**《数据结构》**

**课程设计报告**

|  |  |
| --- | --- |
| 课程名称： | 《数据结构》课程设计 |
| 课程设计题目： | 魔王语言问题 |
| 姓 名： | 杨新莹 |
| 院 系： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 22052316 |
| 学 号： | 22050605 |
| 指导教师： | 张灵均 |

2023年10月22日

1. **需求分析**

#### 功能需求：

创建一个队列和栈,输入一串魔王语言，按解码方式：B->tAdA, A->sae，根据魔王语言与对应汉字关系进行输出打印。

#### 界面需求：

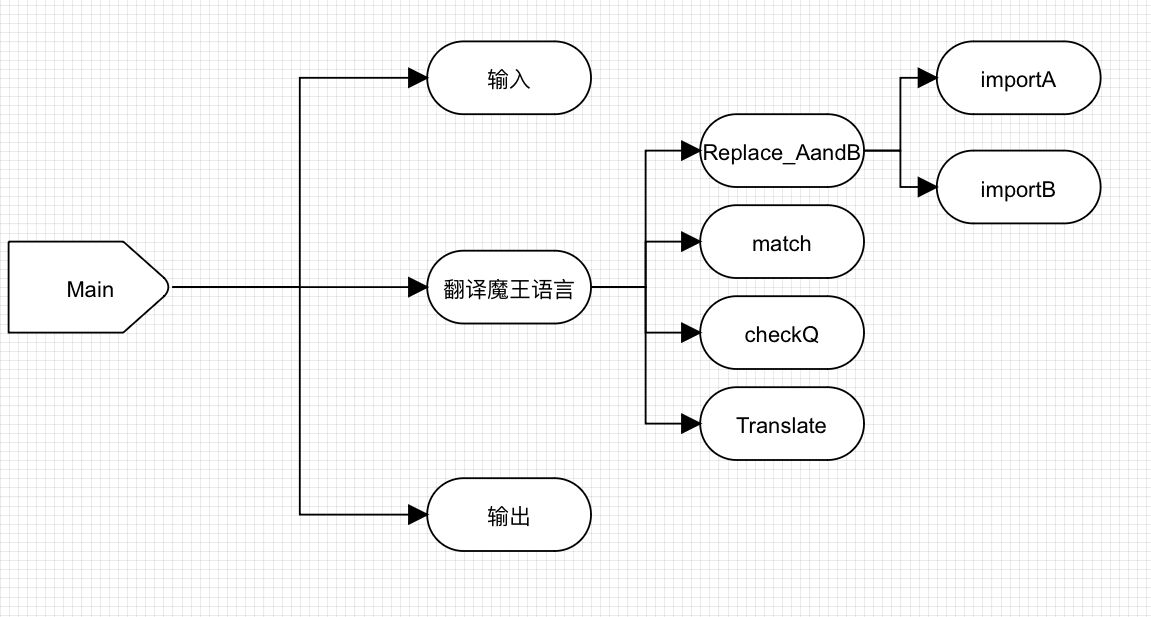
打印出魔王的语言和与之对应的人的语言。

#### 接口需求：

翻译魔王语言的函数、检查输入字符串中的括号是否匹配、检查翻译后的是否还有括号存在

1. **概要设计**

#### 模块层次结构设计



#### 接口设计

void importA(LinkQueue &Q) //翻译字符“A”

void importB(LinkQueue &Q) //翻译字符“B”

void Replace\_AandB(LinkQueue Q, LinkQueue &Q1 ) //检查翻译后的字符，将大写字母（即魔王语言）转化为人的语言

bool match(char\* s,bool &flag) //检查括号是否匹配

bool checkQ(LinkQueue Q) //检查是否所有括号都已经被解开

void Translate(SqStack &S, LinkQueue& Q) //翻译括号内的字符

**界面设计**

输出输入魔王语言 cout << "你所输入的魔王语言是：" << s << endl;

输出翻译后的语言 cout << "输出翻译后的结果为："; ShowQueue(Q1);

