[len1+1];
 string_copy(s1,temp);
 cout<<"Enter String2\t:";
 cin>>temp;
 len2=string_len(temp);
 s2=new char[len2+1];
```

```
 string_copy(s2,temp);
  s3=new char[len1+len2+1];
 string_copy(s3,s1);
 string_concat(s3,s2);
  s4=new char[len1+1];
 string_copy(s4,s1);
 cout<<"\n\nUser-defined String Handling Functions\n\n";
 cout<<"String1\t:"<<s1;
 cout<<"\nString2\t:"<<s2;
 cout<<"\nLength of the String1\t:"<<len1;
 cout<<"\nAfter Concatenating S1 and S2, the String S3 is "<<s3;
 cout<<"\nAfter Copying S1 into S4, the String S4 is "<<s4;
 cout<<"\nAfter Comparing the String S1 and S2, \n\t\t\t";
 flag=string_compare(s1,s2);
 if(flag)
        cout<<"They are Equal";
 else
        cout<<"They are not Equal";
 return 0;
}
int string_len(char *str)
{
 int i=0;
 char *ptr=str;
 while(*ptr!='\0')
 {
        ptr++;
        i++;
 }
 return i;
}
void string_copy(char *str2,char *str1)
```

```
{
        char *s1=str1;
        char *s2=str2;
        while(*s1!='\0')
        {
                *s2=*s1;
                s2++;
                s1++;
        }
        *s2='\0';
}
void string_concat(char *str1,char *str2)
{
        char *s1=str1;
        char *s2=str2;
        while(*s1!='\0')        //move end of string
                s1++;
        while(*s2!='\0')        //append s2 to s1
        {
                *s1=*s2;
          s1++;
          s2++;
        }
        *s1='\0';
}
int string_compare(char *str1,char *str2)
{
 char *s1=str1,*s2=str2;
 while((*s1==*s2) && *s1!='\0' && *s2!='\0')
 {
        s1++;
        s2++;
```

```
 }
 if( *s1=='\0' && *s2=='\0')
        return 1;
 else
        return 0;
}
```

Output:

```
Terminal
Enter String1   :Object-Oriented
Enter String2   :Programming


User-defined String Handling Functions

String1 :Object-Oriented
String2 :Programming
Length of the String1   :15
After Concatenating S1 and S2, the String S3 is Object-OrientedProgramming
After Copying S1 into S4, the String S4 is Object-Oriented
After Comparing the String S1 and S2,
                        They are not Equal
```

Viva questions:

1. **What is a string?**

   Ans: string is a character array.

2. **What are the basic library functions to manipulate string?**

   Ans: strcat(), strcmp(), strlen() etc..

3. **What is the header file containing all the string library functions?**

   Ans: string.h

4. **int *x;char *y;**

   what is the size of x and y.

   Ans: size of x and y both are 2 bytes.

5. **What are the three steps to activate a user defined function?**

   Ans:declaration, definition, call

18. CONSTRUCTOR OVERLOADING

Aim:

Write a C++ program illustrating Constructor overloading (Both parameterized and default).

Algorithm:

Step1: start

Step2: declare a class.

Step3: Declare constructors of different arguments

Sep4: define constructors.

Step5: Create objects with different arguments as that of constructors.

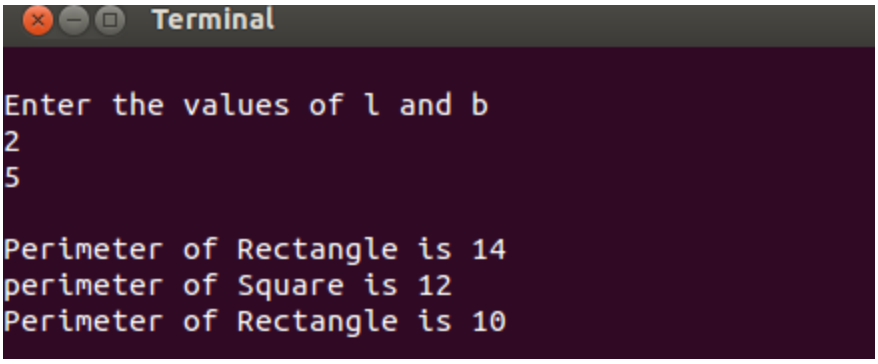
Step6: stop

Program:

```
/*      Program to illustrate Constructor Overloading (Both Parameterized and Default)   */
#include<iostream>
using namespace std;
class perimeter
{
        private:
                int l,b,peri;
        public:
                perimeter()             //default constructor
                {
                        cout<<"\nEnter the values of l and b";
                        cin>>l>>b;
                }
                perimeter(int a)        //Parameterized constructor with single parameter
                {
                        l=b=a;
                }
                perimeter(int l1, int b1)       //Parameterized constructor with two parameters
```

