实验三 栈及其应用

实验周次：第9周 学时：2学时 地点：学院机房

**【实验目的】**

1.领会顺序栈存储结构和掌握顺序栈中各种基本运算算法设计。

2.领会链栈存储结构和掌握链栈中各种基本运算算法设计。

3.掌握栈在求解迷宫问题中的应用。

**【实验内容】**

实验题1.编写一个程序sqstack.cpp,实现顺序栈的各种基本运算（假设顺序栈的元素类型Elem Type为char），并在此基础上设计一个程序exp3-1.cpp完成以下功能。

1. 初始化栈s。
2. 判断栈s是否非空。
3. 依次进栈元素a,b,c,d,e
4. 判断栈s是否非空。
5. 输出出栈序列即依次出栈并显示。
6. 判断栈s是否非空。
7. 释放栈。

提示内容：不会请查看PPT中相关内容。

程序：

Sqstack.cpp

//顺序栈运算算法

#include <stdio.h>

#include <malloc.h>

#define MaxSize 100

typedef *char* ElemType;

typedef *struct*

{

    ElemType data[MaxSize];

*int* top;                //栈指针

} SqStack;                  //声明顺序栈类型

*void* InitStack(SqStack \*&*s*)   //初始化顺序栈

{

*s*=(SqStack \*)malloc(sizeof(SqStack));

*s*->top=-1;

}

*void* DestroyStack(SqStack \*&*s*) //销毁顺序栈

{

    free(*s*);

}

*bool* StackEmpty(SqStack \**s*)     //判断栈空否

{

    return(*s*->top==-1);

}

*bool* Push(SqStack \*&*s*,ElemType *e*)    //进栈

{

    if (*s*->top==MaxSize-1)    //栈满的情况，即栈上溢出

        return false;

*s*->top++;

*s*->data[*s*->top]=*e*;

    return true;

}

*bool* Pop(SqStack \*&*s*,ElemType &*e*)    //出栈

{

    if (*s*->top==-1)     //栈为空的情况，即栈下溢出

        return false;

*e*=*s*->data[*s*->top];

*s*->top--;

    return true;

}

*bool* GetTop(SqStack \**s*,ElemType &*e*)  //取栈顶元素

{

    if (*s*->top==-1)         //栈为空的情况，即栈下溢出

        return false;

*e*=*s*->data[*s*->top];

    return true;

}

Exp3-1.cpp

//文件名:exp3-1.cpp

#include "sqstack.cpp"      //包含顺序栈的基本运算算法

*int* main()

{

    ElemType e;

    SqStack \*s;

    printf("顺序栈s的基本运算如下:\n");

    printf("  (1)初始化栈s\n");

    InitStack(s);

    printf("  (2)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (3)依次进栈元素a,b,c,d,e\n");

    Push(s,'a');

    Push(s,'b');

    Push(s,'c');

    Push(s,'d');

    Push(s,'e');

    printf("  (4)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (5)出栈序列:");

    while (!StackEmpty(s))

    {

        Pop(s,e);

        printf("%c ",e);

    }

    printf("\n");

    printf("  (6)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (7)释放栈\n");

    DestroyStack(s);

    return 1;

}

实验题2.编写一个程序linkstack.cpp,实现链栈的各种基本运算（假设链栈中数据域元素类型Elem Type为char），并在此基础上设计一个程序exp3-2.cpp完成以下功能。

1. 初始化栈s。
2. 判断栈s是否非空。
3. 依次进栈元素a,b,c,d,e
4. 判断栈s是否非空。
5. 输出出栈序列即依次出栈并显示。
6. 判断栈s是否非空。
7. 释放栈。

提示内容：不会请查看PPT中相关内容。

程序：

//linkstack.cpp

//链栈运算算法

#include <stdio.h>

#include <malloc.h>

typedef *char* ElemType;

typedef *struct* linknode

{

    ElemType data;              //数据域

*struct* linknode \*next;      //指针域

} LinkStNode;                   //链栈类型定义

*void* InitStack(LinkStNode \*&*s*)  //初始化链栈

{

*s*=(LinkStNode \*)malloc(sizeof(LinkStNode));

*s*->next=NULL;

}

*void* DestroyStack(LinkStNode \*&*s*)   //销毁链栈

{

    LinkStNode \*p=*s*->next;

    while (p!=NULL)

    {

        free(*s*);

*s*=p;

        p=p->next;

    }

    free(*s*);    //s指向尾节点,释放其空间

}

*bool* StackEmpty(LinkStNode \**s*)  //判断栈空否

{

    return(*s*->next==NULL);

