"C++" Language

This is an Object Oriented Programming Language [OOP].

①

For Turbo C:

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

For Dev C:

```
#include<iostream>
using namespace std;
class test
{
  private:
    int x,y;
    public:
      void input()
       {
         cout<<"\n Enter the value of x:";
         cin >>x;
         cout<<"\n Enter the value of y:";
          cin>>y;
         }
        void display()
        {
          cout<<"\n x="<<x;
          cout<<"\n y="<<y;
        }
};
int main()
{
   test ob,ob1;
   ob.input();
   ob.display();
   ob1.input();
   ob1.display();
   return 0;
}
```

or,

```
cout<<"\n Enter the value of
                x and y:"<<endl;
cin>>x>>y;
```

- In case of structure, access specifier is → Public
- In case of class, access specifiers are → (priority)
  - Private
  - protected
  - public
- Cout → printf
- Cout → scanf
- endl → End of line
  - ↳ This is called manipulator.
- To access private data function we have to use public functions to access.
- Here,
  ob = object, which is representing "test" class.

② Scope Resolution Operator:

```
#include<iostream>
using namespace std;
class test
{
  private:
     int x, y;
   public:
      void input();
      void display();
};
void test::input()
{
  cout << "\n Enter the value of x:";
  cin >> x;
  cout << "\n Enter the value of y:";
  cin>> y;
}
void test::display()
{
  cout << " x = " <<x;
  cout << " y = " <<y;
}
int main()
{
   test ob, ob1;
   ob.input();
   ob. display();
   ob1. input();
   ob1. display();
   return 0;
}
```

:: → Scope Resolution Operator.

↳ This is used to access the data globally.

Exercise ⇒

Advantages of C++:

1. Flexibility → We can take input variable at any place.
2. Type Casting → See Book.

Void pointer: —

```
void *P
char *S
P=S ✓      // void pointer can be converted into any type
S=P ✗ (error) // but any type can not be converted into void.
```

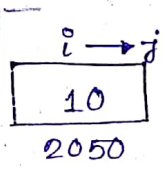
Scope Resolution Operator :

If no variable are there, then it will print the local variable's value

```
int a = 10;
void main()
{
  int a = 15;                  // output = 15   [Local variable]
  cout << a << :: a;           // output = 10   [Global variable]
  :: a = 20;
  cout << a << ::a ;           // output = 20
}                              // output = 15
```

Reference :

```
void main()
{
  int i = 10;
  int &j = i ;
  cout << i << j ; // i=10, j=10
  j = 20;
  cout << i << j ; // i=20, j=20
  i = 30;
  cout << i << j ; // i=30, j=30
  i++;
  cout << i << j ; // i=31, j=31
  j++;
  cout << i << j ; // i=32, j=32
}
```

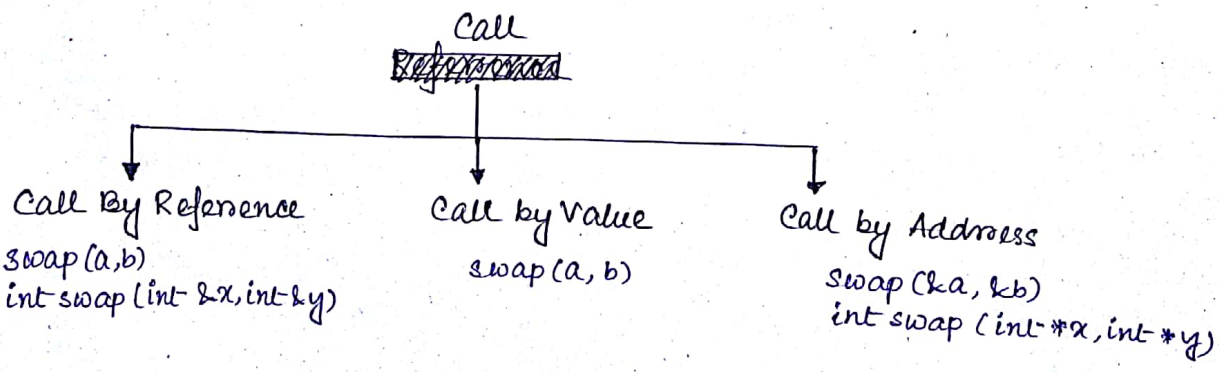


&j → Reference of i.

```
int &j ;
j = i ;
```
Error

- as reference is performed as nick name, so if we change the value of i, then j also must be changed.

There are 3 type of Reference call.



Constant: —

```
char *p = "HELLO";  // Pointer is variable, so is string
*p = 'M';  // Work
 p = "BYE";// work
const char *q = "HELLO"; // string is fixed, but pointer is not fixed
*q = 'M';  // Error
 q = "BYE";// work
char const *s = "HELLO"; // String is fixed, but pointer is not fixed.
*s = 'M'; // Error
 s = "BYE";// work
char *const t = "HELLO"; // Pointer is fixed, but string is not fixed.
*t = 'M';  // work
 t = "BYE";// Error
const char *const u = "HELLO";// string is fixed and point is also fixed
*u = 'M';  // Error
 u = "BYE";// Error.
```

Example:

```
char *str1 = "Rain Rain";
char *str2 = str1;
Cout << str1 << str2; // Str1 = Rain Rain , str2 = Rain Rain
*str1 = 'M'; // Str1 = Main Rain, str2 = Main Rain
*str2 = 'P'; // Str1 = Pain Rain, str2 = Pain Rain
```

BOOL [Boolean Data Type]:

```
int main()
{
bool x,y;
int a=10, b=20, c=30;
x = a<b;
y = b>=c;
cout << x << y; // x=1, y=0
}
```

Example,

```
int main()
{
 bool b=32;
 int i = false; or i=0
 cout << b << i; // b=1, i=0
 int j = b+b;
 bool k=b+b;
 cout << j << k; // j=2, k=1
}
```

Default Argument

15/07/2018

```
void f1 (int x=5, int y=3 , int z=4)
{
  cout << x << y << z;
}
f1(7,8,9); // The values will be changed. [∴ x=7, y=8, z=9 (over write)]
f1(9,10, ); // x=9, y=10, z=4
f1( ,11,12); ✗
f1(9, ,10); ✗
f1(10, , ); // x=10, y=3, z=4
```

Formal argument → actual argument

actual argument → Formal argument

[actual argument can not be over written by formal argument]

- There are 3 pilers of Object Oriented Program.
  - — Inheritance
  - — Polymorphism
  - — Encapsulation

- Function Overloading :—

```
class A
{ public:
void int f1 (int x)
    {

    }
void int f1 (int x, float y)
    {

    }
    void f1 (float x)
    {

    }
};
int main()
{
  test ob;
  ob.f1(6);
  ob.f1 (6, 7.8);
  [illegible];
  ob.f1 (1.2, 8);
  ob.f1 (12.5); → [illegible]
  [illegible]
  return 0;
}
```

* void f1 (float x, int y)
{
}

☑ signature of function overloading

1. no. of arguments.
2. order of arguments.
3. ~~return~~ type of the ~~function~~ argument.

Inline Function

Inline function is used for small program, not for large program.