#### 数据结构设计

使用栈来存放魔王语言，并使用栈来翻译魔王语言，使用队列来存放翻译后的语言

SqStack S;//存放输入进来的原始字符串

LinkQueue Q,Q1;//存放翻译后的文字

SqStack S1;//倒放带翻译字符串

SqStack S2;//用于翻译过程，用作辅助栈

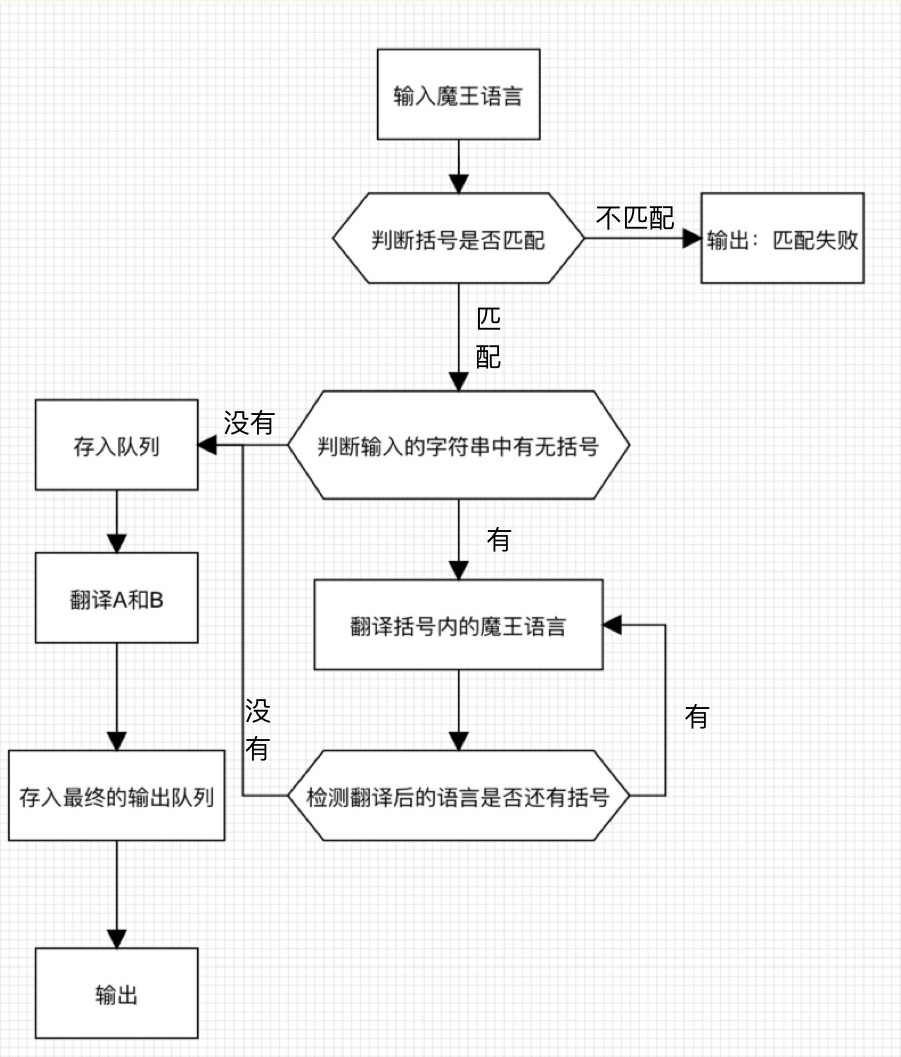
SqStack S3;//用于翻译过程，存放括号内的字符（包括括号）

SqStack S4;//用于翻译过程，存放翻译后的括号内的字符

1. **详细设计**

由于语言的数量并不确定，为避免溢出，采用带头结点的单链表实现队列的抽象数据类型。同时由于栈的操作总是在栈的顶端进行的，并不涉及数据元素的移动和删除，因此采用顺序表来存储栈。

这下面是程序的流程图：

“stack.h”

#ifndef \_STACK\_H\_

#define \_STACK\_H\_

//顺序栈结构

struct SqStack

{

char\* base;

int top;

int size;

};

//栈的初始化,分配m个结点的顺序空间

void InitSqStack(SqStack &s,int m)

{

s.top = 0;

s.base = new char[m];

s.size = m;

}

//判断栈是否为空

bool SqStackEmpty(SqStack s)

{

return s.top==0;

}

//求栈中的元素的个数

int SqStackLength(SqStack s)

{

return s.top;

}

//取栈顶的元素

bool GetTop(SqStack s, char& e)

{

if (!SqStackEmpty(s))

{

e = s.base[s.top - 1];

return true;

}

else

return false;

}

//入栈

void PushSqStack(SqStack& s, char e)

{

if (s.top >= s.size)

{

char\* newbase;

newbase = new char[s.size + 10];

for (int i = 0; i < s.top; i++)

newbase[i] = s.base[i];

delete[]s.base;

s.base = newbase;

s.size += 10;

}

s.base[s.top++] = e;