```
        {
                l=l1;
                b=b1;
        }
        void calculate()        //function to calculate the permeter
        {
                peri=2*(l+b);
                cout<<peri;
        }
};
int main()
{
        perimeter obj1, obj2(3), obj3(2,3);
        cout<<"\nPerimeter of Rectangle is ";
        obj1.calculate();
        cout<<"\nperimeter of Square is ";
        obj2.calculate();
        cout<<"\nPerimeter of Rectangle is ";
        obj3.calculate();
        return 0;
}
```

Output:



Viva Questions:

1. **What is a constructor?**

   Ans: A function that is called automatically when an object is created is called as constructor

2. **What is a default constructor?**

   Ans: A constructor that has no argument is a default constructor.

3. **What is the use of default constructor?**

   Ans:

   - It is a constructor that does not accept any parameters.

   - If there is no user-defined constructor for a class, the compiler declares a default parameterless constructor called default constructor. It is an inline public member of its class.

   When the compiler uses this constructor to create an object – the constructor will have no constructor initializer and a null body.

4. **Mention the ways in which parameterized can be invoked. Give an example of each.**

   Ans: Parameterized constructor can be invoked in the following ways:

   • Implicit Calling: By implicit calling, we mean that the constructor's name is not specified in the calling statement.

   General Form: class_name object_name(value1, value2, ...);

   Example:X o1(4,5);

   • Explicit Calling: By explicit calling, we mean that the constructor's name is specified in the calling statement.

   General Form: class_name object_name = constructor_name(value1, value2, ...);

   Example :X o1 = X(4,5);

5. **What are the restrictions apply to constructors and destructors?**

   Ans: The following restrictions apply to constructors and destructors

   Constructors and destructors don't return values.

   The addresses of constructors and destructors can't be taken so we can't use references and pointers on them.

   Constructors cannot be declared with the keyword virtual.

   Constructors and destructors cannot be declared static, const, or volatile.

19. COPY CONSTRUCTOR

Aim:

Write a C++ program illustrating Copy constructor.

Algorithm:

Step1: start

Step2: define a class with multiple constructors. One of them should have an argument of same class object reference.

Step3: In that constructor update the same class data member values as that of argument.

Step4: Create object in function

Step5: create another object with old object as an argument

Step6: display all the objects

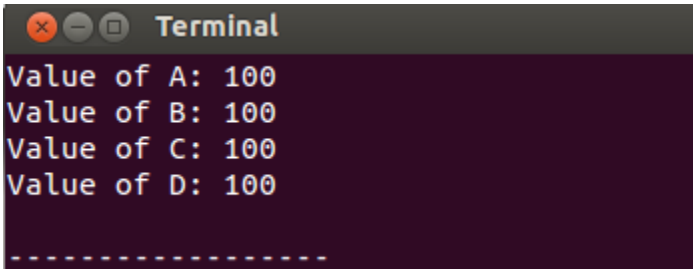
Step7: stop

Program:

```
/*      C++ program illustrating Copy constructor.  */
#include<iostream>
using namespace std;
class example
{
        private:
                        int val;
        public:
                example()
                {       }
                example(int a)
                {
                        val=a;
                }
                example(example &x)
                {
```

```
                val=x.val;
        }
        void display()
        {
                cout<<val;
        }
};
int main()
{
        example a(100);         //object is created and initialized
        example b(a);  //copy constructor is called
        example c=a;   //copy constructor is called
        example d;
        d=a;     //assigning
        cout<<"Value of A: ";
        a.display();
        cout<<"\nValue of B: ";
        b.display();
        cout<<"\nValue of C: ";
        c.display();
        cout<<"\nValue of D: ";
        d.display();
        return 0;
}
```

Output:



Viva questions:

1. **Explain Copy Constructor.**

   Ans: It is a constructor which initializes it's object member variable with another object of the same class. If you don't implement a copy constructor in your class, the compiler automatically does it.

2. **When do you call copy constructors?**

   Ans: Copy constructors are called in these situations:

   i.)when compiler generates a temporary object

   ii.)when a function returns an object of that class by value

   iii.)when the object of that class is passed by value as an argument to a function

   iv.)when you construct an object based on another object of the same class

3. **Name the implicit member functions of a class.**

   Ans: i). default constructor ii). copy constructor

   iii). assignment operator iv). default destructor

4. **Differentiate between a copy constructor and an overloaded assignment operator?**

   Ans: A copy constructor constructs a new object by using the content of the argument object while an overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

5. **In which case Copy Constructor is invoked?**

   Ans:

   - Creation and initialization of an object simultaneously.
   - When an object is passed to a function by value.
   - When an object is returned from a function by value.

6. **What are the various situations where a copy constructor is invoked?**

   Ans: Various situations where a copy constructor is invoked are as follows:

   • When an object is defined and initializes with the values of another object of the same type, then copy constructor is invoked.

   • When an object is passed by value method, then copy constructor is invoked to create the copy of the passed object for the function.

   • When a function returns an object, then copy constructor is invoked to create a temporary object to hold the return value in the memory.

20. ACCESSING DATA MEMBERS AND MEMBER FUNCTIONS USING THIS POINTER

Aim:

Write a C++ program illustrating access data members & member functions using "THIS' pointer.

Algorithm:

step1: start

step2: create a class with two member functions getdata(), setdata() display() and product().

step3: use this pointer in these functions

step4: create an object in main

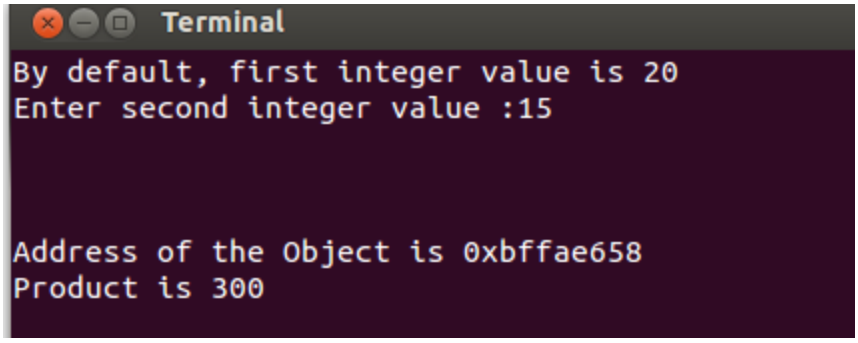
step5: call these functions through the object in main

Program:

