\*\*\*\*\*\*\*\*\*\*最大堆（Max Heap）实现\*\*\*\*\*\*\*\*\*\*

1. **堆的抽象类(heap.h)：**

template <typename E> class heap {

private:

void operator =(const heap&) {}

heap(const heap&) {}

virtual void siftdown(int pos) = 0;

public:

heap() {}

virtual ~heap() {}

//建堆操作

virtual void buildHeap() = 0;

//向堆中插入一个元素

virtual void insert(const E& it) = 0;

//删除最大值

virtual E removefirst() = 0;

//删除指定位置的元素

virtual E remove(int pos) = 0;

};

1. **堆节点类(Heapnode.h)：**

template <typename E>

class Heapnode{

public:

bool isLeaf(int pos,int n) const

{

return (pos >= n/2) && (pos < n);

}

int leftchild(int pos) const

{

return 2\*pos + 1;

} // Return leftchild position 返回左孩子下标

int rightchild(int pos) const

{

return 2\*pos + 2;

} // Return rightchild position 返回右孩子下标

int parent(int pos) const // Return parent position 返回父结点下标

{

return (pos-1)/2;

}

}

1. **堆基于数组的物理实现(Aheap.h)：**

template <typename E, typename Comp> class Aheap {

private:

E\* Heap; //Pointer to the heap array 指向堆数组

int maxsize; // Maximum size of the heap 堆最大容量

int n; // Number of elements now in the heap 堆中当前元素个数

// Helper function to put element in its correct place

//下拉元素到正确位置

void siftdown(int pos) {

while (!isLeaf(pos)) { // Stop if pos is a leaf 是叶子节点就停止

int j = leftchild(pos); int rc = rightchild(pos);

if ((rc < n) && Comp::prior(Heap[rc], Heap[j]))

j = rc;

if (Comp::prior(Heap[pos], Heap[j])) return;

swap(Heap, pos, j);

pos = j;

}

}

public:

Aheap(E\* h, int num, int max) //构造函数

{ Heap = h; n = num; maxsize = max; buildHeap(); }

int size() const //返回堆大小

{ return n; }

void buildHeap() // Heapify contents of Heap 建堆

{ for (int i=n/2-1; i>=0; i--) siftdown(i); }

// Insert "it" into the heap 插入元素到堆中

void insert(const E& it) {

Assert(n < maxsize, "Heap is full");

int curr = n++;

Heap[curr] = it;

// Now sift up until curr's parent > curr

//下拉直到位置正确

while ((curr!=0) &&(Comp::prior(Heap[curr], Heap[parent(curr)]))) {

swap(Heap, curr, parent(curr));

curr = parent(curr);

}

} // Remove first value 删除最大元素

E removefirst() {

Assert (n > 0, "Heap is empty");

swap(Heap, 0, --n); // Swap first with last value将最大值与最后一个元素交换

if (n != 0) siftdown(0); // Siftdown new root val 下拉元素

return Heap[n]; // Return deleted value 返回删除的元素

}

// Remove and return element at specified position

//删除指定位置的元素

E remove(int pos) {

Assert((pos >= 0) && (pos < n), "Bad position");

if (pos == (n-1)) n--; // Last element, no work to do若是最后一个则直接删除

else

{

swap(Heap, pos, --n); // Swap with last value 与最后一个交换

while ((pos != 0) &&

(Comp::prior(Heap[pos], Heap[parent(pos)]))) {

swap(Heap, pos, parent(pos));

pos = parent(pos);

}

if (n != 0) siftdown(pos);

}

return Heap[n];

}

};

1. **常用函数类（book.h）：**

//book.h文件中定义了常用的swap、Assert函数、Int类、<<运算符重载

#include <iostream>

#include <cstdlib>

using std::cout;

using std::endl;

using std::string;

using std::ostream;

void Assert(bool val, string s) {

if (!val) {

cout << "Assertion Failed: " << s << endl;

exit(-1);

}

}

// Swap two elements in a generic array

//交换数组中的两个元素

template<typename E>

inline void swap(E A[], int i, int j) {

E temp = A[i];

A[i] = A[j];

A[j] = temp;

}

// Your basic int type as an object.

