Vector

```
#ifndef ___vector___cpp___
#define ___vector___cpp___
template <class T>
class Vec_ite{
        T *cur;
        public:
                Vec_ite(T *c=0){cur=c;}
                Vec_ite<T>&operator=(Vec_ite<T> &it) {cur=it.cur; return *this;}
                bool operator!=(Vec_ite<T> it) {return cur!=it.cur;}
                T&operator*() {return *cur;}
                Vec_ite<T> operator++(int)  //it++ {
                        Vec_ite<T> t=cur;
                        cur=cur-1;
                        return t;
                }
                Vec_ite<T> operator++()  //++it {
                        cur=cur-1;
                        return *this;
                }
};

template <class T>
class Vector {
	T *buf=0;
	int n,cap;
		void extend(int newcap) {
			cap=newcap;
			T *tem=new T[cap];
			for(int i=0;i<n;i++) tem[i]=buf[i];
			if(buf) delete []buf;
			buf=tem;
		}
	public:
		Vector() {buf=0; n=cap=0;}
		Vector(int m,T x) {n=cap=m; buf=new T[n]; for(int i=0;i<n;i++)
buf[i]=x;}
		~Vector() {if(buf) delete[]buf;}
		void clear() {n=0;}
		int size()       {return n;}
		int capacity() {return cap;}
		bool empty() {return n==0;}
		T &front() {return buf[0];}
		T &back()  {return buf[n-1];}
		T &operator[](int k) {return buf[k];}
		void push_back(T x) {
			if(n==cap) extend(cap?cap*2:1);
			buf[n++]=x;
		}
```

```
void pop_back() {n--;}
void resize(int num) {
        if(num>cap) extend(num);
        n=num;
}
void resize(int num,T x) {
        if(num>cap) extend(num);
        for(int i=n;i<num;i++) buf[i]=x;
        n=num;
}
Vector(Vector<T> &V) //toan tu copy {
        this->cap=V.cap;
        this->n=V.n;
        if(this->cap==0) this->buf=0;
        else {
                this->buf=new T[cap];
                for(int i=0;i<n;i++) this->buf[i]=V.buf[i];
        }
}
Vector<T>&operator=(Vector<T> &V) //toan tu gan {
        this->cap=V.cap;
        this->n=V.n;
        if(this->cap==0) this->buf=0;
        else    {
                if(this->buf) delete []this->buf;
                this->buf=new T[cap];
                for(int i=0;i<n;i++) this->buf[i]=V.buf[i];
        }
        return *this;
}
typedef T* iterator;
iterator begin() {return buf;}
iterator end()   {return buf+n;}
void insert(iterator &it,T x) //chen x vao vi tri it       {
        if(n==cap)    {
                int k=it-buf;  //vi tri no tro tren buf cu
                extend(cap*2);
                it=buf+k;                  //vi tri moi tren buf moi
        }
        for(T *p=buf+n;p>it;p--) *p=*(p-1);
        *it=x;
        n++;
}
void erase(iterator it)       {
        while(it<buf+n) {*it=*(it+1); it++;}
        n--;
}
typedef Vec_ite<T> reverse_iterator;
```

```
                reverse_iterator rbegin() {return buf+n-1;}
                reverse_iterator rend() {return buf-1;}
};
#endif
```

Slist

```
template <class T>
class node{
        T elem;
        node *next;
        public:
                node(T x,node<T> *N=0) {elem=x;next=N;}
                T &getsetelem() {return elem;}
                node<T>* &getsetnext() {return next;}
};
template <class T>
class slist_ite{
        node<T>*cur;  //tro vao vi tri hien thoi
        public:
                node<T>* getcur() {return cur;}
                slist_ite(node<T>*c=0) {cur=c;}
                T&operator*() {return cur->getsetelem();}
                slist_ite<T> &operator++(int) //it++
                {
```

```
            slist_ite<T> tem(cur);
            cur=cur->getsetnext();
            return tem;
        }
        slist_ite<T> &operator++()      //++it {
            cur=cur->getsetnext();
            return *this;
        }
        //operator=
        bool operator!=(slist_ite<T> sit) {return cur!=sit.cur;}
        bool operator==(slist_ite<T> sit) {return cur==sit.cur;}
};

template <class T>
class slist   //single list {
    node<T>*Head,*Trai;   //Head tro den phan tu dau danh sach, Trai tro phan
tu cuoi ds
    unsigned n;
        void Delete() {
            node<T> *p=Head;
            while(p) {
                p=p->getsetnext();
                delete Head;
                Head=p;
            }
        }
public:
        typedef slist_ite<T> iterator;
        iterator begin() {return Head;}
        iterator end() {return 0;}
        slist(){Head=Trai=0;n=0;}
        slist(slist<T> &sL)   {
                //cout<<"\ncopy\n";
                Head=Trai=0;n=0;