```
#include<iostream>
using namespace std;
class Example
{
        private:
                        int a,b;

                        void getdata()
                        {
                                cout<<"Enter second integer value :";
                                cin>>a;
                        }
        public:
                void setdata()
                {
                this->b=20;  //accessing data members
                cout<<"By default, first integer value is 20\n";
                this->getdata();  //accessing member functions
```

```
                }
                void display()
                {
                        cout<<"\n\n\nAddress of the Object is "<<this<<endl;
        }
                int product()
                {
                        return(a*b);
                }
};
int main()
{
        Example A;
        A.setdata();
        A.display();
        cout<<"Product is "<<A.product();
        return 0;
}
```

Output:



Viva questions:

1. **What is "this" in c++?**

   Ans: "this" is a keyword in C++ used as a pointer to point the members of the object currently invoked.

2. **Give an example where this pointer is implicitly used? How?**

   Ans: operator overloading. We pass only one argument to the function, other one is implicitly passed using this pointer.

3. **Can we use it in return statement?**

   Ans: yes.

4. **What is the difference between this pointer and void pointer?**

   Ans: Void pointer can point to any data type variable where as this pointer points to currently invoked object.

5. **What is the use of this pointer?**

   Ans: This pointer points to an object. It can be explicitly used in a class and used to return an object.

6. **Which pointer is implicit pointer passed as the first argument for non-static member functions?**

   this pointer

7. **What is the type of "this" pointer? When does it get created?**

   Ans: It is a constant pointer type. It gets created when a non-static member function of a class is called.

8. **When do you use this pointer?**

   Ans: 'this pointer' is used as a pointer to the class object instance by the member function. The address of the class instance is passed as an implicit parameter to the member functions.

21. OVERLOADING UNARY OPERATORS

Aim:

Write a C++ program illustrating for overloading ++ operator to increment data.

Algorithm:

Step1: start

Step2: increment all the members by one

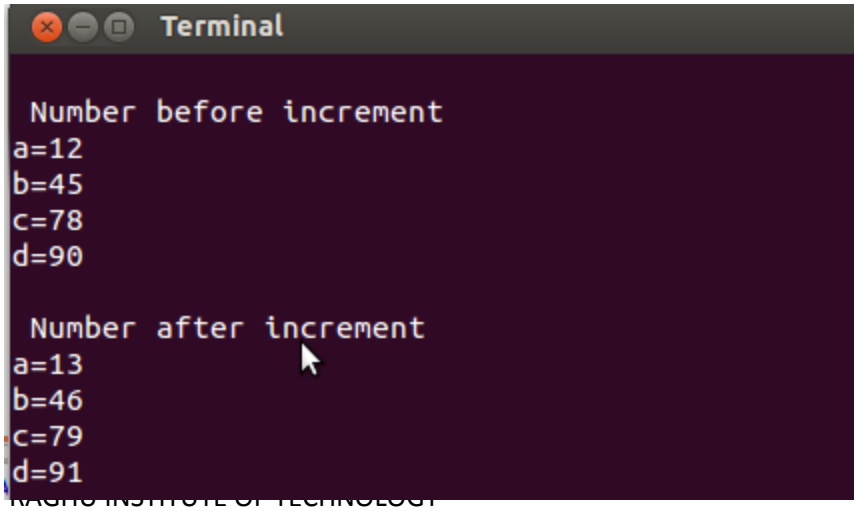
Step3: stop

Program:

```
/*      Write a C++ program illustrating for overloading ++ operator to increment data    */
#include<iostream>
using namespace std;
class number
{
        private:
                int a,b,c,d;
        public:
                number(int j,int k,int l,int m)
                {
                        a=j;
                        b=k;
                        c=l;
                        d=m;
                }
                void show(void);
                void operator ++();
};
void number::show()
{
        cout<<"a="<<a<<endl;
```

