#ifndef HEAP\_H

#define HEAP\_H

template<class ElemType> class MaxHeap{

private:

ElemType \*heapArr;

int CurrentSize;

int MaxSize;

void FilterDown(int Start);

void FilterUp(int End);

public :

MaxHeap(int maxSize);

MaxHeap(ElemType a[],int maxsize,int n);

~MaxHeap(){delete []heapArr;}

Status Insert(const ElemType &e);

Status DeleteTop(ElemType & e);

Status GetTop(ElemType & e)const;

bool IsEmpty()const{return CurrentSize == 0;}

bool IsFull()const{return CurrentSize == MaxSize;}

int SizeOfHeap()const{return CurrentSize;}

void SetEmpty(){CurrentSize=0;}

void Traverse(void (\*Visit)(const ElemType &)) const;

};

template<class ElemType>

MaxHeap<ElemType>::MaxHeap(int maxSize)

{

if (maxSize <= 0) {

cerr << "堆的大小不能小于1" << endl;

exit(1);

}

MaxSize = maxSize;

heapArr = new ElemType[MaxSize];

CurrentSize = 0;

}

template<class ElemType>

MaxHeap<ElemType>::MaxHeap(ElemType a[],int maxSize,int n)

{

if(n <= 0) {

cerr << "堆的大小不能小于1" << endl;

exit(1);

}

MaxSize = maxSize;

heapArr = new ElemType [MaxSize];

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

heapArr[i] = a[i];

CurrentSize = n;

int i = (CurrentSize - 2) / 2;

while(i >= 0) {

FilterDown(i);

i--;

Traverse(Write<ElemType>);

cout << endl;

}

}

template<class ElemType>

void MaxHeap<ElemType>::FilterDown(const int Start)

{

int i = Start,j;

ElemType temp = heapArr[i];

j = 2 \* i + 1;

while(j <= CurrentSize - 1) {

if(j < CurrentSize - 1 && heapArr[j] < heapArr[j+1])

j++;

if(temp >= heapArr[j]) break;

else {

heapArr[i] = heapArr[j];

i = j;

j = 2\*j+1;

}

}

heapArr[i] = temp;

}

template<class ElemType>

void MaxHeap<ElemType>::FilterUp(int End)

{

int j = End,i;

ElemType temp = heapArr[j];

i = (j - 1) / 2;

while(j > 0) {

if (heapArr[i] >= temp) break;

else {

heapArr[j] = heapArr[i];

j = i;

i = (j - 1) / 2;

}

heapArr[j] = temp;

}

}

template<class ElemType>

Status MaxHeap<ElemType>::Insert(const ElemType &e)

{

if (IsFull())

return OVER\_FLOW;

heapArr[CurrentSize] = e;

FilterUp(CurrentSize);

CurrentSize++;

return SUCCESS;

}

template<class ElemType>

Status MaxHeap<ElemType>::DeleteTop(ElemType &e)

{

if(IsEmpty())

return UNDER\_FLOW;

e = heapArr[0];

heapArr[0] = heapArr[CurrentSize-1];

CurrentSize--;

FilterDown(0);

return SUCCESS;

}

template<class ElemType>

Status MaxHeap<ElemType>::GetTop(ElemType &e) const

{

if(IsEmpty())

return UNDER\_FLOW;

e = heapArr[0];

return SUCCESS;

}

template<class ElemType>

void MaxHeap<ElemType>::Traverse(void (\*Visit)(const ElemType &)) const

{

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

(\*Visit)(heapArr[i]);

}

#endif

