



中南大學

CENTRAL SOUTH UNIVERSITY

高级程序设计 实践实验报告

题 目：	高级程序设计实践
学生姓名：	陈康坤
指导教师：	李玺
学 院：	计算机学院
专业班级：	计科 2202 班

本科生院制

2024 年 1 月

目录

第一章 实验一	4
1. 题目要求	4
2.题目的设计思路	4
3.题目的实现过程（包括设计思想、流程图等内容）	4
4. 题目的难点及解决	5
5.题目的运行及测试结果	5
5. 总体心得体会	6
第二章 实验二	7
1. 题目要求	7
2.题目的设计思路	4
3.题目的实现过程（包括设计思想、流程图等内容）	4
4. 题目的难点及解决	5
5.题目的运行及测试结果	5 10
6.总体心得体会	10
7.选做题：计算器	10
.....	12
第三章 实验三	14
1. 题目要求	14
2.题目的设计思路	14
3.题目的实现过程（包括设计思想、流程图等内容）	16
4.题目的难点及解决	16
5.题目的运行及测试结果	17
6.总体心得体会	16
第四章 实验四	17
1. 题目要求	17
2.题目的设计思路	17
3.题目的实现过程（包括设计思想、流程图等内容）	19
4. 题目的难点及解决	20
5.题目的运行及测试结果	20
6.总体心得体会	20

1. 题目要求
2. 题目的设计思路
3. 题目的实现过程 （包括设计思想、流程图等内容）
4. 题目的难点及解决
5. 题目的运行及测试结果
6. 总体心得体会

第一章 实验一

1. 题目要求

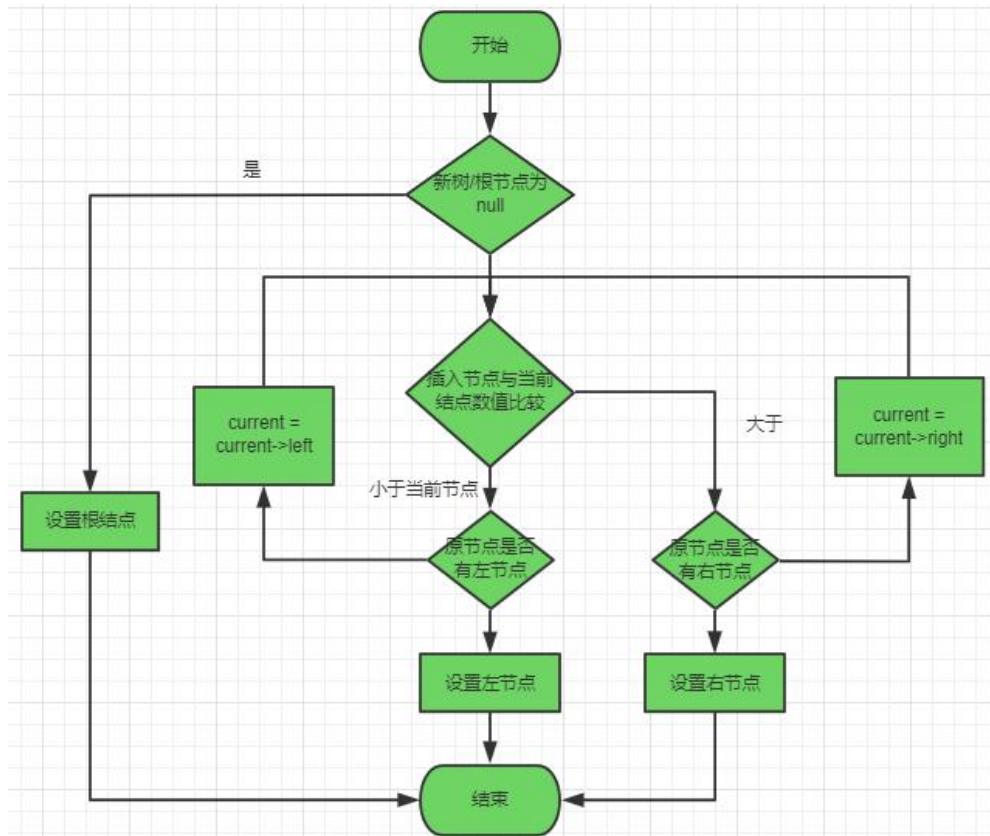
- (1) 编写函数，实现输入一组元素，建立一个带头结点的单链表；对该链表进行非递减排序；实现在非递减有序链表中删除值为 x 的结点；
- (2) 编写函数，采用链式存储和顺序存储实现队列的初始化、入队、出队操作；
- (3) 编写函数，建立有序表，利用二叉排序树的插入算法建立二叉排序树；在以上二叉排序树中删除某一指定关键字元素；采用折半查找实现某一已知的关键字的查找(采用顺序表存储结构)
- (4) 选用 1-3 的数据结构，编写程序实现下述五种算法：简单插入排序，冒泡排序，快速排序，归并排序，堆排序。

2. 题目的设计思路

二叉搜索树 (BST) 用于简单插入排序：使用二叉搜索树 (BST) 来实现简单插入排序。
TreeNode 结构表示 BST 中的每个节点。insert_bst 函数以保持排序顺序的方式将元素插入到 BST 中。排序算法：该程序包括冒泡排序、快速排序、归并排序和堆排序的实现。每个排序算法都被实现为一个独立的函数。输入和输出：该程序接受用户输入的元素数量和元素本身。输出每种排序算法的排序结果。

3. 题目的实现过程 （包括设计思想、流程图等内容）

利用 BST 进行简单插入排序：使用二叉搜索树来构建一个有序序列。
将元素插入 BST，然后执行中序遍历以获取有序序列。排序算法：冒泡排序：
比较元素并根据需要交换它们，迭代列表直到它排序完成。快速排序：
根据中枢元素对元素进行划分，并递归地应用该过程到子数组。归并排序：
将数组划分为两半，并递归地对这些子数组进行排序和合并。堆排序：
构建一个堆，然后重复地移除并放置最大的元素。输入和输出：用户输入包括元素数量和元素本身。打印每个排序算法的排序结果