```
        cout<<"b="<<b<<endl;
        cout<<"c="<<c<<endl;
        cout<<"d="<<d<<endl;
}
void number:: operator ++()          //Post-increment
{
        ++a;
        ++b;
        ++c;
        ++d;
}
int main()
{
        number X(12,45,78,90);
        cout<<"\n Number before increment\n";
        X.show();
        ++X;
        cout<<"\n Number after increment\n";
        X.show();
        return 0;
}
```

Output:



Viva questions:

1. **What is operator overloading?**

   Ans: when one operator is overloaded to do some more task then it's usual task then it is called operator overloading.

2. **What is the syntax?**

   Ans:

   return_type class_name::operator@(op-arg.list)

   {

   //function body

   }

   where @ can be replaced by operators.

3. **Which operators can not be overloaded?**

   Ans: ., *, ::, sizeof, ?:

4. **Can the grammatical rules of an operator changed using operator overloading?**

   Ans: no,it can't be. Number of operands, precedence, and associativity can’t be changed.

5. **Is it mandatory to make it member function?**

   Ans: no. we have two options. Either we will make it member function or friend function.

22. ABSTRACT CLASSES

Aim:

Write a C++ program illustrating Abstract classes.

Algorithm:

Step1: start

Step2: declare a base class

Step3: make the base class abstract base class by declaring an pure virtual function inside it.

Step4: derive a class from base class

Step5: create object of derived class

Step6: try to create object of base class. It will be syntax wrong.

Step7: call display of derived class

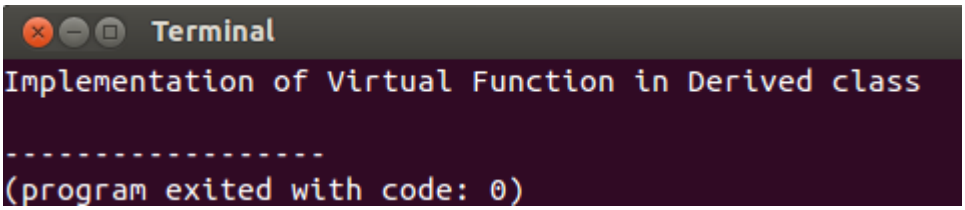
Step8: stop

Program:

```
#include<iostream>
using namespace std;
class Base          //Abstract base class
{
 public:
 virtual void show() = 0;            //Pure Virtual Function
};
void Base :: show()         //Pure Virtual definition
{
 cout << "Pure Virtual definition\n";
}
class Derived:public Base
{
 public:
 void show()
 { cout << "Implementation of Virtual Function in Derived class"; }
```

```
};
int main()
{
 Base *b;
 Derived d;
 b = &d;
 b->show();
}
```

Output:



Viva questions:

1. **What is an abstract class?**

   Ans: the base class which do not create object of its type is known as abstract class.

2. **To make a class abstract class forcibly what we need?**

   Ans: we have to place a pure virtual function in the base class.

3. **How to make a class abstract class.**

   Ans: to make a class abstract class, so that its object can't be created, we need to declare a pure virtual function in that class.

4. **What is the syntax of pure virtual function?**

   Ans: virtual return_type function_name()=0;

5. **Give an example.**

   Ans: virtual void abc()=0;

6. **Is it mandatory to make base class, abstract base class?**

   Ans: no.

23. INHERITANCE

Aim:

Write a C++ program illustrating Inheritance (Multiple, Multilevel Hybrid).

Algorithm:

step1: create base classes.

step2: create derived classes

step3: create the object of derived classes

step4: access the base class members through the object of derived classes

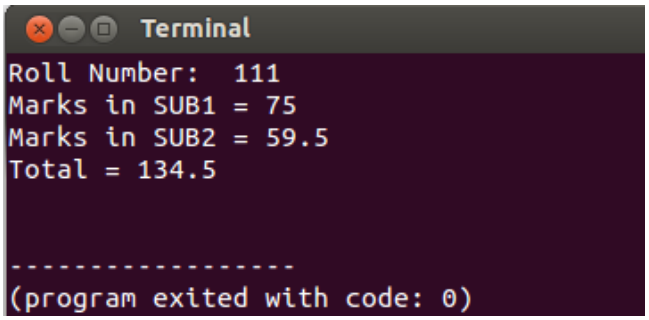
Program:

```
                /*          Multilevel inheritance          */
#include<iostream>
using namespace std;
class student
{
        protected:
                        int roll_number;
        public:
                void get_number(int);
                void put_number(void);
};
void student::get_number(int a)
{
        roll_number=a;
}
void student::put_number()
{
        cout<<"Roll Number:  "<<roll_number<<endl;
}
class test:public student
```

```
{
        protected:
                        float sub1,sub2;
        public:
                        void get_marks(float,float);
                        void put_marks(void);
};
void test::get_marks(float x,float y)
{
        sub1=x;
        sub2=y;
}
void test::put_marks()
{
        cout<<"Marks in SUB1 = "<<sub1<<"\n";
        cout<<"Marks in SUB2 = "<<sub2<<"\n";
}
class result:public test
{
                float total;
        public:
                void display(void);
};
void result::display(void)
{
        total=sub1+sub2;
        put_number();
        put_marks();
        cout<<"Total = "<<total<<"\n";
}
int main()
{
```

