Ex. No:1(a)	PRINTING GROSS PAY OF EMPLOYEE

AIM:

To write a c++ program to calculate Gross pay of an employee.

ALGORITHM:

Step 1: Start the program.
Step 2: In the main function, declare the variables.
Step 3: Input the Basic pay, DA, HVA, PF.
Step 4: Calculate Gross pay.
Step 5: Stop the program.

PROGRAM:

```
#include<iostream>
using namespace std;
int main()
{
int gross_pay,basic_pay,da,hra,pf,DA,HRA,PF;
cout<<"Enter the basic salary:";
cin>>basic_pay;
cout<<"Enter the DA percentage :";
cin>>da;
cout<<"Enter the HRA percentage :";
cin>>hra;
cout<<"Enter the PF percentage :";
cin>>pf;
DA=(basic_pay/100)*da;
HRA=(basic_pay/100)*hra;
PF=(basic_pay/100)*pf;
gross_pay=basic_pay+DA+HRA-PF;
cout<<"Your Gross pay salary is :"<<gross_pay;
return 0;
}
```

OUTPUT:

```
Enter the basic salary:70000
Enter the DA percentage :7
Enter the HRA percentage :8
Enter the PF percentage :12
Your Gross pay salary is :72100
```

RESULT:

Thus, the above c++ program was executed and output was verified successfully.

Ex No:1(b)	PRINTING BIODTA OF STUDENT

AIM:

To write a c++ program to print biodata of an individual.

ALGORITHM:

Step 1: Start the program.
Step 2: Include appropriate header file.
Step 3: In the main method, declare the variables.
Step 4: Input the Name, Date of Birth, Gender, City, College, Branch, Department, Phone number from the user.
Step 5: Print the output.
Step 6: Stop the program.

PROGRAM:

```
#include<iostream>
using namespace std;
int main()
{
char name[30],gender[10],college[50],dept[50],branch[50],city[30],dob[10];
char ph_no[10];
cout<<"Enter the name:";
cin>>name;
cout<<"Enter your Gender :";
cin>>gender;
cout<<"Enter your Date of birth : ";
cin>>dob;
cout<<"Your phone_no :";
cin>>ph_no;
cout<<"Enter your city : ";
cin>>city;
cout<<"Enter the collegename:";
cin>>college;
cout<<"Your Branch: ";
cin>>branch;
cout<<"Your department :";
cin>>dept;
cout<<"******************";
cout<<"\n Name : "<<name;
cout<<"\n Date of Birth : "<<dob;
cout<<"\n Gender : "<<gender;
cout<<"\n City : "<<city;
cout<<"\n Phone number : "<<ph_no;
cout<<"\n College : "<<college;
cout<<"\n Branch : "<<branch;
cout<<"\n Department : "<<dept;
```

```
return 0;
}
```

OUTPUT:

```
Enter the name:V.S.Harshini
Enter your Gender :Female
Enter your Date of birth : 21/09/2003
Your phone_no :9597783109
Enter your city : Coimbatore
Enter the collegename:SREC
Your Branch: B.Tech
Your department :IT
*****************
 Name :
 Date of Birth : 21/09/20039597783109
 Gender : Female
 City : Coimbatore
 Phone number : 9597783109
 College : SREC
 Branch : B.Tech
 Department : IT
```

RESULT:

Thus, the above c++ program was executed and output was verified successfully.

Ex. No:2(a)	CREATING A CLASS

AIM:

To write a C++ program to display the account details of a customer.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Create a class and define it.
Step 3**:** Later define the main method and create an object for the class.
Step 4: Using the created object, call the member functions in the main method.
Step 5: Display the output.
Step 6: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Account
{
   public:
   string account_Num;
   int balance;
   Account(string accNum,int accBalance)
   {
      account_Num=accNum;
      balance=accBalance;
   }
   string getAccountNumber()
   {
      return account_Num;
   }
   int getBalance()
   {
      return balance;
   }
   int credit(int amount)
   {
      balance+=amount;
   }
   int debit(int amount)
   {
      balance-=amount;
   }
};
int main()
{
   string accNum;
```

```
    int accBalance;
    cout<<"Enter account number:"<<endl;
    cin >> accNum;
    cout<<"Enter account balance:"<<endl;
    cin>>accBalance;
    Account account(accNum,accBalance);
    int txnsCount,txnAmount;
    cout<<"Enter the number of transactions:"<<endl;
    cin >> txnsCount;
    for(int ctr=1; ctr <= txnsCount; ctr++){
       cout<<"Enter the transaction amount:"<<endl;
       cin >> txnAmount;
       if(txnAmount >= 0){
          account.credit(txnAmount);
       }else{
          account.debit(-txnAmount);
       }
    }
    cout << account.getAccountNumber() << " balance is "  << account.getBalance();
    return 0;
}
```

OUTPUT:

```
Enter account number:
1044101223064
Enter account balance:
1000
Enter the number of transactions:
2
Enter the transaction amount:
100
Enter the transaction amount:
-98
1044101223064 balance is 1002
```

RESULT:

Thus, the above c++ program was executed and output was verified successfully.

Ex. No:2(b)	CREATING MEMBER FUNCTION

AIM:

To write a c++ program to create a member function for a class.

ALGORITHM:

Step 1: Start the program.
Step 2: Include the appropriate header file.
Step 3: Create a class named Rectangle and declare the function Area with return type.
Step 4: Define Area outside the function using space resolution operator and return area.
Step 5: Input the length and breadth of the rectangle from the user.
Step 6: In the main method, call the function using class object.
Step 7: Print the output.
Step 8: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class space{
   public:
   int a,b, ans;
   int area(int a, int b);
}vs;
int space::area(int a, int b){
   ans=a*b;
   return ans;
}
int main(){
   cout<<"Enter the length : ";
   cin>>vs.a;
   cout<<"Enter the height : ";
   cin>>vs.b;
   cout<<"Area of rectangle is : "<<vs.area(vs.a,vs.b);
   return 0;
}
```

OUTPUT:

```
Enter the length : 5
Enter the height : 5
Area of rectangle is : 25
```

RESULT:

Thus, the c++ program to print area of rectangle using class member function was executed and output was verified successfully.

