****

项目说明文档

**数据结构课程设计**

**——修补围栏**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 功能分析 1](#_Toc58837480)

[1.1 背景分析 1](#_Toc58837481)

[1.2 功能分析 1](#_Toc58837482)

[第2章 设计 2](#_Toc58837483)

[2.1 数据结构设计 2](#_Toc58837484)

[2.2 类结构设计 2](#_Toc58837485)

[2.3 成员与操作设计 2](#_Toc58837486)

[2.4 系统设计 3](#_Toc58837487)

[第3章 实现 4](#_Toc58837488)

[3.1 最小堆插入的实现 4](#_Toc58837489)

[3.1.1 最小堆插入功能流程图 4](#_Toc58837490)

[3.1.2 最小堆插入功能核心代码 5](#_Toc58837491)

[3.2 最小堆删除的实现 6](#_Toc58837492)

[3.2.1 最小堆删除功能流程图 6](#_Toc58837493)

[3.2.2 最小堆删除功能核心代码 7](#_Toc58837494)

[3.3 最小堆查找的实现 8](#_Toc58837495)

[3.3.1 最小堆查找功能流程图 8](#_Toc58837496)

[3.3.2 最小堆查找功能核心代码 9](#_Toc58837497)

[3.4 总体系统的实现 10](#_Toc58837498)

[3.4.1 总体系统流程图 10](#_Toc58837499)

[3.4.2 总体系统代码实现 10](#_Toc58837500)

[3.2.3 总体功能截屏示例 12](#_Toc58837501)

[第4章 测试 13](#_Toc58837502)

[4.1 通常功能测试 13](#_Toc58837503)

[4.2 边界测试 13](#_Toc58837504)

[4.2.1 输入为空 13](#_Toc58837505)

[4.3 出错测试 13](#_Toc58837506)

[4.3.1 输入非数值字符 13](#_Toc58837507)

[4.3.1 输入负数 14](#_Toc58837508)

# 第1章 功能分析

## 1.1 背景分析

农夫要修理牧场的一段栅栏，他测量了栅栏，发现需要N块木头，每块木头长度为整数Li个长度单位，于是他购买了一个很长的，能锯成N块的木头，即该木头的长度是Li的总和。

但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。为简单起见，不妨就设酬金等于所锯木头的长度。例如，要将长度为20的木头锯成长度为8，7和5的三段，第一次锯木头将木头锯成12和8，花费20；第二次锯木头将长度为12的木头锯成7和5花费12，总花费32元。如果第一次将木头锯成15和5，则第二次将木头锯成7和8，那么总的花费是35（大于32）。

## 1.2 功能分析

对项目需求进行分析，发现因为锯木头的总次数恒定，所以当木头中更短的部分被重复锯的次数越多，更长的部分被重复锯的次数也就越少。所以，每次将更长的部分锯出的这一过程，若倒过来思考，则是具有n块木头需要合并，每次取最小的两个进行合并，再将合并后的木头放回堆里。

# 第2章 设计

## 2.1 数据结构设计

如上所述，需要每次取最小的两个元素进行合并，则考虑使用最小堆。

## 2.2 类结构设计

为了最小堆的实现方便和内存节省，考虑使用数组模拟堆。

## 2.3 成员与操作设计

**模板最小堆类（MinHeap）**

template <class T>

class MinHeap {

   private:

    T\* mHeap;       // 数据

    int mCapacity;  // 总的容量

    int mSize;      // 实际容量

   private:

    void FilterDown(int start, int end);  // 最小堆的向下调整算法

    void FilterUp(

        int start);  // 最小堆的向上调整算法(从start开始向上直到0，调整堆)

   public:

    MinHeap();

    MinHeap(int capacity);

    ~MinHeap();

    T Top();               //返回堆顶元素

    int GetIndex(T data);  // 返回data在二叉堆中的索引

    void Remove(T data);   // 删除最小堆中的data

    void RemoveFront();    // 删除最小堆中的第一个元素

    void Insert(T data);   // 将data插入到二叉堆中

    void Print();          // 打印二叉堆

    int Size() const { return mSize; }

    bool Empty() { return (mSize == 0 ? true : false); }

};

## 2.4 系统设计

系统首先调用opening ()函数实现对屏幕的初始化，然后直接在main函数中输入数据并返回结果。

# 第3章 实现

## 3.1 最小堆插入的实现

### 3.1.1 最小堆插入功能流程图



### 3.1.2 最小堆插入功能核心代码

template <class T>

void MinHeap<T>::Insert(T data) {

    mHeap[mSize] = data;  // 将"数组"插在表尾

    FilterUp(mSize);      // 向上调整堆

    mSize++;              // 堆的实际容量+1

}

template <class T>

void MinHeap<T>::FilterUp(int start) {

    int c = start;        // 当前节点(current)的位置

    int p = (c - 1) / 2;  // 父(parent)结点的位置

    T tmp = mHeap[c];     // 当前节点(current)的大小

    while (c > 0) {

        if (mHeap[p] <= tmp)

            break;

        else {

            mHeap[c] = mHeap[p];

            c = p;

            p = (p - 1) / 2;