Inline Function-এ control jump না করে,
Function-এ copy হয়

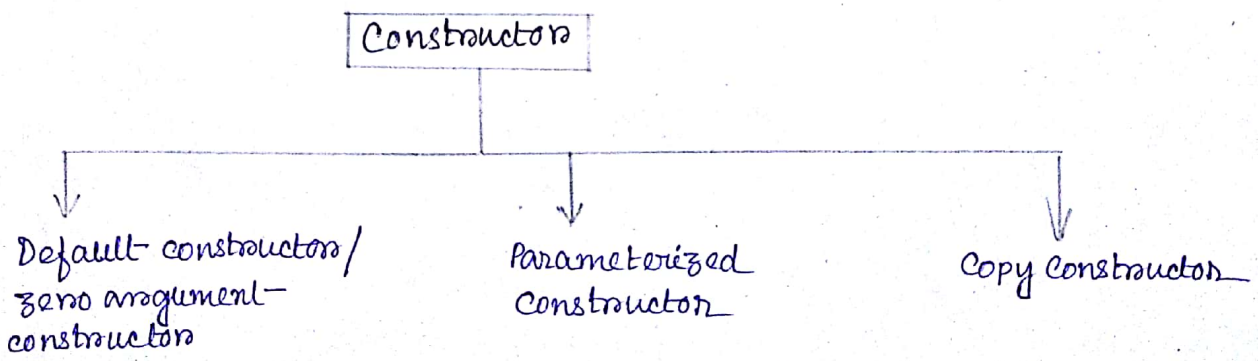
```
class test
{ public:
    inline void fi()
    { cout << " HELLO";
    }
};
void main()
{
   test ob;
   ob.fi();
}
```

☑ Constructor :—

- This is a member function who has no return type and the name of the member function will be same as class name.
- constructor এবং destructor উভয়কে কোথাও implicit call হয়।
- in that case, we can call any object function using constructor function.
- Default constructor is created by itself.
- Destructor is symbolized by negation (~).
- constructor memory-তে address allocate করে।
- Destructor যেখানে memory-র যে position-এ কাজ হয়ে যাওয়ার পর deallocate করে।
- access specifier of constructor is Public.

Constructor and Destructor : —

```
class test
{ private:
    int x, y;
  public:
    test()
    { cout << "\n Enter the value of x:";
      cin >> x;
      cout << "\n Enter the value of y:";
      cin >> y;
    }
    ~test()
    { cout << "\n Inside Destructor";
    }
    int display()
    { cout << " x = " << x;
      cout << " y = " << y;
    }
};
int main()
{
   test ob, ob1, ob2;
   ob.display();
   ob1.display();
   ob2.display();
   return 0;
}
```



Constructor Overloading :—

```
class test
{ private:
    int a,b;
    public:
      test()
      { a=0;
        b=0;
      }
      test(int x)
      { a=x;
        b=0;
      }
      test(int x, int y)
      {
        a=x;
        b=y;
      }
};
int main()
{
  test ob;
  test ob1(6);
```

```
      void display()
      { cout<<"\n a="<<a;
        cout<<"\n b="<<b;
      }
};
int main()
{
  test ob;
  test ob1(6);
  test ob2(7,8);
  ob1.display();
  ob2.display();
  return 0;
}
```

NOTE: If we use parameterized constructor, then we have to use default constructor or zero-argument constructor.

(Most Important)

Copy Constructor :—

```
#include<iostream>
using namespace std;
class test
{ private:
    int a;
   public:
     test()
     {
       a=0;
     }
     test(const test &obj)
     {
      a = obj.a;
     }
     void input()
     {
       cout<<"\n Enter the value of a:";
       cin>>a;
     }
     void display()
     {
        cout<<"a"\n a = "<<a;
     }
};
int main()
{
   test ob;
   ob.input();
   test ob1(ob);
   ob.display();
   return 0;
}
```

or,

```
test ob, ob1;
ob.input();
ob1 =ob;
return 0;
```

[Bitwise Copy]

- test(test obj)
  {
  obj.a=7;
  a=obj.a;
  }

এক্ষেত্রে ob-এর a-তে 5 input নেওয়া হলে, obj-তে 5 যাবে। এরপর obj.a কয়েক যদি 7 input নেওয়া হয়, তাহলে display এর সময় সময় ob-এর a-তে (input 5) 5 আর, ob1-এর a-তে 7 display করবে। অর্থাৎ, copy হয়নি।

- test(test &obj)
  {
  obj.a=7;
  a=obj.a;
  }

এক্ষেত্রে obj.a = 7 করলে, reference-এর মধ্যে 7-value-টা store করবে। As আমরা জানি reference হল main value-এর nick name, অর্থাৎ, reference value change হলে, main value ও change হয়ে যাবে।
so, display-এর সময় ob-এর a-তে 7 এবং ob1-এর a-তেও 7 show করবে।
অর্থাৎ, copy হয়নি।

- test(const test &obj)

– এটা constant type set করা হলে, এর মধ্যে obj.a-তে কোনো value assign করা যাবে না [~~কোনো~~, so, এই line-টা আর execute করবে না], তাই const type use করার জন্য display-এর পর ob1 আর ob-এর a-তে same value show করবে।
অর্থাৎ, successfully copy constructor work করেছে।

29/10/2018

Static Variable: —

①
```
#include<iostream>
using namespace std;
class test
{
 private:
    int y;
     static int x;
     public:
       test()
       { y++;
         x++;
       }
       void show data()
       { cout<<x<<y;
       }
};
int test::x=0; //initializing static variable by 0
int main()
{
  test ob, ob1, ob2;
  ob.showdata(); // x=3, y= garbage value
  ob1.showdata(); // x=3, y= garbage value
  ob2. showdata(); //x=3 y= garbage value
  return 0;
}
```

- as, we have used x as a static variable, so the content of x will be same for ob, ob1 and ob2.
- private → default access specifier
- initially, x=0
- For ob, x=1, y = garbage value
  For ob1, x=2, y = y++
  For ob2, x=3, y = y++
- This program can count no. of objects.

Static Function: —

⑥
```
#include<iostream>
using namespace std;
class test
{ private:
    static int x;
    public:
      test()
      {
        x++;
      }static void showdata()
       {
          cout<<"\n x="<<x;
       }
};
int test::x=0;
int main()
{
    test ob, ob1, ob2;
    ob.showdata(); // 3
    ob1.show_data(); // 3
    ob2.show_data(); // 3 ⟶ test::showdata();
{
```

```
int main()
{
  test ob;
  ob.showdata(); // x=1
  test ob1;
  ob1.showdata(); // x=2
  test ob2;
  ob2.showdata(); // x=3.
}
```

Over Loading :—

Operator Over Loading :—

Operators → +, −, *, /, >>, <<, [ ], ( ), new, delete.