```
        result student1;
        student1.get_number(111);
        student1.get_marks(75.0,59.5);
        student1.display();
        return 0;
}
```

Output:

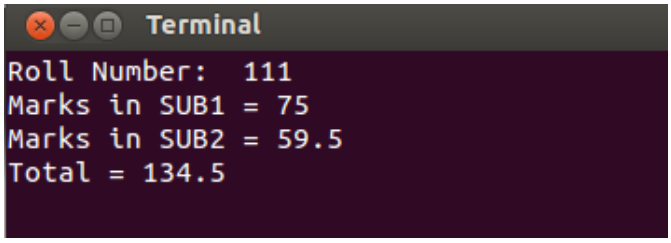


/* Multiple Inheritance */

```
#include<iostream>
using namespace std;
class M
{
	protected:
			int m;
	public:
		void get_m(int);
};
class N
{
	protected:
			int n;
	public:
		void get_n(int);
};
class P:public M,public N
{
	public:
		void display(void);
};
void M::get_m(int x)
{
	m=x;
}
void N::get_n(int y)
{
	n=y;
}
void P::display(void)
{
```

```
        cout<<"m = "<<m<<"\n";
        cout<<"n = "<<n<<"\n";
        cout<<"m*n = "<<m*n<<"\n";
}
int main()
{
        P p;
        p.get_m(10);
        p.get_n(20);
        p.display();
        return 0;
}
```

Output:



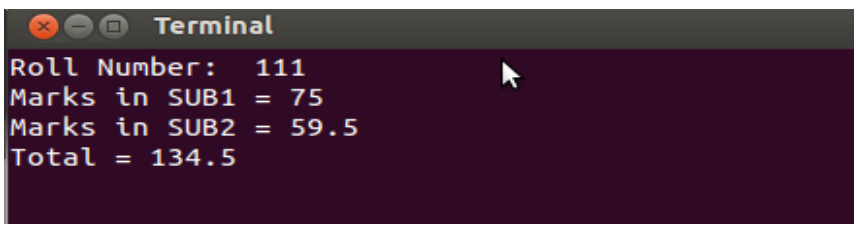
/* Hybrid inheritance */

```
#include<iostream>
using namespace std;
class student
{
	protected:
			int roll_number;
	public:
		void get_number(int);
		void put_number(void);
};
void student::get_number(int a)
{
	roll_number=a;
}
void student::put_number()
{
	cout<<"Roll Number:  "<<roll_number<<endl;
}
class test:public student
{
	protected:
			float part1,part2;
	public:
			void get_marks(float,float);
			void put_marks(void);
};
void test::get_marks(float x,float y)
{
	part1=x;
	part2=y;
}
```

```
void test::put_marks()
{
	cout<<"MArks obtained: "<<"\n";
	cout<<"Part1 = "<<part1<<"\n";
	cout<<"Part2 = "<<part2<<"\n";
}
class sports
{
	protected:
			float score;
	public:
		void get_score(float s)
		{
			score=s;
		}
		void put_score(void)
		{
			cout<<"Sports Wt: "<<score<<"\n\n";
		}
};
class result:public test,public sports
{
		float total;
	public:
		void display(void);
};
void result::display(void)
{
	total=part1+part2+score;
	put_number();
	put_marks();
	put_score();
```

```
        cout<<"Total score = "<<total<<"\n";
}
int main()
{
        result student1;
        student1.get_number(1234);
        student1.get_marks(27.5,33.0);
        student1.get_score(6.0);
        student1.display();
        return 0;
}
```

Output:



Viva questions:

1. **What is an inheritance?**

   Ans: Inheritance is the process by which -Object of one class acquires the properties of objects of another class

2. **List the advantages of inheritance?**

   - Inheritance permits code reusability.

   - Reusability saves time in program development.

3. **What are the different types of Inheritance?**

   - Single Inheritance

     A (parent class) -> B (child class)

   - Multiple Inheritance

     A -> C, B -> C

- Hierarchical inheritance

  A -> B, A -> C, A -> D
- Multilevel inheritance

  A -> B, B -> C
- Hybrid inheritance

  A -> B, A -> C, B -> D, C -> D.

4. **What is multiple inheritance?**

   Ans: When a class is derived from another class ie it inherits functionalities of another class, this phenomenon is known as inheritance. In some cases, a class can inherit from multiple classes, ie a derived class can have multiple base classes, it is known as multiple inheritance.
5. **What is private, public and protected Inheritance?**

   Ans:

   private inheritance: all the public and protected members in base become private.

   protected inheritance: all the public and protected members in base class become protected.

   public inheritance: in case of public inheritance, public remains public and protected remains protected..

24. VIRTUAL CLASSES

Aim:

Write a C++ program illustrating Virtual classes & virtual functions

Algorithm:

Step1: start

Step2: declare a base class A

Step3: derive B1 and B2 from A virtually

Step4: derive C from B1 and B2

Step5: create object of C

Step6: call the member functions of all the classes.

Step7: stop

Program:

```
/*      Write a C++ program illustrating Virtual function.   */
#include<iostream>
using namespace std;
class A
{
        public:
                        int a;
                        void get_a()
                        {
                                cout<<"enter an integer value:";
                                cin>>a;
                        }
};
class B1 : public virtual A
{
        public:
                        int b1;
```