}

//出栈

bool PopSqStack(SqStack& s, char& e)

{

if (SqStackEmpty(s))

return false;

e = s.base[--s.top];

return true;

}

#endif

“queue.h”

#ifndef \_QUEUE\_H\_

#define \_QUEUE\_H\_

using namespace std;

//链队列的结构体

struct LinkNode {

char data;

LinkNode\* next;

};

//带头节点的链队列结构

struct LinkQueue {

LinkNode\* front;

LinkNode\* rear;

};

//构造一个空的链队列

void InitQueue(LinkQueue &Q)

{

Q.front = Q.rear = new LinkNode;

Q.front->next = nullptr;

}

//判断队列是否为空

bool QueueEmpty(LinkQueue Q)

{

return Q.front == Q.rear;

}

//返回链队列中的元素个数

int QueueLength(LinkQueue Q)

{

int i = 0;

LinkNode\* p=Q.front->next;

while (p != nullptr)

{

i++;

p = p->next;

}

return i;

}

//取链队列头元素，先决条件是队列不为空

char GetHead(LinkQueue &Q)

{

return Q.front->next->data;

}

//取链队列尾元素，先决条件是队列不为空

char GetTail(LinkQueue &Q)

{

return Q.rear->data;

}

//链队列入队

void EnQueue(LinkQueue& Q, char e)

{

LinkNode\* p;

p = new LinkNode;

p->data = e;

Q.rear->next = p;

p->next = nullptr;

Q.rear = p;

}

//链队列出队

bool DeQueue(LinkQueue& Q, char &e)

{

if (QueueEmpty(Q))

return false;

LinkNode\* p=Q.front->next;

Q.front->next = p->next;

e = p->data;

if (p == Q.rear)

Q.rear = Q.front;

delete p;

return true;

}

void ShowQueue(LinkQueue Q)

{

LinkNode\* p = Q.front->next;

while (p != nullptr)

{

cout<<p->data;

p = p->next;

}

}

#endif

“main.cpp”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include"stack.h"

#include"queue.h"

using namespace std;

//翻译字符“A”

void importA(LinkQueue &Q)

{

EnQueue(Q, 's');

EnQueue(Q, 'a');

EnQueue(Q, 'e');

}

//翻译字符“B”

void importB(LinkQueue &Q)

{

EnQueue(Q, 't');

importA(Q);

EnQueue(Q, 'd');

importA(Q);

}

//检查翻译后的字符，将大写字母（即魔王语言）转化为人的语言

void Replace\_AandB(LinkQueue Q, LinkQueue &Q1 )

{

char e;

InitQueue(Q1);

int len = QueueLength(Q);

for (int i = 0; i <len ; i++)

{

DeQueue(Q, e);

if (e == 'A')

{

importA(Q1);

}

else if (e == 'B')

{

importB(Q1);

}

else

{

EnQueue(Q1, e);

}

}

}

//检查括号是否匹配，并返回该字符串中是否有括号，用flag作为标志

bool match(char\* s,bool &flag)

{

SqStack S;

InitSqStack(S,100);

char r;

int i = 0;

while (s[i] != '\0')//遍历数组

{

if (S.top == 0)

{

if (s[i] == ')')//直接读入一个右括号，说明没有左括号，一定不匹配

{

return false;

}

else if (s[i] == '(')

{

flag = true;//说明该字符串中有括号

PushSqStack(S, s[i]);//如果读入的是左括号则直接入栈

}

}

else//栈不空的情况

{

if (s[i] == '(')

{

flag = true;//说明该字符串中有括号

PushSqStack(S, s[i]);

}

else if (s[i] == ')')

{

PopSqStack(S, r);

if (r != '(')

{

return false;

}

}

}

i++;

}

if (S.top!= 0)//如果遍历完后栈里还有剩余的符号，一定有没有配对的情况存在

{

return false;

}

return true;

}

//检查是否所有括号都已经被解开

bool checkQ(LinkQueue Q)

{

int len = QueueLength(Q);

char e;

LinkNode\* p;

p = Q.front->next;

while(p!=nullptr)

{

e = p->data;

if (e == '(' || e == ')') return false;//遍历队列

p = p->next;

}

return true;

}

//翻译括号内的字符

void Translate(SqStack &S, LinkQueue& Q)

{

SqStack S1;//倒装准备翻译的字符串

SqStack S2;

SqStack S3;

SqStack S4;

InitSqStack(S1, 100);

InitSqStack(S2, 100);

InitSqStack(S3, 100);

