**数据结构实验报告**

**实验题目：判别回文字符**

|  |  |  |  |
| --- | --- | --- | --- |
| 班级：2020211305 | 姓名：缪奇志 | 学号：2020211345 | 分工：用户使用说明和测试结果 |
| 班级：2020211305 | 姓名：倪玮昊 | 学号：2020211346 | 分工：详细设计和调试分析报告需求分析和概要设计 |
| 班级：2020211305 | 姓名：杨智杰 | 学号：2020211356 | 分工：详细设计和调试分析报告 |

1. **需求分析**

题目任务：

根据输入的一串以#结尾的字符串来判断它是否是回文字符。

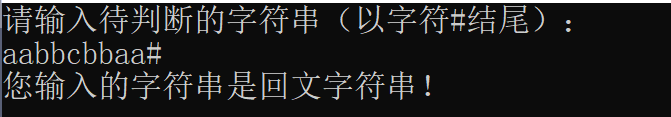
程序的输入和输出：

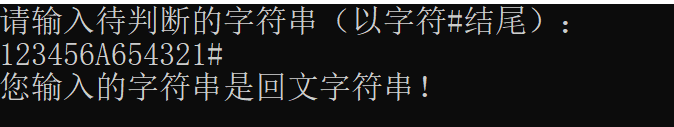
|  |  |  |  |
| --- | --- | --- | --- |
| 输入 | 数据类型 | 值的范围 | 代表意义 |
| data[] | 字符串 | 9999 | data为题目要求我们要判断的字符+# |

程序功能实现：

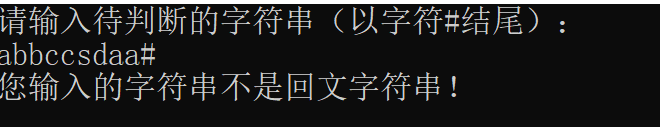
通过输入的以#结尾的字符串来判断该字符串是否为回文字符串。

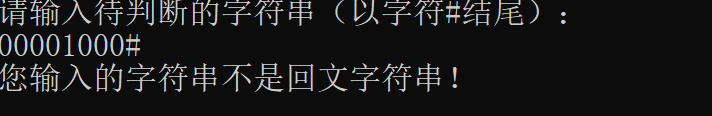
正确数据输出结果范例:

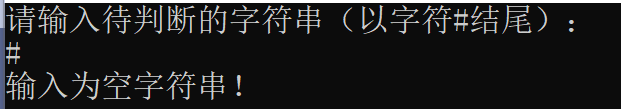




错误数据输出结果范例：





****

1. **概要设计**

**算法思路概述：**

本程序主要实现了:

首先对栈和队列进行初始化，由使用者输入一串字符串，并让字符分别进行进栈和进入队列，由程序判断是否为空字符串,如果为空字符，则进行栈和队列和销毁，并且输出“该字符串为空字符”，如果不是空字符则进行字符比较,在栈中的字符串进行出栈操作，在队列中的字符串则进行出队，因为出栈时在后面的字符先出，而出队列时，在最前面的字符先出，分别将两者出去的字符进行比较，如果不相等则返回bool值false，输出“您输入的字符串不是回文字符串！”，如果直到队和栈中的前置节点等于后置节点（即所有字符都已经出栈和出队）则跳出循环，返回bool值true，程序输出“您输入的字符串是回文字符串”,并进行栈和队列的销毁。

**数据结构类型的定义：**

typedef char Element;

定义结构体 Element。

struct myStack

{

Element\* base;

Element\* top;

int size;

};定义结构体mtStack作为栈，指针base来记录栈的底部位置以及字符，指针top记录栈的顶部位置以及字符。size为栈的容量

struct myQueue

{

Element data[MAXSIZE];

int front;

int rear;

};定义结构体myQueue作为队列，rear为队尾的位置，front为对头的位置，data[]记录输入的字符串

**主程序流程：**

InitStack(stack)

InitQueue(queuePtr)

JudgeString <-Input(stack, queuePtr)

if JudgeString == false

printf("输入为空字符串！\n")

DelStack(stack)

DelQueue(queuePtr)

return 0

JudgePalindrome <- Compare(stack, queuePtr)

if JudgePalindrome == true

printf("您输入的字符串是回文字符串！\n")

else printf("您输入的字符串不是回文字符串！\n")

DelStack(stack)

DelQueue(queuePtr)

return 0

**各层次模块层次调用关系：**

函数开始调用InitStack来创建一个栈，调用IitStack,再调用Input函数来判断输入的字符串是否是空字符串，在Input函数中调用Push进行入栈，调用Enqueue进行入队,如果Input返回的bool值是false则调用DelStack和DelQueue来销毁栈和队列，若返回的是true，则调用函数Compare来判断字符串是否是回文,在compare函数中调用pop进行出栈，使用Dequeue进行出栈，通过循环比较返回布尔值到主函数，通过不同的布尔值输出不同结果，最后统一进行栈和队列的销毁。

1. **详细设计**

**各种函数的设计：（伪代码）**

1. 初始化栈：

s.base <- (Element\*)malloc(sizeof(Element) \* INITSIZE)

if s.base == NULL

printf("分配空间不足，栈初始化失败！\n")

return

s.top <- s.base

s.size <- INITSIZE

return

1. 初始化队列：

qPtr <- (myQueue\*)malloc(sizeof(myQueue));

if qPtr == NULL

printf("没有多余空间分配给队列！\n")

return

}

(\*qPtr).front <- (\*qPtr).rear <- 0

return

1. 让字符串进栈和进队列：

printf("请输入待判断的字符串（以字符#结尾）：\n")

e

scanf("%c", &e)

if e == '#' return false;

while e != '#' do

repeat

Push(s, e)