```
                void get_b1()
                {
                        cout<<"Enter an integer value:";
                        cin>>b1;
                }
};
class B2 : public virtual A
{
        public:
                int b2;
                void get_b2()
                {
                        cout<<"Enter an integer value:";
                        cin>>b2;
                }
};
class C : public B1, public B2
{
        public:
                void display()
                {
                        cout<<"Entered Values are "<<a<<" "<<b1<<" "<<b2;
                        cout<<"\nSum of above integers is "<<a+b1+b2;
                }
};
int main()
{
        C o1;
        o1.get_a();
        o1.get_b1();
        o1.get_b2();
        o1.display();
```

```
        return 0;
}
```

Output:

```
Terminal
enter an integer value:2
Enter an integer value:3
Enter an integer value:4
Entered Values are 2 3 4
Sum of above integers is 9
```

Viva questions:

1. **When it is necessary to make the base class virtual?**

   Ans: multipath inheritance

2. **Why?**

   Ans: to avoid data duplicacy.

3. **What happens when a class is made virtual base class?**

   Ans: C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.

4. **Is it mandatory to make the base class virtual in case of multipath inheritance?**

   Ans: according to syntax it is NO. but to get the correct output, it is YES

5. **Can we use "virtual" and "public" keywords in either order?**

   Ans : yes

25. OVERLOADING FUNCTION TEMPLATE

Aim:

Write a C++ program illustrating overloading function template

Algorithm:

Step1: create a generic class

step2: write a function which argument is the object of the generic class

step3: again define the same function which argument is the object of the generic class with more no. of arguments.

step4: call these functions in main for different arguments.

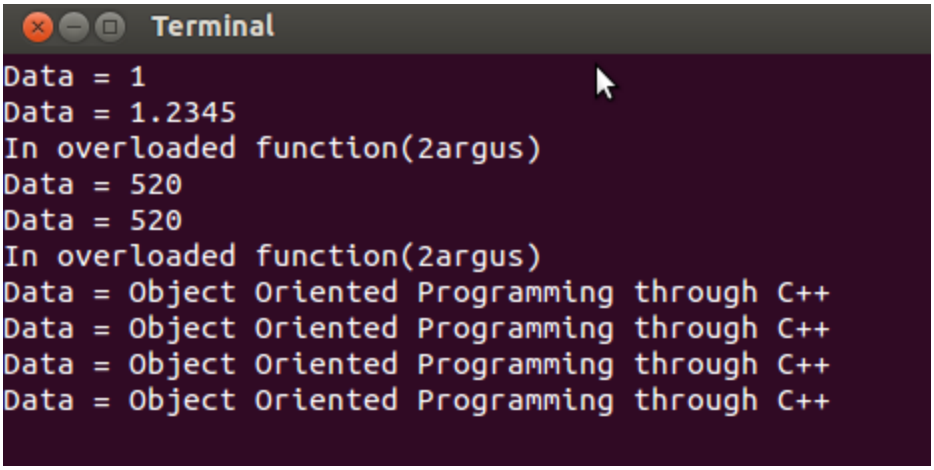
step5: stop

Program:

```
#include<iostream>
using namespace std;
template<class T>
void print(T data)
{
	cout<<"Data = "<<data<<endl;
}
template<class T>
void print(T data,int nTimes)
{
	cout<<"In overloaded function(2argus)\n";
	for(int i=1;i<=nTimes;i++)
		cout<<"Data = "<<data<<"\n";
}
int main()
{
	print(1);
	print(1.2345);
```

```
    print(520,2);
    print("Object Oriented Programming through C++",4);
    return 0;
}
```

Output:



Viva questions:

1. **What is overloading template? Explain it with an example?**

   Ans: A template function overloads itself as needed. But we can explicitly overload it too. Overloading a function template means having different sets of function templates which differ in their parameter list.

2. **What is a function template?**

   Ans: A function having generic data type arguments.

3. **What is its syntax?**

   Ans:

   template<class T>

   return_type function name(args Of type T)

   {

   //body of the function with type T where ever appropriate

   }

4. **Give an example.**

   Ans:

```
template<class T>
void swap(T &x,, T &y)
{        T temp= x; x=y; y= temp;      }
```

5. **Can a template function overloaded. If so what is the priority?**

   Ans: yes. A template function can be overloaded with a function having built-in arguments. If in function call argument is that of the built-in argument then that function will be called, else template function will be called.

6. **Give an example?**

   Ans:

```
template<class T>
void xyz(T x)
{
//body  of the function  }void xyz(int x){//body  of the unction
}
void main()
{
xyz(10);
xyz(10.15);
}
```

In the first call of xyz(), the xyz(int x) will be called.

In the second call of xyz(), the xyz(T x) will be called.

26 CLASS TEMPLATE

Aim:

Write a C++ program illustrating Class template

Algorithm:

Step1: define 2 generic classes T1 and T2

Step2: define a class where the object of the generic classes will be the data members and arguments of member functions

Step3: create object of this class in main.

Step4: call the member functions through the object.

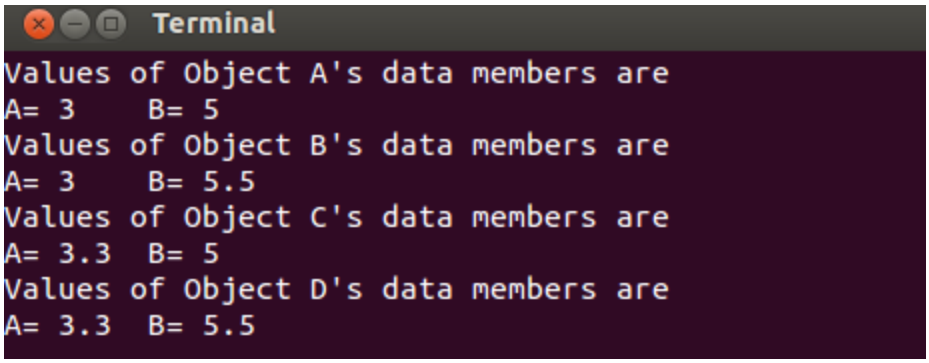
Step5: stop

Program:

```
#include<iostream>
using namespace std;
template<class T1,class T2>
class ex
{
 T1 a;
 T2 b;
 public:
 ex(T1 x,T2 y)
 {
 a=x;
 b=y;
 }
 void show()
 {
  cout<<"A= "<<a<<"\t";
  cout<<"B= "<<b<<endl;
 }
```

```
};
int main()
{
 ex <int,int> A(3,5);
 cout<<"Values of Object A's data members are\n";
 A.show();
 ex <int,float> B(3,5.5);
 cout<<"Values of Object B's data members are\n";
 B.show();
 ex <float,int> C(3.3,5);
  cout<<"Values of Object C's data members are\n";
 C.show();
 ex <float,float> D(3.3,5.5);
 cout<<"Values of Object D's data members are\n";
 D.show();
 return 0;
}
```

Output:



Viva question:

1. **What is a class template?**

   Ans: A class having generic type member.

2. **Are templates conceptually related to polymorphism?**

   Ans: Yes, but compile-time polymorphism

3. **What is meant by template parameter?**

   Ans: A template parameter is a special kind of parameter that can be used to pass any data type variable as argument.

4. **Which parameter is legal for non-type template?**

   Ans: The following are legal for non-type template parameters: integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

5. **What is a template class?**

   Ans: a class created from a class template is called template class. Suppose XYZ is a class template. When we declare an object as:

   XYZ <int> obj;

   then it is called template class used for integer variable.

Additional programs:

Aim:

Write a c++ program illustrating the concepts of static data members.

Algorithm:

Step1: start

Step2: declare a class item and place normal data member and a static data member count .

Step3: define the functions getdata() and getcount() in the public part of the class.

Step4: create the objects a,b,c of class item.

Step5: Invoke the function getcount() by using all the objects defined.

Step6: invoke the function getdata() by using all the objects defined.

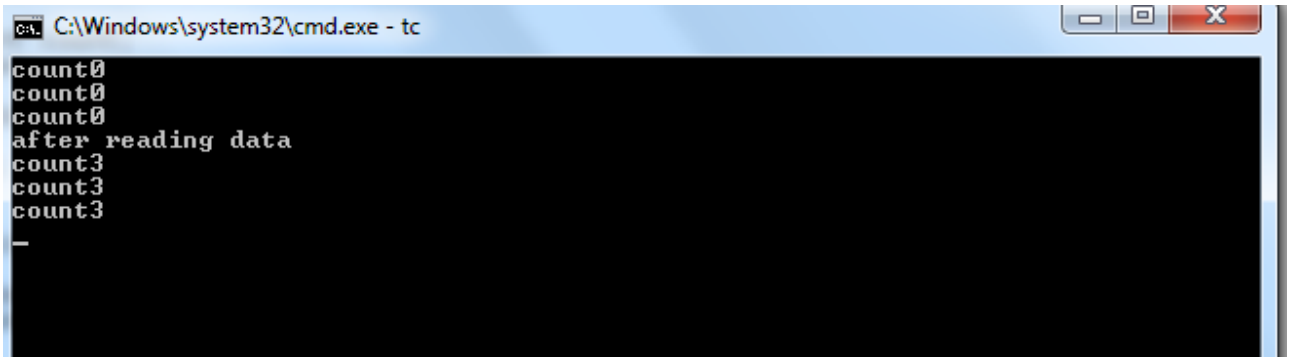
Step7: stop

Program:

```
#include<iostream.h>
Class item
{
static int count;
int number;
public:
void getdata(int a)
{
number=a;
count++;
}
void getcount(void)
{
```