Syntax:

<return type> operator # (class_name) → operator_symbol

Plus operator Overloading :—

```
# include<iostream>
  using namespace std;
  class test
  {
    private :
      int x, y;
      public:
        test()
        { x=0;
          y=0;
        }
        test operator + (test obj)  // "test" return type is used to return the value
        {                              of ob3 into ob which is in the main function
          test ob3;
          ob3.x = x + obj.x;  // x → x of ob1 and obj.x → x of ob2
          ob3.y = y + obj.y;  // y → y of ob1 and obj.y → y of ob2
          return (ob3);
        }
        void input()
        { cout << "\n Enter the value of x: ";
          cin >> x;
          cout << "\n Enter the value of y:";
          cin >> y;
        }
        void display()
        { cout << "\n x = " << x;
          cout << "\n y = " << y;
        }
  };
  int main()
  {
    test ob, ob1, ob2;
    ob1.input();
    ob2.input();
    ob1.display();
    ob2.display();
    ob = ob1 + ob2; // way to call the operator function, [ob = ob1.operator+(ob2)]
    ob.display();   (The function will call through ob1 and argument will be ob2,
    return 0;          which will be stored on ob).
  }
```

Here, x and y can not be accessed directly because those are private

operator
minus overloading ~~operation~~ :-

```
#include<iostream>
using namespace std;
class test
{ private:
    int x, y;
    public:
      test()
      { x = 0;
        y = 0;
      }
      test operator- (test obj)
      {
        test ob3;
        ob3.x = x - obj.x;
        ob3.y = y - obj.y;
        return (ob3);
      }
      void input()
      {
        cout << "\n Enter the value of- x:";
        cin >> x;
        cout << "\n Enter the value of y:";
        cin >> y;
      }
      void display()
      {
        cout << "\n x = " << x;
        cout << "\n y = " << y;
      }
};
int main()
{ test ob, ob1, ob2;
  ob1.input();
  ob2.input();
  ob1.display();
  ob2.display();
  ob = ob1 - ob2;
  ob.display();
  return 0;
}
```

Operator Overloading using switch case :-

```
#include<iostream>
using namespace std;
class test
{
  private:
     int x,y;
      public:
        test()    // default constructor
        {
          x=0;
          y=0;
        }
        test operator+(test obj)
        {
          test ob3;
          ob3.x = x+obj.x;
          ob3.y = y+obj.y;
          return(ob3);
        }
        test operator-(test obj)
        {
          test ob3;
          ob3.x = x-obj.x;
          ob3.y = y-obj.y;
          return(ob3);
        }
        test operator*(test obj)
        {
          test ob3;
          ob3.x = x*obj.x;
          ob3.y = y*obj.y;
          return(ob3);
        }
        test operator/(test obj)
        {
          test ob3;
          ob3.x = x/obj.x;
          ob3.y = y/obj.y;
          return(ob3);
        }
        void input()
        {
          cout<<"\n Enter the value of x:";
          cin>>x;
          cout<<"\n Enter the value of y:";
          cin>>y;
        }
        void display()
        {
          cout<<"\n x = "<<x;
          cout<<"\n y= "<<y;
        }
};
```

```
int main()
{
    test ob, ob1, ob2;
    int x;
    cout<<"\n\t\t\t\tOperator Overloading Operation \t\t\t\n\n";
    ob1.input();
    ob2.input();
    ob1.display();
    ob2.display();
    while(1)
    {
        cout<<"\n\n 1. Addition overloading \n2. Subtraction Overloading
        \n3. Multiplication Overloading \n4. Division overloading \n5. Exit\n";
        cout<<"\n Enter your choice: ";
        cin>>x;
        switch(x)
        {
            case1: ob=ob1+ob2;
                   ob.display();
                   break;
            case2: ob=ob1-ob2;
                   ob.display();
                   break;
            case3: ob=ob1*ob2;
                   ob.display();
                   break;
            case4: ob=ob1/ob2;
                   ob.display();
                   break;
            case5: exit(0);
            default: cout<<"\nWrong Choice!!";
        }
    }
    return 0;
}
```

"This" Pointer

This pointer holds the particular address of an object.

```
#include <iostream>
using namespace std;
class test
{
  private:
    int x, y;
    public:
      test()        // default constructor
      {
        x=0;
        y=0;
      }
      test(int x, int y)
      {
        this->x = x;
        this->y = y;
      }
      test add(test obj)
      {
        this->x = x + obj.x;
        this->y = y + obj.y;
        return *this;
      }
};
int main()
{
  test ob(5,6), ob1(7,8), ob3;
  ob3 = ob.add(ob1);
  return 0;
}
```

Overload [ ]

```
#include <iostream>
using namespace std;
class test
{
  private:
    int a[10];
    public:
      int operator[](int x)
      {
        return a[x];
      }
};
int main()
{
  test obj, ob;
  obj[3] = 6;  ⟶ ob.operator[](3)=6;  [array-ல் 3rd pos-ல் 6 insert]
  return 0;
}
```

Overload"()"

```
# include <iostream>
 using namespace std;
 class test
 {
   private:
     int a, b;
     public:
        test()
        {
          a=0;
          b=0;
        }
        void operator () (int i, int j)
        {
          a=i;
          b=j;
        }
 };
 int main()
 {
   test obj (1,3), ob;
   ob(7,9);    // ob.operator()(7,9)
   return 0;
 }
```

```
* test (int x, int y)
  {
    a = x;
    b = y;
  }
```

Array of Object

```
# include <iostream>
 using namespace std;
 class test
 {
   private:
     int x, y;
     public:
        test()
        {
          x=0;
          y=0;
        }
        test (int i, int j)
        {
          x=i;
          y=j;
        }
 };
 int main()
 {
    test obj [30];   //Default
   //test ob[20] = {1,2}; // parameterized
    return 0;
 }
```

Dynamically Create Array (1D):

```
int *p, *q;
p = new int [size];   → array create করে
q = new int (5);  // q = 5 (initializing q with 5) variable create করে
```

Dynamically Create Array (2D):

```
int **p;
p = new int *[20];  → space
for(i=0; i<20; i++)
{
  p[i] = new int[10];
}
```

no. of rows = 20
no. of column = 10
] 2D array

Friend Function: —

কোনো Function-কে Friend করলে, সেটা class-এর under-এ থাকেনা। Global হয়ে যায়।

```
#include<iostream>
using namespace std;
class test
{
  private:
    int i,j;
    public:
      friend istream &operator >> (istream &din, test &ob)
      {
        cout << "\n inside class i and j:";
        din >> ob.i >> ob.j;
      }
      friend ostream &operator >> (ostream &dout, test &ob)
      {
        cout << "\n i and j is:";
        dout << ob.i << ob.j;
      }
};
int main()
{
  int x;
  test obj;
  cin >> obj;
  cin >> x;
  cout << x;
  cout << obj;
  return 0;
}
```

Both Friend Function :—

```
#include <iostream>
using namespace std;
class test1;  //Forward Reference
class test
{
  private:
    int x;
  public:
    friend void add(test, test1);
    void input()
    { cin >> x;
    }
    test()
    { x=0;
    }
};
class test1
{ private:
    int y;
  public:
    test1()
    { y=0;
    }
    void input()
    { cin>>y;
    }
    friend void add(test, test1);
    void add(test ob, test1 ob1)
    { cout<<ob.x+ob1.y;
    }
};
int main()
{ test ob;
  test1 ob1;
  ob.input();
  ob1.input();
  add(ob, ob1);
  return 0;
}
```

