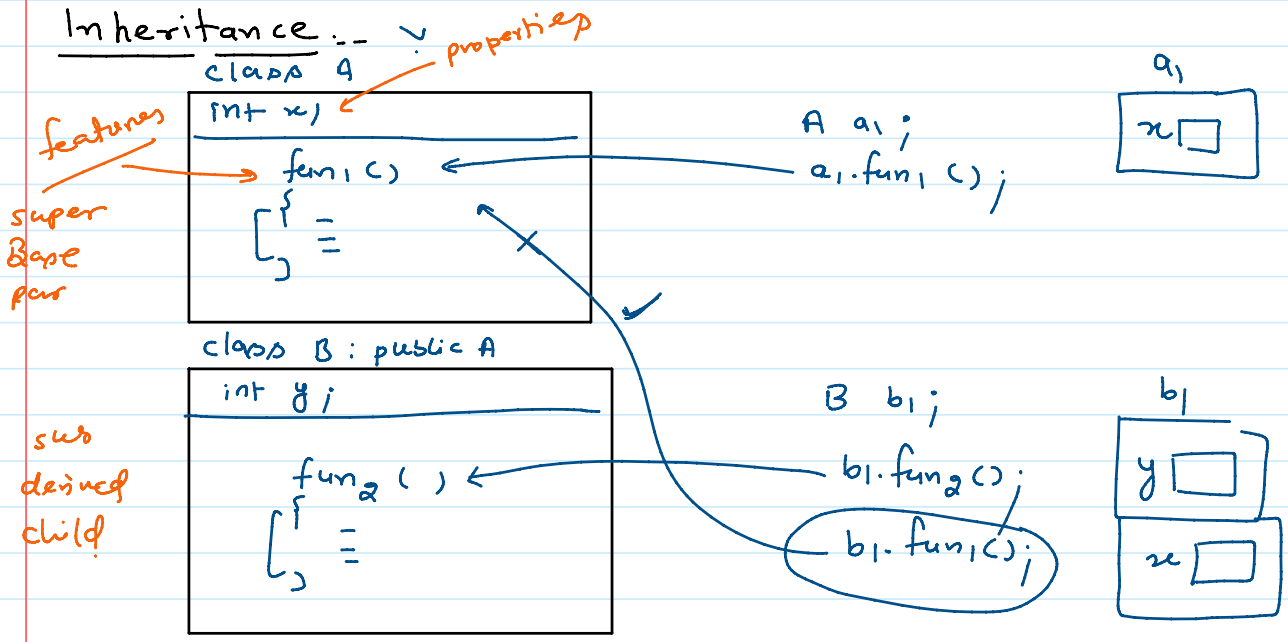
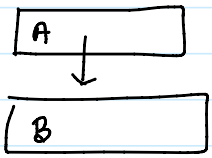
Day 10

