**//Bubble sort**

template <class T>

void bubble(T \*L, T \*R, bool cmp(T,T)=[](T a, T b){return a<b;}){

for(T \*p=L; p<R; p++)

for(T \*q=R-1; q>p; q--)

if(cmp(\*q,\*(q-1))) swap(q[0], q[-1]); //swap(\*q,\*(q-1));

}

**//Selection sort**

template <class T>

void select(T \*L, T \*R, bool cmp(T,T)=[](T a, T b){return a<b;}){

for(T \*p=L; p<R; p++){

T\*r=p;

for(T\*q=p;q<R;q++) if(cmp(\*q,\*r)) r=q;

swap(\*p,\*r);

}

}

**//Insetion sort**

template <class T>

void insert(T \*L, T \*R, bool cmp(T,T)=[](T a,T b){return a<b;}){

for(T \*p=L+1; p<R; p++){

T x=\*p, \*q=p-1;

while(q>=L && \*q>x) \*(q+1)=\*q--;

\*(q+1)=x;

}

}

**//Quick Sort**

**template <class T>**

void quicksort(T \*L, T \*R, bool cmp(T,T)=[](T a,T b){return a<b;})

{

if(L+1>=R) return;

T x=\*(L+(R-L)/2), \*p=L, \*q=R-1;

while(p<q){

while(p<q && cmp(\*p,x)) p++;

while(p<q && cmp(x,\*q)) q--;

if(p<=q){

swap(\*p++, \*q--);

}

}

quicksort(L,q+1,cmp);

quicksort(p,R,cmp);

}

**//HeapSort**

template <class T>

void heapy(T \*L,T \*R,T \*i, bool cmp(T,T)){

T \*p=L+2\*(i-L)+1;

if(p<R){

if(p+1<R && cmp(\*p,\*(p+1))) p++;

if(cmp(\*i,\*p)) {

swap(\*i,\*p);

heapy(L,R,p,cmp);

}

}

}

template <class T>

void heapsort(T \*L, T \*R, bool cmp(T,T)=[](T a,T b){return a<b;}){

for(T \*p=R-1; p>=L; p--) heapy(L,R,p,cmp);

for(T\*p=R-1; p>L; p--){

swap(\*L, \*p);

heapy(L,p,L,cmp);

}

}

**//Merge Sort**

template <class T>

void ms(T \*L, T \*R, bool cmp(T,T), T\*b, T\*c){

if(L+1>=R) return;

T \*M=L+(R-L)/2;

ms(L,M,cmp,b,c); //sort tu L..M-1

ms(M,R,cmp,b,c); //sort tu M..R-1

T\*p=b, \*q=c;

for(T \*z=M-1; z>=L; z--) \*p++=\*z; p--; //p giong n

for(T \*z=R-1; z>=M; z--) \*q++=\*z; q--; //q giong m

for(T \*z=L; z<R; z++)

\*z=p<b?\*q--:(q<c?\*p--:(cmp(\*p,\*q)?\*p--:\*q--));

}

template <class T>

void mergesort(T \*L, T \*R, bool cmp(T,T)=[](T a,T b){return a<b;})

{

T b[R-L+5], c[R-L+5];

ms(L,R,cmp,b,c);

}

**//Cài đặt Tree**

struct Tree

{ int L, R, elem;

Tree \*left, \*right;

Tree(int u, int v) {

L=u; R=v;

if(u==v) left=right=0;

else{

left=new Tree(u, (u+v)/2);

right= new Tree((u+v)/2+1, v);

}

}

};

void update(Tree \*&T, int p, int x)

{

if(T->L==T->right) T->elem->x;

else{

T->left->R>=p? update(T->left, p, x):update (T->right, p, x);

T->elem=max(T->left->elem, T->right->elem);

}

}

int get(Tre \*T, int u, int v)

{

if(T->L==u && T->R==v) return T->elem;

if(T->left->R==v) return get(T->left, u, v);

if(T->right->L==u) return get(T->right, u, v);

return max(get(T->left, u, T->left->R), get(T->right, T->right->L, v));

}

**//Tìm kiếm nhị phân**

template<class T>

T \*find(T \*L, T \*R, T x)

{

for(T\*p=L;p<R;p++) if(\*p==x) return p;

return nullptr;

}

template <class T>

T \*search(T \*L, T \*R, T x)

{

if(L+1>R) return NULL;