Ex. No:2(c)	ARRAY AS AN OBJECT

AIM:

To write a c++ program to access a class using array of objects.

ALGORITHM:

Step 1: Start the program.
Step 2: Include the appropriate header file.
Step 3: Create a class named Hobby.
Step 4: Declare and Define a function to get the hobby of the user.
Step 5: In the main method, create an array of objects for the class.
Step 6: Call the functions using each object.
Step 7: Print the output.
Step 8: Stop the program.

PROGRAM:

```
#include <iostream>
#include <string>
using namespace std;
class Hobby{
   string name, interest;
   public:
   void getdetail(){
      cout<<"Enter your name :"<<endl;
      cin>>name;
      cout<<"Enter your hobby of interest : "<<endl;
      cin>>interest;
   }
   void display(){
      cout<<name<<" is having a interest on doing "<<interest<<"."<<endl;
   }
};
int main()
{
   Hobby vs[10];
   int n;
   cout<<"Enter number of candidates : "<<endl;
   cin>>n;
   for(int i=0;i<n;i++){
      vs[i].getdetail();
   }
   for(int i=0; i<n;i++){
      vs[i].display();
```

```
    }
    return 0;
}
```

OUTPUT:

```
Enter number of candidates :
3
Enter your name :
padmasri
Enter your hobby of interest :
artworks
Enter your name :
harshini
Enter your hobby of interest :
reading
Enter your name :
sowmiya
Enter your hobby of interest :
cooking
padmasri is having a interest on doing artworks.
harshini is having a interest on doing reading.
sowmiya is having a interest on doing cooking.
```

RESULT:

Thus, the c++ program to print hobbies of candidate using array of objects was executed and output was verified successfully.

Ex. No:2(d)	OBJECT AS FUNCTION ARGUMENT AND RETURNING OBJECTS

AIM:

To write a c++ program to demonstrate passing object as function argument and return the object.

ALGORITHM:

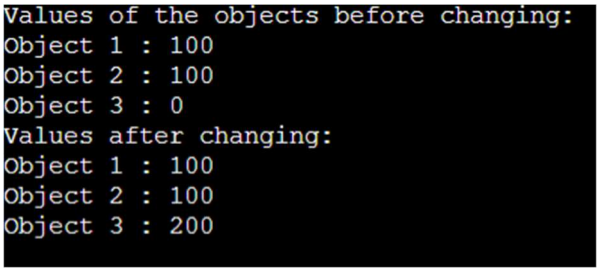
Step 1: Start the program.
Step 2: Create a class and declare the variables inside the public access specifier.
Step 3: Create a function with class objects as arguments.
Step 4: Return the appropriate object.
Step 5: Inside the main method, declare the objects for the created class.
Step 6: Assign values to the class variables.
Step 7: Call the function with object.
Step 8: Print the output.
Step 9: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class addition{
   public:
   int a;
   addition add(addition x, addition y){
      addition z;
      z.a=x.a+y.a;
      return z;
   }
};
int main()
{
 addition r,s,t;
 r.a=100;
 s.a=100;
 t.a=0;
 cout<<"Values of the objects before changing:"<<endl;
 cout<<"Object 1 : "<<r.a<<endl;
 cout<<"Object 2 : "<<s.a<<endl;
 cout<<"Object 3 : "<<t.a<<endl;
 t=t.add(s,r);
 cout<<"Values after changing:"<<endl;
 cout<<"Object 1 : "<<r.a<<endl;
 cout<<"Object 2 : "<<s.a<<endl;
```

```
  cout<<"Object 3 : "<<t.a<<endl;
  return 0;
}
```

OUTPUT:



RESULT:

Thus, the above program to pass objects as function arguments and returning objects is executed and output was verified successfully.

Ex. No:3a	CONSTRUCTOR

AIM:

To write a c++ program to demonstrate the concept of constructor.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a class named sample and declare the variables inside the public specifier.
Step 3: Create constructor using the same name as class and initialize the variables.
Step 4: Create a function to display the result.
Step 5: Inside the main method, call the function using class object
Step 6: Print the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Matrix{
   public:
   int row,col;
   int value[100][100];
   Matrix(int ro, int co){
      row=ro;
      col=co;
   }
   void storeValue(int val,int ro, int co){
     value[ro][co]=val;
   }
   void printTranspose(){
      for(int i=0;i<col;i++){
         for(int j=0;j<row;j++){
            cout<<value[j][i]<<" ";
         }
         cout<<endl;
      }
   }
};
int main()
{
   int R,C,currValue;
   cout<<"ENTER NUMBER OF ROWS AND COLUMNS IN THE MATRIX:"<<endl;
   cin >> R >> C;
   Matrix matrix(R,C);
   cout<<"ENTER ELEMENTS OF THE MATRIX:"<<endl;
   for(int row=0; row<R; row++)
   {
      for(int col=0; col<C; col++)
```

```
        {
            cin >> currValue;
            matrix.storeValue(currValue,row,col);
        }
    }
    cout<<"TRANSPOSE OF THE MATRIX:"<<endl;
    matrix.printTranspose();
    return 0;
}
```