16 April 2024 01:01 PM

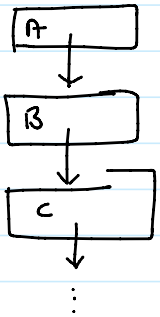


Types of Inheritance

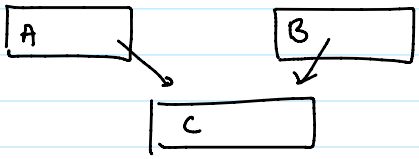
1. Single level inheritance



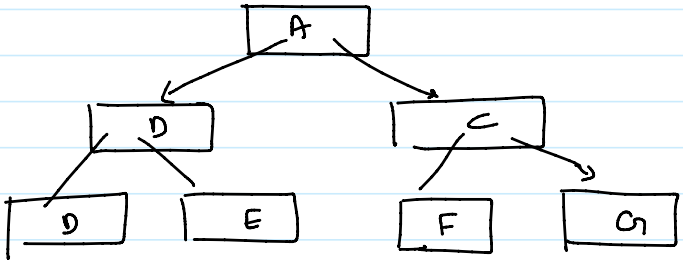
2. Multi level Inheritance



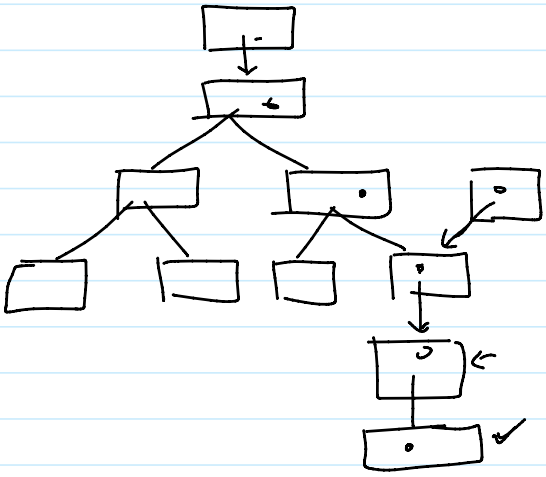
3. Multiple inheritance



4. (Tree) | Heirarchical inheritance



5. Hybrid Inheritance



.

```
#include<iostream>
using namespace std;
class A
{
    int x;
    public:
        A()
        {
            cout<<"Default constructor of class A\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Parameterized constructor of class A\n";
        }
        void output()
        {
            cout<<"X="<<x<<" ";
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
```

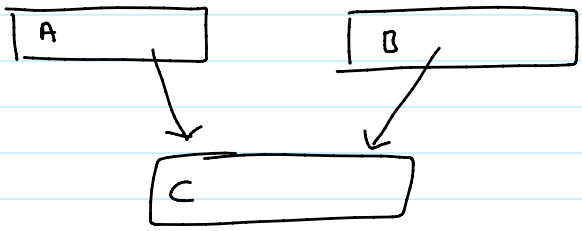
```
                cout<<"Default constructor of class B\n";
            }
            B(int x1, int y1):A(x1)
            {
                y=y1;
                cout<<"Parameterized constructor of class B\n";
            }
            void output()
            {
                A::output();
                cout<<"Y="<<y<<endl;
            }
    };
    int main()
    {
        B b1;
        B b2(10,20);
        b2.output();
    }
```

.

```
#include<iostream>
using namespace std;
class A
{
    int x;
    public:
        A()
        {
            cout<<"Default constructor of class A\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Parameterized constructor of class A\n";
        }
        void output()
        {
            cout<<"X="<<x<<" ";
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
            cout<<"Default constructor of class B\n";
        }
        B(int x1, int y1):A(x1)
        {
            y=y1;
            cout<<"Parameterized constructor of class B\n";
        }
        void output()
        {
            A::output();
            cout<<"Y="<<y<<" ";
        }
};
class C:public B
{
    int z;
    public:
```

```
        C():B()
        {
            cout<<"Default constructor of class C\n";
        }
        C(int x1, int y1, int z1):B(x1,y1)
        {
            z=z1;
            cout<<"Parameterized constructor of class C\n";
        }
        void output()
        {

            cout<<"Z="<<z<<endl;
        }
};
int main()
{
    C c1;
    C c2(10,20,30);
    c2.output();
}
```



.

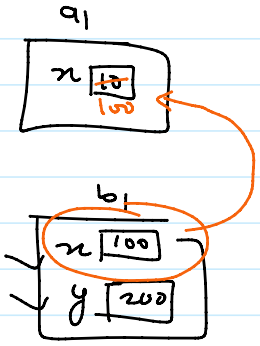
```
#include<iostream>
using namespace std;
class A
{
    int x;
    public:
        A()
        {
            cout<<"Default constructor of class A\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Parameterized constructor of class A\n";
        }
        void output()
        {
            cout<<"X="<<x<<" ";
        }
};
class B{
    int y;
    public:
        B()
        {
            cout<<"Default constructor of class B\n";
        }
        B(int y1)
        {
            y=y1;
            cout<<"Parameterized constructor of class B\n";
        }
        void output()
```

```
        {
            cout<<"Y="<<y<<" ";
        }
};
class C:public A,public B
{
    int z;
    public:
        C():B(),A()
        {
            cout<<"Default constructor of class C\n";
        }
        C(int x1, int y1, int z1):B(y1),A(x1)
        {
            z=z1;
            cout<<"Parameterized constructor of class C\n";
        }
        void output()
        {
            A::output();
            B::output();
            cout<<"Z="<<z<<endl;
        }
};
int main()
{
    C c1;
    C c2(10,20,30);
    c2.output();
}
```