T \*M=L+(R-L)/2;

if(\*M==x) return M;

if(\*M>x) return search(L, M, x);

return search(M+1, R, x);

}

#ifndef HASHTABLE\_H

#define HASHTABLE\_H 1

#include"SList.cpp"

template<class Keys,class T>

class **Hashtable**

{

private:

int N;

SingleList<Keys,T> \*table;

public:

Hashtable(int n);

Node<Keys,T> \*Add(Keys key, T obj, int (\*hash)(Keys,int));

void Remove(Keys key, int (\*hash)(Keys,int));

Node<Keys,T> \* Find(Keys key, int (\*hash)(Keys,int));

int Contains(Keys key, int (\*hash)(Keys,int));

int Count();

};

template<class Keys,class T>

Hashtable<Keys,T>::Hashtable(int n)

{

N = n;

table= new SingleList<Keys,T>[N];

}

template<class Keys,class T>

Node<Keys,T>\* Hashtable<Keys,T>::Add(Keys key, T obj,

int (\*hash)(Keys,int))

{

Node<Keys,T> \*p;

int h = hash(key,N);

p = table[h].getNode(key);

if(p==NULL)

return table[h].insertLast(key,obj);

else

return NULL;

}

template<class Keys,class T>

void Hashtable<Keys,T>::Remove(Keys key,

int (\*hash)(Keys,int))

{

int h = hash(key,N);

Node<Keys,T> \*p;

p = table[h].getNode(key);

if(p!=NULL)

table[h].remove(p);

}

template<class Keys,class T>

Node<Keys,T> \* Hashtable<Keys,T>::Find(Keys key,

int (\*hash)(Keys,int))

{

int h = hash(key,N);

return table[h].getNode(key);

}

template<class Keys,class T>

int Hashtable<Keys,T>::Contains(Keys key, int (\*hash)(Keys,int))

{

int h = hash(key,N);

if(table[h].getNode(key)==NULL)

return 0;

else

return 1;

}

template<class Keys,class T>

int Hashtable<Keys,T>::Count()

{

int t = 0;

for(int i=0;i<N;i++)

t = t + table[i].size();

return t;

}

#endif

**//Quản lý sinh viên dung list**

using namespace std;

class NVien{

private: string Id, Name, Gender;

**int** salary;

public:

friend istream &operator>>(istream &is, NVien &a){

cout << **"Nhap ma nhan vien: "**; is >> a.Id;

cin.ignore();

cout << **"Nhap ho va ten: "**; getline(is, a.Name);

cout << **"Nhap gioi tinh: "**; is >> a.Gender;

cout << **"Nhap luong: "**; is >> a.salary;

}

friend ostream &operator<<(ostream &os, NVien a){

os << **"Ma nhan vien: "** << a.Id << endl;

os << **"Ho va ten: "** << a.Name << endl;

os << **"Gioi tinh: "** << a.Gender << endl;

os << **"Luong: "** << a.salary << endl;

}

string getId(){

**return** Id;

}

**int** getSalary(){

**return** salary;

}

};

class QL{private: **int** n;

list<NVien> ql;

public:

**void** Create(){

cout << **"Nhap so nhan vien: "**;cin >> n;

cin.ignore();NVien a;

**for**(**int** i = 0; i < n; i++){

cin >> a;

ql.push\_back(a);

}

}

**void** Del(){string Id;

cout << **"Nhap ma nhan vien can xoa: "**;cin >> Id;

**for** (list<NVien>::iterator it = ql.begin(); it != ql.end(); it++)

**if** ((\*it).getId() == Id)

ql.erase(it);

}

**void** Display(){

**for**(list<NVien>::iterator it = ql.begin(); it != ql.end(); it++)

**if**((\*it).getSalary() > 2000000)

cout << \*it;

}

**void** DisplayAll(){

**for**(list<NVien>::iterator it = ql.begin(); it != ql.end(); it++)

cout << \*it;

}

};

**int** main()

{ QL a;

a.Create();

*// a.Del();*

a.Display();

}

**Sinh viên vector**

class SV{

private:

string ten,ma;

int tuoi;

public:

friend istream& operator>>(istream &is,SV &a){

is.ignore();

getline(is,a.ten);

is>>a.ma;

is>>a.tuoi;

return is;

}

friend ostream& operator<<(ostream &os,SV a){

cout<<a.ten<<setw(15)<<a.ma<<setw(15)<<a.tuoi<<"\n";