Any one Friend Function :—

```
#include<iostream>
using namespace std;
class test1;
class test
{
  private:
    int x;
    public:
    void input()
        { cin >>x;
        }
    void add(test 1);
};
class test 1
{
  int y;
  private:
      int y;
      public:
      void input()
          { cin >>y;
          }
          friend void test :: add (test 1);
};
int test ::add(test1 obj)
{ cout<< x+obj.y;
}
int main()
 { test ob;
   test1 ob1;
   ob.input();
   ob1.input();
   ob.add(ob1);
   return 0;
 }
```

Date: 13.08.2018

Inheritance :—

```
class <class_name1>                    // Base class
{
  private:
    // member variable
    public:
      // member function
};
class <class_name2> : accessspecifier <classname1>  // Derived class
{
  private:
    // member variable
     public:
      // member function
};
```

- **Publicly inherit করলে:**

Base class-এর private variable-গুলো derived class-এ আসবেনা। Protected variable-গুলোও আসবেনা। কেবলমাত্র, public member variable-গুলো আসবে [এবং ওগুলো public থাকবে], এক্ষেত্রে protected variable-গুলো, derived class-এ protected-ই থাকবে।

- **Protected inherit করলে:**

Base class-এর private variable-গুলো derived class-এ আসবেনা। Public variable-গুলো derived class-এ এসে protected হয়ে যাবে। আর protected-গুলো protected-ই থাকবে।

- **Privately inherit করলে:**

Base class-এর private variable-গুলো derived class-এ আসতে পারবেনা। Protected আর public variable-গুলো derived class-এ এসে private হয়ে যাবে।

[এক্ষেত্রে input আর display-গুলো directly access করা যাবেনা]

1. Input and Display By inheritance :—

```
#include<iostream>
using namespace std;
class A
{
  private:
    int i,j;
    public:
     void input()
      {
        cout<<"\n Enter the value of i:";
        cin>>i;
        cout<<"\n Enter the value of j:";
        cin>>j;
      }
     void display()
      {
        cout<<"\n i= "<<i;
        cout<<"\n j= "<<j;
      }
};
class B : public A
{
  private:
    int a, b;
     public:
      int get()
      {
        cout<<"\n Enter the value of a:";
        cin>>a;
        cout<<"\n Enter the value of b:";
        cin>>b;
      }
      int show()
      {
        cout<<"\n a= "<<a;
        cout<<"\n b= "<<b;
      }
};
int main()
{
   B obj;
   obj.input();
   obj.display();
   obj.get();
   obj.show();
   return 0;
}
```

Function use করে কিভাবে protected or private data inherit করা যায়?

(2)

```
#include<iostream>
using namespace std;
class A
{
  private:
    int i,j;
  public:
    void input()
    {
      cout<<"\n Enter the value of i:";
      cin>>i;
      cout<<"\n Enter the value of j:";
      cin>>j;
    }
    void display()
    {
      cout<<"\n i= "<<i;
      cout<<"\n j= "<<j;
    }
};
class B: ~~eta~~ private A
{
  private:
    int a,b;
  public:
    void get()
    {
      cout<<"\n Enter the value of a:";
      cin>>a;
      cout<<"\n Enter the value of b:";
      cin>>b;
      input();
    }
    void show()
    {
      cout<<"\n i= "<<i;
      cout<<"\n j= "<<j;
      display();
    }
};
int main()
{
  B obj;
/* obj.input();
   obj.display(); */
  obj.get();
  obj.show();
  return 0;
}
```

এই 2-টা line -এর জন্য error হবে, কারণ class A -কে privately inherit করা হয়েছে, তাই সব function-গুলো derived class B-তে এসে private হয়ে যাবে। আর private function-কে directly access করা যায় না। তাই line 2 টা কে comment করে দিলে program run করবে, ~~এছাড়া~~, private function -কে derived-class-এ call করে use করা যায়।

```
#include<iostream>
using namespace std;
class Base
{
  protected:
    int a,b;
    public:
      Base()
      {
        a=0;
        b=0;
      }
      void get()
      {
        cout<<"\n\n Enter the value of a:";
        cin>>a;
        cout<<"\n Enter the value of b:";
        cin>>b;
      }
      Base(int i, int j)
      {
        a=i;
        b=j;
      }

};
class derived: public Base
{
  protected:
      int i,j;
      public:
        derived()
        {
          i=0;
          j=0;
        }
        void input()
        { cout<<"\n Enter the value of i:";
          cin>>i;
          cout<<"\n Enter the value of j:";
          cin>>j;
        }
        derived(int w, int x, int y, int z): Base(w,x)
        {
          i=y;
          j=z;
        }

};
```

```
int main()
{
  derived obj;
  obj.get();
  obj.input();
  obj.display();
  return 0;
}
```

Multilevel Inheritance

```
class A
{

}
class B : public A
{

}
class C : protected B
{


}
```

```
#include<iostream>
using namespace std;
class A
{
  private:
    int x;
  protected:
    int y;
  public:
    int z;
};
class B : public A
{ private:
    int a;
  protected:
    int b;
  public:
    int c;
};
```

```
class C : protected B
{
 private :
      int m;
       protected:
         int n;
           public :
             int p;
             void display()
             {
                 cout << "p = " << p;
             }
};
int main()
{
    B ob;
 /* ob.a = 5;
    ob.b = 6;
    ob.c = 7;
    ob.x = 11;
    ob.y = 20;
    ob.z = 18; */
     C ob1;
   /* ob1.a = 12;
      ob1.b = 25;
      ob1.c = 3 ;
      ob1.x = 45;
      ob1.y = 16;
      ob1.z = 10; */
      ob1.p = 22;
      ob1.display();
      return 0;
}
```

Scanned by CamScanner

Multiple Inheritance :—

```
#include<iostream>
using namespace std;
class A
{
  private:
    int a;
     public:
       void input()
       {
         cout<<"\n Enter the value of a:";
         cin>> a;
       }
       void display()
       {
         cout<<"\n a= "<<a;
       }
};
class B
{
  private:
     int b;
      public:
        void input()
        {
          cout<<"\n Enter the value of b:";
          cin>>b;
        }
        void display()
        {
          cout<<"\n b= "<<b;
        }
};
class c : private A, protected B
{
  private:
     int c;
      public:
       void input()
       {
         A :: input();
         B:: input();
         cin>>c;
       }
       void display()
       {
         A :: display();
         B:: display();
         cout<<"\n c= "<<c;
       }
};
```

cout<<"\n Enter the value of c:";

```
int main()
{
  c ob;
  ob.input();
  ob.display();
   return 0;
}
```