4. 题目的难点及解决

用于简单插入排序的二叉搜索树：难点：确保 BST 保持正确的排序。

解决方案：insert_bst 函数递归地插入元素，同时保持二叉搜索树的性质。排序算法：难点：理解和实现各种排序算法。解决方案：每个排序算法都被独立实现，具有清晰的逻辑和步骤。

5.题目的运行及测试结果

The screenshot shows a C++ IDE with a project named 'C.PROJECT1'. The file explorer on the left lists various files including '1.1.c', '1.2.cpp', '1.3.cpp', '1.4.1.c', '2.1.1.cpp', '2.1.2.cpp', and '2.2.cpp'. The main editor displays the code for '1.1.c', which includes a function to insert a node into a linked list. The code is as follows:

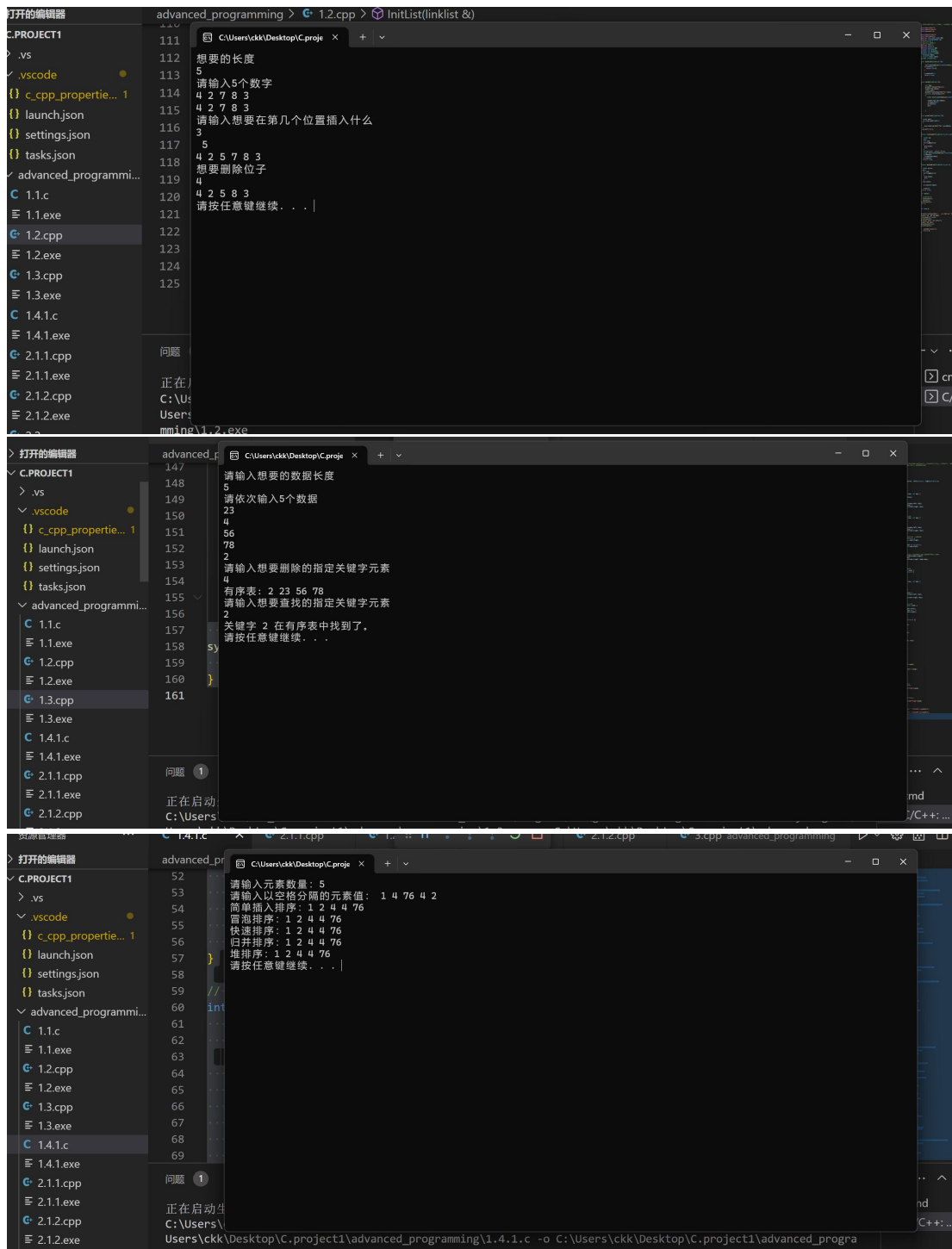
```

60 }
61 }
62 return 1;
63 }
64 }
65 void printList() {
66     while (head != NULL) {
67         cout << head->data << " ";
68         head = head->next;
69     }
70     cout << endl;
71 }
72 }
73 int main() {
74     std::cout << "请输入元素个数\n";
75     int n;
76     while (n < 1 || n > 10) {
77         std::cout << "输入元素值\n";
78         int x;
79         while (x < 1 || x > 10) {
80             std::cout << "原始链表\n";
81             int a[10];
82             for (int i = 0; i < n; i++) {
83                 a[i] = i + 1;
84             }
85             printList();
86             std::cout << "排序后的链表 1 2 3 5 7\n";
87             std::cout << "输入要删除的值\n";
88             int val;
89             while (val < 1 || val > 10) {
90                 std::cout << "删除值为 2 的结点后的链表 1 3 5 7\n";
91                 printList();
92                 std::cout << "请按任意键继续... \n";
93                 getChar();
94             }
95             deleteNode(val);
96             printList();
97         }
98     }
99 }
100 
```

The output window at the bottom shows the command prompt with the following text:

```

正在启动生成...
C:\Users\ckk\x86_64-8.1.0-release-win32-seh-rt_v6-rev0\mingw64\bin\g++.exe -fdiagnostics-color=always -g C:\Users\ckk\Desktop\C.project1\advanced_programming\1.1.c -o C:\Users\ckk\Desktop\C.project1\advanced_programming\1.1.exe
  