}

};

class QLSV{

private:

vector<SV> V;

public:

void nhap(){

int n;

SV a;

cin>>n;

while(n--){

cin>>a;

V.insert(V.size(),a);

}

}

void xoa(){

SV a;

int k;

cout<<"Nhap vi tri can xoa ";cin>>k;

if( V.remove(k,a) ==true ){

cout<<"\nXoa thanh cong ";

cout<<"\nSinh vien vua xoa la: "<<a;

}

else cout<<"\nXoa that bai";

}

void in(){

vector\_itr<SV> it(&V);

cout<<"\n";

while(it.hax\_next()) cout<<it.next()<<" ";

}

};

int main(){

QLSV A;

A.nhap();

A.xoa();

A.in();

}

**Cầu thủ vector**

#define ll long long

class cthu{

private:

int soao;

string hoten;

int nam,thuong,banthang,gio;

public:

cthu(){}

friend istream &operator>>(istream &it, cthu &p)

{

cout<<"Nhap so ao";

it>>p.soao;

cout<<"Nhap ho va ten";

fflush(stdin);

getline(it, p.hoten);

cout<<"Nhap nam sinh:";

it>>p.nam;

cout<<"Nhap thuong:";

it>>p.thuong;

cout<<"Nhap so ban thang:";

it>>p.banthang;

cout<<"Nhap so gio:";

it>>p.gio;

return it;

}

friend ostream &operator<<(ostream &it, cthu &p){ it<<p.soao<<"||"<<p.hoten<<"||"<<p.nam<<"||"<<p.thuong<<"||"<<p.banthang<<"||"<<p.gio<<endl;

return it;

}

int getao(){

return soao;}

};

class quanli

{ private: vector<cthu> a;

public:

void them(){

cthu b;

cin>>b;

a.push\_back(b);

}

void xoa(){

int x,i;

cin>>x;

for(int i=0;i<a.size();i++)

if(a[i].getao()==x)

break;

a.erase(a.begin()+i);

}

void chen(){

int x; cin>>x; cthu b; cin>>b;

a.insert(a.begin()+x, b);

a.erase(a.begin()+x+1);

}

void in(){

for(int i=0;i<a.size();i++)cout<<a[i];

}

};

int menu(){

int chon;

cout<<"====MENU===\n";

cout<<"1. them mot cau thu:\n";

cout<<"2. Xao mot cau thu theo so ao:\n";

cout<<"3. thay the mot cau thu tai vi tri x:\n";

cout<<"4. In danh sach ra man hinh:\n";

cin>>chon;

return chon;

}

int main(){

quanli a; int chon;

while(true){ switch(chon=menu()){

case 1: a.them();break;

case 2:a.xoa()break;

case 3: a.chen();break;

case 4: a.in();break;

default: break;

}

int y; cout<<"chon tiep(1) thoat(0):"; cin>>y;

if(y==0)break;

if(y==1)continue;

}

}

**//Sắp xếp dãy dung Heap**

**void** Heapy(**int** \*a, **int** n, **int** i){

**if**(2\*i+1>= n) **return**;

**int** p=2\*i+1;

**if**(p<n){

**if**(p+1<n && a[p+1]>a[p]) p++;

**if**(a[i] < a[p]) {

swap(a[i], a[p]);

Heapy(a, n, p);

}

}

}

**void** HeapSort(**int** \*a, **int** n){ *// sap xep day tu a[0]->a[n-1]*

**for**(**int** i=n-1; i>=0; i--) Heapy(a,n,i); *//tao dong*

**for**(**int** i=n-1; i>0; i--){

swap(a[0],a[i]);

Heapy(a,i,0);

}

}

template<class T>

**int** binarySearch(T \*arr, **int** l, **int** r, **int** x)

{

**if** (r >= l) {

**int** mid = l + (r - l) / 2;

**if** (arr[mid] == x)

**return** mid;

**if** (arr[mid] > x)

**return** binarySearch(arr, l, mid - 1, x);

**return** binarySearch(arr, mid + 1, r, x);

}

**return** -1;

}

class Dayso{

private: **int** n;

**int** \*a;

public:

**void** nhap(){

cout << **"Nhap so phan tu: "**;

cin >> n;

a = new **int**[n + 1];

cout << **"Nhap cac phan tu: "** << endl;

**for**(**int** i = 0; i < n; i++)

cin >> a[i];