object slicing :-

A $a_1 = 10;$

B $b_1(100,200);$



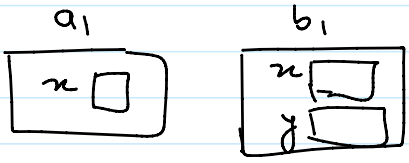
C.O $b_1 = a_1;$ P.O ✗

P.O $a_1 = b_1;$ C.O ✓

Pointer :-

A $a_1;$

B $b_1;$



A $*p_1;$ $p_1 = \&a_1;$

B $*p_2;$ $p_2 = \&b_1;$

$p_2 = \&a_1;$ ✗

C.P = & P.O ✗

$p_1 = \&b_1;$ ✓

P.P = & C.O ✓

protected :-

```
#include<iostream>
using namespace std;
class A
{
    protected:
        int x;
    public:
        A()
        {
            cout<<"Default constructor of class A\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Parameterized constructor of class A\n";
        }
        void output()
        {
            cout<<"X="<<x<<" ";
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
            cout<<"Default constructor of class B\n";
        }
        B(int x1,int y1):A(x1)
        {
            y=y1;
            cout<<"Parameterized constructor of class B\n";
        }
        void output()
        {
            //A::output();
            cout<<"X="<<x<<endl;
            cout<<"Y="<<y<<end;
        }
};
int main()
{
    B b1(10,20);
    b1.output();
    //b1.A::x;      error
```

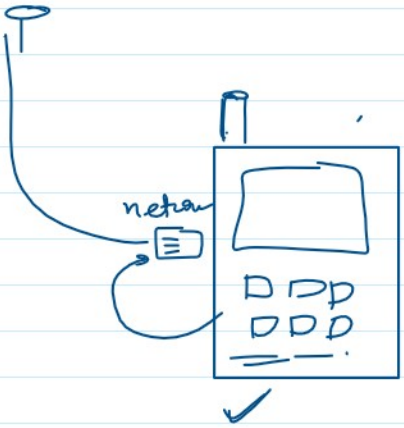
}

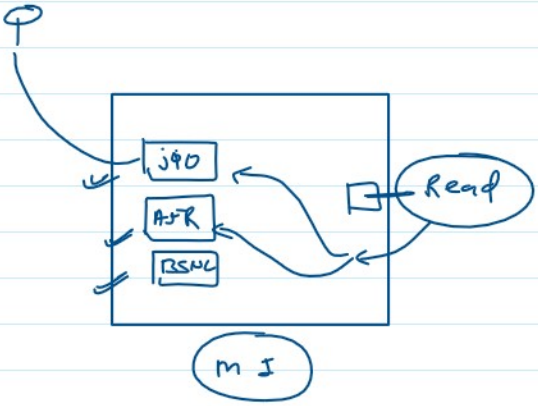
virtual function | Dynamic Binding | Run time polymorphism.

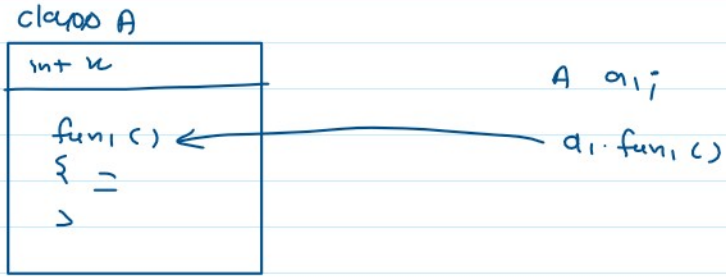
default

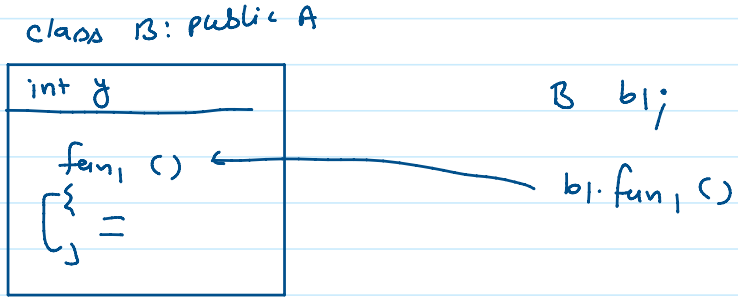
Compile time Binding (static Binding)