                for(auto z:sL) push_back(z);
        }
        slist(int k,T x) {
                Head=Trai=0;n=0;
                while(k--) push_back(x);
        }
        ~slist() {
                clear();
        }
        void clear()  {
                Delete(); n=0;
        }
        bool empty() {return n==0;}
        unsigned size() {return n;}
        T &front() {return Head->getsetelem();}
```

```
T &back()  {return Trai->getsetelem();}
void push_back(T x)  {
        Trai=(n?Trai->getsetnext():Head)=new node<T>(x);
        n++;
}
void push_front(T x) {
        Head=new node<T>(x,Head);
        if(n==0) Trai=Head;
        n++;
}
void pop_back() {
        if(n==1) {delete Head; Head=Trai=0;}
        else    {
                node<T>*p=Head;
                while(p->getsetnext()!=Trai) p=p->getsetnext();
                p->getsetnext()=0;
                delete Trai;
                Trai=p;
        }
        n--;
}
void pop_front()    {
        if(n==1) {delete Head; Head=Trai=0;}
        else    {
```

```
                    node<T>*p=Head->getsetnext();
                    delete Head;
                    Head=p;
                }
                n--;
            }
            void travel() {
                for(node<T>*p=Head;p;p=p->getsetnext()) cout<<p-
>getsetelem()<<" ";
            }
            void insert(iterator it,T x){
                if(it==begin()) return push_front(x);
                if(it==end()) return push_back(x);
                node<T>*p=Head;
                while(iterator(p->getsetnext())!=it) p=p->getsetnext();
                p->getsetnext()=new node<T>(x,it.getcur());
                n++;
            }
            void erase(iterator &it)    {
                if(it==end()) return;
                if(it==begin()) return pop_front();
                if(it==iterator(Trai)) return pop_back();
                node<T>*p=Head;
                while(iterator(p->getsetnext())!=it) p=p->getsetnext();
                p->getsetnext()=p->getsetnext()->getsetnext();
                delete it.getcur();
                it=p->getsetnext();
                n--;
        }
        void erase(T x) {
                node<T>*p=Head;
                while(p && p->getsetelem()!=x) p=p->getsetnext();
                if(p)   {
                        iterator it=p;
                        erase(it);
                }
        }
        void sort()   {
                for(node<T>*p=Head;p;p=p->getsetnext())
                for(node<T>*q=p->getsetnext();q;q=q->getsetnext())
                if(p->getsetelem()>q->getsetelem()) swap(p->getsetelem(),q-
>getsetelem());
        }
};
#endif;
```

Dlist

```
template <class T>
class node {
        T elem;
        node *next,*prev;
        public:
                node(T x,node<T> *P=0,node<T>*N=0) {elem=x;prev=P; next=N;}
                T &getelem() {return elem;}
                node<T>* &getprev() {return prev;}
                node<T>* &getnext() {return next;}
};
template <class T>
class dlist_ite {
        node<T>*cur;  //tro vao vi tri hien thoi
        public:
                node<T>* getcur() {return cur;}
                dlist_ite(node<T>*c=0) {cur=c;}
                T&operator*() {return cur->getelem();}
                dlist_ite<T> &operator++(int) //it++    {
                        dlist_ite<T> tem(cur);
                        cur=cur->getnext();
                        return tem;
                }
                dlist_ite<T> &operator++()        //++it {
                        cur=cur->getnext();
                        return *this;
                }
                bool operator!=(dlist_ite<T> sit) {return cur!=sit.cur;}
                bool operator==(dlist_ite<T> sit) {return cur==sit.cur;}
};
template <class T>
class dlist_rite {
        node<T>*cur;  //tro vao vi tri hien thoi
        public:
                node<T>* getcur() {return cur;}
                dlist_rite(node<T>*c=0) {cur=c;}
                T&operator*() {return cur->getelem();}
                dlist_rite<T> &operator++(int) //it++   {
                        dlist_rite<T> tem(cur);
                        cur=cur->getprev();
                        return tem;
                }
                dlist_rite<T> &operator++()        //++it {
                        cur=cur->getprev();
                        return *this;
                }
                bool operator!=(dlist_rite<T> sit) {return cur!=sit.cur;}