        }

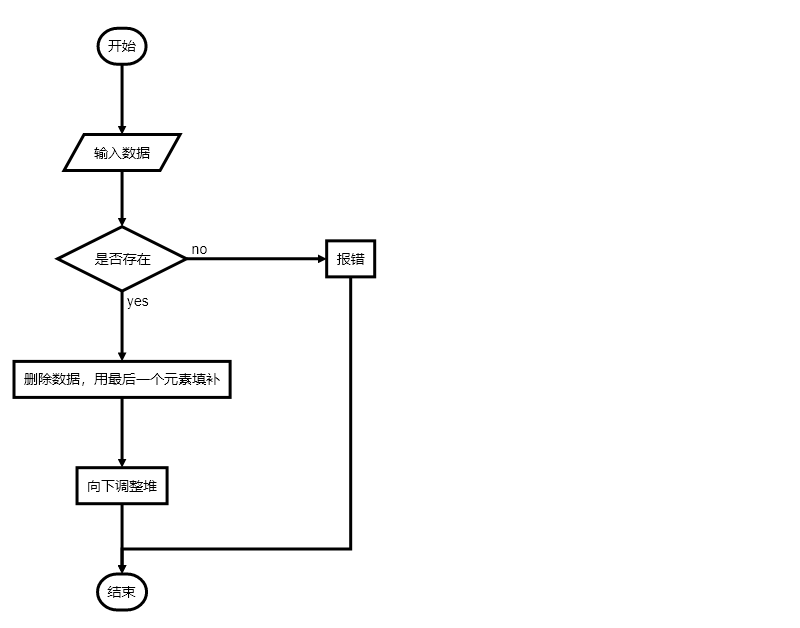
    }

    mHeap[c] = tmp;

}

## 3.2 最小堆删除的实现

### 3.2.1 最小堆删除功能流程图



### 3.2.2 最小堆删除功能核心代码

template <class T>

void MinHeap<T>::Remove(T data) {

    int index;

    if (this->Empty()) throw "Error: The heap is Empty.";

    // 获取data在数组中的索引

    index = GetIndex(data);

    if (index == -1) throw "Error: Data not found.";

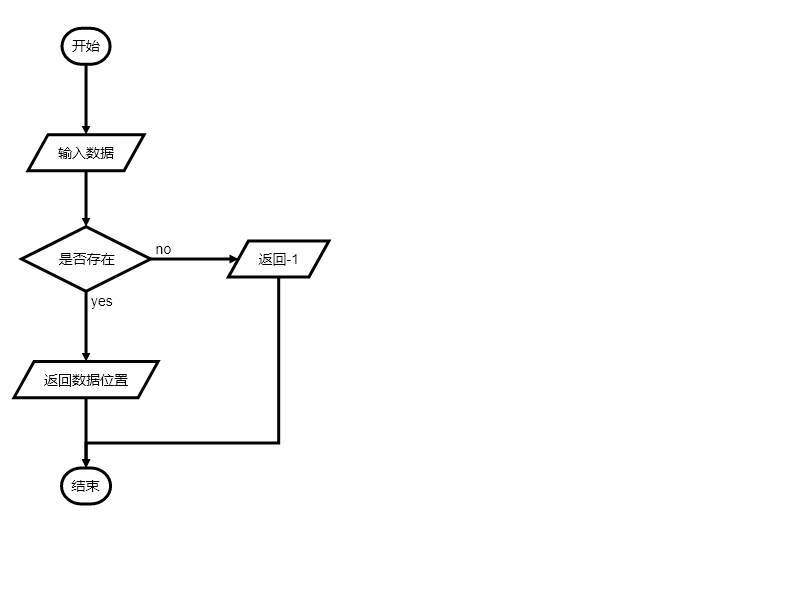
    mHeap[index] = mHeap[--mSize];  // 用最后元素填补

    FilterDown(index, mSize - 1);  // 从index号位置开始自上向下调整为最小堆

}

## 3.3 最小堆查找的实现

### 3.3.1 最小堆查找功能流程图



### 3.3.2 最小堆查找功能核心代码

template <class T>

int MinHeap<T>::GetIndex(T data) {

    for (int i = 0; i < mSize; i++)