InitSqStack(S4, 100);

char r;

char e;

int i;

//将准备翻译的字符串倒装进S1

for (i = SqStackLength(S) - 1; i >= 0; i--)

{

PopSqStack(S, e);

PushSqStack(S1, e);

}

PopSqStack(S1, r);

while (r != ')')

{

PushSqStack(S3, r);

PopSqStack(S1, r);

}

PushSqStack(S2, r);//把右括号压入S2中

do

{

PopSqStack(S3, r);

PushSqStack(S2, r);

} while (r != '(');

//得到第一层内括号

PopSqStack(S2, r);//弹出左括号

PopSqStack(S2, e);//把第一个元素弹出并保存

PopSqStack(S2, r);

//翻译括号内的魔王语言，存入S4

while (r != ')')

{

PushSqStack(S4, e);

PushSqStack(S4, r);

PopSqStack(S2, r);

}

PushSqStack(S4, e);

//最后一次会弹出右括号，就不用在放入栈S4里了

while (!SqStackEmpty(S3))

{

PopSqStack(S3, r);

PushSqStack(S2, r);

}

//将各个栈的字符统统压入到队列中

InitQueue(Q);

while (!SqStackEmpty(S2))

{

PopSqStack(S2, r);

EnQueue(Q, r);

}

while (!SqStackEmpty(S4))

{

PopSqStack(S4, r);

EnQueue(Q, r);

}

while (!SqStackEmpty(S1))

{

PopSqStack(S1, r);

EnQueue(Q, r);

}

}

int main()

{

bool flag=false;//用于标志字符串中有无括号

char s[100];

int len;

SqStack S;

LinkQueue Q,Q1;

InitSqStack(S, 100);

InitQueue(Q);

InitQueue(Q1);

cout << "请输入魔王语言：";

cin >> s;

len = strlen(s);

cout << "你所输入的魔王语言是：" << s << endl;

if(match(s,flag))//括号匹配

{

printf("匹配成功\n");

if (flag == false)//没有括号，直接压入队列

{

for (int i = 0; i < len; i++)

{

EnQueue(Q, s[i]);

}

}

Else//有括号

{

for (int i = 0; i < len; i++)

{

PushSqStack(S, s[i]);

}

do //翻译括号

{

if (S.top == 0)//非第一次翻译，需要重新将字符压会栈S

{

while (!QueueEmpty(Q))

{

char e;

DeQueue(Q, e);

PushSqStack(S, e);

}

}

Translate(S, Q);//翻译

} while (!checkQ(Q));//检查括号是否已被完全翻译

}

Replace\_AandB(Q, Q1);//翻译A/B

cout << "输出翻译后的结果为：";

ShowQueue(Q1);//输出翻译后的人类语言

cout << endl;

}

else //括号不匹配

{

cout << "匹配失败，输入错误" << endl;

return 0;

}

system("pause");

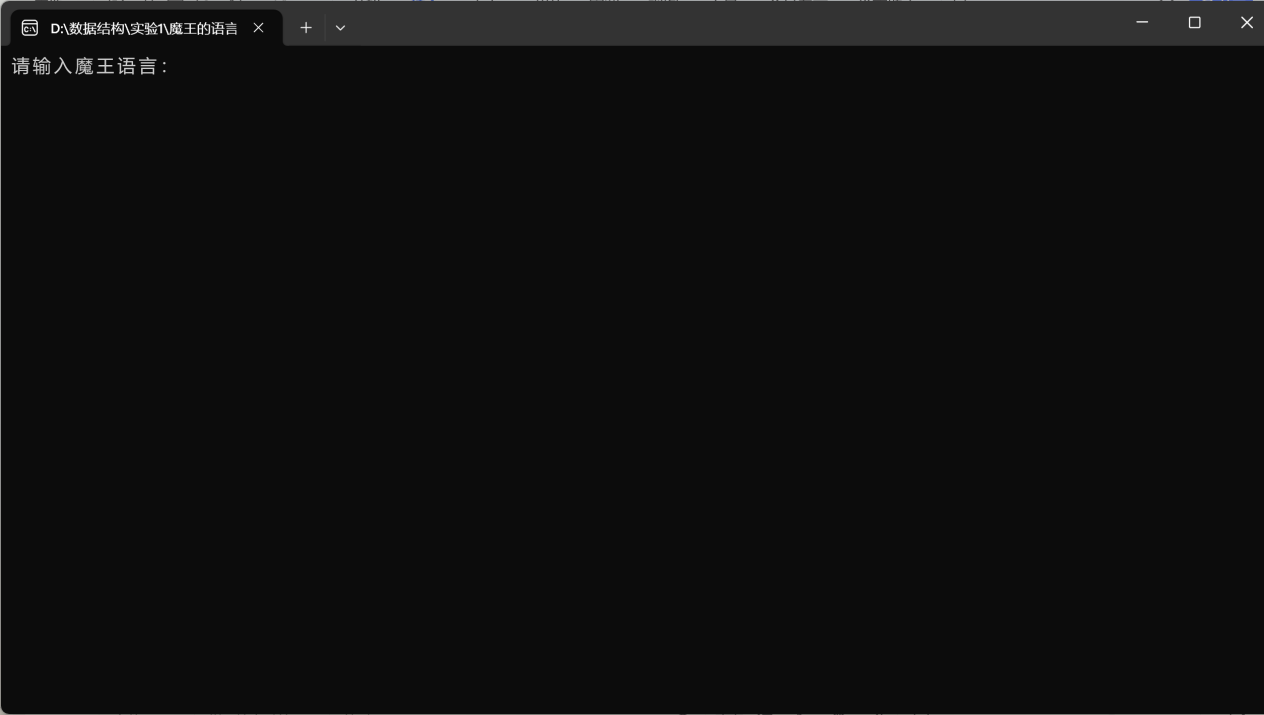
return 0;