//定义Int类

class Int {

private:

int val;

public:

Int(int input=0) { val = input; }

int key() const { return val; }

// Overload = to support Int foo = 5 syntax 运算符重载

Int operator= (int input) { val = input; return val; }

};

// Let us print out Ints easily 运算符重载

ostream& operator<<(ostream& s, const Int& i)

{ return s << i.key(); }

ostream& operator<<(ostream& s, const Int\* i)

{ return s << i->key(); }

1. **比较类（compare.h）：**

//比较两个Int值

class maxIntCompare {

public:

static bool prior(Int x, Int y) { return x.key() > y.key(); }

};

1. **主函数测试文件（heapmain.cpp）：**

#include "book.h"

#include "compare.h"

#include "Aheap.h"

using namespace std;

int main() {

int i, j;

int n=10,count=0;

Int A[10] = {9,3,0,2,4,7,5,6,8,1};

Int C[10] = {73, 6, 57, 88, 60, 34, 83, 72, 48, 85};

cout<<"初始数据："<<endl;

for (i=0; i<10; i++)

cout << C[i] << " ";

cout<<endl<<"建堆ing..."<<endl;

Aheap<Int, maxIntCompare> BH(C, 0, 20);

cout<<"建堆完成" <<endl<<endl;

cout<<"按照层次遍历的顺序依次输出堆元素："<<endl;

for (i=0; i<10; i++)

cout << C[i] << " ";

cout << endl;

//演示依次插入数据建堆过程

cout<<endl<<"-------演示依次插入数据建堆过程-------"<<endl;

cout<<"初始堆为空"<<endl;

Aheap<Int, maxIntCompare> H(A, 0, 10);

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

cout<<"插入元素"<<A[i];

H.insert(A[i]);

count++;

cout<<",输出目前已建好的堆元素："<<endl;

for(j=0;j<count;j++){

cout << A[j] << " ";

}

cout<<endl;

}

cout<<endl<<"删除数组中下标为3的元素:"<<H.remove(3)<<endl;

cout<<"输出删除后堆中元素："<<endl;

for (i=0; i<9; i++)

cout << A[i] << " ";

cout<<endl;

cout<<endl<<"删除堆中最大值："<<H.removefirst()<<endl;

cout<<"输出删除后堆中元素："<<endl;

for (i=0; i<8; i++)

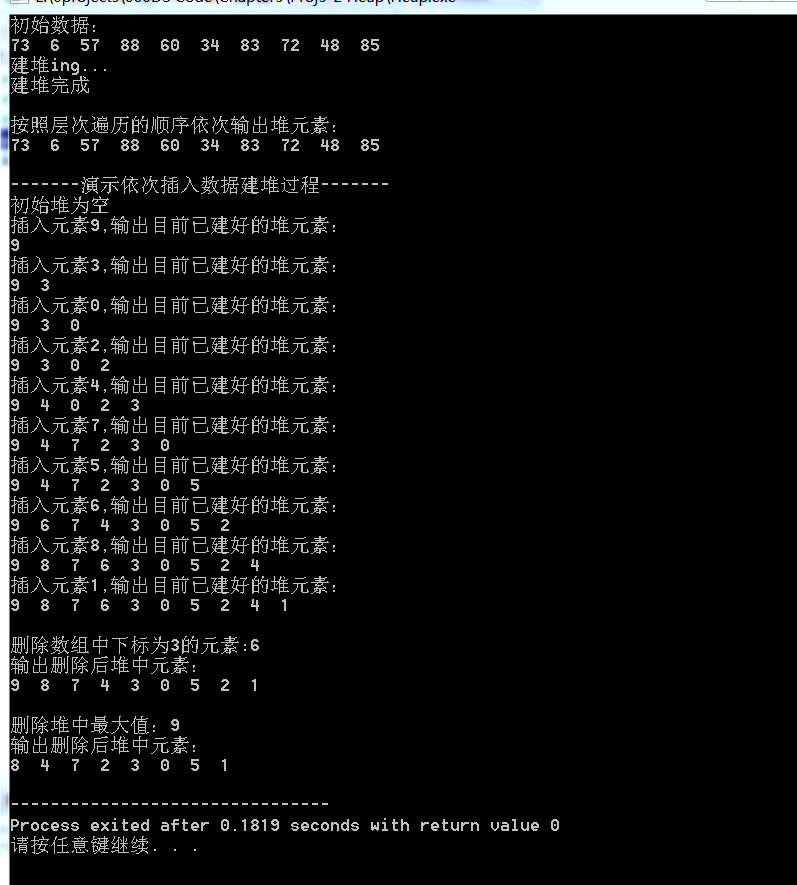
cout << A[i] << " ";

cout<<endl;

return 0;

}

1. **测试结果示例：**



**【附录】**

版本信息声明：

Dev-C++ 5.11

TDM-GCC 4.9.2 64-bit Release

资料整理人：  
物联网工程1402班 宁静仪

物联网工程1402班 吴彦彦