STACK OPERATION :— (using class)

```
#include<iostream>
using namespace std;
class stack
{
  private:
    int s[30], top, max, data;
    public:
      stack()
      {
        top=-1;
        cout<<"\n Enter the max size :";
        cin>>max;
      }
      void push()
      {
        if(top==max-1)
        { cout<<"\n The stack is full!!";
        }
        else
        {
          top++;
          cout<<"\n Enter the data:";
          cin>>s[top];
        }
      }
      void pop()
      {
        if (top==-1)
        {
          cout<<"\n The stack is empty !!";
        }
        else
        {
          cout<<"\n The deleted data is :"<<s[top];
          top--;
        }
      }
      void display()
      {
        if (top==-1)
        {
          cout<<"\n The stack is empty !!";
        }
        else
        {
          for(int i=top; i>=0; i--)
          {
            cout<<s[i]<<endl;
          }
        }
      }
};
```

```
int main()
{
  stack ob;
  int i;
  while(1)
  {
   cout<<"\n\t\t STACK OPERATION \t\t\n\n";
   cout<<"\n1. PUSH operation";
   cout<<"\n2. POP operation";
   cout<<"\n3. DISPLAY";
   cout<<"\n4. EXIT";
   cout<<"\n Enter your choice:";
   cin>>i;
   switch(i)
   {
    case 1: ob.push();
            break;
    case 2: ob.pop();
            break;
    case3: ob.display();
            break;
    case 4: exit(0);
    default: cout<<"\n Wrong Choice!!";
   }
  }
  return 0;
}
```

Stack using inheritance :—

```
#include<iostream>
using namespace std;
class stack
{
  protected:
    int s[30], top, max;
    public:
      stack()
      { top=-1;
        cout<<"\n Enter the max size:";
        cin>>max;
      }
      void push()
      {
        top++;
        cout<<"\n Enter the data:";
        cin>>s[top];
      }
      void pop()
      {
        cout<<"\n Deleted data is:"<<s[top];
        top--;
```

```
class stack1 : public stack
{
  public:
     void push()
     { if (top == max-1)
          cout <<"\n stack is full!!";
       else
          stack:: push();
     }
     void ~~display~~ pop()
     {
       if (top==-1)
          cout<<"\n stack is empty !!";
       else
          stack:: pop();
     }
     void display()
     {
        for(int i = top ; i>=0 ; i--)
        {
          cout <<s[i] <<endl;
        }
     }
     }
  };
int main()
{
  stack1 ob;
  int i;
  while(1)
  {
    cout <<"\n1. PUSH \n2. POP \n3. DISPLAY \n4. EXIT";
    cout <<"\n Enter your choice :";
    cin>>i;
    switch(i)
    {
      case 1: ob.push();
              break;
      case 2: ob.pop();
              break;
      case 3: ob.display();
              break;
      case 4: exit(0);
      default: cout<<"\n Wrong Choice!!";
    }
  }
  return 0;
}
```

Queue Operation [using class] :-

```
#include <iostream>
using namespace std;
class queue
{
  private:
    int q[30], max, front, rear;
  public:
    queue()
    {
      rear = -1;
      front = -1;
      cout << "\n Enter the max size :";
      cin >> max;
    }
    void enqueue()
    {
      if(rear == max-1)
      {
        cout << "\n The queue is full!!";
      }
      else
      {
        rear++;
        cout << "\nEnter the item :";
        cin >> q[rear];
        if(rear == 0)
          front = 0;
      }
    }
    void dequeue()
    {
      if(front == -1)
      {
        cout << "\n The queue is empty !!";
      }
      else
      {
        cout << "\n The deleted element is :" << q[front];
        if(rear == front)
        {
          rear = -1;
          front = -1;
        }
        else
        {
          front++;
        }
      }
    }
    void display()
    {
      if(rear == -1)
      {
        cout << "\n The queue is empty !!";
      }
      else
      {
        for(int i = front; i <= rear; i++)
        { cout << q[i] << " "; }
      }
    }
};
```

```
int main()
{
  queue ob;
  int i;
  cout<<"\n\t\t Queue Operation \t\t\n";
  while(1)
  {
   cout<<"\n\n1. PUSH \n2. POP \n3. DISPLAY \n4. EXIT";//or,
   cout<<"\n Enter your choice : ";
   cin>>i;
   switch(i)
   {
    case 1: ob.enqueue();
            break;
    case 2: ob.dequeue();
            break;
    case 3: ob.display();
            break;
    case 4: exit(0);
   default: cout<<"\n wrong choice!!";
   }
  }
  return 0;
}
```

[cout<<"\n\n1.ENQUEUE \n2.DEQUEUE \n3.DISPLAY\n 4. EXIT]

Queue Operation using inheritance :—

```
#include <iostream>
using namespace std;
class queue
{
  protected :
      int q[30], front, rear, max;
    public :
        queue()
        { rear = -1;
          front = -1;
          cout << "\n Enter the max size :";
          cin >> max;
        }
        void enqueue()
        { rear++;
          cout << "\n Enter the item :";
          cin >> q[rear];
          if (rear == 0)
            front = 0;
        }
        void dequeue()
        {
          cout << "\n The deleted data is :" << q[front];
          if (rear == front)
          { rear = -1;
            front = -1;
          }
          else
          { front++;
          }
        }
};
class queue1 : public queue
{ public :
    void enqueue()
    { if (rear == max-1)
      { cout << "\n The queue is full !! ";
      }
      else
        queue :: enqueue();
    }
    void dequeue()
    { if (front == -1)
      { cout << "\n The queue is empty !! ";
      }
      else
        queue :: dequeue;
    }
```

```
void display()
{ if (rear==-1)
  {
    cout<<"\n The queue is empty !!";
  }
  else
  {
    for (int i=front; i<=rear; i++)
    { cout<<q[i]<<" ";
    }
  }
}
};
int main()
{ queue1 ob;
  int i;
  cout<<"\n\t\t QUEUE OPERATIONS \t\t\n";
  while(1)
  { cout<<"\n1. Enqueue \n2. Dequeue \n3. Display \n 4. Exit";
    cout<<"\n Enter your choice:";
    cin>>i;
    switch(i)
    { case1: ob.enqueue();
             break;
      case 2: ob.dequeue();
             break;
      case3: ob.display();
             break;
      case4: exit(0);
             ~~break~~
      default: cout<<"\n Wrong choice !!";
    }
  }
  return 0;
}
```

Example (Multi-level inheritance):—

```
#include<iostream>
using namespace std;
class A
{ public:
   A()
    {
      cout << "\n In constructor A";
    }
   ~A()
    {
      cout << "\n In destructor B";
    }
};
class B : public A
{public:
   B()
    {
      cout << "\n In constructor B";
    }
  ~B()
    {
      cout << "\n In destructor B";
    }
};
class C : public B
{ public:
    C()
    {
      cout << "\n In constructor C";
    }
    ~C()
    {
      cout << "\n In destructor C";
    }
 };
int main()
{
   C ob;
}
```

main function-এ object create হলে, constructor call হয়। First-এ C-class-এর through B class, তারপর B class-এর through A class-এর constructor call হবে (কারণ, C-class, B-এর derived class; আবার C-এর base class B, A class-এর derived class। তাই আগে class A, তারপর class B আর তারপর class C-এর constructor call হবে।
Destructor call হয় reverse ভাবে। অর্থাৎ, আগে class C-এর, তারপর B আর তারপর class A-এর destructor call হয়।

Virtual Function :——

```
#include <iostream>
using namespace std;
class A
{ public:
      virtual void display()
       { cout << "In A";
       }
};
class B : public A
{ public:
      void display()
       { cout << "In B";
       }
};
class C : public A
{ public:
      void display()
      { cout << "In C";
      }
};
int main()
{  A *p;
   B ob1;
   C ob2;
   p = &ob1;
   p -> display(); // A class-এর display call হবে, যদি void display-কে virtual করা না হয়।
   p = &ob2;
   p -> display(); // A class-এর display call হবে, যদি void display-কে virtual করা না হয়।
   return 0;
}
```

- যদি virtual mention করা থাকে, তাহলে B আর C-এর content display করবে। Otherwise, class A-এর content display করবে।
- virtual mention করা না থাকলে, P-এ type check করবে, সেহেতু, P class A-এর, তাই প্রথম p-কে display করলে, A class-এর display call হবে, similarly for 2nd.

