**实 验 报 告**

**课程名称：** 数据结构C

**实验项目：**栈或队列的基本操作与应用

**实验仪器：** 计算机

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目** | **报告格式** | **写作质量** | **逻辑、注释，思想描述** | **复杂度分析** | **报告合计** |
| **百分比(%)** | **15** | **25** | **40** | **20** | **100** |
| 得分 |  |  |  |  |  |

**系 别： 计算机学院**

**专 业： 数据科学与大数据技术**

**班级姓名： 大数据1802 王恒烨**

**学 号： 2018010518**

**日 期：**

**成 绩：**

**同组成员：**

**指导教师：** 丁 濛

# 实验目的

1. 熟练掌握栈和队列的基本操作原则；

2. 掌握栈和队列的顺序存储和链式存储实现方式；

3. 实现一个循环队列；(验证)

4. 会用栈解决实际问题（设计、综合）。

# 实验内容

1. 实现一个链式存储的栈结构并分析每个功能的复杂度，要求具有栈的基本功能。  
Implement a linked-list based stack, analyze the complexity of each operation. Your stack structure should provide at least following basic operations: Pop(), Top(), Push(), Size().

2. 实现顺序存储的循环队列，并分析每个功能的复杂度，要求具有队列的基本功能。

Implement a array-based circulation queue, analyze the complexity of each operation. Your circulation queue structure should provide at least following basic operations: EnQueue(), DeQueue(), Size().

3. 利用栈，完成一个后缀四则表达式的计算，并返回结果。

Given a postfix expression, calculate its value using a stack.

4. 完成实验平台上实验二的其他题目

# 实验过程（应包括算法逻辑描述、复杂度分析、实现代码、实验结果等内容）

基本思想：

①初始化即将栈顶元素设为空，元素个数设为零 复杂度：时间θ（1），空间θ（1）

②清空数据，释放结点内存，实际上就是pop所有数据 复杂度：时间θ（n）， 空间θ（1）

③判断栈是否为空，通过栈中的元素个数。 复杂度：时间θ（1），空间θ（1）

④获取元素个数也是直接返回count的值。复杂度：时间θ（1），空间θ（1）

⑤获取顶部元素返回头指针所指的数据。复杂度：时间θ（1），空间θ（1）

⑥弹栈：栈顶指针指向要弹出元素前置结点，然后释放弹出元素内存空间，然后count-1.复杂度：时间θ（1），空间θ（n）

⑦压栈：先将压入元素放入到链表表中，然后再将栈顶指针指向压入的元素，然后count+1.复杂度：时间θ（1），空间θ（n）

⑧打印整个栈的内容时直接对栈中的元素进行遍历。复杂度：时间θ（n），空间θ（1）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#define ERROR 0

#define OK 1

#define TRUE 1

#define FALSE 0

typedef int Status;

typedef int EleType;

typedef struct StackNode {

EleType data;//结点数据域

struct StackNode\* next;//结点指针域

}StackNode,\* LinkStackPoi;

//链栈的数据结构

typedef struct LinkStack {

LinkStackPoi top;//栈顶结点

int count;//元素个数

}LinkStack;

//初始化

Status InitLinkStack(LinkStack\* stack)

{

if (!stack) //如果stack不存在，返回错误

{

return ERROR;

}

stack->top = NULL;//栈顶设空

stack->count = 0; //元素个数为零

return OK;

}

//清空数据，释放结点内存，实际上就是pop所有数据

Status ClearLinkStack(LinkStack\* stack)

{

if (!stack||!stack->count)//如果stack不存在或者stack中没有元素，返回错误

{

return ERROR;

}

while (stack->count)//当stack中的元素不为空时，执行以下操作

{

StackNode\* node = stack->top;//指针指向栈顶元素

stack->top = node->next;//栈顶指针指向新的栈顶元素

free(node);//释放节点内存

stack->count--;//元素总数减一

}

return OK;

}

//判断链栈是否为空

Status EmptyLinkStack(LinkStack\* stack) {

if (!stack)

{

return ERROR;

}

return stack->count == 0 ? 1 : 0;//如果为空返回1，不为空返回0

}

//获取元素个数

int GetLengthLinkStack(LinkStack\* stack)

{

if (!stack )

{

return -1;

}

return stack->count;//直接返回栈中的个数

}

//获取顶部元素

int GetTop(LinkStack\* stack)

{

if (!stack)

{

return ERROR;

}

return stack->top->data;//返回栈顶元素的数据

}

/\*

弹栈

栈顶指针指向要弹出元素前置结点，然后释放弹出元素内存空间，然后count-1

\*/

Status pop(LinkStack\* stack,EleType \*e)

{

if (!stack && stack->count)

{

return ERROR;

}

StackNode\* node = stack->top;

\*e = node->data;

stack->top = node->next;//栈顶指针指向新的栈顶元素

free(node);//释放元素空间

stack->count--;

return OK;

}

/\*

压栈

先将压入元素放入到链表表中，然后再将栈顶指针指向压入的元素，然后count+1.