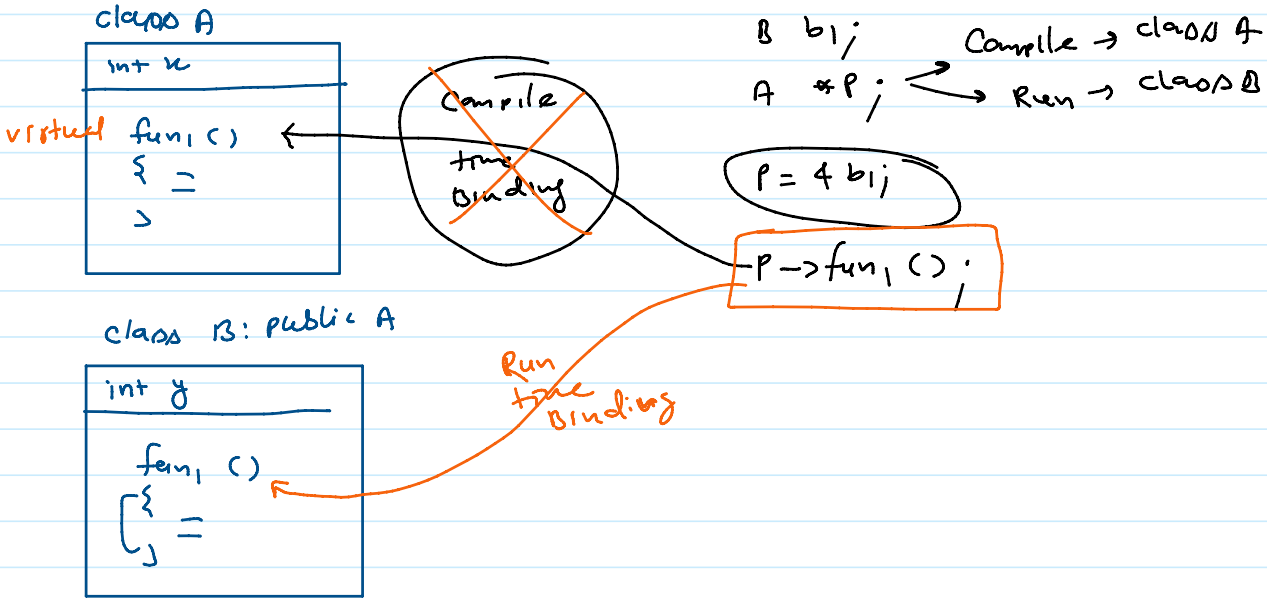
Run time Binding (Dynamic Binding)





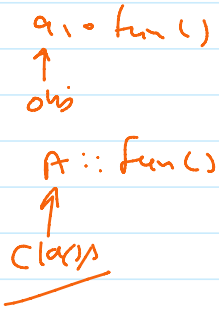






Static fun

```cpp
#include<iostream>
using namespace std;
class A
{
    public:
        virtual void fun1()
        {
            cout<<"fun1 of class A\n";
        }
};
class B:public A{
    public:
        void fun1()
        {
            cout<<"fun1 of class B\n";
        }
};
int main()
{
    A *p;
    B b1;
    p=&b1;
    p->fun1();
}
```



```cpp
#include<iostream>
using namespace std;
class mobile    //abstract class
{
    public:
        virtual void network()=0;   //pure virtual function
};
class BSNL:public mobile
{
    public:
        void network()
        {
            cout<<"Class BSNL\n";
        }
};
class jio:public mobile
{
    public:
        void network()
        {
            cout<<"Class jio\n";
        }
};
int main()
{
    mobile *p;
    int n;
    cout<<"1. BSNL\n2. jio ";
    cin>>n;
    if(n==1)
        p=new BSNL;
    else
        p=new jio;
    p->network();
    //mobile mi;        error
    delete p;
}
```

STL (standard template library)