```



5. 总体心得体会

该程序全面展示了不同排序算法，并展示了它们在 C++ 中的实现。

使用二叉搜索树进行简单插入排序是一种独特的方法。

代码反映了对算法概念及其在排序中的实际实现的扎实理解。

总体而言，该程序展示了排序算法的多样性以及它们在不同场景中的适用性。

第二章 实验二

1. 题目要求

- (1) 编写函数，分别采用链式存储和顺序存储实现栈的初始化、入栈、出栈操作； 3.编写函数，采用链式存储实现队列的初始化、入队、出队操作
- (2) 编写函数，建立二叉树的二叉链表；实现二叉树的前中后序的递归和非递归遍历算法。

2.题目的设计思路

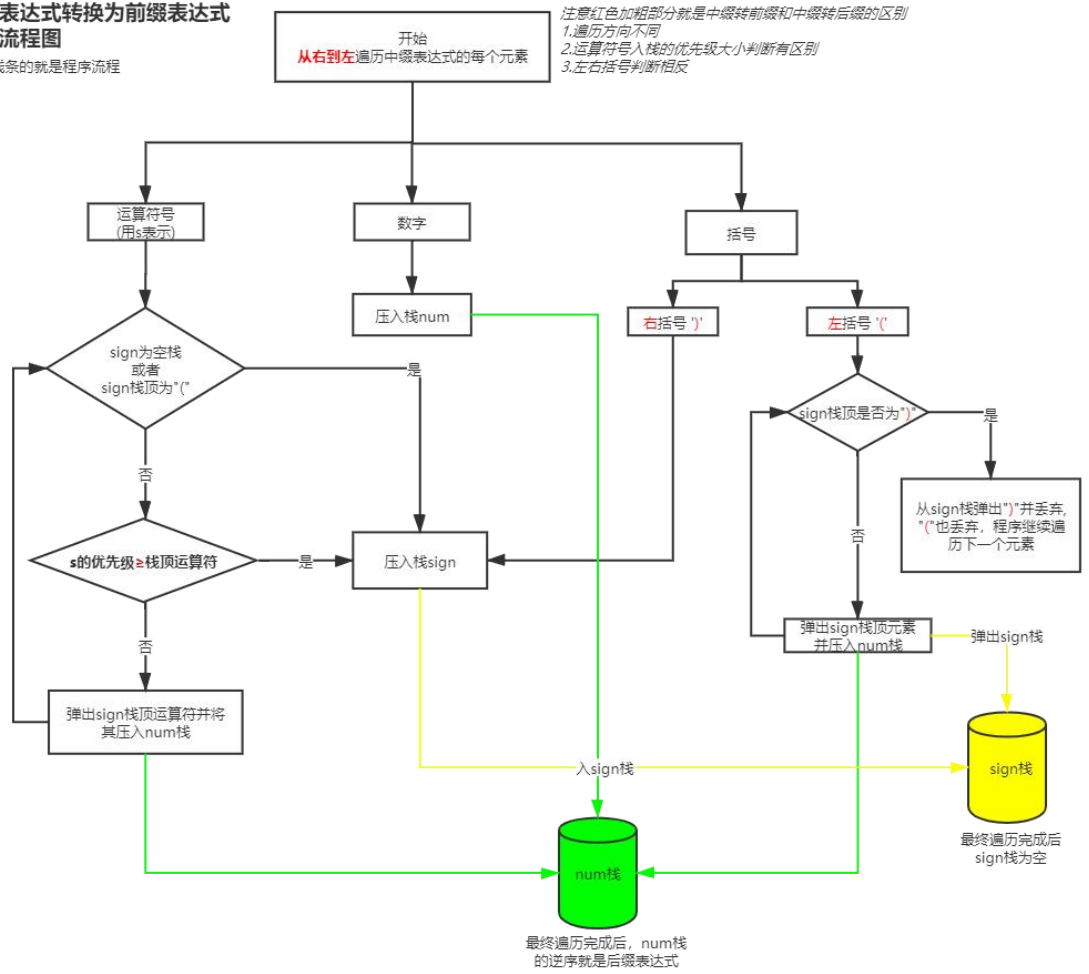
使用链表作为数据结构，通过节点之间的指针关系来表示栈的元素。Node 结构体用于表示链表中的节点，每个节点包含一个整数数据成员和一个指向下一个节点的指针。LinkedStack 类包含一个指向栈顶的指针。

3.题目的实现过程（包括设计思想、流程图等内容）

push(int value) 方法：创建新节点，将其链接到栈顶，并更新栈顶指针。pop() 方法：检查栈是否为空，如果不为空，则从栈顶弹出一个节点，返回其数据值，并释放节点内存。isEmpty() 方法：检查栈是否为空，返回布尔值。

