```
};
    void main()
    {
      A  *k;
      B  ob;
      clrscr();
      k=&ob;
      k->xyz();
      getch();
    }
```

Output

```
Member of base class
```

k->xyz(); explanation-

In late binding, function call is resolved at runtime compiler determines the type of object at runtime and then binds the function call.

Virtual Function-

Virtual function is a function in base class which is overrided in derived class and which tells the compiler to perform late binding on this function.

Pure Virtual Function-

It is a function in base class which consist of function declaration but not the function defination and is overrided in the derived class and which tells the compiler to perform late binding on this function.

Teacher's Signature

PAGE NO.:

Pure Virtual Function-

e.g.→
```
#include<iostream.h>
#include<conio.h>
class A
{
 public: virtual void xyz()=0;
};
 class B: public A
 {
   public: void xyz()
        {
         cout<<"Member of derived class";
        }
 };
  void main()
  {
   A *k;
   B ob;
   clrscr();
   k=&ob;
   k->xyz();
   getch();
  }
```

Scanned by CamScanner

Operator Overloading-

In C++, operators are defined to work with only primitive data types operands like int, float, char, double, etc. We can not use operators with objects if we want to use operators on objects when we have to overload then operator after overloading an operator we can use it with the object of that class. Some operators can not be overloaded (like size of, ::, ternary, ., etc).

To overload an operator we have to define a function in the class.

Syntax:

```
returntype operator<operator symbol>()
{
    // function body
}
```

There are two types of operator overloading:-

1.) Unary Operator Overloading
2.) Binary Operator Overloading

1.) Unary Operators-

Unary operators work on only one operand. Unary operator overloading means overloading of ++, --, logical not (!), 1's complement (~).

The Unary operators operate on the object for which they are called and normally this operator appears on the left side of the object (as --obj, !obj, ~obj) but sometimes they can be used postfix (like obj++, obj--)

e.g.→
```
#include<iostream.h>
#include<conio.h>
class demo
{
public: void operator!()
        {
        cout<<"Hello";
        }
};
      void main()
      {
      demo d1;
      !d1; // we can also call as d1.operator!()
      getch();
      }
```

```
#include<iostream.h>
#include<conio.h>
class demo
{
   int i;
   public: void getdata()
           {
            cout<<"Enter value";
            cin>> i;
           }
           void operator ++()
           {
            i=i+1;
           }
           void display()
           {
            cout<<i;
           }
};
           void main()
           {
            demo d1;
            d1. getdata();
            d1 ++;
            d1. display();
            getch();
           }
```

Binary Overloading-

Similar to Unary Operators, Binary Operators can also be overloaded.

Binary Operators require two operands and they are overloaded by defining a member function. Right side operand is passed as argument and left side operand calls the function.

```
#include<iostream.h>
#include<conio.h>
{
  int i, j;
  public: void getdata()
          {
           cout<<"Enter value";
           cin>>i>>j;
          }
          demo operator+(demo obj)
          {
           demo ob;
           ob.i= i+obj.i;
           ob.j= j+obj.j;
           return ob;
          }
          void show()
          {
           cout<<i<<" "<<j<<endl;
          }
};
```

d3=d1+d2;

Teacher's Signature

```
void main()
{
 demo d1, d2, d3;
 d1. getdata();
 d2. getdata();
 d3 = d1 + d2;  → d3=d1. operator+(d2);
 d3. show();
 getch();
}
```

In above statement the object d1 calls the function operator+() and the object d2 is passed as argument for the function.

The above statement can also be written as follows-

d3=d1. operator+(d2);

The data members of d1 are passed implicitly and data members of d2 are passed as argument while overloading binary operators the left side operand calls the function and the right side operand is passed as argument.

DATE : / /

PAGE NO. :

Template-

Templates are used to perform the same operation on different data types. Templates allows to develop reusable software components like functions, classes etc. Supporting different data types in a single frame work.

The templates declared for a function is called function template and those declared for a class is called class template.

Function Template - There are several functions to be used frequently within different data types. The limitation of such function is that they works only on a particular data type. A function template specifies how an individual function can be used with different data types.