OUTPUT:

```
ENTER NUMBER OF ROWS AND COLUMNS IN THE MATRIX:
3 3
ENTER ELEMENTS OF THE MATRIX:
1 2 3
4 5 6
7 8 9
TRANSPOSE OF THE MATRIX:
1 4 7
2 5 8
3 6 9
```

RESULT:

Thus, the c++ program demonstrate constructor was executed and output was verified successfully.

Ex. No:3b	DESTRUCTOR

AIM:

To write a c++ proram to demonstrate the concept of Destructor.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a class named sample and declare the variables inside the public specifier.
Step 3: Create constructor using the same name as class and initialize the variables.
Step 4: Create a function to display the result.
Step 5: Create a destructor using ~.
Step 6: Inside the main method, create an object for the class.
Step 7: Call the function using the object.
Step 8: Print the output.
Step 9: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Matrix{
   public:
   int row,col,ir;
   int value[100][100];
   Matrix(int ro, int co){
      row=ro;
      col=co;
   }
   void storeValue(int val, int ro, int co){
      value[ro][co]=val;
   }
   void printRow(int rows){
      ir=rows;
      for(int i=0;i<col;i++){
         cout<<value[ir-1][i]<<" ";
      }
   }
   ~Matrix(){
      cout<<"MatrixDestructor:LastPrintedRow="<<ir;
   }
};
int main()
{
   int R,C,currValue;
   cout<<"ENTER THE ROWS AND COLUMN OF THE MATRIX"<<endl;
   cin >> R >> C;
   Matrix matrix(R,C);
```

```
    cout<<"ENTER THE ELEMENTS OF THE MATRIX"<<endl;
    for(int row=0; row<R; row++)
    {
       for(int col=0; col<C; col++)
       {
          cin >> currValue;
          matrix.storeValue(currValue,row,col);
       }
    }
    int rowToPrint;
    cout<<"ENTER THE ROW TO PRINT"<<endl;
    cin >> rowToPrint;
    matrix.printRow(rowToPrint);
    cout << endl;
    return 0;
}
```

OUTPUT:

```
ENTER THE ROWS AND COLUMN OF THE MATRIX
3 3
ENTER THE ELEMENTS OF THE MATRIX
1 2 3
4 5 6
7 8 9
ENTER THE ROW TO PRINT
2
4 5 6
MatrixDestructor:LastPrintedRow=2
```

RESULT:

Thus, the c++ program to demonstrate the concept of Destructor was executed and output was verified successfully.

Ex. No:4(a)	FRIEND FUNCTION

AIM:

To write c++ program to implement friend function.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a class named mini and declare variables inside public specifier.
Step 3: Initialize the variables and declare friend function using friend keyword within the class.
Step 4: Define the friend function outside the class.
Step 5: In the main method, create object for the class and call appropriate function.
Step 6: Print the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class mini{
   public:
   int a, b;
   void func(){
     a=10;
     b=40;
   }
   friend int num(mini vs);
};
int num(mini vs){
   if(vs.a<vs.b){
      return vs.a;
   }
   else{
      return vs.b;
   }
}
int main()
{
   mini s;
   s.func();
   cout<<"The smallest number is "<<num(s);
   return 0;
}
```

OUTPUT:

```
The smallest number is 10
```

RESULT:

Thus, the above program to demonstrate the concept of friend function was executed and verified successfully.

Ex. No:4(b)	FRIEND CLASS

AIM:

To write a c++ program to execute friend class.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a class named mom and initialize the member variable within constructor.
Step 3: Declare friend class daughter inside the mom class.
Step 4: Create an object for the mom class and access the mom variables.
Step 5: In the main method, call the function.
Step 6: Print the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class daughter;
class mom{
   private:
   string quality;
   friend class daughter;
   public:
   mom(){
      quality="classical dancer";
   }
 };
class daughter{
      private:
      string talent;
      public:
      daughter(){
         talent="poeter";
      }
      void display(){
         mom s;
         cout<<"Mom is a "<<s.quality<<". Daughter is a "<<talent<<"."<<endl;
      }
}vs;

int main()
{
   //daughter vs;
   cout<<"Both got talent "<<endl;
   vs.display();
   return 0;
}
```

OUTPUT:

```
Both got talent
Mom is a classical dancer. Daughter is a poeter.
```

RESULT:

Thus, the c++ program to implement friend class was executed and output was verified successfully.

Ex. No:5(a)	FUNCTION OVERLOADING

AIM:

To write ac++ program to implement function overloading.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a class and declare the variables inside the public specifier.
Step 3: Declare functions with same name and different arguments.
Step 4: Define each function.
Step 5: In the main method, create object for the class.
Step 6: Call each function with appropriate arguments.
Step 7: Print the output.
Step 8: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class function{
   public:
   void disp(int r){
      cout<<"the roll no is :"<<r<<endl;
   }
   void disp(string roll){
      cout<<"the roll no is :"<<roll<<endl;
   }
};
int main()
{
   cout<<"This program is demonstrating Function Overloading!!"<<endl;
   function vs;
   vs.disp(39);
   vs.disp("thirty nine");

   return 0;
}
```

OUTPUT:

```
This program is demonstrating Function Overloading!!
the roll no is :39
the roll no is :thirty nine
```

RESULT:

Thus, the c++ program to demonstrate function overloading was executed and output was verified successfully.