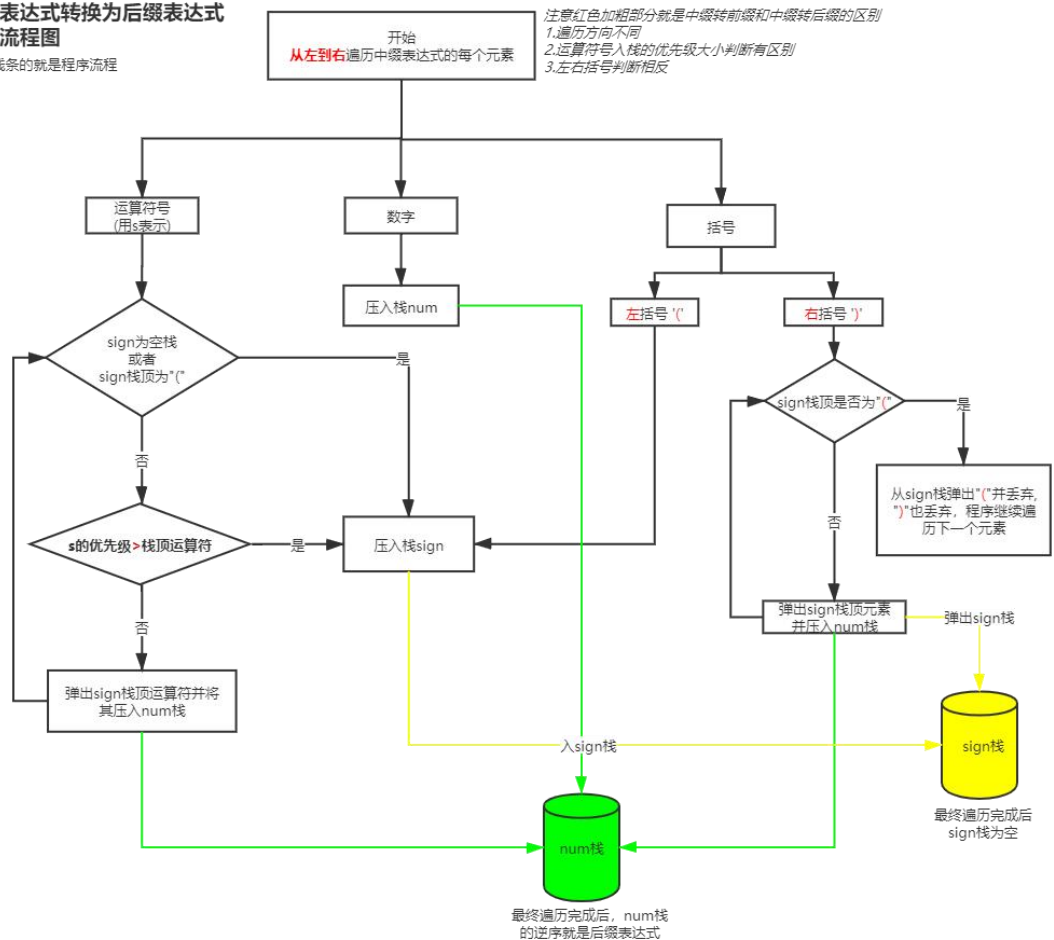
中缀表达式转换为前缀表达式 程序流程图

黑色线条的就是程序流程



中缀表达式转换为后缀表达式 程序流程图

黑色线条的就是程序流程



4.题目的难点及解决

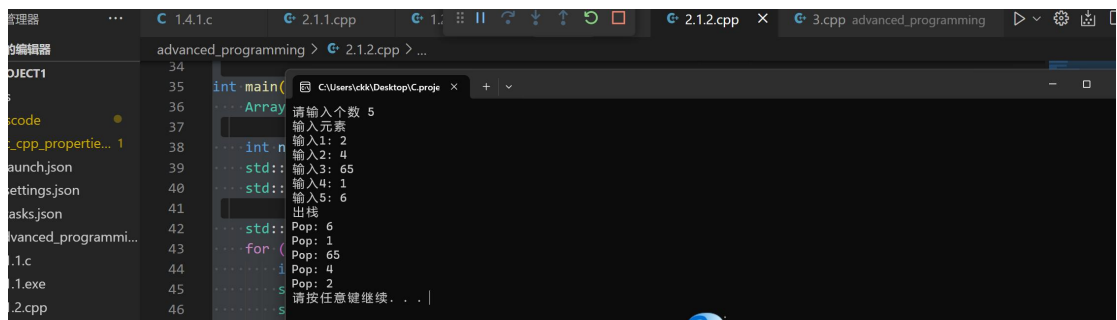
难点：确保链表维持正确的栈顺序。

解决：push 方法中通过将新节点链接到栈顶来维护栈的顺序，pop 方法通过更新栈顶指针来正确弹出元素。

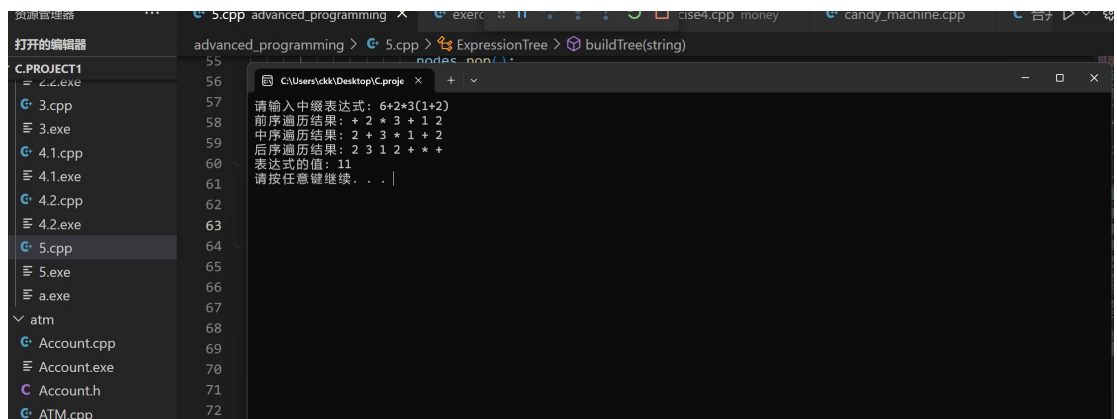
5.题目的运行及测试结果

```

advanced_programmi...
26  请输入个数 5
27  依次输入:
28  1 3 2 7 5
29  出栈元素
30  5
31  7
32  2
33  3
34  1
35  请按任意键继续... |
  
```



```
34
35 int main()
36 {
37     Array
38     {
39         std::
40         std::
41         出栈
42         std::
43         for (
44         Pop: 6
45         Pop: 1
46         Pop: 65
47         Pop: 4
48         Pop: 2
49         请按任意键继续. . . |
```



