**数据结构与程序设计专题实验报告**

实验一表达式求值

1 实验目的：

熟练掌握栈、队列等基本操作及其在实际问题中的应用

2 实验要求；

(1)实现栈的push、pop基本操作；

(2)检测表达式的输入是否是正确的数学表达式；

(3)对于正确的数学表达式求取其值。

1）数学表达式需支持小数、加减乘除、小括号、中括号、幂运算。为简化逻辑，不需要考虑负数情况。

2）不正确的表达式可能包括：字母、非运算符号、括号不匹配，运算符的排列不符合表达式形式等多种情况。

3）表达式通过命令行参数读取 。

3 设计思路；

表达式求值是程序设计中的一个重要的问题。它的实现是栈应用的案例。我在进行操作中首先置操作数栈为空栈，表达式起始符“#”为运算符栈的栈底元素。依次读入表达式中的每个字符，如果操作数则进OPND栈，若是运算符则和OPTR栈的栈顶运算符比较优先权后做相应的操作，直到整个表达式求值完毕。

但是显然，如果要具体到题目，我们还需考虑一些如对于括号的检查等问题，这些问题都可以通过对栈的合理应用来解决。

4 运行结果；



5 问题与思考；

问题:在程序编写的过程中，我们不可避免的遇到了一些问题，这其中包括对于字符串的运算和运算栈使用了两个栈，利用了先入后出的方法来实现，但是在做题的过程中，发现对于树的前序，中序，后序遍历就是对于表达式的前缀，中缀，后缀。

解答：显然可以，我在空余时间发现，我们可以先用中序遍历生成一棵树，然后借助一个栈，实现对前序遍历的计算。最后我花费了接近2个小时，完成了部分，但是确实大大增加了空间复杂度。

思考：我一直不清楚为什么要把计算式分为前缀，中缀和后缀。

最终我看了计算机原理，原来，虽然人的大脑很容易理解与分析中缀表达式，但对计算机来说中缀表达式却是很复杂 的，因此计算表达式的值时， 通常需要先将中缀表达式转换为前缀或后缀表达式， 然后再进 行求值。对计算机来说，计算前缀或后缀表达式的值非常简单。

5.源代码（输入命令行参数输入）



实验二基于串的模式匹配

1 实验目的；

熟练掌握串的实现，学会使用KMP算法解决字符串模式匹配问题

2 实验要求；

输入：通过命令行参数输入原字符串和模式字符串。输出：（1）命令行参数不正确输出字符串ERROR\_01；（2）如果查找到模式串，输出关键字在字符串中的位置（计数从1开始）；（3）如果未找到模式串则输出-1 。