}

**void** Sort(){

HeapSort(a, n);

}

**void** Search(){

cout << **"Nhap phan tu muon tim: "**;

**int** x;

cin >> x;

cout << **"Vi tri phan tu can tim la: "** << binarySearch(a, 0, n - 1, x) + 1 << endl;

}

**void** xuat(){

cout << **"Day so: "** << endl;

**for**(**int** i = 0; i < n; i++)

cout << a[i] << **" "**;

}

};

**int** main(){

Dayso D;

D.nhap();

D.Sort();

D.xuat();

}

**//Slist\_Node**

template<class T>

class node

{

T elem;

node \*next;

public:

node<T>() {next=NULL;}

node<T>(T x,node<T> \*N=NULL) {elem=x;next=N;}

T &getelem() {return elem;}

node<T> \*&getnext() {return next;}

void setelem(T x) {elem=x;}

void setnext(node<T> \*N=0) {next=N;}

};

**//Slist\_Iterator**

template<class T>

class slist\_ite

{

node<T> \*curr;

public:

slist\_ite<T>(node<T> \*c=0) {curr=c;} //ham tao

node<T> \*getcurr() {return curr;}

slist\_ite<T> &operator=(slist\_ite<T> \*it)

{

this->curr=it->getcurr();

return \*this;

}

T &operator\*() {return curr->getelem();}

slist\_ite<T> operator++(int)

{

curr=curr->getnext();

return curr;

}

slist\_ite<T> operator++()

{

slist\_ite<T> it=curr;

curr=curr->getnext();

return it;

}

bool operator!=(slist\_ite<T> it) {return curr!=it.getcurr();}

};

**//Slist**

template<class T>

class slist

{

node<T> \*head,\*trail;

int num;

public:

slist<T>() {head=trail=0;num=0;}

slist<T>(int k,T x)

{

head=trail=0;num=0;

while(k--) push\_front(x);

}

bool empty() {return num==0;}

int size() {return num;}

T &front() {return head->getelem();}

T &back() {return trail->getelem();}

void push\_front(T x)

{

if(num==0) {head=trail=new node<T>(x);}

else head=new node<T>(x,head);

num++;

}

void push\_back(T x)

{

if(num==0) {head=trail=new node<T>(x);}

else

{

trail->setnext(new node<T>(x,0));

trail=trail->getnext();

}

num++;

}

void pop\_front()

{

head=head->getnext();

num--;

if(num==0) trail=0;

}

void pop\_back()

{

if(num==1) return pop\_front();

node<T> \*p=head;

while(p->getnext()!=trail) p=p->getnext();

p->setnext(0);

trail=p;

num--;

}

typedef slist\_ite<T> iterator;

iterator begin() {return head;}

iterator end() {return iterator(NULL);}

void insert(iterator it,T x)

{

if(it.getcurr()==head) return push\_front(x);

node<T> \*p=head;

while(p->getnext()!=it.getcurr()) p=p->getnext();

p->setnext(new node<T>(x,it.getcurr()));

num++;

}

void erase(iterator it)

{

if(it.getcurr()==head) return pop\_front();

if(it.getcurr()==trail) return pop\_back();

node<T> \*p=head;

while(p->getnext()!=it.getcurr()) p=p->getnext();

p->setnext(it.getcurr()->getnext());

num--;

}

};

**//Dlist\_Node**

template<class T>

class node

{ T elem;

node<T> \*next,\*prev;

public:

node<T>(T e,node<T>\*P=0,node<T>\*N=0)

{

elem=e;

prev=P; next=N;

}

T &getelem() {return elem;}

node<T>\*&getnext() {return next;}

node<T>\*&getprev() {return prev;}

void setelem(T e) {elem=e;}

void setnext(node<T>\* N) {next=N;}

void setprev(node<T>\* P) {prev=P;}

};

**//Dlist\_iterator**

template<class T>

class dlist\_ite

{ node<T> \*curr;

public:

dlist\_ite<T>(node<T> \*c=0) {curr=c;} //ham tao

node<T> \*getcurr() {return curr;}

dlist\_ite<T> &operator=(dlist\_ite<T> \*it)

{

this->curr=it->getcurr();

return \*this;

}

T &operator\*() {return curr->getelem();}

dlist\_ite<T> operator++(int)

{

curr=curr->getnext();

return curr;

}

dlist\_ite<T> operator++()

{

dlist\_ite<T> it=curr;

curr=curr->getnext();

return it;