}

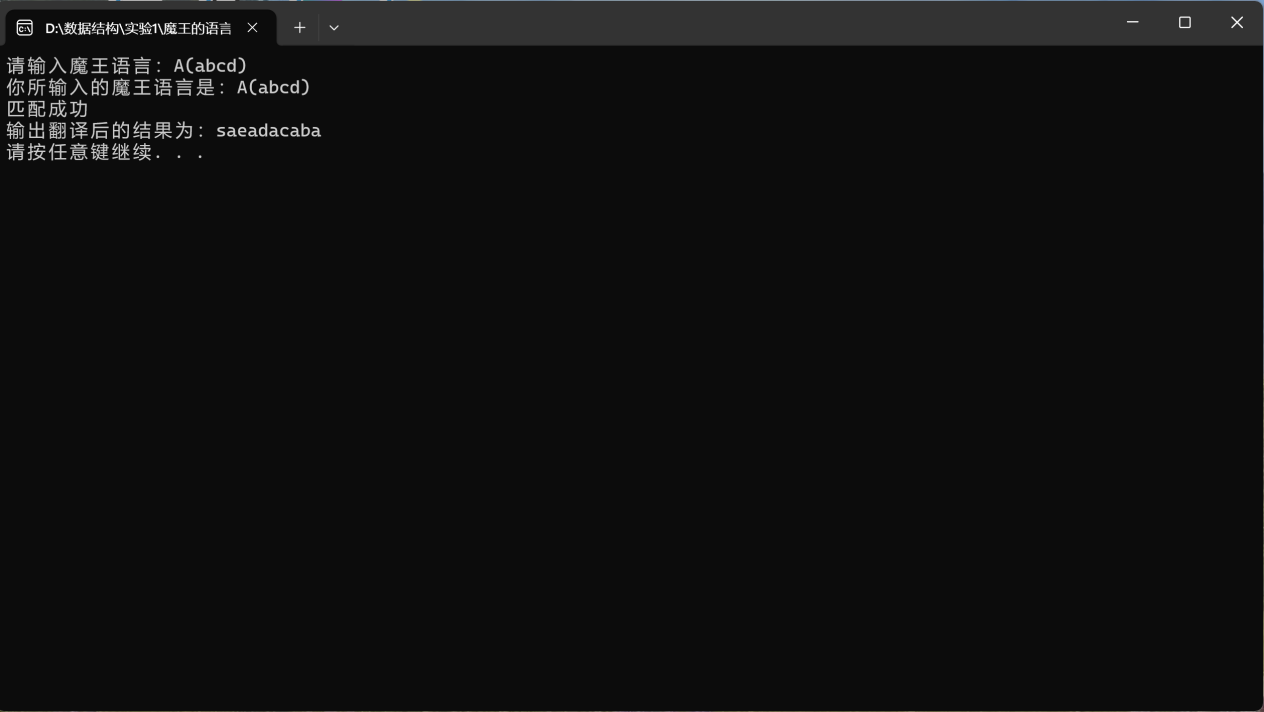
1. **调试分析**
2. 本次作业要求实现魔王语言的翻译，要求准确的翻译出魔王的语言
3. 本程序的一个难点在于输入的魔王的语言中括号的处理方式以及语言中嵌套多个括号或存在多个括号并列的情况。
4. 先将问题分解为翻译一个括号，代码使用了4个栈进行括号的处理，先将字符从尾到头读入栈S1，之后一个一个抛出至S3栈，直到遇到第一个括号，将其压入栈S2，再将S3弹出并以此压入S2，直到遇到第一个左括号，再将其压入S2，得到第一重括号内的字符，将栈S2内的栈顶左括号弹出，设置一个变量来保存它，再将其他元素和该变量按照翻译的要求依次压入到栈S4中。可以明显看出当有并列括号时上述过程也能实现
5. 当多层括号嵌套的时候，需要重复上述操作，为此增添一个函数，用于检查翻译后的字符串中是否还存在括号，若存在则重复上述操作
6. 同时在初步程序的时候还注意到当语言中没有括号时无法进行输出，调试分析修改代码后，增加了一个判定标志变量，对语言中是否存在括号进行了判定，使当语言中没有括号也能进行正常的翻译和输出
7. 同时，整个程序运行的界面中，也增加了提示性语言，方便用户使用。
8. **用户手册**