A function template is prefixed with the keyword template and a list of template type arguments.

Syntax:-

```
template <class template name>
returntype function name (arg list)
{
   // function body
}
```

Teacher's Signature

DATE: / /

PAGE NO.:

```
#include<iostream.h>
#include<conio.h>
template <class T>
void Swap (T &a, T &b)
{
    T c;
    c=a;
    a=b;
    b=c;
}
   void main ()
   {
    clrscr();
    int i,j;
    float p,q;
    char m,n;
    cout<<"Enter two integers:";
    cin>> i>>j;
    Swap(i,j);
    cout<< "After Swapping:"<< i<<" "<<j<<endl;
    cout<<"Enter two real numbers:";
    cin>>p>>q;
    Swap(p,q);
    cout<<"After Swapping:"<<p<<" "<<q<<endl;
    cout<<"Enter two characters:";
    cin>>m>>n;
    Swap(m,n);
    cout<<"After Swapping"<<m<<" "<<n<<endl;
    getch();
   }
```

Teacher's Signature

```
#include<iosfream.h>
#include<conio.h>
template <class T>
 class sum
 {
   Ta; Tb;
   public: sum(T i, Tj)
    {
     a=i;
     b=j;
    }
    void showsum()
    {
     cout<< a+b<< endl;
    }
 };
    void main()
    {
     clrscr();
     sum<int> ob1(10,20);
     sum <float> ob2(2.4, 8.3);
     ob1.showsum();
     ob2.showsum();
```

Teacher's Signature

PAGE NO. :

```
#include <iostream.h>
#include <conio.h>
template <class temp>
class myclass
{
  temp a,b,c;
  public: void getdata (temp j, temp k)
  {
    a=j;
    b=k;
  }
   void result ()
   {
     c=a;
     a=b;
     b=c;
     cout <<"After Swapping" <<a<<" "<<b <<endl;
   }
};
   void main()
   {
     int i, j;
     float p,q;
      char m,n;
     myclass< int> ob1;
     myclass<float> ob2;
     myclass<char> ob3;
      clrscr();
```
~~cout<<"Before Swapping i="<<i<<endl;~~

Teacher's Signature

Scanned by CamScanner

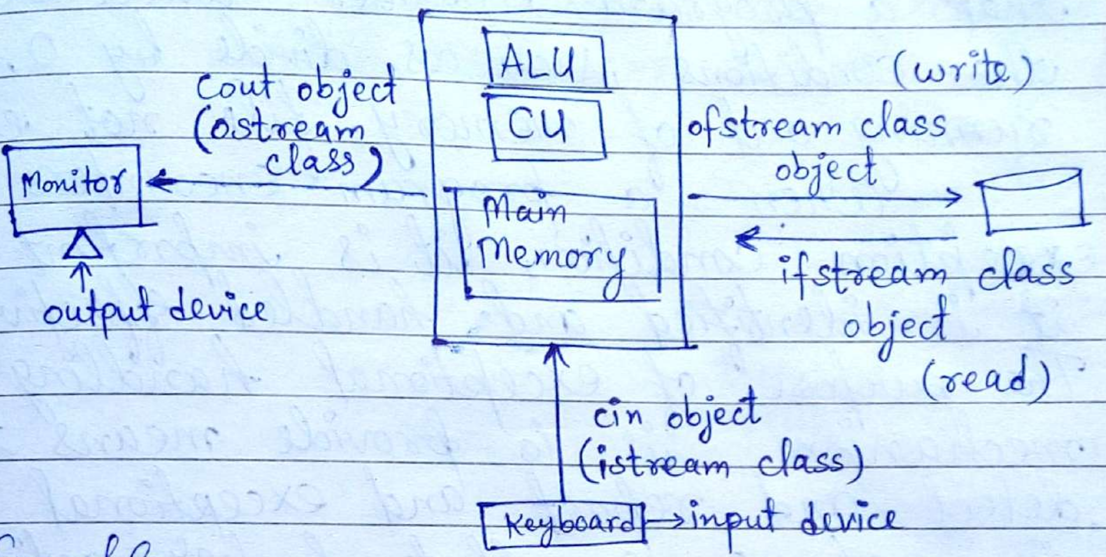
```
cout<<"Enter two integers:";
 cin>>i>>j;
 cout<<"Before Swapping i="<<i<<endl;
 cout<<"Before Swapping j="<<j<<endl;
 ob1.getdata(i,j);
 ob1.result();
 cout<<"Enter two real numbers:";
  cin>>p>>q;
  cout<<"Before Swapping p="<<p<<endl;
  cout<<"Before Swapping q="<<q<<end;
  ob2.getdata(p,q);
  ob2.result();
 cout<<"Enter two characters:";
 cin>>m>>n;
  cout<<"Before Swapping m="<<m<<endl;
 cout<<"Before Swapping n="<<n<<endl;
 ob3.getdata(m,n);
 ob3.result();
  getch();
}
```