Enqueue(qPtr, e)

scanf("%c", &e)

end

return true

1. 具体入队：

if ((\*qPtr).rear + 1) % MAXSIZE == (\*qPtr).front

printf("队列已满，%c无法入队！", e)

return

(\*qPtr).rear <- ((\*qPtr).rear + 1) % MAXSIZE

(\*qPtr).data[(\*qPtr).rear] <- e

return

1. 具体入栈：

if s.top - s.base + 1 > s.size

s.base <- (Element\*)realloc(s.base, sizeof(Element) \* (s.size + ADDSIZE))

if s.base == NULL

printf("栈满且没有多余空间可分配，新增元素%c无法入栈！\n", e)

return

s.size <- s.size + ADDSIZE

\*(s.top) <- e

s.top<-s.top+1

return

1. 比较字符串：

while s.top != s.base && (\*qPtr).rear != (\*qPtr).front

repeat

if Pop(s) != Dequeue(qPtr)

return false;

end

return true

1. 出队：

if \*qPtr).rear == (\*qPtr).front

printf("队列为空，无法出队！")

(\*qPtr).front <- ((\*qPtr).front + 1) % MAXSIZE

return (\*qPtr).data[(\*qPtr).front]

1. 出栈:

if s.top == s.base

printf("栈空，没有可取出的元素！\n")

return ERROR;

s.top<-s.top-1

return \*(s.top)

1. 销毁队列:  
   free(qPtr);

return

1. 销毁栈:

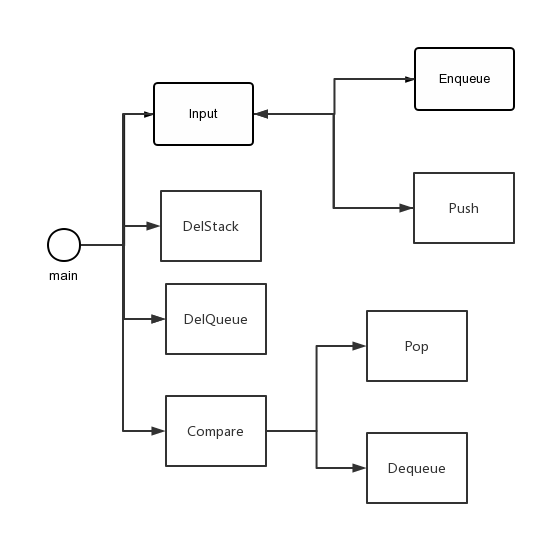
free(s.base)

s.base <- s.top = NULL

s.size <- 0

return

**函数调用关系图：**



1. **调试分析报告**

**调试过程中遇到的问题：**  
1.队列的初始化函数忘记传引用，导致指向malloc开辟空间的指针为在函数内的局部变量，在函数运行结束后就不再存在，新开辟的空间无法访问。通过对指针的传引用解决了这一问题。  
2.进行出队操作时忘记了front指向的区域并没有存放队列元素，导致运行出错误结果。后来通过先移动front指针再取出元素的步骤解决了这一问题。  
**算法时空分析：**

Input:o(n)

DelStack:o(1)

DelQueue:o(1)

Compare:o(n)

InitQueue:o(1)

InitStack:o(1)

Pop:o(1)

Dequeue:o(1)

Enqueue:o(1)

Push:o(1)

**改进设想：**

可以尽量将不同的函数整合在一起，用一次循环来解决分多次循环解决的问题。

**经验体会**：

1. 心得：  
   通过本次实验，进一步地理解了栈和队列这两个数据结构的概念，同时牢记了队列front指向队列首个元素前驱这一细节。通过巧妙运用栈先入后出和队列先入先出的性质，对同一个字符串分别入栈和入队，然后出栈和出队便可以判断字符串是否回。虽然使用的是静态队列，但是取模循环这一处理也增大了队列容量，足够应对较长的字符串输入。同时本实验需要特别考虑输入为空字符串的情形，即“#”，此时栈和队列均为空，并没有元素可以出栈和出队判断是否相等。
2. **用户使用说明**

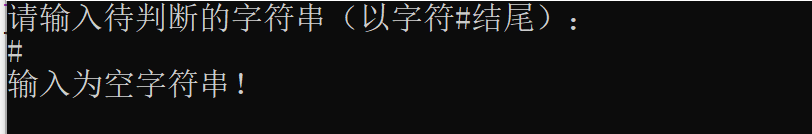
步骤：

用户打开程序，输入需要判断的字符串。

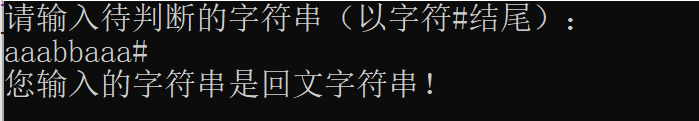
结果：

1. 如果输入的是空字符串则打印“输入为空字符串！”
2. 如果输入的是回文字符串 则打印“您输入的字符串是回文字符串！”
3. 如果输入的不是回文字符串 则打印“您输入的字符串不是回文字符串！”
4. **测试结果**

输入的字符串为空字符串：



输入的字符串为回文字符串时:



输入的字符串不是回文字符串的时候：