1、本程序的执行文件为：魔王的语言.exe

2、进入演示程序后，将显示如下的界面



1. 输入魔王语言后，按Enter键就可以看到翻译后的魔王语言



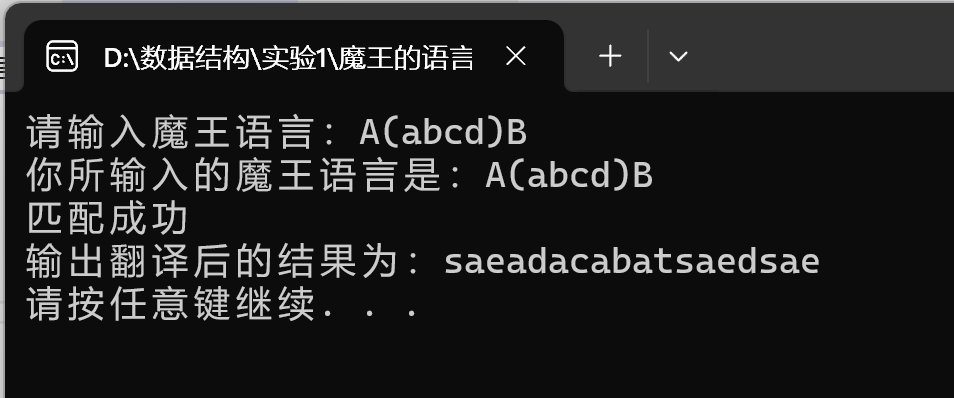
1. **测试结果**

由于翻译魔王语言是根据输入的字符串决定最后输出的内容，下面展示几种不同情况的翻译结果

1. 字符串中只有一个括号，即题目中的例子

输入：A(abcd)B

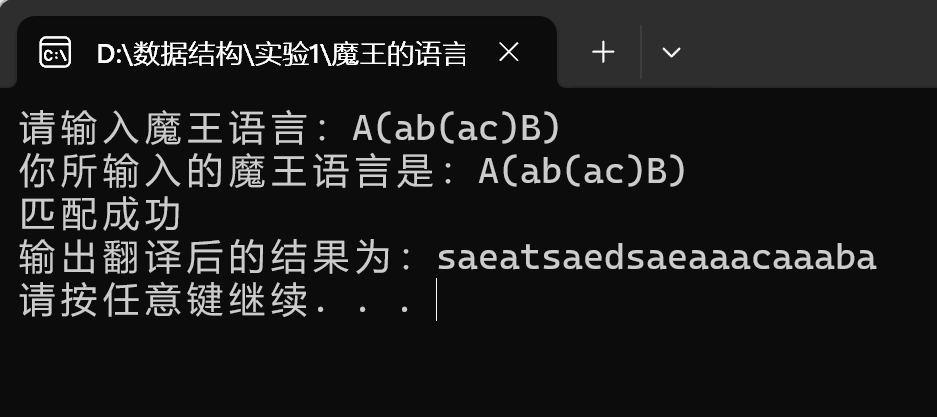