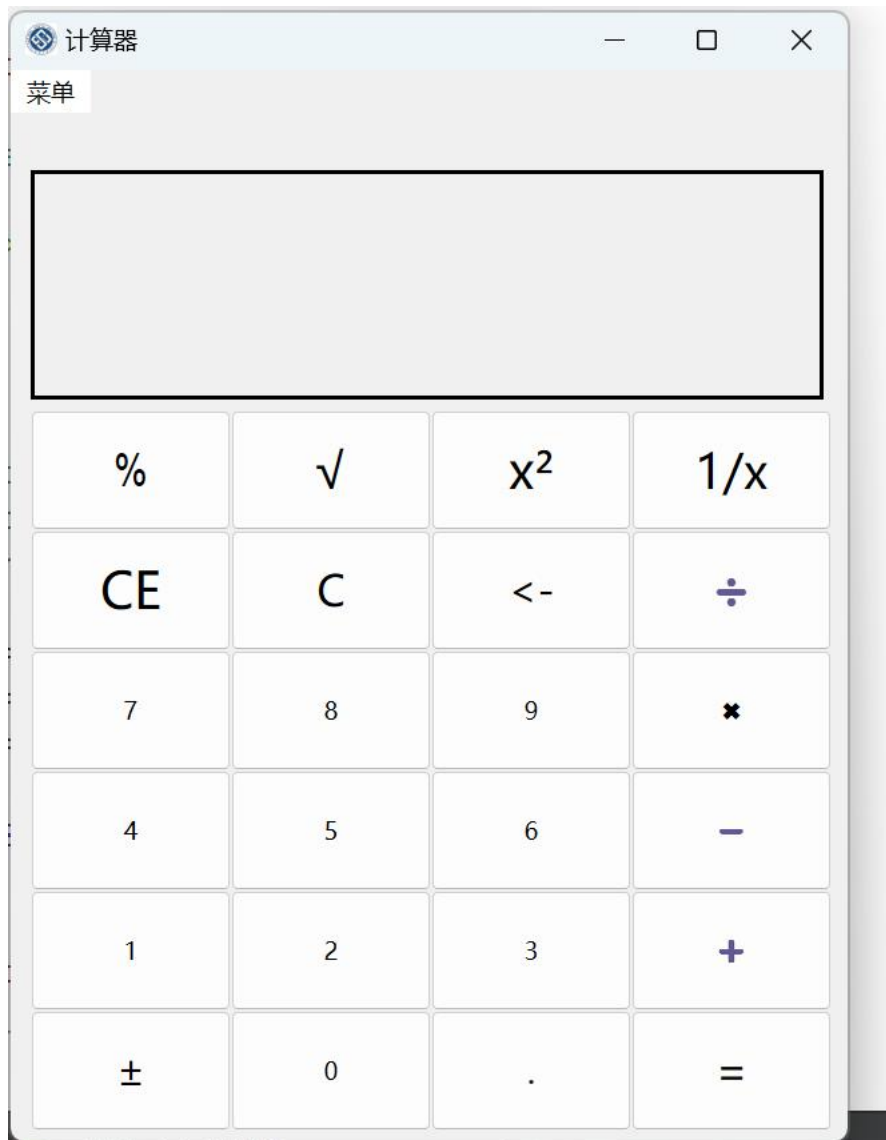
```
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

6.总体心得体会

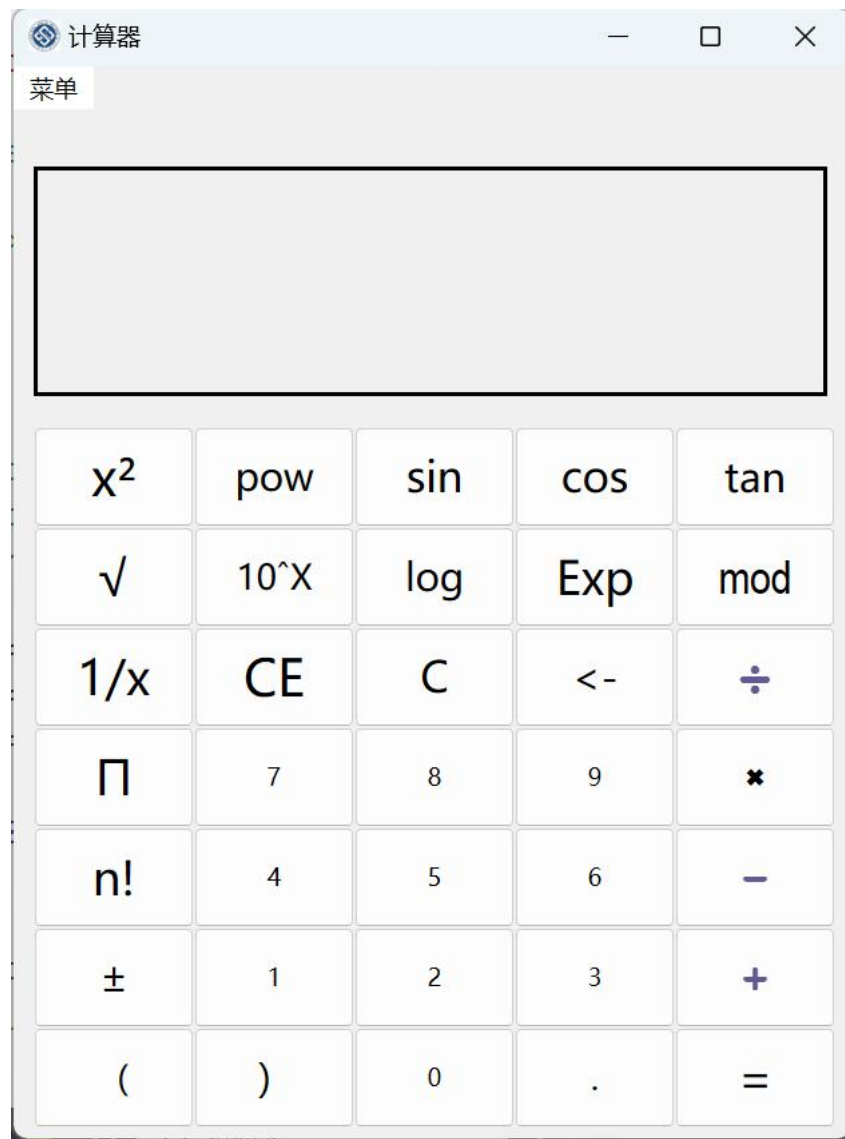
这两个程序展示了栈的两种常见实现方式：链表和数组。使用链表实现的栈更加灵活，可以动态调整大小，但可能会占用更多的内存。使用数组实现的栈在空间上可能更有效，但需要预先定义大小，且可能无法灵活扩展。在设计栈时，需要考虑到各种边界情况，如栈满或栈空的处理。理解不同实现方式的优势和局限性有助于根据具体需求选择适当的数据结构。

7.选做题：计算器

标准模式



科学计算器模式



信息界面



1. 设计思路:

界面设计: 通过 Qt 框架创建一个图形用户界面 (GUI) 的计算器应用。使用不同的按钮和布局来支持标准、科学和其他功能。

计算逻辑设计: 设计计算器的计算逻辑, 支持基本的四则运算、括号、科学函数 (如 \sin 、 \cos 、 \log) 等。使用栈数据结构处理中缀表达式转后缀表达式, 然后进行计算。

状态管理: 引入状态管理, 包括初始状态、输入中、计算完成等, 以正确处理用户的输入和计算过程。

2. 实现过程:

界面实现: 使用 Qt 的设计器创建计算器的图形界面, 包括数字按钮、操作符按钮、显示屏等组件。连接按钮的信号和槽, 使得按钮点击时触发相应的操作。

计算逻辑实现: 在 `Widget` 类中实现计算器的逻辑, 包括输入处理、后缀表达式转换、运算等功能。使用自定义的 `Calculator` 类处理计算逻辑, 包括中缀转后缀、运算等。

状态管理实现: 通过枚举类型 `'Mycase'` 管理计算器的状态, 以确保在不同状态下进行正确的操作。

3. 难点及解决:

中缀表达式转后缀表达式: 这是一个经典的计算器实现难点, 通过使用栈数据结构, 逐个处理输入的表达式, 将中缀表达式转换为后缀表达式。

科学函数的处理: 对于涉及到科学函数的计算, 需要正确处理函数的输入、输出, 并在界面上显示正确的计算结果。

状态管理: 确保在不同状态下按钮的可用性、显示屏的显示等状态切换。