                bool operator==(dlist_rite<T> sit) {return cur==sit.cur;}
};

template <class T>
class dlist   //double list
```

```
{
        node<T>*Head,*Trai;   //Head tro den phan tu dau danh sach, Trai tro phan
tu cuoi ds
        unsigned n;
                void Delete(){
                        node<T> *p=Head;
                        while(p){
                                p=p->getnext();
                                delete Head;
                                Head=p;
                        }
                }
        public:
                typedef dlist_ite<T> iterator;
                iterator begin() {return Head;}
                iterator end() {return 0;}
                typedef dlist_rite<T> reverse_iterator;
                reverse_iterator rbegin() {return Trai;}
                reverse_iterator rend()   {return 0;}

                dlist(){Head=Trai=0;n=0;}
                dlist(dlist<T> &sL)  {
                        //cout<<"\ncopy\n";
                        Head=Trai=0;n=0;
                        for(auto z:sL) push_back(z);
```

```
}
dlist(int k,T x){
        Head=Trai=0;n=0;
        while(k--) push_back(x);
}
~dlist() {clear();}
void clear()  {Delete(); n=0;}
bool empty() {return n==0;}
unsigned size() {return n;}
T &front() {return Head->getelem();}
T &back()  {return Trai->getelem();}
void push_back(T x)  {
        if(n==0) Head=Trai=new node<T>(x);
        else Trai = Trai->getnext() = new node<T>(x,Trai,0);
        n++;
}
void push_front(T x) {
        if(n==0) Head=Trai=new node<T>(x);
        else   Head = Head->getprev() = new node<T>(x,0,Head);
        n++;
}
void pop_back() {
        if(n==1) {delete Head; Head=Trai=0;}
        else {
```

```
                        node<T> *p=Trai;
                        Trai=Trai->getprev();
                        Trai->getnext()=0;
                        delete p;
                }
                n--;
        }
        void pop_front()    {
                if(n==1) {delete Head; Head=Trai=0;}
                else
                {
                        node<T> *p=Head;
                        Head=Head->getnext();
                        Head->getprev()=0;
                        delete p;
                }
                n--;
        }
        void travel() {
                for(node<T>*p=Head;p;p=p->getnext()) cout<<p-
>getelem()<<" ";
        }
        void insert(iterator it,T x){
                if(it==begin()) return push_front(x);
                if(it==end()) return push_back(x);
        node<T> *p=it.getcur()->getprev();
        node<T> *q=new node<T>(x,p,it.getcur());
        p->getnext()=q;
        it.getcur()->getprev()=q;
        n++;
}
void erase(iterator it)      {
        if(it==end()) return;
        if(it==begin()) return pop_front();
        if(it.getcur()==Trai) return pop_back();
        node<T>*p=it.getcur()->getprev();
        node<T>*q=it.getcur()->getnext();
        p->getnext()=q;
        q->getprev()=p;
        delete it.getcur();
        n--;
}
void erase(T x){
        node<T>*p=Head;
        while(p && p->getelem()!=x) p=p->getnext();
        if(p) erase(p);
}
void sort()    {
        for(node<T>*p=Head;p;p=p->getnext())
```

```
                        for(node<T>*q=p->getnext();q;q=q->getnext())
                        if(p->getelem()>q->getelem()) swap(p->getelem(),q-
>getelem());
                }
};
#endif;
```

Bảng băm

```
template <class T>
class hashtable {
        vector< list<T> >  buf;
        int n,cap;
        int myhash(T x)     {
                hash<T> H;
                return H(x)/cap%cap;
        }
        public:
                hashtable(int _cap=113)  {
                        cap=_cap;
                        buf.resize(cap);
                        n=0;
                }
                void insert(T x) {
                        int k=myhash(x);
                        buf[k].push_back(x);
```

```
            n++;
        }
        bool find(T x) {
            int k=myhash(x);
            for(auto z:buf[k]) if(z==x) return true;
            return false;
        }
        void erase(T x) {
            int k=myhash(x);
            auto p=buf[k].begin();
            while(p!=buf[k].end()&& *p!=x) p++;
            if(p!=buf[k].end()) {buf[k].erase(p); n--;}
        }
        void travel() {
            for(auto z: buf)
            for(auto x:z) cout<<x<<" ";
        }

};
```