\*/

Status push(LinkStack\* stack,EleType e)

{

if (!stack)

{

return ERROR;

}

StackNode\* node = (StackNode\*)malloc(sizeof(StackNode));

node->next = stack->top;//将元素加入链表中

node->data = e;

stack->top = node;//栈顶元素指向压入元素

stack->count++;

return OK;

}

//打印整个栈的内容

void PrintfLinkStack(LinkStack\* stack)

{

if (!stack&&stack->count)

{

return;

}

StackNode\* node = stack->top;

while (node)//用node对整个栈中的元素进行遍历

{

printf("%d,", node->data);

node = node->next;

}

puts("");

return;

}

int main(int argc, char \*argv[])

{

LinkStack stack;

int i;

InitLinkStack(&stack);//初始化

push(&stack, 1);

push(&stack, 2);

push(&stack, 3);

push(&stack, 4);

push(&stack, 5);

puts("链栈元素：");

PrintfLinkStack(&stack);

i = GetTop(&stack);

printf("%d\n",i);

printf("链栈元素个数：%d\n", GetLengthLinkStack(&stack));

EleType e1,e2;

pop(&stack, &e1);

printf("弹出第一个元素：%d\n", e1);

pop(&stack, &e2);

printf("弹出第二个元素：%d\n", e2);

puts("链栈元素：");

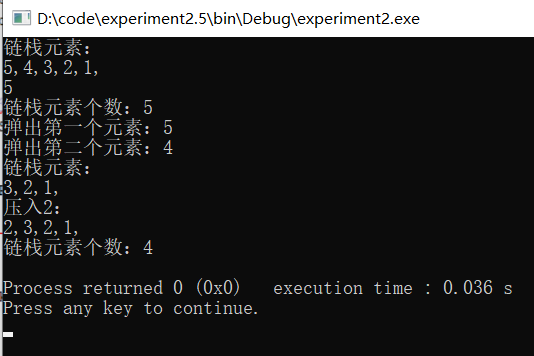
PrintfLinkStack(&stack);

push(&stack, 2);

printf("压入2：\n");

PrintfLinkStack(&stack);

printf("链栈元素个数：%d", GetLengthLinkStack(&stack));

 printf("\n");

return 0;

}

①构造空队列时将头指针和尾指针直接赋值为0. 复杂度：时间θ（1），空间θ（n）

②销毁队列时将头指针和尾指针赋值为0，并释放出空间。复杂度：时间θ（1），空间θ（1）

③置空队列时将头指针和尾指针赋值为0。复杂度：时间θ（1），空间θ（1）

④判断是否为空队列时，直接判断头指针和尾指针指的是否为同一元素。

复杂度：时间θ（1），空间θ（1）

⑤返回队列的长度，即返回Q的元素个数。复杂度：时间θ（1），空间θ（1）

⑥在队尾插入元素时，先判断队列有没有满。若不满，先插入元素，再将尾指针往后移一位。复杂度：时间θ（1），空间θ（1）

⑦删除头元素时，先判断队列是否为空。若不为空，则先将data赋值给e，再将头指针往后移一位即可将第一位元素删除。复杂度：时间θ（1），空间θ（1）

⑧遍历时用i来做遍历队列中的每一个数据。复杂度：时间θ（n），空间θ（n）

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int Status;

typedef char QElemType;

// -----循环队列――――队列的顺序存储结构-----

#define MAXQSIZE 100 // 最大队列长度

typedef struct { // 队列链式结构

QElemType \*base; // 初始化的动态分配存储空间

int front; // 头指针，若队列不空，指向队列头元素

int rear; // 尾指针，若队列不空，指向队列尾元素的下一个位置

} SqQueue;