4. 总体心得体会:

完成了一个基本的计算器应用, 通过 Qt 框架的使用, 熟悉了图形界面的设计和实现。

对计算器的计算逻辑有了更深入的理解, 特别是中缀表达式到后缀表达式的转换和计算的过程。

学会了如何使用信号和槽机制来连接用户界面和后台逻辑, 实现了用户与应用之间的交互。

第三章 实验三

1. 题目要求

- (1) 编写函数, 实现创建哈夫曼树的算法;
- (2) 编写函数, 实现求哈夫曼编码的算法;
- (3) 根据已知的字符及其权值, 建立哈夫曼树, 并输出哈夫曼编码

2. 题目的设计思路

哈夫曼编码的构建思路:

输入字符的个数和每个字符的权值。创建优先队列, 按照频率从小到大排序。依次取出两个频率最小的节点, 创建新节点, 频率为两子节点频率之和, 加入队列。重复上述步骤, 直至队列中只剩一个节点, 即为哈夫曼树的根节点。

生成哈夫曼编码的思路:

从哈夫曼树的根节点开始递归遍历, 每次遍历左子树加 "0", 右子树加 "1"。当遍历到叶子节点时, 将字符和对应的哈夫曼编码存储起来。

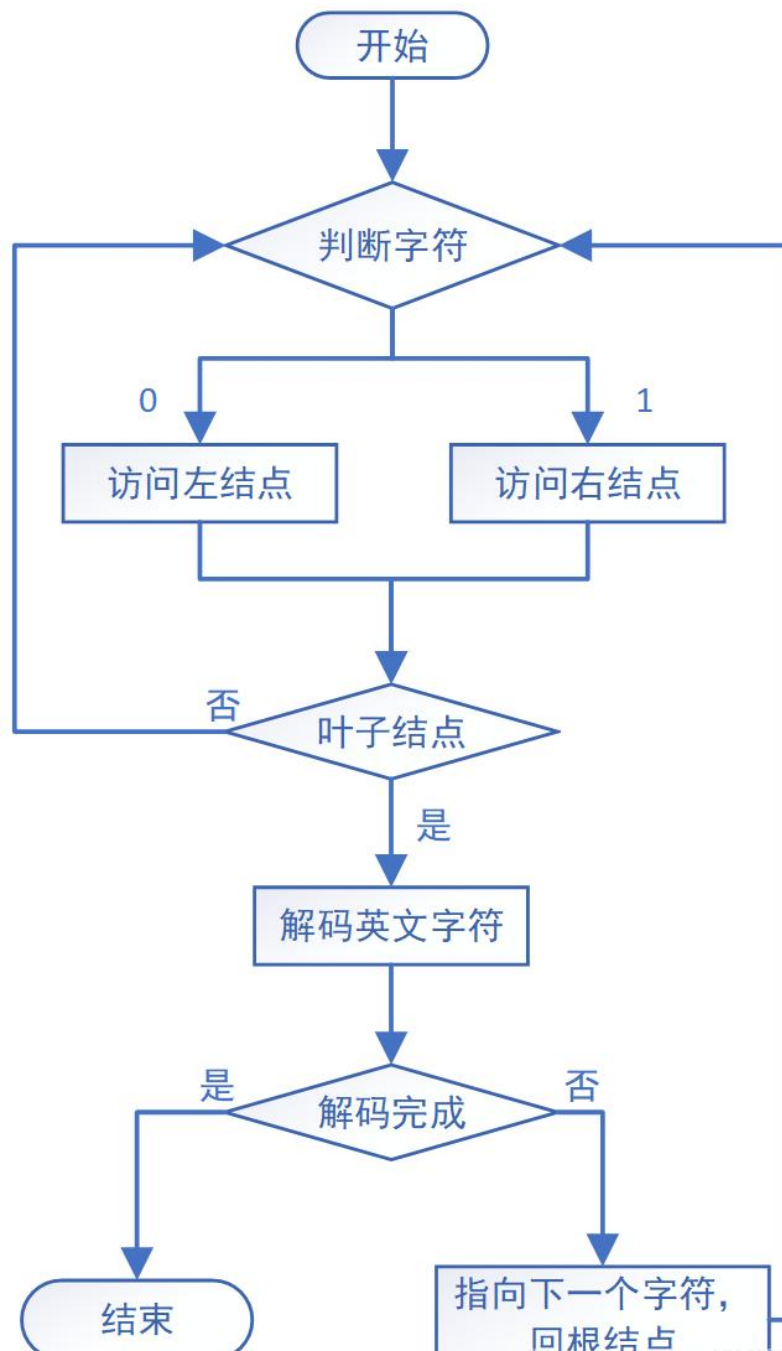
3. 题目的实现过程 (包括设计思想、流程图等内容)

构建哈夫曼树：

用户输入字符的个数和每个字符的权值。创建一个优先队列 minHeap, 存储 HuffmanNode* 类型的指针, 按照频率从小到大排序。将字符及其权值转换为叶子节点, 加入优先队列。循环取出两个频率最小的节点, 创建新节点作为它们的父节点, 频率为两子节点频率之和, 加入队列。当队列中只剩一个节点时, 表示哈夫曼树构建完成, 返回根节点。

生成哈夫曼编码：

使用递归函数 generateHuffmanCodes, 从哈夫曼树的根节点开始。每次递归左子树加 "0", 右子树加 "1"。当遍历到叶子节点时, 将字符和对应的哈夫曼编码存储在数组 huffmanCodes 中。



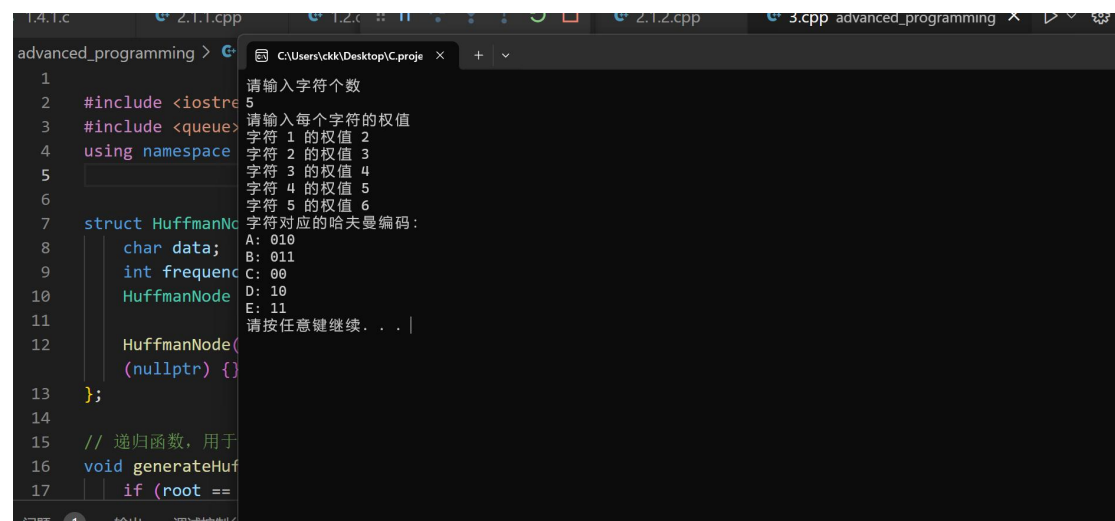
4.题目的难点及解决

难点：构建哈夫曼树并生成哈夫曼编码。

解决：

使用优先队列存储节点，便于每次取出频率最小的节点。通过递归遍历哈夫曼树，生成每个字符的哈夫曼编码。

5.题目的运行及测试结果