Pure Virtual Function :—

```
class A
{
  public:
     virtual void display()=0; // This line implies that, the function is a pure virtual function.
};
class B : public A
{
  public :
     void display() // এই function-টি derive করা না থাকলে, overwrite হয়ে যাবে class B টির abstract হয়ে যাবে।
     {

     }
};
```

- Pure virtual function has no "body".
- কোনো class-এর ভেতর pure virtual function থাকলে, তাকে abstract-class বলে।
- abstract class-এর object তৈরী করা যায়না।
- class-এ যতগুলো member function থাকে, তাদের মধ্যে যে-কোনো একটি pure virtual হলেও abstract হয়ে যায়।

[Important: Virtual Function (maintenance Vehicle) → Kanitkar]

Pre-increment and Post increment Operator :—

```
# include<iostream>
 using namespace std;
 class test
 { private:
     int i;
     public:
       test()
       { i=0;
       }
       test operator++() //pre-increment
       { test ob;
         ob.i = ++i;
         return (ob);
       }
       testoperator++(int) //post increment
       { test obj;
         obj.i = i++;
         return (obj);
       }
```

```
void display()
 {
   cout<<i;
 }
};
int main()
 {
   test a,b,c,d;
   b=++a;   // a.operator++()
   b.display();
   a.display();
   d=c++;   //c.operator++()
   d.display();
   c.display();
   return 0;
 }
```

2.

1. Write a program to calculate area and perimeter of a circle, a triangle, a square and a rectangle using virtual function.

```
#include<iostream>
using namespace std;
```

```
#include<iostream>
#include<math.h>
using namespace std;
class shape
{
  protected:
      float a, p;
      public:
         virtual void area()
         {

         }
         virtual void perimeter()
         {

         }
};
class square : public shape
{
  private:
      float s;
        public:
           virtual void area
            square()
            {
             cout<<"\n\n Enter the value of the sides of the square:";
             cin>>s;
             a=0;
              p=0;
            }
            void area()
            {
               a=s*s;
               cout<<"\n\nThe area of the square is: "<<a;
            }
            void perimeter()
            {
               p = 4 *s ;
               cout<<"\n\n The perimeter of the square is: "<<p;
            }
};
```

```
class triangle : public shape
{ private:
    float x,y,z,s;
    public:
        triangle()
        {
          cout<<"\n\n Enter the value of 1st side of the triangle: ";
          cin>>x;
          cout<<"\n Enter the value of 2nd side of the triangle: ";
          cin>>y;
          cout<<"\n Enter the value of 3rd side of the triangle: ";
          cin>>z;
           s=0;
           a=0;
           p=0;
        }
        void area()
        {
           s=(x+y+z)/2;
           a=sqrt(s*(s-x)*(s-y)*(s-z));
           cout<<"\n\n The area of the triangle is: "<<a;
        }
        void peremetor()
        {
           p=(x+y+z);
           cout<<"\n\n The peremetor of the triangle is: "<<p;
        }
};
class rectangle : public shape
{ private:
    float l,b;
    public:
      rectangle()
      {
       cout<<"\n\n Enter the value of the length of the rectangle: ";
       cin>>l;
       cout<<"\n Enter the value of the breadth of the rectangle: ";
       cin>>b;
        a=0;
        p=0;
      }
      void area()
      {
         a=l*b;
         cout<<"\n\n The area of the rectangle is: "<<a;
      }
      void perimetor()
      {
         p=2*(a+b);
         cout<<"\n\n The perimetor of the rectangle is: "<<p;
      }
};
```

```
class circle : public shape
{
  private:
      float r;
      public:
          circle()
          {
           cout <<"\n Enter the value of radius of the circle :";
           cin >>r;
           a=0;
           p=0;
          }
          void area()
          {
             a = 3.14*r*r;
             cout<<"\n The area of the circle is : "<<a;
          }
          void perimeter()
          {
             p = 2*3.14*r;
             cout<<"\n The perimeter of the circle is : "<<p;
          }
};
int main()
{
    shape *p;
    int i;
    while(1)
    {
      cout<<"\n\n 1. Square \n 2. Triangle \n 3. Rectangle \n 4. Circle
            \n 5. Exit \n";
      cout<<"\n Enter your choice : ";
      cin>>i;
      switch(i)
      {
        case 1: {
                  square s;
                  p=&s;
                  p->area();
                  p->perimeter();
                  break;
                }
        case 2: {
                  triangle T;
                  p=&T;
                  p->area();
                  p->perimeter();
                  break;
                }
```

```
        case 3: {
                    rectangle R;
                    P = &R;
                    P->area();
                    P->perimeter();
                    break;
                }
        case 4: {
                    circle C;
                    P = &C;
                    P->area();
                    P->perimeter();
                    break;
                }
        case 5: exit(0);
        default : cout<<"\n Wrong Choice!!";
    }
  }
  return 0;
}
```