Teacher's Signature

DATE : / /

PAGE NO. :

File Handling-



Example-

```
#include <fstream.h>
~~#include <conio.h>~~
~~#include<iostream.h>~~
void main()
{
  ofstream ob;
  ob.open("xyz.txt");
  ob << "Hello";
  ~~ob << "Friends";~~
  ob.close();
}
```

Teacher's Signature

DATE : / /
PAGE NO. :

Exceptions are runtime unusual conditions that a program encounter while executing the conditions such as divide by 0, running out of memory, file not exists.
When a program encounters an exception condition, it is important that, it is identified and handled effectively. The purpose of exceptional handling mechanism is to provide means to detect and report and exceptional circumstance so that proper action can be taken. The following tasks are performed:

1. Find the problem (hit the exception)
2. Inform that an ~~occur has~~ error has occur (throw the exception)
3. Receive the error information (catch the exception)
4. Proper action handle the exception.

Exception Handing Mechanism uses these three keywords (try, catch, throw) The keyword try is used to preface a block statement which may generate exception. This block is known as try block when an exception is detected. It is thrown using a throw statement in the try block.

Teacher's Signature

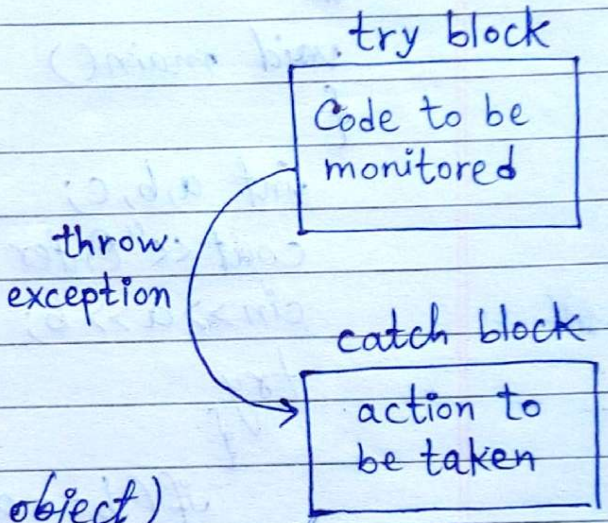
A catch block is defined by the keyword catch. This block catches the exception thrown by the throw statement in the try block.

Syntax:-

```
Type
 {
   ≡
   throw exception;
   ≡
 }
catch (type exception object)
 {

 }
catch (type exception object)
 {

 }
 ≡
```



When a try block throws an exception the program control leaves the try block and enters the catch statement of catch block.

If the type of the argument matches in the catch statement then the catch block is executed if they don't match, the program is aborted.

DATE : / /

PAGE NO. :

When no execution is detected, the control goes to the next statement after the catch block.

```
#
#
void main()
{
 int a,b,c;
 cout<<"Enter two values";
 cin>>a>>b;
 try
   {
     if(b==0)
       {
        throw b;
       }
     else
       {
        c=a/b;
        cout<<c;
       }
   }
   catch (int i)
   {
    cout<<"Exception caught";
   }
 getch();
};
```

Teacher's Signature