}

bool operator!=(dlist\_ite<T> it) {return curr!=it.getcurr();}

};

**//Dlist\_re\_iterator**

template<class T>

class dlist\_re\_ite

{

node<T> \*curr;

public:

dlist\_re\_ite<T>(node<T> \*c=0) {curr=c;} //ham tao

node<T> \*getcurr() {return curr;}

dlist\_re\_ite<T> &operator=(dlist\_re\_ite<T> \*it)

{

this->curr=it->getcurr();

return \*this;

}

T &operator\*() {return curr->getelem();}

dlist\_re\_ite<T> operator++(int)

{

curr=curr->getprev();

return curr;

}

dlist\_re\_ite<T> operator++()

{

dlist\_re\_ite<T> it=curr;

curr=curr->getprev();

return it;

}

bool operator!=(dlist\_re\_ite<T> it) {return curr!=it.getcurr();}

};

**//Dlist**

template <class T>

class dlist

{ node<T> \*head,\*trail;

int num;

public:

public:

dlist<T>() {head=trail=0;num=0;}

dlist<T>(int k,T x)

{

head=trail=0;num=0;

while(k--) push\_front(x);

}

bool empty() {return num==0;}

int size() {return num;}

T &front() {return head->getelem();}

T &back() {return trail->getelem();}

void push\_front(T x)

{

if(num==0) {head=trail=new node<T>(x);}

else

{

head=new node<T>(x,0,head);

head->getnext()->setprev(head);

}

num++;

}

void push\_back(T x)

{

if(num==0) {head=trail=new node<T>(x);}

else

{

trail=new node<T>(x,trail,0);

trail->getprev()->setnext(trail);

}

num++;

}

void pop\_front()

{

head=head->getnext();

if(head) head->setprev(0);

num--;

if(num==0) trail=0;

}

void pop\_back()

{

trail=trail->getprev();

if(trail) trail->setnext(0);

num--;

if(num==0) trail=0;

}

typedef dlist\_ite<T> iterator;

iterator begin() {return head;}

iterator end() {return iterator(NULL);}

typedef dlist\_re\_ite<T> reverse\_iterator;

reverse\_iterator rbegin() {return trail;}

reverse\_iterator rend() {return reverse\_iterator(NULL);}

void insert(iterator it,T x)

{

if(it.getcurr()==head) return push\_front(x);

node<T> \*q=it.getcurr(),\*p=q->getprev(),\*r=new node<T>(x,p,q);

p->setnext(r);

q->setprev(r);

num++;

}

void erase(iterator it)

{

if(it.getcurr()==head) return pop\_front();

if(it.getcurr()==trail) return pop\_back();

node<T> \*p=it.getcurr()->getprev();

node<T> \*q=it.getcurr()->getnext();

p->setnext(q);

q->setprev(p);

num--;

}

};

**//STACK**

template <class T>

class STACK

{

int num; //So phan tu dong thoi la dinh cua stack

int cap; //suc chua

T \*buff; //Mang luu cac phan tu cua stack

public:

STACK() {cap=num=0; buff=NULL;}

~STACK() {if(buff) {delete []buff; buff=NULL;}}

int size() {return num;}

bool empty() {return num==0;}

T &top() {return buff[num-1];}

void pop() {num--;}

void push(T x)

{

if(num==cap)

{

cap=cap\*2+5;

T \*tem=new T[cap];

for(int i=0;i<num;i++) tem[i]=buff[i]; //sao du lieu

if(buff) delete []buff;

buff=tem;

}

buff[num++]=x;

}

};

**//Queue**

template <class T>

class Queue

{

T \*buff;

int num,cap,F,L;

public:

Queue() {buff=0;num=cap=0;F=L=0;}

~Queue() {if(buff) delete[]buff; buff=0;}

bool empty() {return num==0;}

int size() {return num;}

T &front() {return buff[F];}

T &back() {return L==0?buff[cap-1]:buff[L-1];}

void pop() {F=(F+1)%cap;num--;}

void push(T x)

{

if(num==cap)

{

int newcap=cap\*1.7+5;

T \*tem=new T[newcap];

for(int i=F,j=0;i<F+cap;i++,j++) tem[j]=buff[i%cap];

if(buff) delete[]buff;

buff=tem;

F=0;L=cap;cap=newcap;

}

buff[L]=x;

L=(L+1)%cap;

num++;

}

};