```
cout<<"count";
cout<<count<<"\n";
}
};
int item :: count;
int main()
{
item a,b,c;
a.getcount();
b.getcount();
c.getcount();
a.getdata(100);
b.getdata(200);
c.getdata(300);
cout<<"after reading data"<<"\n";
a.getcount();
b.getcount();
c.getcount();
return 0;
}
```

Output:



Viva question:

1. What is the visibility mode of a static data member?

   It is visible only with in the class, but its lifetime is the entire program.

2. What is the other name for the static variables?

   Static variables are also known as class variables.

3. What is the important constraint for a static data member?

   It is initialized to zero when the first object of its class is created.

Aim:

Write a c++ program to illustrate the concept of manipulators.

Algorithm:

Step1: start

Step2: declare the variables basic allowance and total with some values.

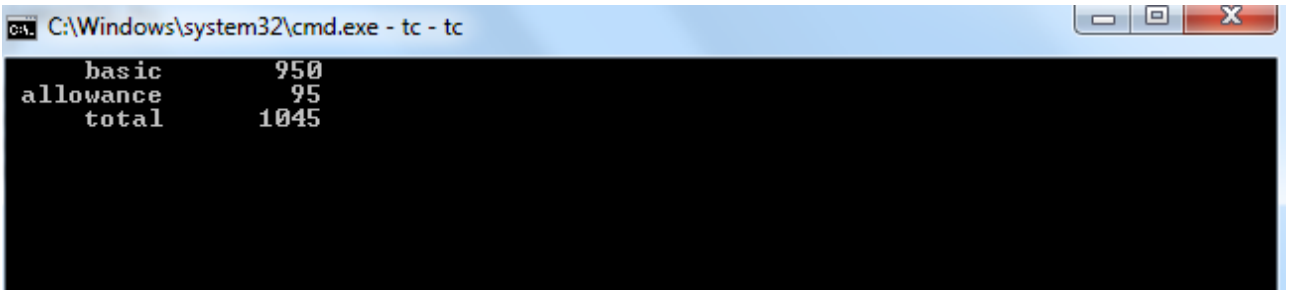
Step3: use the predefined function setw() for allocating the required space for the variables and their values.

Step4: stop

Program:

```
#include<iostream.h>

#include<conio.h>

#include<iomanip.h>

int main()

{

int basic=950,allowance=95,total=1045;

cout<<setw(10)<<"basic"<<setw(10)<<basic<<endl

    <<setw(10)<<"allowance"<<setw(10)<<allowance<<endl

    <<setw(10)<<"total"<<setw(10)<<total<<endl;

return 0;

}
```

Output:



<u>Viva question:</u>

1) What are manipulators?

   Manipulators are the operators that are used to format the data display.

2) List some examples of manipulators?

   Setw() ,endl, setprecision are some of the examples.

3) How the manipulators work with respect to the character strings?

   All the character strings are printed right justified with respect to the manipulators.

Aim:

Write a c++ program to illustrate the concept of pointers.

Algorithm:

Step1: start

Step2: declare a normal variable, pointer variable and a double pointer variable.

Step3: Assign the address of a to ptr1;

Step4: Assign the address of ptr1 to ptr2;

Step5: increment the locations of ptr1 and ptr2 with two address locations and then print the result.

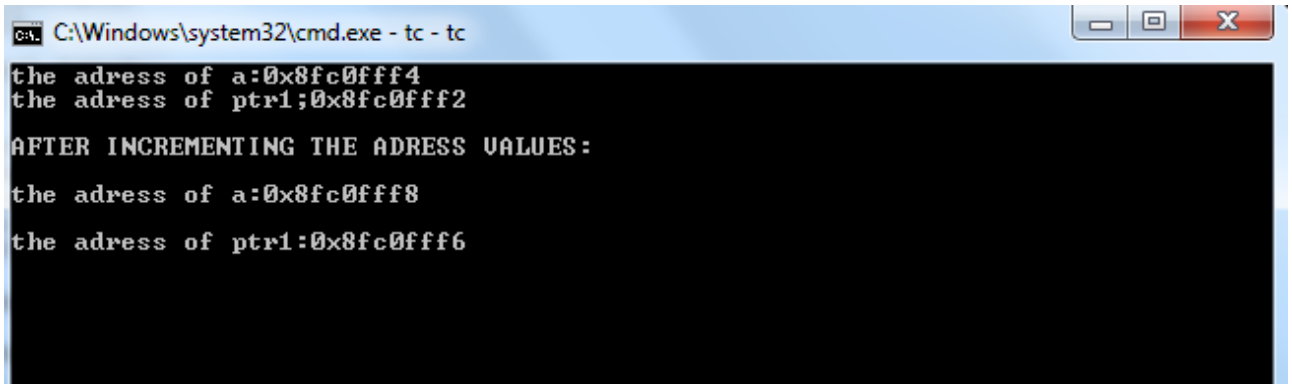
Step6: stop

Program:

```
#include<iostream.h>

#include<conio.h>

int main()

{

int a,*ptr1,*ptr2;

clrscr();

ptr1=&a;

ptr2=&ptr1;

cout<<"the adress of a:"<<ptr1<<"\n";

cout<<"the adress of ptr1;"<<ptr2;

cout<<"\n\n";

cout<<"AFTER INCREMENTING THE ADRESS VALUES:"\N\N";

ptr1+=2;

cout<<"the adress of a:"<<ptr1<<"\n\n";

ptr2+=2;
```

```
cout<<"the adress of ptr1:"<<ptr2<<"\n\n";

return 0;

}
```

Output:



<u>Viva question:</u>

1) What is a pointer?

   Pointer is a variable which holds the address of another variable.

2) What is the syntax for a pointer variable?

   Data_type *pointer_variable

3) What is the difference between the address(&) and asterisk(*) in pointers?

   Address (&) operator gives the address location of a normal variable or a pointer variable whereas the asterisk(*) gives the value at that location.