Singly Link-list:

```
#include<iostream>
using namespace std;
class node
{
  protected:
      int data;
      node * next;
      public:
         node()
         {
           next = NULL;
         }
         void setdata(int x)
         { data=x;
         }
         void setnext(node *p)
         {
           next = p;
         }
         int rdata()
         {
           return(data);
         }
         node * rnext()
         {
            return(next);
         }
};
class LL : public node
{
  private:
      node * head, * last;
      public:
         void insert_beg();
         void insert_end();
         void insert_afterpos();
         void insert_aftervalue();
         void delete_beg();
         void delete_end();
         void delete_afteranypos();
         void display();
         LL()
         {
           head = NULL;
         }
};
```

```
void LL :: insert_beg()
{
    node *P;
    int n;
    p= new node();
    cout<<"\n Enter the data: ";
    cin>>n;
    p->setdata(n);
    if(head==NULL)
    {
       head=p;
       head->setnext(NULL);
       Last = head;
    }
    else
    {
      p->setnext(head);
      head=p;
    }
    cout<<endl<<p->rdata()<<" is inserted"<<endl;
}
void LL:: insert_end()
{
    node *p;
    int n;
    p= neonode();
    cout<<"\n Enter the data: ";
    cin>>n;
    p->setdata(n);
    if(head==NULL)
    {
      head=P;
      head->setnext(NULL);
      Last = head;
    }
    else
    {
      Last->setnext(p);
      Last = p;
    }
    cout<<endl<<p->rdata()<<" is inserted"<<endl;
}
void LL:: insert_afterpos()
{
    node *p, *q;
    int n, i, pos;
    p= neonode();
    cout<<"\n Enter the data:";
    cin>>n;
    p->setdata(n);
    cout<<"\n Enter the position after which you want to insert:";
    cin>>pos;
```

```
    if(head==NULL)
    {
      head=p;
      head->setnext(NULL);
      last=head;
    }
    else
    {
      q=head;
      i=1;
      while(i!=pos && q->nnext()!=NULL)
      {
        i=i+1;
        q=q->nnext();
      }
      if(i==pos && q->nnext()==NULL)
      {
        q->setnext(p);
        last=p;
      }
      else if(i==pos && q->nnext()!=NULL)
      {
        p->setnext(q->nnext());
        q->setnext(p);
      }
      else
      {
        cout<<"\n The position is not found !!";
      }
    }
}
void LL::insert_aftervalue()
{
  node *p,*q;
  int n, val;
  p=new node();
  cout<<"\n Enter the data:";
  cin>>n;
  p->setdata(n);
  cout<<"\n Enter the value after which you want to insert: ";
  cin>>val;
  if(head==NULL)
  {
    head=p;
    head->setnext(NULL);
    last=head;
  }
  else
  {
    q=head;
    while(val!=q->ndata() && q->nnext()!=NULL)
    {
      q=q->nnext();
    }
```

```
        if (val == q->ndata() && q->ndata() == NULL)
        {
          q->setnext(p);
          last = p;
        }
        else if (val == q->ndata() && q->ndata() != NULL)
        {
            p->setnext(q->next());
            q->setnext(p);
        }
        else
        {
          cout << "\n The value is not found !!";
        }
    }
}
void LL::delete_beg()
{
    node *p = head;
    if (head == NULL)
    {
      cout << "\n The list is empty !!";
    }
    else
    {
        head = head->next();
        p->setnext(NULL);
        cout << "\n The deleted element is : " << p->ndata();
    }
}
void LL::delete_end()
{
    node *p = head;
    if (head == NULL)
    {
      cout << "\n The list is empty !!";
    }
    else
    {
      while (p->next() != last)
      {
        p = p->next();
      }
      p->setnext();
      p->setnext(NULL);
      cout << "\n The deleted element is : " << last->ndata();
    }
    last = p;
}
```

```
void LL::delete_afteranypos()
{
  node *p=head;
  node *q;
  int pos,i;
  cout<<"\n Enter the position:";
  cin>>pos;
  if(head==NULL)
  {
    cout<<"\n The list is empty !!";
  }
  else
  {
    i=1;
    q=p;
    if(pos==1)
    {
      head=q->nnext();
      p->setnext(NULL);
    }
    else
    {
      while(i!=pos && q->nnext()!=NULL)
      { i=i+1;
        q=q->nnext;
      }
      if(i==pos && q->nnext()==NULL)
      {
        q->setnext(NULL);
        cout<<"\n The deleted data is :"<<q->ndata();
      }
      else if(i==pos && q->nnext()!=NULL)
      {
        q=q->nnext();
        p->setnext(q->nnext());
        q->setnext(NULL);
        cout<<"\n The deleted data is:"<<q->ndata();
      }
    }
  }
}
void LL::display()
{
  node *p=head;
  if(head==NULL)
  {
    cout<<"\n Empty list !!";
  }
  else
  {
    cout<<"\n List is: ";
```

```
        while(p!=NULL)
        {
            cout<<p->ndata()<<"->";
            p=p->nnext();
        }
        cout<<" NULL";
    }
}

int main()
{
    LL ob;
    int i;
    cout<<"\n\t\t\t SINGLY LINKLIST \t\t\t\n\n";
    while(1)
    {
        cout<<"\n 1. Insert at beginning ";
        cout<<"\n 2. Insert at end ";
        cout<<"\n 3. Insert at any position";
        cout<<"\n 4. Insert after any value";
        cout<<"\n 5. Delete from beginning ";
        cout<<"\n 6. Delete from end";
        cout<<"\n 7. Delete from after any position";
        cout<<"\n 8. Display";
        cout<<"\n 9. Exit";
        cout<<"\n Enter your choice:";
        cin>>i;
        switch(i)
        {
            case 1: ob.insert_beg();
                    break;
            case 2: ob.insert_end();
                    break;
            case 3: ob.insert_afterpos();
                    break;
            case 4: ob.insert_aftervalue();
                    break;
            case 5: ob.delete_beg();
                    break;
            case 6: ob.delete_end();
                    break;
            case 7: ob.delete_afteranypos();
                    break;
            case 8: ob.display();
                    break;
            case 9: exit(0);
            default: cout<<"\n Wrong choice!";
        }
    }
    return 0;
}
```

1. When a class makes virtual ?

```
#include<iostream>
using namespace std;
class Base
{
  protected:
     int x;
     public:
      int y;
}
class Base1 : public Base.
{
  public:
    int i;
};
class Base2 : public Base
{
  public:
    int j;
};
class derived : public Base1, public Base2
{
  public:
    int k;
    void show()
    {
      cout << k;
      cout <<x<< y;
      cout << i <<j;
    }
}
```

```
#include<iostream>
using namespace std;
class Base
{ protected:
    int x;
    public:
     int y;
}
class Base1 : virtual public Base
{
  public:
    int i;.
};
class Base2 : virtual public Base
{
  public:
    int j;
};
class derived : public Base1, public Base
{
  public:
    int k;
    void show()
    {
      cout << k;
      cout << x<< y;
      cout << i << j;
    }
}
```

virtual করা হয়েছে as class Base1-এ x, y inherit করে আসবে, আবার class Base2-তেও x, y inherit হয়ে আসবে। তাই derived class-এ যেন x, y-ই inherit হয়ে আসবে সেটা নিয়ে confusion হবে, তাই virtual use করা হবে। যাতে direct base থেকে x, y-কে inherit করা যায়।

Template

—— Template Function

—— Template Class (Generic class)

1. Template class

```
template <class T>
void f1 (T x)
{
  T a;
  cin>>a;
  a=a+x;
  cout <<a;
}
void main()
{
  int x=6;
  f1(x);
  float b=5.5;
  f1(b);
}
```

2.
```
#include<iostream>
using namespace std;
template <class T1, class T2>
void f1 (T1 x, T2 y)
{
    cout<<x<<y;
}
template <class T>
void f1 (T1 x)
{
  cout<<x;
}
int main()
{
  int a=5;
  float b=2.4;
  char x='a';
  f1(a,x);
  f1(x);
  f1(b);
  f1(x,b);
  return 0;
}
```

3.

```
#include <iostream>
using namespace std;
template <class T>
int * lsearch (T *a, int n, T key)
{
  int i, f=0;
  for(i=0; i<n ; i++)
  {
    if (a[i] == key)
    {
      f=1;
      break;
    }
  }
  if(f=1)
    return(i);
  else
    return(-1);
}
int main()
{
  int *a, key, n, i;
  cin>> key;
  cin>>n;
  a = new int [n];
  for (i=0; i<n; i++)
  {
    cin >>a[i];
  }
  i= lsearch(a, n, key);
  if (i==-1)
    cout<<" Not found ";
  else
    cout << " Found at"<< i;
}
```