// 构造一个空队列Q。

Status InitQueue(SqQueue &Q) {

Q.base = (QElemType\*)malloc(MAXQSIZE \* sizeof(QElemType));//分配内存

if(!Q.base) // 存储分配失败

exit(OVERFLOW); // exit(-2)程序异常退出

Q.front = Q.rear = 0;

return OK;

}

// 销毁队列Q，Q不再存在。

Status DestroyQueue(SqQueue &Q) {

Q.front = Q.rear = 0; //使头指针和尾指针全部归零

free(Q.base); //释放内存

return OK;

}

// 把Q置为空队列。

Status ClearQueue(SqQueue &Q) {

Q.front = Q.rear = 0;

return OK;

}

// 若Q为空队列，返回TRUE，否则返回FALSE

Status QueueEmpty(SqQueue Q) {

if(Q.front == Q.rear)

return TRUE; // 返回1

else

return FALSE; // 返回0

}

// 返回Q的元素个数，即队列的长度。

int QueueLength(SqQueue Q) {

return (Q.rear - Q.front + MAXQSIZE) % MAXQSIZE;

}

// 若Q为非空队列，则用e返回Q的队头元素。

Status GetHead(SqQueue Q, QElemType &e) {

if(Q.front == Q.rear)

return ERROR; // 空队列

e = Q.base[Q.front]; // 取队头元素

printf("获取的队头元素：%c\n", e);

return OK;

}

// 插入元素e为Q的新的队尾元素。

Status EnQueue(SqQueue &Q, QElemType e) {

// 少用一个元素空间，约定以“队列头指针在队列尾指针的下一位置上”作为队列满的标志。

if((Q.rear + 1) % MAXQSIZE == Q.front)

return ERROR; // 队列满

Q.base[Q.rear] = e; // 插入元素e

Q.rear = (Q.rear+1)%MAXQSIZE; // rear后移

printf("插入的队尾元素：%c\n", e);

return OK;

}

// 删除Q的队头元素，并用e返回其值。

Status DeQueue(SqQueue &Q, QElemType &e) {

if(Q.front == Q.rear)

return ERROR; // 空队列

e = Q.base[Q.front];

Q.front = (Q.front+1)%MAXQSIZE; // front后移

printf("删除的队头元素：%c\n", e);

return OK;

}

// 从队头到队尾，依次对Q的每个数据元素进行打印。

Status QueueTraverse(SqQueue Q) {

if(Q.front == Q.rear) {

printf("队列为空!\n");

return ERROR; // 空队列

}

int i = Q.front; // i指向队头元素下标

while(i != Q.rear) {

printf("%c ->",Q.base[i]);

i = (i+1) % MAXQSIZE;

}

printf("\n");

return OK;

}

int main() {

SqQueue Q;

// 构造空队列

if(InitQueue(Q)) {

// 插入元素

for(int i=0; i<26; i++)

EnQueue(Q, 'A' + i);

}

// 求队列长

printf("队列的大小：%d\n", QueueLength(Q));

// 返回队列的队头元素

QElemType e;

if(GetHead(Q, e))

QueueTraverse(Q);

// 删除队列的队头元素

if(DeQueue(Q, e))

QueueTraverse(Q);

// 判空

if(QueueEmpty(Q))

printf("队列为空。\n");

else

printf("队列非空。\n");

// 置空

ClearQueue(Q);

QueueTraverse(Q);

// 判空

if(QueueEmpty(Q))

printf("队列为空。\n");