```
1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 struct HuffmanNode
6 {
7     char data;
8     int frequency;
9     HuffmanNode* left;
10    HuffmanNode* right;
11
12    HuffmanNode(char data, int frequency, HuffmanNode* left, HuffmanNode* right) : data(data), frequency(frequency), left(left), right(right) {}
13 };
14
15 // 递归函数，用于生成哈夫曼编码
16 void generateHuffmanCode(HuffmanNode* node, string code)
17 {
18     if (node == nullptr) return;
19     if (node->data != '\0')
20     {
21         cout << node->data << ": " << code << endl;
22     }
23     generateHuffmanCode(node->left, code + "0");
24     generateHuffmanCode(node->right, code + "1");
25 }
```

请输入字符个数
5
请输入每个字符的权值
字符 1 的权值 2
字符 2 的权值 3
字符 3 的权值 4
字符 4 的权值 5
字符 5 的权值 6
字符对应的哈夫曼编码：
A: 010
B: 011
C: 00
D: 10
E: 11
请按任意键继续. . .

6.总体心得体会

通过实现哈夫曼编码，深入理解了哈夫曼树的构建和编码生成的原理。了解了优先队列在哈夫曼树构建中的应用，有效地处理了频率排序的问题。通过递归实现哈夫曼编码的生成，增强了对树形结构的理解。总体来说，通过这个题目，加深了对数据结构和算法的认识，提升了编程和解决问题的能力。

第四章 实验四

1. 题目要求

(1) 编写用邻接矩阵表示无向带权图时图的基本操作的实现函数，主要包括：①初始化邻接矩阵表示的无向带权；②建立邻接矩阵表示的无向带权图；③输出邻接矩阵表示的无向带权图；编写生成最小生成树的 Prim 算法函数以及输出边集数组的函数

(3) 利用邻接矩阵构造有向带权图，并求出某一顶点到其余顶点的最短路径并打印输出；编写求最短路径的 Dijkstra 算法函数，该算法求从顶点 i 到其余顶点的最短路径与最短路径长度；3.编写打印输出从源点到每个顶点的最短路径及长度的函数；

2. 题目的设计思路

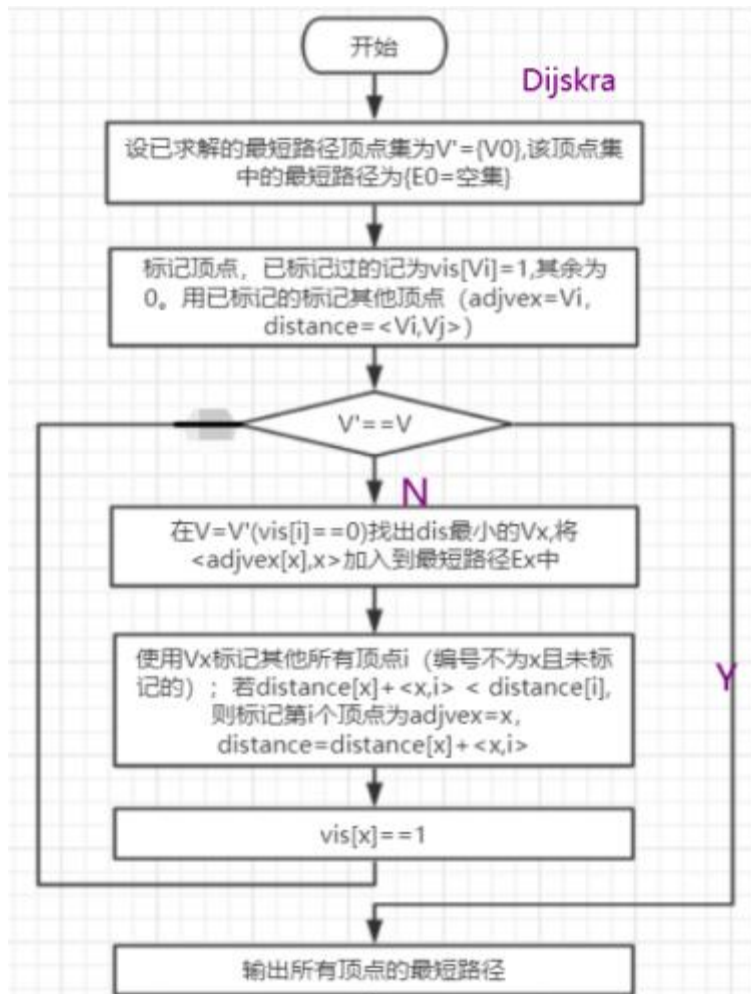
图的表示：

使用邻接矩阵表示有向图，其中 $\text{adjacencyMatrix}[i][j]$ 表示从顶点 i 到顶点 j 的边的权重。

Dijkstra 算法：

使用集合 (set) 维护待处理的顶点，并按照距离从小到大排序。初始化距离数组 distance 和父节点数组 parent ，其中 $\text{distance}[i]$ 表示从源点到顶点 i 的最短路径长度， $\text{parent}[i]$ 表顶点 i 在最短路径中的前一个顶点。从源点开始，逐步更新距离数组和父节点数组，直到所有顶点都被处理

3. 题目的实现过程（包括设计思想、流程图等内容）



图的表示:

使用邻接矩阵表示有向图, 通过 `addDirectedEdge` 方法添加边的权重。

Dijkstra 算法:

使用 `set` 来存储待处理的顶点, 通过 `dijkstraShortestPath` 方法实现 Dijkstra 算法。在每一步迭代中, 选择距离最小的顶点 `u`, 并更新与 `u` 相邻的顶点的距离和父节点。最终通过 `printShortestPaths` 方法打印从源点到各顶点的最短路径和长度

4.题目的难点及解决

难点:

确保正确实现 Dijkstra 算法的核心逻辑, 包括集合的维护和距离数组的更新。

解决:

使用 `set` 来维护待处理的顶点, 并按照距离排序, 确保每次选择距离最小的顶点。

使用 `distance` 数组和 `parent` 数组来动态更新最短路径的信息。

5.题目的运行及测试结果