Ex. No:5(b)	UNARY OPERATOR OVERLOADING

AIM:

To write a C++ program using overloading unary operator.

ALGORITHM:

Step 1**:** Start the program.
Step 2: Create a class.
Step 3**:** Create a constructor to initialize the variables.
Step 4**:** Define the method to overload any unary operator.
Step 5**:** In the main method, create an object for the class.
Step 6**:** Display the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Overloading
{
private:
int value;
public:
Overloading()
{
value=1000;
}
void operator ++ ()
{
++value;
}
void operator ++ (int)
{
value++;
}
void display()
{
cout << "Count: " << value << endl;
}
};
int main()
{
Overloading count1;
count1++;
cout<<"POST INCREMENT\n";
```

```
count1.display();
++count1;
cout<<"PRE INCREMENT\n";
count1.display()
return 0;
}
```

OUTPUT:

```
POST INCREMENT
Count: 1001
PRE INCREMENT
Count: 1002
```

RESULT:

Thus, the c++ program to overload unary operator was executed and output was verified successfully.

Ex. No:5(c)	BINARY OPERATOR OVERLOADING

AIM:

To write a C++ program to demonstrate binary operator overloading.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare a class named caterpillar. Inside the class declare a variable. Initialize the variable through constructor.
Step 3: Declare binary operator and perform required action.
Step 4: Inside the main method, Call the binary operator using third object
Step 5: Print the output.
Step 6: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class caterpilar{
   public:
   int move;
   caterpilar(){
      move= 0;
   }
   caterpilar(int m){
      move=m;
   }
   caterpilar operator+(caterpilar& vs2){
      caterpilar vs3;
      vs3.move=move+vs2.move;
      return vs3;
   }
};
int main()
{
  caterpilar vs1(10);
  caterpilar vs2(10);
  caterpilar vs3;
  vs3=vs1+vs2;
  cout<<"The caterpilar has moved :"<<vs3.move << " distance forwardly in the
morning."<<endl;
  return 0;
}
```

OUTPUT:

```
The caterpilar has moved :20 distance forwardly in the morning.
```

RESULT:

Thus, the c++ program to implement binary operator overloading was executed and output was verified successfully.

Ex. No:6(a)	POINTER OPERATIONS

AIM:

To write a C++ program to demonstrate the concept of pointers.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Inside the main method, input a value from the user.
Step 3**:** Declare a pointer to store the address of the value.
Step 4**:** Print the value.
Step 5**:** Print the address of the value.
Step 6**:** Display the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
int main()
{
   int value;
   int* a;
   cout<<"ENTER A VALUE:";
   cin>>value;
   a = &value;
   cout << "VALUE:" << value << endl;
   cout << "ADDRESS OF VALUE(&value):" << &value<< endl;
   cout << "ADDRESS OF VALUE USING POINTER(*a):" << a << endl;
   cout << "CONTENT OF THE ADDRESS POINTED BY THE POINTER(*a):" << *a<<
endl;
   return 0;
}
```

OUTPUT:

```
ENTER A VALUE:21
VALUE:21
ADDRESS OF VALUE(&value):0x7ffc809d906c
ADDRESS OF VALUE USING POINTER(*vs):0x7ffc809d906c
CONTENT OF THE ADDRESS POINTED BY THE POINTER(*vs):21
```

RESULT:

Thus, the c++ program to demonstrate the concept of pointers was executed and output was verified successfully.

Ex. No:6(b)	PASSING POINTERS TO FUNCTIONS

AIM:

To write a C++ program to demonstrate the concept of passing pointers to functions.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare a method and pass pointers as the parameters.
Step 3: Inside the main method, input values from the user.
Step 4: Print values before swapping.
Step 5: Call the method by passing address of the values as arguments.
Step 6: Print values after swapping.
Step 7: Define the method to swap two numbers.
Step 8: Display the output.
Step 9: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
void swap(int& x, int& y)
{
	int z = x;
	x = y;
	y = z;
}

int main()
{
	int a = 45, b = 35;
	cout << "Before Swap\n";
	cout << "a = " << a << " b = " << b << "\n";

	swap(a, b);

	cout << "After Swap with pass by reference\n";
	cout << "a = " << a << " b = " << b << "\n";
}
```

OUTPUT:

```
Before Swap
a = 45 b = 35
After Swap with pass by reference
a = 35 b = 45
```

RESULT:

Thus, the c++ program to implement pointers as function arguments was executed and output was verified successfully.