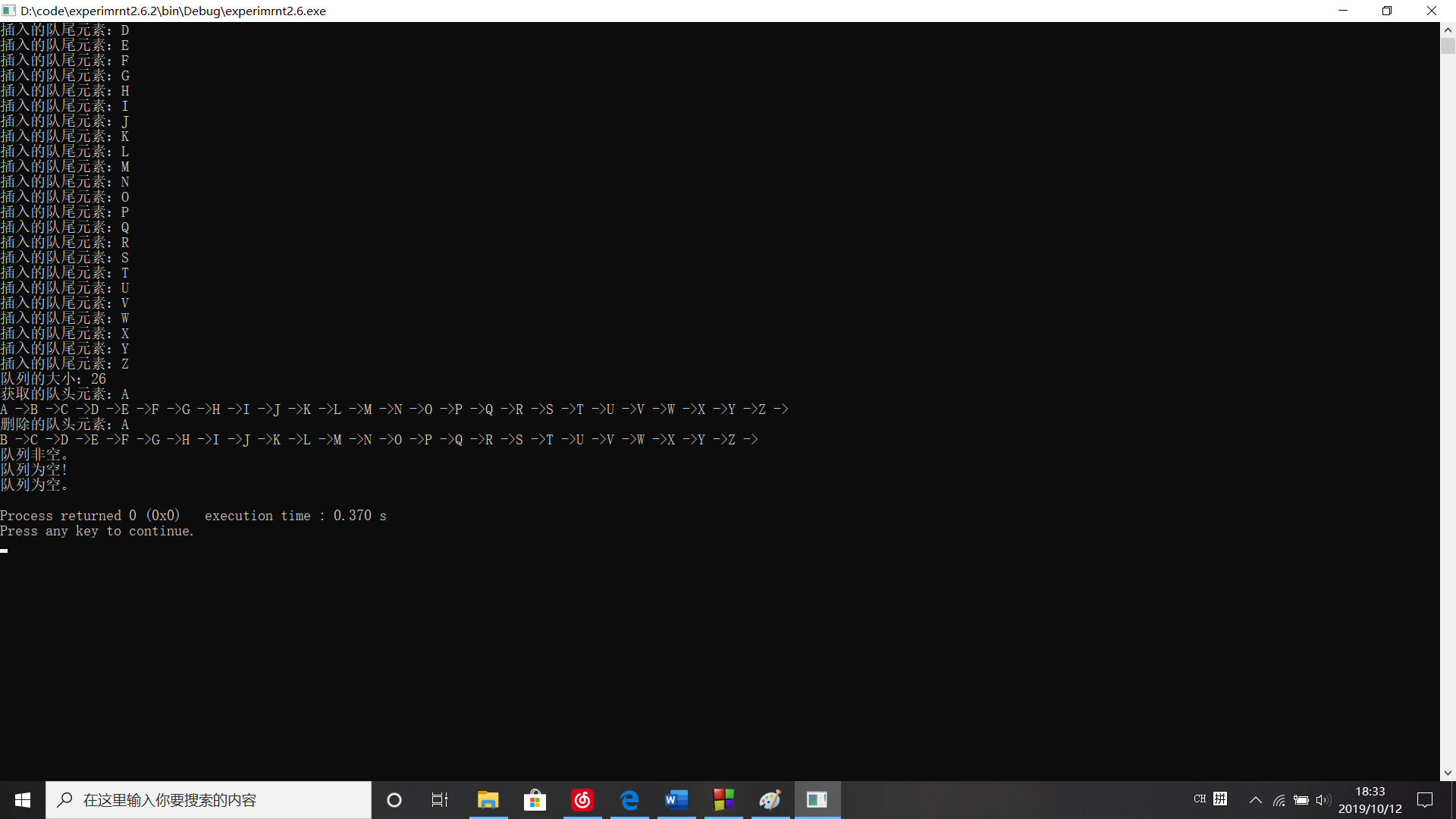
else

printf("队列非空。\n");

// 销毁

DestroyQueue(Q);

return 0;

}

①利用栈计算后缀表达式，栈存数字，遇到符号就出栈，进行运算后把结果重新进栈。复杂度：时间θ（n），空间θ（n）

#include <iostream>

#include <stack> //调用C++相对于C特有的stack库

#include <stdio.h>

using namespace std;

double cal(double a,double b,double c){//定义一个函数，传入参数为两个数字和一个符号。根据符号进行相关的加减乘除等操作

if(c=='+') return a+b;

else if(c=='-') return a-b;

else if(c=='\*') return a\*b;

else if(c=='/') return a/b;

else return 0;

}

double postcal(char\* post){//对栈进行操作

stack <double > s; //调用C++库里的函数stack

while(\*post){ //当post不为空时，执行以下循环

if(\*post >='0'&& \*post<='9'){//如果读取到的为数字，则进栈

s.push(\*post); //数字进栈

}

else{//如果为其他，执行以下操作

double a = s.top()-48;//读取栈顶元素，并将其赋值给a

s.pop(); //将栈顶元素剔除

double b = s.top()-48;//读取栈顶元素，并将其赋值给a

s.pop(); //将栈顶元素剔除

double c = cal(b,a,\*post);//进行运算并将结果赋值给c

s.push(c+48); //c的值进栈

}

post++; //post往后移一位，为下次循环做准备

}

return s.top()-48; //将栈顶元素作为结果返回

}

int main(int argc, const char \* argv[]) {

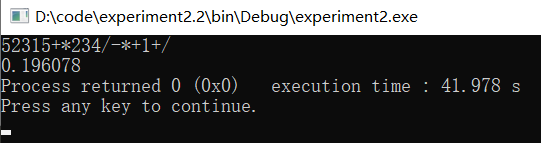
char\* post = new char;

cin>>post; //输入将要运算的后缀表达式

double res = postcal(post); //调用函数进行运算

printf("%lf", res); //将结果打印

return 0;