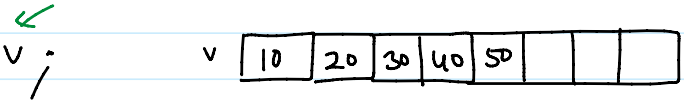
1. Iterator (Pointer)
2. Container (store) → vector, stack, queue, list, set, map)
3. Algorithm
4. Functors

#include <bits/stdc++.h>

vector :-

#include <vector>

class → vector <int> v; (template: int; object: v)



```
v.push_back(10);
v.push_back(20);
v.push_back(30);
v.push_back(40);
v.push_back(50);
```

No of elements → `v.size();` → 5

`v.capacity();` → 8

Iterator :-

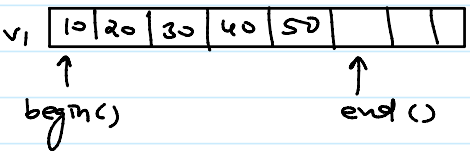
1. iterator — begin(), end() | Read & write
2. reverse iterator — rbegin(), rend() | read & write
3. constant iterator → cbegin(), cend() | read only
4. constant Reverse iterator → crbegin(), crend() | read only

iterator :—

```
vector<int> :: iterator i;
```

begin() → & first element

end() → & last + 1 element




```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> v1;
    cout<<"Size="<<v1.size()<<endl;
    cout<<"Capacity="<<v1.capacity()<<endl;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
```

```
        v1.push_back(50);
        cout<<"Size="<<v1.size()<<endl;
        cout<<"Capacity="<<v1.capacity()<<endl;
        vector<int>::iterator i;
        for(i=v1.begin();i!=v1.end();i++)
        {
            *i += 2;
            cout<<*i<<" ";
        }
        cout<<endl;
        vector<int>::reverse_iterator j;
        for(j=v1.rbegin();j!=v1.rend();j++)
        {
            *j -= 2;
            cout<<*j<<" ";
        }
        cout<<endl;
        vector<int>::const_iterator k;
        for(k=v1.cbegin();k!=v1.cend();k++)
        {
            //*k += 2;
            cout<<*k<<" ";
        }
        cout<<endl;
        vector<int>::const_reverse_iterator l;
        for(l=v1.crbegin();l!=v1.crend();l++)
        {
            //*l += 2;
            cout<<*l<<" ";
        }
        cout<<endl;
        for(auto x=v1.crbegin();x!=v1.crend();x++)
        {
            cout<<*x<<"  ";
        }

    }
```

