第一题

（1）

#include<iostream>

using namespace std;

template <class T>

class linearlist

{

public:

T ele[];

int current;

int max;

linearlist(T length)

{

ele = new T[length];

current = 0;

max = length;

}

void add(T a)

{

ele[current] = a;

current++;

}

void deletes(int i)

{

if(i>current){

cout<<"超出范围"<<endl;

}

for(int k = i;k<current;k++)

{

ele[k] = ele[k+1];

}

ele[current].~T();

}

void copys()

{

if(current <= max/4)

{

T kkk[] = new T[max/2];

for(int i = 0;i<=max/4;i++){

kkk[i] = ele[i];

}

delete ele;

ele = kkk;

}

cout<<"bumanzutiaojian"<<endl;

}

} ;

int main()

{

linearlist <int>aaa;

aaa = new linearlist(4);

aaa.add(2);

aaa.add(3);

aaa.deletes(1);

aaa.copys();

}

（2）

6. T [] Alternate(T[] a,T[] b)

{

int a1 = a.length();

int b1 = b.length();

T [] c = new T[a1+b1];

for(int i = 0;i<a1+b1;i++)

{

t = i/2;

if(i>2\*a1){

c[i] = b[t];

}

if(i>2\*b1+1){

c[i] = a[t];

}

if(i%2==0){

c[i] = a[t];

}

if(i%2==1){

c[i] = b[t];

}

}

return c;

}

复杂度：4a+4b+4

7. template <class T>

LinearList<T>& LinearList<T>::

Merge(const LinearList<T>& A, const LinearList<T>& B)

{// Merge the two sorted lists A and B

int al = A.Length();

int bl = B.Length();

length = al + bl; // length of result

if (length > MaxSize)

throw NoMem();

// inadequate space for result

int ca = 0; int cb = 0;

while ((ca < al) && (cb < bl)) {

if (A.element[ca] <= B.element[cb])

element[ct++] = A.element[ca++];

else

element[ct++] = B.element[cb++];

}

if (ca == al) // A is finished

for (int q = cb; q < bl; q++)

element[ct++] = B.element[q];

else for (int kkk = ca;kkk < al; kkk++)

element[ct++] = A.element[kkk];

return \*this;

}

27.Chain<T>& Chain<T>::Reverse()

{

ChainNode<T> \*last = 0,

\*current = first,

\*next;

while (current) {

next = current->link;

current->link = last;

last = current;

current = next;

}

first = last;

return \*this;

}

28.

template<class T>

void ReverseChain(Chain<T> &a.Chain<T> &b){

ChainNode \*tmp,\*acur,\*bcur;

acur = a.first.link;

tmp = acur.link;

acur.link = null;

bcur = b.first.link = acur;

acur = tmp;

a.first = null;

while(acur){

tmp = acur.link;

acur.link = bcur;

bcur = acur;

b.first.link = acur;

acur = tmp;

}

}

29:

void Alternate(const Chain<T>&A,const Chain<T>& B, Chain<T>& C)

{

ChainIterator<T> a, b;

T \*DataA = a.Initialize(A);

T \*DataB = b.Initialize(B);

C.Erase();

while (DataA && DataB) {

C.Append(\*DataA);

C.Append(\*DataB);

DataA = a.Next();

DataB = b.Next();

}

while(DataA) {

C.Append(\*DataA);

DataA = a.Next();

}

while(DataB) {

C.Append(\*DataB);

DataB = b.Next();

}

}

31. template<class T>

void Merge(const Chain<T>& A,const Chain<T>& B, Chain<T>& C)

{

ChainIterator<T> a, b;

T \*DataA = a.Initialize(A);

T \*DataB = b.Initialize(B);

C.Erase();

while (DataA && DataB) {

if (\*DataA <= \*DataB) {

C.Append(\*DataA);

DataA = a.Next();}

else {

C.Append(\*DataB);

DataB = b.Next();}

}

if (DataA)

while(DataA) {

C.Append(\*DataA);

DataA = a.Next();

}

else while(DataB) {

C.Append(\*DataB);

DataB = b.Next();

}

}

32.

template<class T>

void Chain<T>::Spilt(Chain<T> &a,Chain<T> &b){

ChainNode<T> \*cur = this.first,\*acur,\*bcur;

if(c.first){

a.first = cur;cur = cur.link;

acur = a.first.link;

if(cur){

b.first = cur;

cur = cur.link;

bcur = b.first.link;

}

while(cur){

acur = cur;

cur = cur.link;

acur = acur.link;

if(cur){

bcur = cur;cur = cur.link;bcur = bcur.link;

}else{break;}

}

}

}

33. template <class T>

void Split(Chain<T>& A, Chain<T>& B, Chain<T>& C)

{ B.Erase();

C.Erase();

ChainIterator<T> a;

T \*e = a.Initialize(A);

while (e) {

B.Append(\*e);

e = a.Next();

if (!e) break;

C.Append(\*e);

e = a.Next();

}

}

67.

template<class T>

class SimChainIterator {

public:

T\* Initialize(SimChain<T>& c)

{location = c.first;

s = &c.s;

if (location == -1) return 0;

return &s.node[location].data;

}

T\* Next()

{if (location == -1) return 0;

location = s.node[location].link;

if (location == -1) return 0;

return &s.node[location].data;

}

SimSpace<T> \*s;

int location;

};