}

①利用栈计算后缀表达式，栈存数字，遇到符号就出栈，进行运算后把结果重新进栈。复杂度：时间θ（n），空间θ（n）

#include <iostream>

#include <stack> //调用C++相对于C特有的stack库

#include <stdio.h>

using namespace std;

double cal(double a,double b,double c){//定义一个函数，传入参数为两个数字和一个符号。根据符号进行相关的加减乘除等操作

if(c=='+') return a+b;

else if(c=='-') return a-b;

else if(c=='\*') return a\*b;

else if(c=='/') return a/b;

else return 0;

}

double postcal(char\* post){//对栈进行操作

stack <double > s; //调用C++库里的函数stack

while(\*post){ //当post不为空时，执行以下循环

if(\*post >='0'&& \*post<='9'){//如果读取到的为数字，则进栈

s.push(\*post); //数字进栈

}

else{//如果为其他，执行以下操作

double a = s.top()-48;//读取栈顶元素，并将其赋值给a

s.pop(); //将栈顶元素剔除

double b = s.top()-48;//读取栈顶元素，并将其赋值给a

s.pop(); //将栈顶元素剔除

double c = cal(b,a,\*post);//进行运算并将结果赋值给c

s.push(c+48); //c的值进栈

}

post++; //post往后移一位，为下次循环做准备

}

return s.top()-48; //将栈顶元素作为结果返回

}

int main(int argc, const char \* argv[]) {

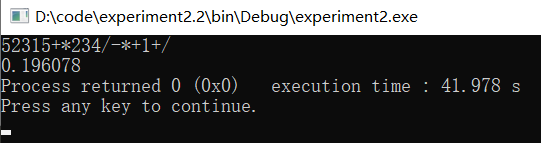
char\* post = new char;

cin>>post; //输入将要运算的后缀表达式

double res = postcal(post); //调用函数进行运算

printf("%lf", res); //将结果打印

return 0;

}

本题和计算后缀表达式的值稍有不同。设置两个栈，第一个栈中存储的是符号，第二个栈存储数字。并且还要注意符号优先级的问题。复杂度：时间θ（n），空间θ（n^2）

#include<iostream>

#include<cstring>

#include<algorithm>

#include<stack>

using namespace std;

stack<char> s1;//存储符号

stack<char> s2;//存储中间结果

int f(const char str){

int yxj; //设置优先级

switch(str){ //

case '\*':yxj=5;break;

case '/':yxj=5;break;

case '+':yxj=4;break;

case '-':yxj=4;break;

}

return yxj;

}

int main(){

char c[100];

cin>>c;

int lenc=strlen(c);//读取字符串长度

for(int i=0;i<lenc;i++){

if(c[i]>='0'&&c[i]<='9'){//如果是数字，直接压入s2

s2.push(c[i]);

}else if(c[i]=='+'||c[i]=='-'||c[i]=='\*'||c[i]=='/'){//如果是运算符

while(true){

if(s1.empty()||s1.top()=='('){//s1为空，或者栈顶为（

s1.push(c[i]);

break;

}else if(f(c[i])>f(s1.top())){//当前运算符的优先级大于s1栈顶元素的优先级

s1.push(c[i]);

break;

}

else{ //当前运算符的优先级小于或等于s1栈顶元素的优先级，先pop出栈顶元素，使当前运算符入栈，再将之前pop出的元素入栈，一直循环下去，直到栈中某个元素或者到了有（的地方或者到了栈底。

char cc=s1.top();

s1.pop();

s2.push(cc);

}

}

}else{

if(c[i]=='('){//如果是（直接读入

s1.push(c[i]);

}else{ //如果是），将栈顶元素取出并压入s2中，一直到）停止。

while(s1.top()!='('){

char ccc=s1.top();

s1.pop();

s2.push(ccc);

}

s1.pop();//将（pop出

}

}

}

while(!s1.empty()){//如果符号栈不空，将符号栈中的元素全部放到s2中

char cccc=s1.top();

s2.push(cccc);

s1.pop();

