```
#ifndef Heap_h
#define Heap_h 1
#include<iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <limits>
using namespace std;
//---------------------------------------------------------------
// HeapException Constructor
//---------------------------------------------------------------
struct HeapException{
    HeapException(char* m)
    {   cout << endl;
        cout << "I am the Heap and I am " << m << ".";
        cout << endl;
    }
};
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
template <class T>
class Heap {
    int size;                   //Number of available entries
    int count;                  //Number of occupied entries
    T* H;                       //Points to an array that stores elements of the
                                //Heap
    const T MIN;                //Mininum value of type T
    //-------------------------------------------------------------------------
    //Function Graph prints the elements of the heap by performing an inorder traversal
    //and indenting each element according to its depth in the heap.
    //-------------------------------------------------------------------------
    void Graph(ostream& o, int depth, int n )
    {   if (n>count) return;
        Graph(o, depth + 1, 2 * n);
        o << endl;
        for (int a = 0; a < depth; a++) o << "      ";      //8 spaces
        o << H[n];
        Graph(o, depth + 1, 2 * n + 1);
    }
```

```
void Title(ostream& o)
{    o << endl;
     o << setw(8) << "min";
     o << setw(8) << "last";
     o << setw(8) << "count";
     o << setw(8) << "a";
     o << setw(8) << "a*2";
     o << setw(8) << "c";
     o << setw(8) << "c+1";
     o << setw(8) << "H[c]";
     o << setw(8) << "H[c+1]";
}
public:
//---------------------------------------------------------------------
// Heap Constructor
//---------------------------------------------------------------------
Heap(T m,int sz=100):MIN(m),size(sz),count(0){H=new T[size];H[0]=MIN;}
//---------------------------------------------------------------------
// Heap Destructor
//---------------------------------------------------------------------
~Heap(){ if (H) delete [] H; }
//-----------------------------------------------------------------------------
//Function IsFull determines if the Heap is full.
//-----------------------------------------------------------------------------
bool IsFull(void){return count >= size - 1; }
//-----------------------------------------------------------------------------
//Function IsEmpty determines if the Heap is empty.
//-----------------------------------------------------------------------------
bool IsEmpty(void){ return count == 0; }
//-----------------------------------------------------------------------------
//Function Insert inserts a new value v into the heap
//-----------------------------------------------------------------------------
void Insert(T v)
{    if (IsFull()) throw HeapException(" full ");
     int a = ++count;                    // pre-increment count
     while( H[a/2] > v ) {
          H[a] = H[a/2];                 // child = parent
          a /= 2;                        // go to parent
     }
     H[a] = v;
}
```

```
//-------------------------------------------------------------------------
//Function remove removes the smallest element
//-------------------------------------------------------------------------
T Remove(void)
{    if (IsEmpty() ) throw HeapException(" empty ");
      T min=H[1];
     T last=H[count--];
     int a,child;
     for (a=1;a*2<=count;a=child) {
          child=a*2;              // child is left child of parent
          if ((child!=count) && (H[child+1]<H[child])) child++;
          if (last>H[child]) H[a]=H[child]; else break;
     }
     H[a] = last;
     return min;
}
//-------------------------------------------------------------------------
//Function print first prints a title and then prints the heap as a linear list.
//-------------------------------------------------------------------------
void Print(ostream& o, char* title)
{    o << endl;
     o << title;
     o << endl;
     for (int a=1;a<=count;a++) o << setw(5) << H[a];
     o << endl;
}
//-------------------------------------------------------------------------
//Function Scan reads values from input stream i and stores them into the Heap.
//-------------------------------------------------------------------------
void Scan(istream& i)
{    for (;;) {
          T v;
          i >> v;
          if (i.eof()) break;
          Insert(v);
     }
}
```

```
    //-------------------------------------------------------------------------
    //Function Sort sorts the Heap.  Elements of the Heap are sorted
    //in ascending order.
    //-------------------------------------------------------------------------
    void Sort(void)
    {   T* J=new T[size];
        J[0]=MIN;
        int a=1;
        while (!IsEmpty()) {
            T v=Remove();
            J[a++]=v;
        }
        if (H) delete H;
        H=J;
        count=a-1;
    }
    //-------------------------------------------------------------------------
    //Function Graph call private member function Graph to
    //print the heap using an inorder traversal and indenting each element
    //according to its depth in the Heap.
    //-------------------------------------------------------------------------
    void Graph(ostream& o,char* title)
    {   o << endl;
        o << title;
        Graph(o,0,1);
        o << endl;
    }
    //-------------------------------------------------------------------------
    //Function Print prints the Heap as a linear list.
    //-------------------------------------------------------------------------
    void Print(ostream& o)
    {   o << endl;
        for (int a=1;a<=count;a++) o << setw(5) << H[a];
        o << endl;
    }
    //-------------------------------------------------------------------------
    //Function Dump prints all member data stored in the Heap
    //-------------------------------------------------------------------------
    void Dump(ostream& o)
    {   o << endl;
        o << "size=" << size << " count=" << count;
        Print(o);
    }
};
#endif
```