```
advanced_programming > 4.1.cpp > ...  
C:\Users\ckk\Desktop\C.proje  
请输入顶点数: 5  
请输入边数: 3  
请输入边的信息 (起点 终点 权重):  
1 2 4  
4 1 5  
0 2 3  
邻接矩阵表示的无向带权图:  
INF INF 3 INF INF  
INF INF 4 INF 5  
3 4 INF INF INF  
INF INF INF INF INF  
INF 5 INF INF INF  
最小生成树的边集数组:  
0 - 2  
2 - 1  
1 - 4  
请按任意键继续. . . |
```

```
advanced_programming > 4.2.cpp > ...  
1 #include <iostream>  
2 #include <climits>  
  
C:\Users\ckk\Desktop\C.proje x + v  
2.1.1.c  
2.1.1.e 请输入顶点数和边数：4 2  
2.1.1.e 请输入每条边的起点、终点和权重：  
2.1.2.c 1 2 45  
2.1.2.c 3 0 3  
2.1.2.e 请输入源顶点：1  
2.2.cp 从顶点 1 到各顶点的最短路径和长度：  
2.2.ex 到顶点 0 的最短路径：无路径到达，长度：2147483647  
2.2.ex 到顶点 2 的最短路径：1 2，长度：45  
2.2.ex 到顶点 3 的最短路径：无路径到达，长度：2147483647  
3.cpp 请按任意键继续... |  
3.exe  
4.1.cp  
4.1.ex  
4.2.cp  
4.2.ex  
5.cpp  
5.exe  
a.exe  
atm  
Accou  
Accou  
Accou
```

6.总体心得体会

通过实现 Dijkstra 算法，加深了对图的最短路径问题的理解。学会使用集合和数组来有效地维护算法的状态，确保算法的正确性和效率。通过输入不同的图结构和权重，验证算法在不同情境下的表现，提高了对算法性能的认识。