期望输出：saeadacabatsaedsae

结果：

1. 字符串中有嵌套的括号

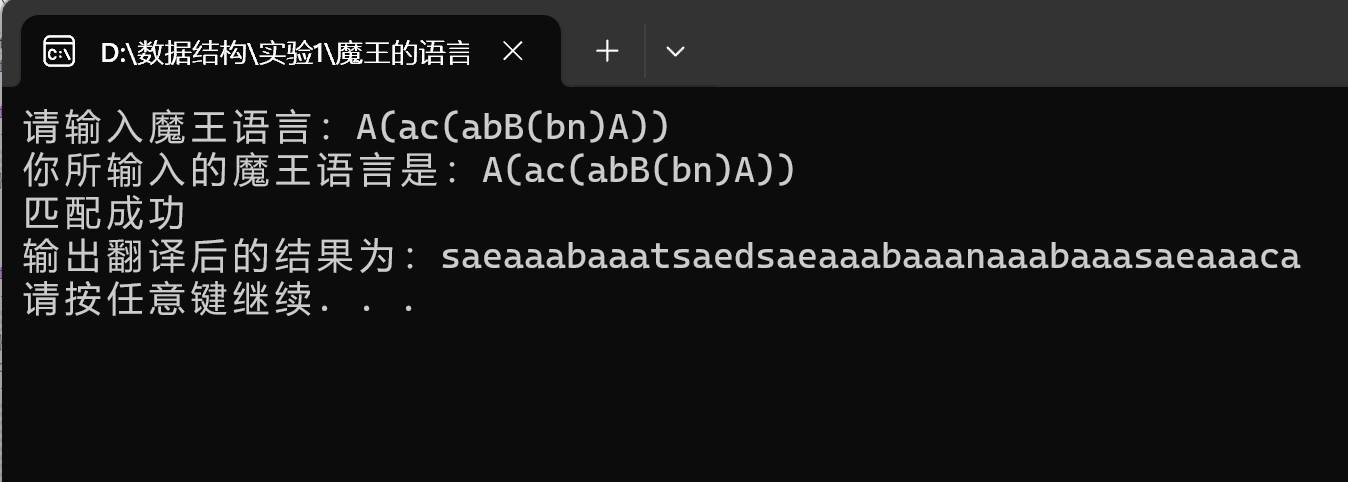
输入：A(ab(ac)B)

期望输出：saeatsaedsaeaaacaaaba

结果：

输入：A(ac(abB(bn)A))

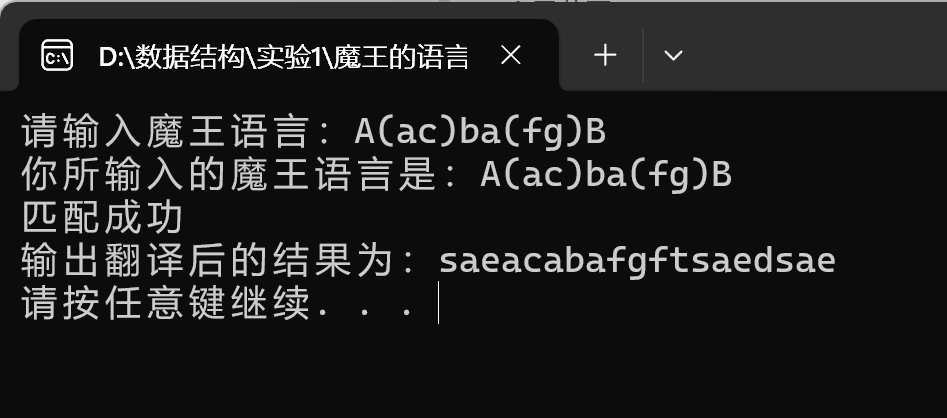
期望输出：saeaaabaaatsaedsaeaaabaaanaaabaaasaeaaaca

