**数据结构实验报告**

**实验题目：2/8进制转换器**

|  |  |  |  |
| --- | --- | --- | --- |
| 班级：2020211305 | 姓名：缪奇志 | 学号：2020211345 | 分工：需求分析，概要设计 |
| 班级：2020211305 | 姓名：倪玮昊 | 学号：2020211346 | 分工：详细设计，使用说明 |
| 班级：2020211305 | 姓名：杨智杰 | 学号：2020211356 | 分工：调试分析报告，测试用例，详细设计 |

1. **需求分析**

题目任务：

输入端输入一串用0/1表示的二进制数，以“#”结束，输出这个数的8进制表示形式。要求数据结构使用栈来实现。

程序的输入和输出：

输入：以0/1组成，最后以“#”结尾的字符串。

输入的数据类型：字符。

值的范围：字符串必须是0/1组成而且最后“#”结尾。

代表意义：表示一个数的二进制表示（以“#”表示结尾）。

程序功能实现：

1. 成功创建两个栈分别用来存放2进制数和8进制数并完成栈的初始化。
2. 检测用户输入是否规范。会对用户输入的数据进行判断，如果输入不规范则会结束程序并消除两个栈；如果输入合法，将数据存入二进制栈。
3. 在完成数据的输入之后，完成数据二进制转换为八进制，并存入对应八进制栈。
4. 转换完成后会对数据进行打印，之后销毁所创建的栈，释放内存。

正确数据输出结果范例：

文本

描述已自动生成

错误数据输出结果范例：

**文本

描述已自动生成**

1. **概要设计**

**算法思路概述：**

本程序主要依次实现了四个功能模块进而完成整个主程序的实现。

1. 模块一：成功创建两个对应的栈并对栈进行初始化。

思路：创建栈的结构体，其中有三个元素分别对应栈底、栈顶、栈的容量。创建时设置初始容量为100，并采取动态申请的方法实现。完成创建之后再让栈顶等于栈底，完成栈的初始化。

1. 模块二：在用户输入完成后程序判断输入的数据是否符合要求并将值存入二进制栈中。

思路：利用bool型来判断输入是否合格，如果错误，直接销毁栈结束程序。如果符合要求则让栈顶不断存入数据之后让栈顶不断加1，如果储存空间不够时每次申请50个单位容量直到无法再申请程序会给出提示。

1. 模块三：用出栈操作完成二进制对八进制的转换再存入对应的八进制栈。

思路：创建一个整型的变量count，作为位数计数器，利用一个循环只要栈顶不等于栈底程序就一直循环下去。同时命名要存入八进制的数据为e，如果count=0，e=e，count+1；count=1，e=e\*2，count+1；count=2，e=e\*4，count+1；count=3或栈中只有一个元素时，e=e，同时让count=0，把e输入进入八进制栈，八进制栈顶+1。

1. 模块四：完成数据的输出，释放内存并结束程序。

思路：设置一个条件为栈顶不等于栈底的循环，完成对数据的输出。之后再利用free释放内存最后结束程序。

**数据结构类型的定义：**

定义结构体myStack表示栈。

定义字符类型 StackElement表示栈内的元素。

分别定义字符指针StackElement\* base和 StackElement\* top表示栈底和栈顶；定义整型 size为栈的容量。

typedef char StackElement;

struct myStack

{

StackElement\* base;

StackElement\* top;

int size;

};

**主程序流程：**

stack2, stack8

InitStack(stack2)

InitStack(stack8)

flag <- Input(stack2)

if (flag == false)

DelStack(stack2)

DelStack(stack8)

return 0

B2O(stack2, stack8)

Print(stack8)

DelStack(stack2)

DelStack(stack8)

return 0

**各层次模块层次调用关系：**

创建好两个表示栈的数据结构后首先调用InitStack初始化栈，再调用Input判断输入是否合法，Input里调用Push函数实现入栈操作。完成输入之后主程序调用B2O，B2O再调用Pop和Push分别完成入栈和出栈的操作。最后主程序调用Print完成输出，调用DelStack释放栈所占的空间，程序结束。

1. **详细设计**

**各种函数的设计：**

1. 栈的初始化

s.base <- (StackElement\*)malloc(sizeof(StackElement) \* 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;

1. 入栈操作

if (s.top - s.base + 1 > s.size) // 判断栈满，如果满了重新申请更大的空间

s.base = (StackElement\*)realloc(s.base, sizeof(StackElement) \* (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++

1. 出栈操作

if (s.top == s.base)

printf("栈空，没有可取出的元素！\n");

return ERROR;

s.top--

return \*(s.top)

1. 栈的销毁

free(s.base)

s.base <- s.top <- = NULL

s.size <- 0

1. 判断输入的二进制串是否合法

printf("请输入待转换的二进制串（以字符#结尾）：\n");

StackElement e

scanf("%c", &e)

while (e != '#')

if (e == '0' || e == '1')

Push(s, e);

else

printf("非法输入，程序停止！\n")

return false

scanf("%c", &e)

return true

1. 二进制转换为八进制

int count <- 0;

while (s2.top != s2.base)

switch (count)

case 0: e <- Pop(s2) - '0'

break

case 1: e <- e + 2 \* (Pop(s2) - '0')

break

case 2: e <- e + 4 \* (Pop(s2) - '0')

break

count++

if (count == 3 || s2.top == s2.base)

e <- e + '0'

Push(s8, e)

count <- 0

return

1. 八进制栈的输出

if (s.top == s.base)

printf("输出为空串！\n")

return

while (s.top != s.base)

printf("%c", Pop(s))

return

**函数调用关系图：**

电脑萤幕

低可信度描述已自动生成

1. **调试分析报告**

**调试过程中遇到的问题与解决：**

1.最开始模拟静态的栈，发现对栈的容量可能出现过大或者过小的情况，前者导致空间浪费，后者极大限制可处理二进制串的长度。

解决：改为malloc加头尾指针的模拟动态栈。

2.最开始函数传参的时候没有传引用，导致所有函数操作都是对栈上开辟的形参进行的，主函数里传递的栈变量并没有被修改。

解决：通过传引用的方式解决了这一问题。

3.忽略了字符‘0’‘1’和整型01的区别，在二进制串和八进制串的转换时出现问题，导致输出结果错误。通过隐式转换进行运算将字符转换为数字。

解决：通过隐式转换进行运算将字符转换为数字。

**算法时空分析：**

InitStack:o(1)

Push:o(1)

Pop:o(1)

DelStack:o(1)

Input:o(n)

B2O:o(n)

Print:o(n)

main:o(n)

**改进设想：**

可以尝试再减小栈对空间的占用，实现真正的动态申请，需要多少就申请多少的存储空间，减小空间的占用。

**经验体会**：

通过malloc动态分配栈的容量使得栈的空间在满足条件的情况下开销尽可能小，比静态的模拟栈更有优势，体会到了动态分配空间的优势。通过自己模拟栈的各类操作，对栈这一数据结构有了更深入的认识。通过传参遇到的问题，加深了对函数形参和实参的概念理解，体会到了传引用的便利性。字符‘0’和整型0这种小细节上的错误，也警告我们需要打好基础，扎扎实实地学习基础知识，毕竟所有数据结构的实现都需要这些基础知识。

1. **用户使用说明**

用户使用程序时，输入规定的二进制代码串即可（由0/1组成同时最后务必以“#”结尾表示代码串的完结）。

1. **测试结果**

文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成