**//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++;

}

};

**//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);

}