结果：

1. 字符串中有并列括号

输入：A(ac)ba(fg)B

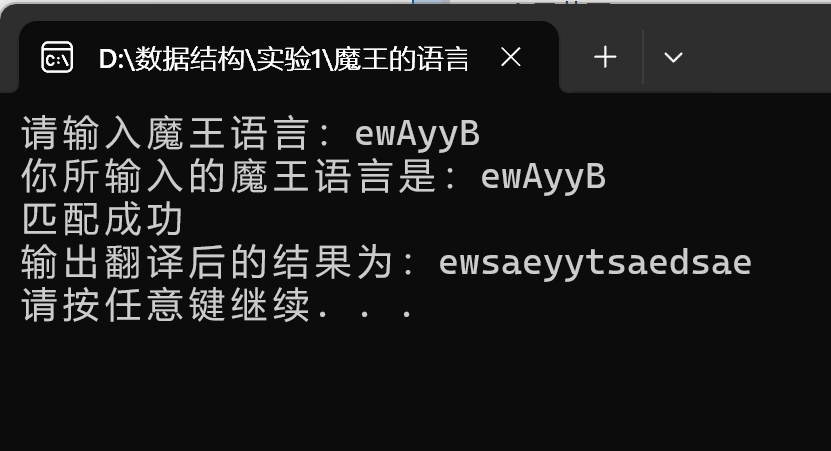
期望输出：saeacabafgftsaedsae

结果：

1. 字符串中没有括号

输入：ewAyyB

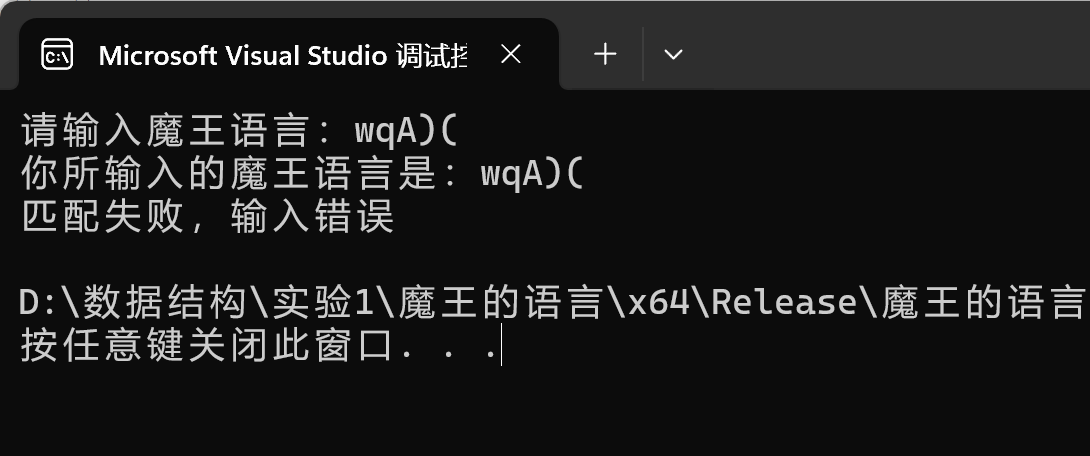
期望输出：ewsaeyytsaedsae

结果：

1. 字符串中括号不匹配

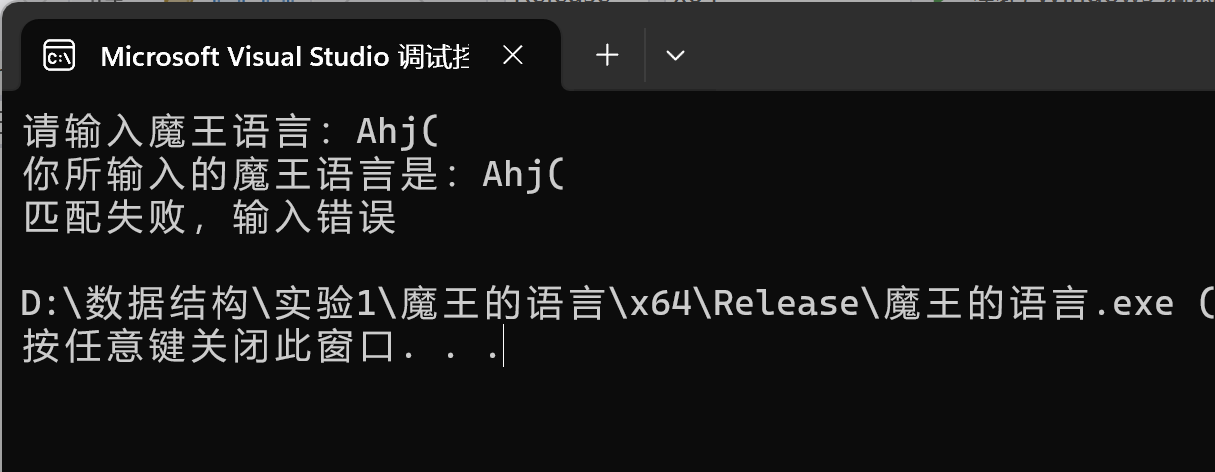
输入：wqA)(

期望输出：匹配失败，输入错误

结果：

输入：Ahj(

期望输出：匹配失败，输入错误

结果：

PS：详细的运行结果请打开魔王的语言.exe文件

1. **附录**

源程序文件名清单：main.cpp，魔王的语言.exe, Stack.h,Queue.h