Ex. No:6(c)	PASSING AN ENTIRE ARRAY TO A FUNCTION

AIM:

To write a C++ program to demonstrate the concept of passing an entire array to a function.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Inside the main method, create an array.
Step 3**:** Declare a pointer to store the address of elements in the array.
Step 4**:** Display the addresses using array.
Step 5**:** Display the addresses using pointers.
Step 6: Display the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class pointers{
   public:
   void fun(int* arr){
      int i;
      unsigned int n = sizeof(arr) / sizeof(arr[0]);
      for (i = 0; i < n; i++)
            cout << " " << arr[i];
}
}vs;
int main()
{
      int arr[] = { 1, 2 };
      vs.fun(arr);
      return 0;
}
```

OUTPUT:

```
This program function takes array of pointers as arguments.
 1 2
```

RESULT:

Thus, the c++ program to pass array of pointers as function arguments was executed and output was verified successfully.

Ex. No:7(a)	SINGLE INHERITANCE

AIM:

To write a C++ program to demonstrate single inheritance concept.

ALGORITHM:

Step 1: Start the program.
Step 2: Create a parent class and class method.
Step 3: Create a child class using the syntax: class class_name: specifier parent_class_name.
Step 4: In the main method, create an object for the child class.
Step 5: Display the output.
Step 6: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Account {
   public:
   void student(){
      cout<<"My name is Harshini."<<endl;
      cout<<"Roll number is 39."<<endl;
   }
};
class academics : public Account {
   public:
   void program(){
      cout<<"This program is executed using Single Inheritance."<<endl;
   }
};
int main()
{
   academics vs;
   vs.student();
   vs.program();
   return 0;
}
```

OUTPUT:

```
My name is Harshini.
Roll number is 39.
This program is executed using Single Inheritance.
```

RESULT:

Thus, the c++ program to demonstrate the concept of Single Inheritance was executed and output was verified successfully.

Ex. No:7(b)	MULTIPLE INHERITANCE

AIM:

To write a c++ program to implement the concept of Multiple Inheritance.

ALGORTITHM:

Step 1: Start the program.
Step 2**:** Declare two parent classes.
Step 3: Declare access specifiers and member functions.
Step 4: Inherit both parent classes to the child class.
Step 5: Create an object for the child class.
Step 6: Using the object created, call member functions of both parent and child class.
Step 7: Display the output.
Step 8: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class father {
public:
   father() { cout << "Father name is Swaminathan.\n"; }
};
class mother {
public:
   mother()
   {
      cout << "Mother name is Jayalakshmi.\n";
   }
};
class daughter : public father, public mother {
   public:
   daughter(){
      cout<<"Daughter name is Harshini.\n";
   }
};
int main()
{
   daughter vs;
   return 0;
}
```

OUTPUT:

```
Father name is Swaminathan.
Mother name is Jayalakshmi.
Daughter name is Harshini.
```

RESULT:

Thus, the c++ program to demonstrate the concept of Multiple inheritance was executed and output was verified successfully.

Ex. No:7(c)	HIERARCHICAL INHERITANCE

AIM:

To write a C++ program to demonstrate the concept of hierarchical inheritance.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare a parent classes.
Step 3**:** Declare access specifiers and member functions.
Step 4**:** Inherit the parent class to both the child class.
Step 5**:** Create an object for the child class.
Step 6: Using the objects created, call both member functions of parent and child class.
Step 7**:** Display the output.
Step 8: Stop the program.

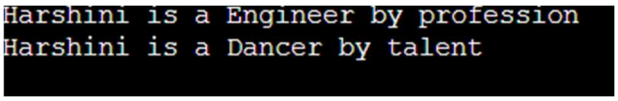
PROGRAM:

```
#include <iostream>
using namespace std;
class person{
   public:
   string n;
   person(){
      n="nithya";
   }
};
class profession: public person {
   public:
   string p="Engineer";
   profession(){
      cout<<n<<" is a "<<p<<" by profession"<<endl;
   }
};
class talent: public person{
   public:
   string t="Dancer";

   talent(){
      cout<<n<<" is a "<<t<<" by talent"<<endl;
   }
};
int main()
{
   profession vs1;
   talent vs2;
   return 0;
```

```
}
```

OUTPUT:



RESULT:

Thus, the c++ program to demonstrate the concept of Hierarchical Inheritance was executed and output was verified successfully.

Ex. No:7(d)	MULTILEVEL INHERITANCE

AIM:

To write a C++ program to demonstrate the concept of multilevel inheritance.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare a parent classes.
Step 3: Declare access specifiers and member functions.
Step 4: Inherit the parent class to the child class.
Step 5: Inherit the above child class as parent class to the next child class.
Step 6: Create an object for the child class and call the member functions.
Step 7: Display the output.
Step 8: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class length{
   protected:
   int l;
   public:
   length(){
      cout<<"Enter the length of the rectangle ";
      cin>>l;
   }
};
class breadth : public length{
   protected:
   int b;
   public:
   breadth(){
      cout<<"Enter the breadth of the rectangle ";
      cin>>b;
   }
};
class area: public breadth{
   protected:
   int a;
   public:
   area(){
      a=l*b;
      cout<<"Area of the rectangle is "<<a;
   }
};
int main()
{
```

```
    area vs;
    return 0;
}
```

OUTPUT:

```
Enter the length of the rectangle 21
Enter the breadth of the rectangle 2
Area of the rectangle is 42
```

RESULT:

Thus, the c++ program to demonstrate the concept of Multilevel Inheritance was executed and output was verified successfully.