}

while(!s2.empty()){//顺序倒置

char c=s2.top();

s1.push(c);

s2.pop();

}

while(!s1.empty()){//打印s1中元素并清空s1

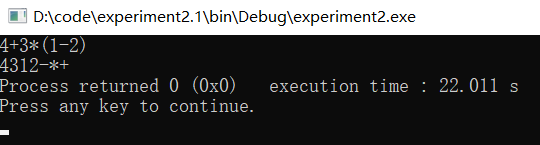
cout<<s1.top();

s1.pop();

}

return 0;

}



本题利用了栈的特点，如果读取到（则进栈，读取到）就出栈，如果读取到）栈为空或者读取到最后栈中还有元素说明不配对，否则就是配对。复杂度：时间θ（n），空间θ（n）

#include<iostream>

#include<cstring>

#include<algorithm>

#include<stack>

using namespace std;

int main()

{

stack<char> s1;//设置一个栈用来存储（

char c[1000];

int i = 0;

cin>>c;

int len = strlen(c);//读取字符串长度

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

{

if(c[i] == '(')//如果(则进栈

{

s1.push(c[i]);

}

else if(c[i] == ')')//如果为）

{

if(s1.empty())//如果栈空，说明不是配对的，直接输出NO并返回

{

cout<<"NO"<<endl;

return 0;

}

else//如果栈不空，就把其中一个”（“pop掉

{

s1.pop();

}

}

}

if(s1.empty())//如果执行完之后栈正好是空的，则输出YES

{

cout<<"YES"<<endl;

}

else//如果栈不空，输出NO

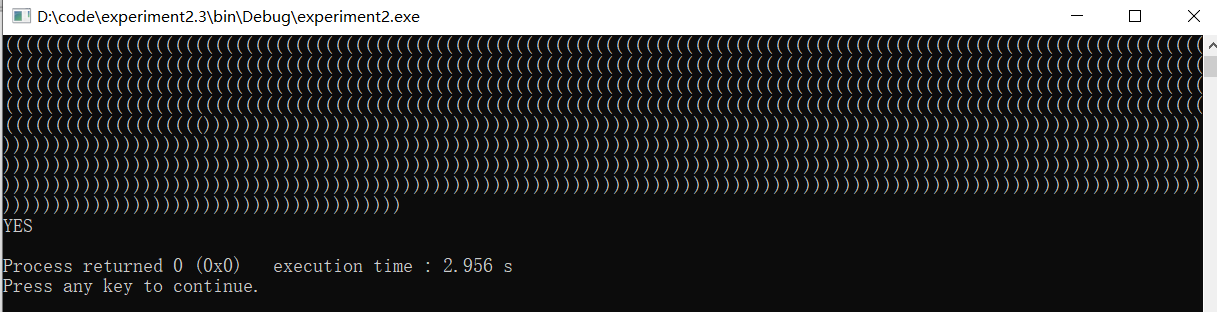
{

cout<<"NO"<<endl;

}

return 0;

}



本体采用取余法，设置一个数组用于存储余数。最后将结果（若为零则不包含）倒置。复杂度：时间O（n），空间θ（n）

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//整体采用的是取余法

int main()

{

int a,n,i;

scanf("%d %d", &a, &n);

int t[1000];

for(i = 0;; i++)

{

t[i] = a % n;//对n进制进行取余

a = ( a -(a % n) )/ n;//取商继续进行下一步操作

if(a < n)//如果a比n小，说明不足以进行下一步操作

{

if(a == 0)//如果说a是0，减一位，若不减则会打印一个0

{

i--;

break;

}

t[i+1] = a;//把最后的得数赋给t[i+1]

break;

}

}

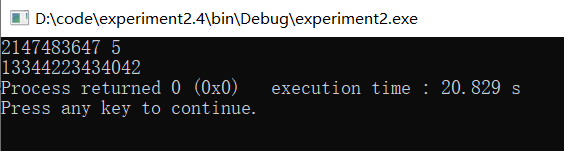
for(i = i + 1; i >= 0; i--)

{

printf("%d",t[i]);//打印结果

}

}



# 实验总结

本次实验我学会了如何定义栈和队列，如何使用栈和队列，以及关于后缀表达式和中缀表达式的一些转换和计算方法，还有栈的一些巧妙地用途。在今后的学习中，要加强对写代码的训练，同时还要锻炼自己的编程思维，利用业余时间多看一些算法，有助于自己的提高。