Stack using Template :—

```
#include<iostream>
using namespace std;
template<class T>
class stack
{
  private:
    T s[30];
    int top, max, data;
  public:
    stack()
    {
      cout<<"\n Enter the max size:";
      cin>>max;
      top=-1;
    }
    void push()
    {
      if(top==max-1)
        cout<<"\nStack is full !! ";
      else
      {
        cout<<"\n Enter data: ";
        cin>>data;
        top++;
        s[top]=data;
      }
    }
    void pop()
    {
      if(top==-1)
        cout<<" \n Stack is empty !! ";
      else
      {
        data=s[top];
        top--;
        cout<<" The deleted data is: "<<data;
      }
    }
    void display()
    {
      if(top==-1)
        cout<<"\nStack is empty !! ";
      else
      {
        for(int i=top;i>=0;i--)
        {
          cout<<s[i]<<endl;
        }
      }
    }
```

चरहरू push(), pop() अनि display() function हरुको definition यस्तो हुन्छ,

```
template<class T>
void stack<T>::push()
void stack<T>::pop()
void stack<T>::display()
```

```
int main()
{
  int i,j;
  while(1)
  {
   cout<<"\n Stack using ~~inheritor~~ template \n";
   cout<<"\n 1. PUSH \n2. POP \n3. DISPLAY \n 4. EXIT \n";
   cout<<" \n Enter your choice :";
   cin>>j;
   switch(j)
   {
    case 1: ob1.push();
            break;
    case 2: ob1.pop();
            break;
    case 3: ob1.display();
            break;

    case 4: break;

    ~~display~~
    default: cout<<"\n Wrong choice !! ";
   }
   if(j==4)
      break;
  }
  break;
  return 0;
}
```

EXCEPTION HANDLING

28/10/2018

Syntax:

```
try
{
  // try BLOCK
}
catch (args)
{
  // catch back
}
main()
{
  int i = 1;
  try
  { if (i)
      throw(i);
      cout << "HELLO"; //
  }
  catch (int x)
  {
    cout << x;
  }
}
```

1.
```
main()
{
  try
  {
    f1(0);
    f1(1);
    f1(2);
  }
  catch(int i)
  {
    cout << i;
  }
}
```

```
f1(int x)
{ if (x)
    throw(x);
}
```

main-এর পরে try-এর মধ্যে, f1 function-এ যাবে, 0 প্রথমের x-এর মধ্যে 0 store হবে, তারপর check করবে if(0). As if(0) মানে condition-টা "False" তাই throw করবেনা। আবার control পরের f1' function-এ ফিরে আসবে। আবার control function-এ যাবে, parameter হিসাবে 1 যাবে, if(1) অর্থাৎ true হবে, তারপর throw(x) অর্থাৎ 1 value কে throw করবে। catch function-এর মধ্যে ঢুকে i-কে print করবে, অর্থাৎ, এই program-টার o/p 1 হবে। 0 আর 2 হবেনা। যেহেতু 0 হলে condition False হয়। আর 1 print করার পর control আর back করবেনা যেহেতু terminate হয়ে যায়।

1

2.

```
#include<iostream>
using namespace std;
class test
{ private:
  char *s;
  int x;
   public:
     test()
     {
       x=0;
       s="Hello";
     }
     test(char *s1, int j)
     {
       strcpy (s,s1);
       x=j;
       cout<<s1<<x;
     }
};
void main()
{
  test obj ("Hello",6);
  int x=1;
  try
  {
    if(x==1)
       throw test();
    else if (x==2)
       throw test ("error",6);
  }
  catch(test ob)
  {
   cout<<"Inside catch";
  }
}
```

*s - আমরা string-এর main address.

test() → default constructor.

test(char *s1, int j) → parameterized constructor.

main()-এ, test type-এর একটা obj create করা হলো যার মধ্যে as a string "Hello" আর একটা number 6 pass করা হলো।

x- নামের একটা variable নেওয়া হলো, যাকে 1 দিয়ে initialize করা হয়েছে।

try → এ ঢোকার পর, if condition check করবে।

as, এখানে প্রথমেই x কে 1 দিয়ে initialize করা হয়েছে, তাই (x==1) condition-টি true হবে।

[এরপর test class-এর test parameterized test function-এ control-টি throw করবে। সেখানে, *s-এর মধ্যে "Hello" string টির base address আর j-এর মধ্যে 6 যাবে।

যেহেতু কোনো string-কে direct copy করা যায়না, তাই base এ "strcpy" function-টি use করা হয়েছে। আর x-এর মধ্যে j-এর মধ্যে থাকা value টি copy হয়ে, "Hello" আর 6-কে print করবে]

—এরকম হওয়ার কথা ছিল, কিন্তু হয়নি। যেহেতু throw করার সময় কোনো parameter দেওয়া হয়নি, তাই একটা temporary object catch করে "Inside catch" print করাবে।

আর "error", 6 print করাবে।

Derived class exception :——

```
class base
{

};
class derived:: public base
{

};
int main()
{
  derived ob;
  try
  {
    throw derived();   // temporary object create হবে। এটা Throw করলে derived
                       catch করে করবে।
  }
  catch (base obj)
  {

  }
  catch (derived obj) // object type "derived" হওয়ায়।
  {


  }
};
```

Catching all exception :——

... → catching all exception.

```
main()
{ int i;
  cin>>i;
  try
  {
    if (i<0)
      throw 5 ;  // 6.12, a, 7123014.... [Without "string"]
  }
  catch(...)
  {
    cout << " caught one exception";
  }
}
```

Rethrowing :—

```
void f1(int x, int y)
{
  try
  {
    if (x>48 && y<98)
        throw x;   // 1st throw
  }
  catch(int x)
  {
    cout << "caught" << x;
    throw;          // again throw হওয়ার জন্য rethrowing এই throw কে again
  }                    x-কে throw করবে।
}
void main()
{
  int x, z;
  cin>> x>> z;
  try
  {f1(z, x);
  }
  catch(int z)  //second throw - এই catch - এ catch করবে।
  {
    cout << z;
  }
}
```

- matrix overloading (H.W)

"FILE"

```
Read mode
if stream in  ──────────► class
in.open("Input.text");
if stream in("Input.text");

of stream out
out.open("output.text");
of stream out("output.text");
Write mode
```

Reading from file :—

```
void main()
{
  if stream in;
  in.open("Input.txt");
  if(!in)
  {
    cout << "\n can't open file!";
    exit(0);
  }
  int x,y;
  in >> x >> y;
  cout << x << y;
  in.close();
}
```

```
void main()
{
  if stream in;
  in.open("Input.txt");
  if(!in)
  {
    cout << "\n can't open file!";
    exit(0);
  }
  int x,y;
  while(in)
  {
    in >> x >> y;
    cout << x << y;
  }
  in.close();
}
```

Writing From file :—

```
void main()
{
  of stream out;
  out.open("output.text");
  if(!out)
  {
    cout << "can't open file!";
    exit(0);
  }
  int x,y;
  while((cin >> x) > 0)
  {
    cin >> y;
    out << x << y << endl;
  }
  out.close();
}
```

```
void main()
{
  of stream out;
  out.open("output.text");
  if(!out)
  {
    cout << "\n Can't open file!!";
    exit(0);
  }
  int x,y;
  char ch;
  do
  {
    cin >> x >> y;
    out << x << y << endl;
  } while(ch=='y');
```