Vector :- (Dynamic Array)

```
vector<int> v1;
```

v1

```
vector<int> v1 = {10,20,30,40,50};
```

v1 | 10 | 20 | 30 | 40 | 50 |

```
vector<int> v1(5);
```

v1 | 0 | 0 | 0 | 0 | 0 |

```
vector<int> v1(5,10);
```

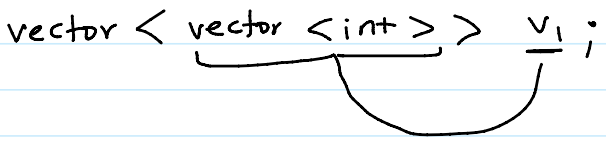
v1 | 10 | 10 | 10 | 10 | 10 |

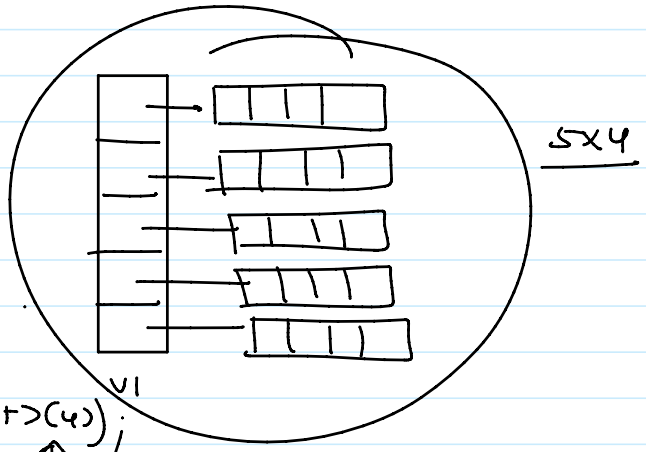
default Value

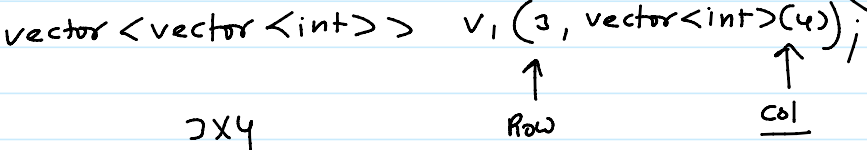
```
#include<iostream>
```

```cpp
#include<vector>
using namespace std;
void output(vector<int> &v)
{
    // for(int i=0;i<v.size();i++)
    //     //cout<<v.at(i)<<" ";
    //     cout<<v[i]<<" ";
    for(int i:v)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    vector<int> v1={10,20,30,40,50};
    cout<<"Size "<<v1.size()<<endl;
    cout<<"Capacity "<<v1.capacity()<<endl;
    v1.push_back(60);
    cout<<"Size "<<v1.size()<<endl;
    cout<<"Capacity "<<v1.capacity()<<endl;
    v1.shrink_to_fit();
    cout<<"Size "<<v1.size()<<endl;
    cout<<"Capacity "<<v1.capacity()<<endl;
    output(v1);
    v1.insert(v1.begin()+2,100);
    output(v1);
    v1.pop_back();
    output(v1);
    v1.erase(v1.begin()+3);
    output(v1);
    vector<int> v2={100,200,300};
    v1.swap(v2);
    output(v1);
    output(v2);
    v2.erase(v2.begin()+2,v2.end());
    output(v2);


}
```







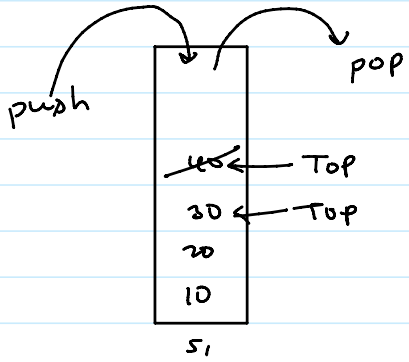
-

```cpp
#include<iostream>
#include<vector>
using namespace std;
void output(vector<int> &v)
{
```

```
    // for(int i=0;i<v.size();i++)
    //     //cout<<v.at(i)<<" ";
    //     cout<<v[i]<<" ";
    for(int i:v)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    vector<vector<int>> v1={
                            {10,20,30,40},
                            {11,12,13,14},
                            {21,22,23,24},
                            {31,32,33,34}
                            };
    for(vector<int> i:v1)
    {
        for(int j:i)
            cout<<j<<" ";
        cout<<endl;
    }
}
```

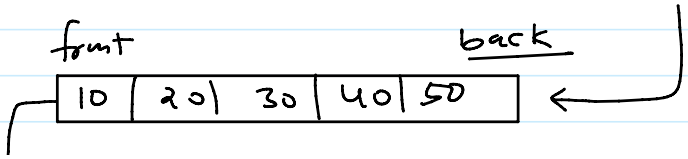
Stack :→ (LIFO)

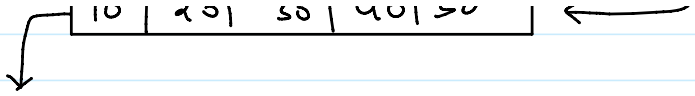
stack <int> s1;




```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
    stack<int> s1;
    s1.push(10);
    s1.push(20);
    s1.push(30);
    s1.push(40);
    s1.push(50);
    while(!s1.empty())
    {
        cout<<s1.top()<<endl;
        s1.pop();
    }
}
```

queue (FIFO)





.

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main()
{
    queue<int> q1;
    q1.push(10);
    q1.push(20);
    q1.push(30);
    q1.push(40);
    q1.push(50);
    cout<<q1.front()<<endl;
    cout<<q1.back()<<endl;
    q1.pop();
    q1.push(60);
    cout<<q1.front()<<endl;
    cout<<q1.back()<<endl;
}
```

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main()
{
    priority_queue<int> q1;
    q1.push(10);
    q1.push(50);
    q1.push(20);
    q1.push(30);
    while(!q1.empty())
    {
        cout<<q1.top()<<endl;
        q1.pop();
    }
}
```