1、实现串的StrAssign、StrCompare、ClearString等基本操作；

2、使用KMP算法实现模式匹配

注：1）使用堆分配的方式申请串内存空间；

2）表达式通过命令行参数读取，并判断命令行参数的数量是否正确

编码要求：

(1)优选C语言，禁止直接调用C++ STL库；

(2)除循环变量外，其它变量命名使用有明确含义的单词或缩写，不建议使用拼音；

(3)禁止出现魔鬼数字；

(4)添加必要的程序注释；

(5)统一代码格式，例如：{}和空行；

(6)变量初始化，不要依赖默认赋值；

(7)入参检查，“外部输入输入不可靠”，指针判空（一级指针、二级指针……），循环变量上下限；

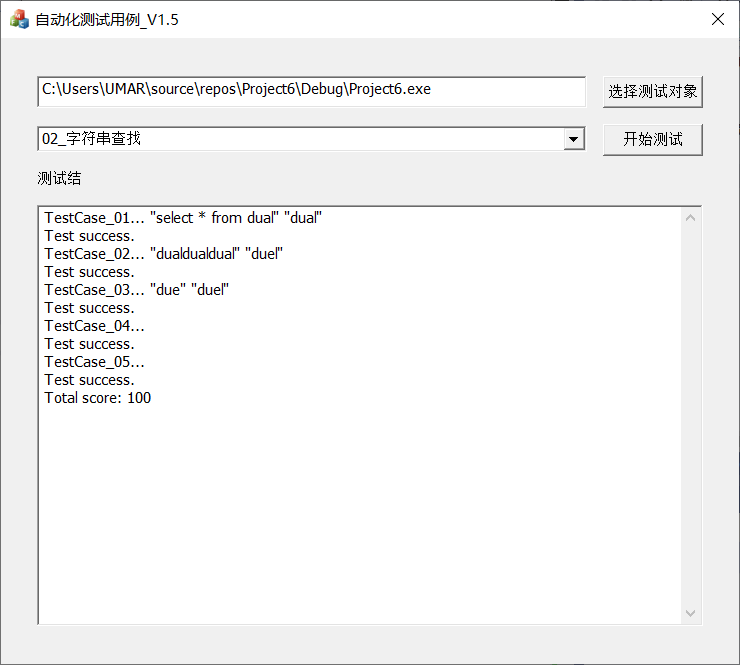
(8)malloc与free配对。

3.设计思路；

假设主串为”S1S2\*\*\*Sn” 模式串为”P1P2P3\*\*\*Pn” 显然当匹配过程中产生“失配”时，模式串“向右滑动”可行的距离多远，当主串中第i个字符与模式中第j个字符“失配”时，直接改便j的数值来达到运算目的。那么就很显然，如果已经得到一个已知的NEXT数组，代码形式基本与普通的顺序查找相类似。但是求解NEXT的数组本身是很有难度的，我们不妨采用使用在NEXT数组去寻找NEXT数组的方式进行构造。也就是说此时，在NEXT数组里寻找NEXT数组.

在实验中我们不可避免的还遇到一些问题，如我们如何解决对于堆形式的字符储存的基本运算，但实际上这个是很容易的，因为它强调的是储存方式，所以比较，清除等思想基本与原来相同。

4.运行结果；



5 问题与思考；

我们在使用KMP算法的时候，不难发现当算法出现011115，会使得NEXT数组呈现递增顺序，而这显然会造成新的问题，因为他会不断的母串进行比较，也就是指向子串的指针在不断做无效的回溯，这显然是我们不期望的。因而我们在此处采用新的NEXTVAL改进法则，这种法则使得像如上情况发生时，NEXTVAL[i]=NEXTVAL[j].

我对于堆这种储存方式有着自己的思考，我认为堆的本质就是会生成一个动态的指针，指针始终指向字符串的初始点，&ch[0],而长度的申请则完全取决与我们的操作，可以很大程度上解决空间存储浪费的问题。

6.源代码



实验三 基于哈夫曼树的编码/译码

1.实验目的；

掌握二叉树的生成、遍历等操作，及哈夫曼编码/译码的原理。

2.实验要求；

程序要求：

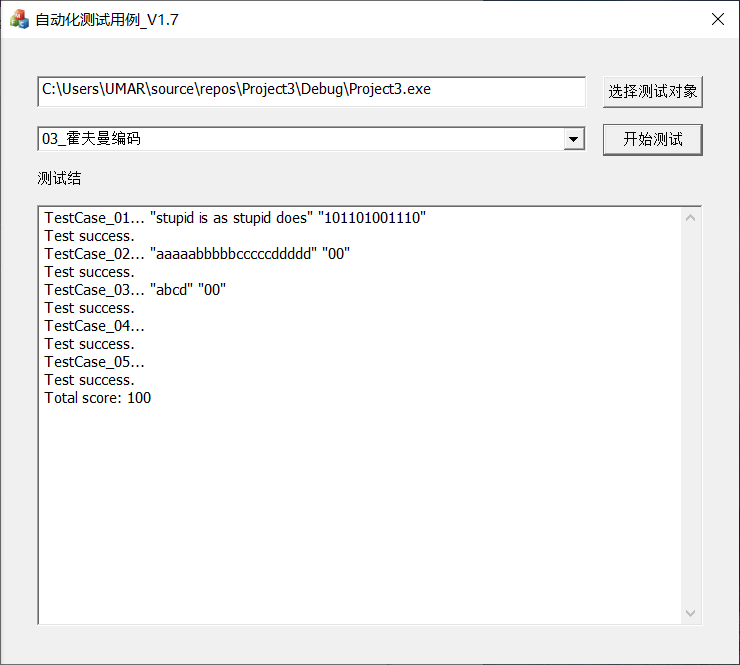
1 基于该哈夫曼树，实现非递归的先序遍历算法，输出该树所有的节点、节点的权值、节点的度和节点所在的层数；