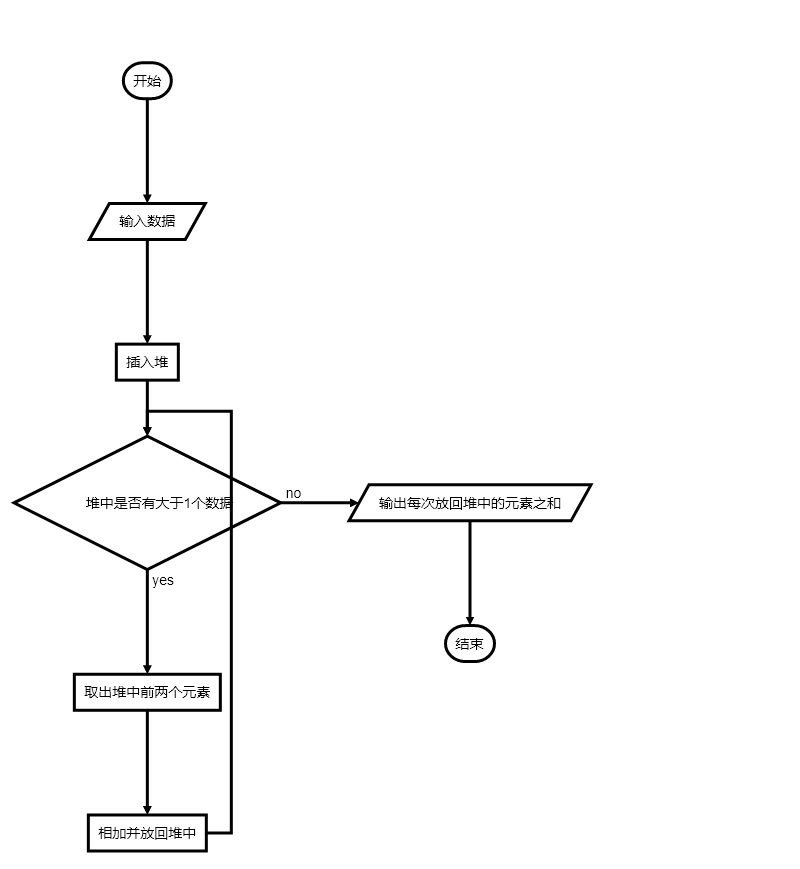
        if (data == mHeap[i]) return i;

    return -1;

}

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统代码实现

void opening() {

    cout << "Repair Ranch" << endl

         << "The input is a line of positive integers. " << endl

         << "The first number is N followed by N pieces of wood. "

         << endl

         << "Now enter a line of positive integers. " << endl;

    return;

}

bool loop(){

    opening();

    MinHeap<int> heap;

    string temp;

    int num = 0;

    while (true) {

        cin >> temp;

        if (!isNum(temp))

            cout << "Error: Invalid input detected. Please try again." << endl;

        else {

            num = str2num(temp);

            break;

        }

    }

    for (int i = 0; i < num; i++) {

        cin >> temp;

        if (!isNum(temp)) {

            i--;

            cout << "Error: Invalid input detected. Please try again." << endl;

        }

        else

            heap.Insert((int)str2num(temp));

    }

    int sum = 0;

    while (heap.Size() > 1) {

        int temp = 0;

        temp += heap.Top();

        heap.RemoveFront();

        temp += heap.Top();

        heap.RemoveFront();

        heap.Insert(temp);

        sum += temp;

    }

    cout << sum << endl;

    std::cout << "Again? (y for yes, n for no)";

    char c;

    std::cin >> c;

    if (c == 'y')

        return true;

    else

        return false;

}

int main() {

    while (loop())

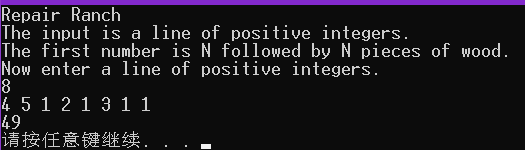
        ;

    std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 3.2.3 总体功能截屏示例



# 第4章 测试

## 4.1 通常功能测试

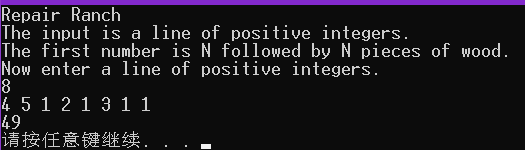
**测试用例**：

8 4 5 1 2 1 3 1 1

**预期结果**：

49

**实验结果**



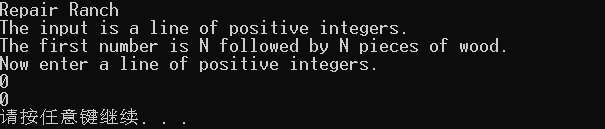
## 4.2 边界测试

### 4.2.1 输入为空

**测试用例：**0

**预期结果：**输出0，因为不需要锯木头

**实验结果：**



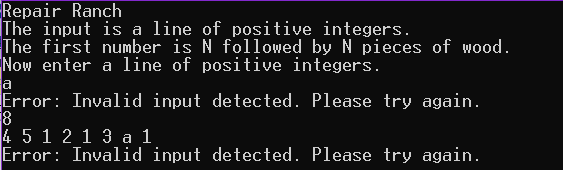
## 4.3 出错测试

### 4.3.1 输入非数值字符

**测试用例：**输入非数值，如a，8 4 5 1 2 1 3 a 1。

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**



### 4.3.1 输入负数

**测试用例：**输入负数，如-1，8 2 1 3 9 4 -1 12 15。

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**