Ex. No:7(e)	HYBRID INHERITANCE

AIM:

To write a C++ program to demonstrate the concept of hybrid inheritance.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Declare a parent class.
Step 3: Declare access specifiers, data members and member functions.
Step 4**:** Inherit the parent class to the child class.
Step 5**:** Declare another parent class.
Step 6: Inherit the new parent class and the child class to another child class.
Step 7**:** Create an object for the last child class and call the member functions of both parent and child class.
Step 8**:** Display the output.
Step 9**:** Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class person{
   public:
   string n;
   person(){
      n="Jayalakshmi";
   }
};
class profession: public person {
   private:
   string p="Proffesor";
   public:
   profession(){
      cout<<"Mother name is "<<n<<endl;
   }
};
class talent: public person{
   public:
   string t;
   talent(){
      t="Dancer";
      cout<<"She is a "<<t<<" by talent"<<endl;
   }
};
class father{
   private:
   string f="swaminathan";
   public:
   father(){
```

```
        cout<<"Father name is "<<f<<endl;
        cout<<"He is an Business man."<<endl;
    }
};
class child: public talent, public father{
    private:
    string c;
    public:
    child(){
        c="Nithya";
        cout<<c<<" wants to became like her father."<<endl;
        cout<<"She is also "<<t<<" like her mother "<<n<<"."<<endl;

    }
};
int main()
{
    profession s;
    child vs;
    return 0;
}
```

OUTPUT:

```
Mother name is Jayalakshmi
She is a Dancer by talent
Father name is swaminathan
He is an Business man.
Nithya wants to became like her father.
She is also Dancer like her mother Jayalakshmi.
```

RESULT:

Thus, the c++ program to demonstrate the concept of Hybrid Inheritance was executed and output was verified successfully.

Ex. No:8	FUNCTION OVERRIDING

AIM:

To create a C++ program to demonstrate the concept of function overriding.

ALGORITHM:

STEP 1: Start the program.
STEP 2: Create a base class and derived class.
STEP 3: Define methods with same name in both the classes.
STEP 4: In the main method, create objects for both base and derived class.
STEP 5: Use the objects to call their respective member functions.
STEP 6: Display the output.
STEP 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class base{
   public:
   void overriding(){
      cout<<"This function is in base class."<<endl;
   }
};
class child : public base{
   public:
   void overriding(){
      cout<<"Base class function have been overrided"<<endl;
   }
};
int main()
{
   cout<<"Base class function before overriding :"<<endl;
   base s;
   s.overriding();
   child vs;
   cout<<"After overriding : "<<endl;
   vs.overriding();
   return 0;
}
```

OUTPUT:

```
Base class function before overriding :
This function is in base class.
After overriding :
Base class function have been overrided
```

RESULT:

Thus, the c++ program to implement function overriding was executed and output was verified successfully.

Ex. No:9(a)	VIRTUAL BASE CLASS

AIM:

To write a C++ program to demonstrate the concept of virtual base class.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare a parent class.
Step 3: Declare access specifiers, data members and member functions.
Step 4**:** Inherit the parent class virtually to both the child classes.
Step 5: Inherit both the child classes as parent class to another child class.
Step 6: In the main method, create an object for the child class and call the member functions.
Step 7: Display the output.
Step 8: Stop the program.

PROGRAM:

```
#include<iostream>
using namespace std;
class Student
{
public:
char name[20];
int rollno;
void getdata()
{
cout<<"Enter your rollno:";
cin>>rollno;
cout<<"Enter your name:";
cin>>name;
}
};
class Test:virtual public Student
{
public:
int m1,m2,m3;
void getmarks()
{
cout<<"Enter the marks : ";
cin>>m1>>m2>>m3;
}
};
class Sports:public virtual Student
{
public:
int score;
void getscore()
{
cout<<"Enter sports score :";
```

```
cin>>score;
}
};
class Result:public Test,public Sports
{
public:
void display_result()
{
cout<<"Total marks and score of "<<name<<" is "<<m1+m2+m3<<" and "<<score<<endl;
}
};
int main()
{
Result r;
r.getdata();
r.getmarks();
r.getscore();
r.display_result();
return 0;
}
```

OUTPUT:

```
Enter your rollno: 39
Enter your name: harshu
Enter the marks : 90 90 90
Enter sports score : 70
Total marks and score of harshu is 270 and 70
```

RESULT:

Thus, the c++ program to demonstrate the concept of file manipulation was executed and output was verified successfully.

Ex. No:9(b)	PURE VIRTUAL FUNCTION

AIM:

To write a C++ program to demonstrate the concept of pure virtual function.

ALGORITHM:

Step 1: Start the program.
Step 2**:** Create a class.
Step 3**:** Declare access specifiers, data members and member functions.
Step 4**:** Declare the pure virtual function in the base class.
Step 5**:** Inherit the parent class to both the child classes and define the virtual function in both the child classes.
Step 6: Create an object for both the child classes created and call the member functions.
Step 7**:** Display the output.
Step 8**:** Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
class Animal {
  public:
    virtual void move() = 0;
};
class Lion: public Animal {
  public:
    void move() {
      cout << "Lions walk in an unusual manner." << endl;
    }
};
class Wolf: public Animal {
  public:
    void move() {
      cout << "Wolves can move for hours at a speed of 5-6 miles per hour." << endl;
    }
};
int main() {
  Lion l;
  Wolf w;
  l.move();
  w.move();
  return 0;
}
```

OUTPUT:

```
Lions walk in an unusual manner.
Wolves can move for hours at a speed of 5-6 miles per hour.
```

RESULT:

Thus, the c++ program to implement virtual function was executed and output was verified successfully.

Ex. No:10	FILE MANIPULATION

AIM:

To write a C++ program to demonstrate the concept of file manipulation.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Include the fstream header file.
Step 3**:** Create an object for the fstream class.
Step 4**:** Open a file using open() method.
Step 5: Write contents into the file.
Step 6**:** Read the contents in the value.
Step 7: Close the value and display the output.
Step 8**:** Stop the program.

PROGRAM:

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
   fstream FileName;
   FileName.open("FileName", ios::out);
   if (!FileName)
   {
      cout<<"Error while creating the file";
   }
   else
   {
      cout<<"File created successfully\n";
      FileName.close();
   }
   FileName.open("FileName.txt", ios::out);
   if (!FileName)
   {
      cout<<" Error while creating the file ";
   }
   else
   {
      cout<<"File was created and contents have been inserted.\n";
      FileName<<"THIS FILE WAS CREATED USING C++ PROGRAM\n";
      FileName.close();
   }
   FileName.open("FileName.txt", ios::in);
   if (!FileName)
```

```
    {
       cout<<"File doesn't exist.";
    }
    else
    {
       char x;
       while (1)
       {
          FileName>>x;
          if(FileName.eof())
             break;
          cout<<x ;
       }
    }
    FileName.close();
    return 0;
}
```

OUTPUT:

```
File created successfully
File was created and contents have been inserted.
THISFILEWASCREATEDUSINGC++PROGRAM
```

main.cpp | FileName | FileName.txt

```
1  THIS FILE WAS CREATED USING C++ PROGRAM
2
```

RESULT:

Thus, the c++ program for file manipulation was executed and output was verified successfully.

Ex. No:11(a)	CLASS TEMPLATES

AIM:

To write a C++ program to demonstrate the concept of templates.

ALGORITHM:

Step 1: Start the program.
Step 2**:** Declare a template class.
Step 3**:** Create a class calculator and define member functions for performing arithmetic operations.
Step 4**:** Inside the main method, create object for the calculator class and call the required member functions.
Step 5**:** Display the output.
Step 6**:** Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
template <class T>
class Calculator
{
private:
T num1, num2;
public:
Calculator(T n1, T n2)
{
num1 = n1;
num2 = n2;
}
void displayResult()
{
cout << "Numbers: " << num1 << " and " << num2 <<endl;
cout << num1 << " + " << num2 << " = " << add() << endl;
cout << num1 << " - " << num2 << " = " << subtract() << endl;
cout << num1 << " * " << num2 << " = " << multiply() << endl;
cout << num1 << " / " << num2 << " = " << divide() << endl;
}
T add()
{
return num1 + num2;
}
T subtract()
{
return num1 - num2;
}
T multiply()
{
return num1 * num2;
```

```
}
T divide()
{
return num1 / num2;
}
};
int main()
{
int a,b;
float c,d;
cout<<"Enter an integer number:"<<endl;
cin>>a;
cout<<"Enter another integer number:"<<endl;
cin>>b;
Calculator<int> intCalc(a,b);
cout << "Results:" << endl;
intCalc.displayResult();
cout<<"Enter a floating number:"<<endl;
cin>>c;
cout<<"Enter another floating number:"<<endl;
cin>>d;
Calculator<float> floatCalc(c,d);
cout << "Results:" << endl;
floatCalc.displayResult();
return 0;
}
```

OUTPUT:

```
Enter an integer number:
45
Enter another integer number:
3
Results:
Numbers: 45 and 3
45 + 3 = 48
45 - 3 = 42
45 * 3 = 135
45 / 3 = 15
Enter a floating number:
3.7
Enter another floating number:
1.3
Results:
Numbers: 3.7 and 1.3
3.7 + 1.3 = 5
3.7 - 1.3 = 2.4
3.7 * 1.3 = 4.81
3.7 / 1.3 = 2.84615
```

RESULT:

Thus, the above program on class templates has been executed and the output is verified successfully.

Ex. No:11(b)	FUNCTION TEMPLATES

AIM:

To write a C++ program to demonstrate the concept of function templates.

ALGORITHM:

Step 1**:** Start the program.
Step **2:** Define a user defined template for multiplication.
Step 3**:** In the main method, input values from the user.
Step 4**:** Call the template function for multiplication.
Step 5: Display the output.
Step 6**:** Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
template <typename T>
T Multiply(T num1, T num2)
{
return (num1 * num2);
}
int main()
{
int result1,a,b;
double result2,c,d;
cout<<"Enter an integer value:"<<endl;
cin>>a;
cout<<"Enter another integer value:"<<endl;
cin>>b;
result1 = Multiply<int>(a,b);
cout <<a<<"*"<<b<<"="<<result1<<endl;
cout<<"Enter a floating number:"<<endl;
cin>>c;
cout<<"Enter another floating number:"<<endl;
cin>>d;
result2 = Multiply<double>(c,d);
cout <<c<<"*"<<d<<"="<<result2<<endl;
return 0;
```

OUTPUT:

```
Enter an integer value:
89
Enter another integer value:
7
89*7=623
Enter a floating number:
7.8
Enter another floating number:
4.2
7.8*4.2=32.76
```

RESULT:

Thus, the above program on function templates have been executed and the output was verified successfully.

Ex. No:12(a)	BUILT-IN EXCEPTIONS

AIM:

To write a C++ program to demonstrate the concept of exception handling.