2 在实现时要求哈夫曼树的左右孩子的大小关系满足，左孩子节点权值小于右孩子节点权值，若左右孩子权值相等，按字母顺序排列（序号小的字母在左孩子）。

3.设计思路；

由于赫夫曼树中没有度为1结点，则一棵树有n个叶子节点的赫夫曼树共有2N-1个节点，可以储存在一个2N-1的一维数组中。那么如何选定节点结构呢？由于在构成赫夫曼树之后，为求编码需要从根中走出一条从根到叶子的路径。而栈的遍历就更加的简单了，因为我们显然可以发现，在实际中，我们只需要把节点存进来，在从栈顶取得一个元素，将他的左右节点存入栈中，实现先来后出的原则。

4.运行结果；



5.问题与思考；

我们这时候思考的主要问题就是霍夫曼树的建立和其中的原理。实际上是很简单的，但是老师在要求中特别的加入了一些条件，包括但是不限于要求我们在权重相同时解决排序，这等于是说我们生成的霍夫曼树是唯一的。

这个问题最初让我思考了很长的时间，我不是太明白如何去储存这个输入部分，在CSDN上有使用结构体来解决这个问题，通过霍夫曼树进行比较和判断，可以非常快的得到相关的顺序。

思考：霍夫曼树的应用是用于压缩文件，使得文件得到了充分的压缩，同时值得强调的是在我的实际应用中，我发现压缩的并不是太多，因为压缩比取决于概率分布，如果要压缩的符号都是等权值的，那实际上并没有起到压缩的作用，只是提供一种翻译的方法而已。

6.源代码（采用了命令行参数）



实验四 无向图最短路径搜索

1 实验目的；

熟练掌握图的操作，掌握 Dijkstra 算法的原理

2 实验要求；

输入：通过命令行参数输入起点和终点的位置名称。输出：（1）命令行参数不正确输出ERROR\_01; （2）获取最短路径失败时输出ERROR\_02; （3）获取最短路径成功时输出路径长度。

1校内知名建筑物示意平面图（其中“传送门”用于增加网络复杂度），以边表示建筑物间的路径，各条路径上方的数字表示路径长度。

2 针对该图进行构建数据结构和算法，通过命令行参数输入任意两建筑物的名称，可查询建筑物间的最短路径长度，并输出最短路径。

编码要求：

1、优选C语言，禁止直接调用C++ STL库；

2、除循环变量外，其它变量命名使用有明确含义的单词或缩写，不建议使用拼音；

3、禁止出现魔鬼数字；

4、添加必要的程序注释；

5、统一代码格式，例如：{}和空行；

6、变量初始化，不要依赖默认赋值；

7、入参检查，“外部输入输入不可靠”，指针判空（一级指针、二级指针……），循环变量上下限；

8、malloc与free配对；

9、尽量少用全局变量；

10、编译错误解决，从前往后处理，提示出错的行不一定是错误的根因；

3 设计思路；

这个问题本质上就是解决在有权重的图中，如何求从某一源点到其余各点的最短路径，那我们可以先引入一个辅助的向量D[W],他表示从该点到其余个点的最短距离，我们每次将到源点最短的点进行更新，通过这个新的纳入的点，我们去进行判断，寻找之后的最短距离。这是极为有趣的。也就是更新D[W],同时值得强调的是我们在寻找中，还要记录到已知点的上一个节点，方便之后对其进行回溯。

4 运行结果；



5 问题与思考；

图的查找，如果想要去求得到任意两点的最短距离，有一个解决方式就是每次以一个顶点为源点，重复执行迪杰斯特拉算法n次，这样，便可以求得每一对顶点之间的最短路径，总的执行时间为o(n3)。

思考：看到课本上提到的FLOID算法，这种思想是，假设求从顶点Vi到Vj的最短路径，

Floyid算法的基本思想是：递推产生一个N阶方阵序列，其中Ak[i][j]表示从顶点vi到顶点vj的路径长度，K表示绕行第K个顶点的运算步骤。初始时，对于任意两个顶点vi和vj,如果它们之间存在一个边，则以此边上的权值作为他们之间的最短路径长度；若他们之间不存在有向边，那我们用无穷大作为他们之间的最短路径长度。以后我们逐渐尝试在原来路径上加入顶点k作为中间顶点。若增加中间顶点后，得到的路径比原来的路径长度减少了，则以此新路径代替原路径。

思考：dijkstra算法本质上就是对于广度遍历的重演，他通过寻找每次到达已知点的最小值，实现最终对于所有部分的遍历。

6.源代码：（采用命令行参数进行输入）