ALGORITHM:

Step 1**:** Start the program.
Step 2: Define a method divide() to perform division of two numbers.
Step 3**:** Include the throw keyword to throw an exception if divisor is zero.
Step 4**:** In the main method, use try and catch block to handle the exceptions.
Step 5: Call the member function.
Step 6: Display the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
using namespace std;
double division(int a, int b)
{
if( b == 0 )
{
throw "Division by zero condition!";
}
return (a/b);
}
int main ()
{
int x,y;
double z = 0;
cout<<"ENTER A NUMBER:"<<endl;
cin>>x;
cout<<"ENTER ANOTHER NUMBER:"<<endl;
cin>>y;
try
{
z = division(x, y);
cout << z << endl;
}
catch (const char* msg)
{
cerr << msg << endl;
}
return 0;
}
```

OUTPUT:

```
ENTER A NUMBER:
8
ENTER ANOTHER NUMBER:
0
Division by zero condition!
```

RESULT:

Thus, the above program on user defined exceptions has been executed and output was verified successfully.

Ex.No:12(b)	USER DEFINED EXCEPTION

AIM:

To write a C++ program for user defined exception in exceptional handling.

ALGORITHM:

Step 1**:** Start the program.
Step 2**:** Create a class inheriting the exception class.
Step 3**:** Inside another class, input values from the user.
Step 4**:** Use try and catch block to check for exceptions.
Step 5**:** Display the output.
Step 7**:** Stop the program.

PROGRAM:

```
#include<iostream>
#include<exception>
using namespace std;
class lesse:public exception
{
public:
void what()
{
cout<<"Mark less than 0"<<endl;
}
};
class more:public exception
{
public:
void what()
{
cout<<"Mark greater than 100"<<endl;
}
};
class student
{
string name;
string rollno;
int marks[5];
public:
void get();
};
void student::get()
{
cout<<"ENTER NAME:"<<endl;
cin>>name;
cout<<"ENTER ROLL NUMBER:"<<endl;
```

```
cin>>rollno;
cout<<"ENTER MARKS:"<<endl;
for(int i=0;i<5;i++)
{
try{
cin>>marks[i];
if(marks[i]>100)
{
more d;
throw d;
}
}
catch(more &e)
{
throw ;
}
try
{
if(marks[i]<0)
{
lesse d;
throw d;
}
}
catch(lesse &e)
{
throw ;
}
}
}
int main()
{
student s;
more e;
lesse e1;
try
{
s.get();
}
catch(more &e)
{
e.what();
}
catch(lesse &e1)
{
e1.what();
}
return 0;
}
```

OUTPUT:

```
ENTER NAME:
asmi
ENTER ROLL NUMBER:
21
ENTER MARKS:
109
Mark greater than 100
```

RESULT:

Thus, the c++program to implement user defined exception was executed and output was verified successfully.

Ex.No:13	PRIORITY QUEUE USING STANDARD TEMPLATE LIBRARY

AIM:

To know about the concept of priority queue using standard template library.

ALGORITHM:

Step 1**:** Start the program.
Step 2: Define the desired header files.
Step 3**:** Create function and return the result.
Step 4**:** Call the function inside the main function.
Step 5: Use push() and pop() function to insert and delete an element in queue.
Step 6**:** Display the output.
Step 7: Stop the program.

PROGRAM:

```
#include <iostream>
#include <queue>
#include <string>
#include <cstdlib>
using namespace std;
int main()
{
priority_queue<int> pq;
int choice, item;
  while(1)
  {
  cout<<"Priority Queue Implementation in Stl"<<endl;
  cout<<"1.Insert Element into the Priority Queue"<<endl;
  cout<<"2.Delete Element from the Priority Queue"<<endl;
  cout<<"3.Size of the Priority Queue"<<endl;
  cout<<"4.Top Element of the Priority Queue"<<endl;
  cout<<"5.Exit"<<endl;
  cout<<"Enter your Choice: ";
  cin>>choice;
  switch(choice)
  {
  case 1:
  cout<<"Enter value to be inserted: ";
  cin>>item;
  pq.push(item);
  break;
  case 2:
  item = pq.top();
  if (!pq.empty())
  {
```

```
    pq.pop();
    cout<<"Element "<<item<<" Deleted"<<endl;
    }
    else
    {
    cout<<"Priority Queue is Empty"<<endl;
    }
    break;
    case 3:
    cout<<"Size of the Queue: ";
    cout<<pq.size()<<endl;
    break;
    case 4:
    cout<<"Top Element of the Queue: ";
    cout<<pq.top()<<endl;
    break;
    case 5:
    exit(1);
    break;
    default:
    cout<<"Wrong Choice"<<endl;
    }
    cout<<"\n"<<endl;
    }
    return 0;
}
```

OUTPUT:

```
Priority Queue Implementation in Stl
1.Insert Element into the Priority Queue
2.Delete Element from the Priority Queue
3.Size of the Priority Queue
4.Top Element of the Priority Queue
5.Exit
Enter your Choice: 1
Enter value to be inserted: 21


Priority Queue Implementation in Stl
1.Insert Element into the Priority Queue
2.Delete Element from the Priority Queue
3.Size of the Priority Queue
4.Top Element of the Priority Queue
5.Exit
Enter your Choice: 1
Enter value to be inserted: 4


Priority Queue Implementation in Stl
1.Insert Element into the Priority Queue
2.Delete Element from the Priority Queue
3.Size of the Priority Queue
4.Top Element of the Priority Queue
5.Exit
Enter your Choice: 1
Enter value to be inserted: 23
```

RESULT:

Thus, the c++ program to demonstrate the concept of priority queue using standard template library was executed and output was verified successfully.