}

*void* Push(LinkStNode \*&*s*,ElemType *e*)    //进栈

{   LinkStNode \*p;

    p=(LinkStNode \*)malloc(sizeof(LinkStNode));

    p->data=*e*;              //新建元素e对应的节点p

    p->next=*s*->next;        //插入p节点作为开始节点

*s*->next=p;

}

*bool* Pop(LinkStNode \*&*s*,ElemType &*e*)    //出栈

{   LinkStNode \*p;

    if (*s*->next==NULL)      //栈空的情况

        return false;

    p=*s*->next;              //p指向开始节点

*e*=p->data;

*s*->next=p->next;        //删除p节点

    free(p);                //释放p节点

    return true;

}

*bool* GetTop(LinkStNode \**s*,ElemType &*e*)  //取栈顶元素

{   if (*s*->next==NULL)      //栈空的情况

        return false;

*e*=*s*->next->data;

    return true;

}

Exp3-2.cpp

//文件名:exp3-2.cpp

#include "listack.cpp"      //包含链栈的基本运算算法

*int* main()

{

    ElemType e;

    LinkStNode \*s;

    printf("链栈s的基本运算如下:\n");

    printf("  (1)初始化栈s\n");

    InitStack(s);

    printf("  (2)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (3)依次进栈元素a,b,c,d,e\n");

    Push(s,'a');

    Push(s,'b');

    Push(s,'c');

    Push(s,'d');

    Push(s,'e');

    printf("  (4)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (5)出栈序列:");

    while (!StackEmpty(s))

    {

        Pop(s,e);

        printf("%c ",e);

    }

    printf("\n");

    printf("  (6)栈为%s\n",(StackEmpty(s)?"空":"非空"));

    printf("  (7)释放栈\n");

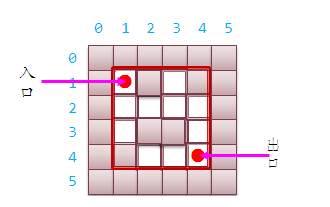
    DestroyStack(s);

    return 1;

}

实验题3. 迷宫路径求解。

例如，M=4，N=4，图中的每个方块，用空白表示通道，用阴影表示障碍物。为了算法方便，一般在迷宫外围加上了一条围墙。如下图所示。（根据PPT上讲的，做了调整）



请定义迷宫数组及相关的数据结构类型，并给出实现代码及调试结果。

//文件名:exp3-5.cpp

#include <stdio.h>

#define M 4                 //行数

#define N 4                 //列数

#define MaxSize 100         //栈最多元素个数

*int* mg[M+2][N+2]={          //一个迷宫,其四周要加上均为1的外框

{1,1,1,1,1,1},

{1,0,1,0,1,1},

{1,0,0,0,0,1},

{1,0,1,1,0,1},

{1,1,0,0,0,1},

{1,1,1,1,1,1}

};

*struct*

{

*int* i,j;

*int* di;

} St[MaxSize],Path[MaxSize];    //定义栈和存放最短路径的数组

*int* top=-1;                     //栈顶指针

*int* count=1;                    //路径数计数

*int* minlen=MaxSize;             //最短路径长度

*void* dispapath()                //输出一条路径并求最短路径

{

*int* k;

    printf("%5d: ",count++);    //输出第count条路径

    for (k=0;k<=top;k++)

        printf("(%d,%d) ",St[k].i,St[k].j);

    printf("\n");

    if (top+1<minlen)           //比较找最短路径

    {

        for (k=0;k<=top;k++)    //将最短路径存放在path中

            Path[k]=St[k];

        minlen=top+1;           //将最短路径长度存放在minlen中

    }

}

*void* dispminpath()              //输出第一条最短路径

{

    printf("最短路径如下:\n");

    printf("长度: %d\n",minlen);

    printf("路径: ");

    for (*int* k=0;k<minlen;k++)

        printf("(%d,%d) ",Path[k].i,Path[k].j);

    printf("\n");

}

*void* mgpath(*int* *xi*,*int* *yi*,*int* *xe*,*int* *ye*) //求迷宫路径

{

*int* i,j,i1,j1,di;

*bool* find;

    top++;                          //进栈

    St[top].i=*xi*;

    St[top].j=*yi*;

    St[top].di=-1;mg[*xi*][*yi*]=-1;    //初始方块进栈-1

    while (top>-1)                  //栈不空时循环

    {

        i=St[top].i;j=St[top].j;di=St[top].di;

        if (i==*xe* && j==*ye*)         //找到了出口

        {

            dispapath();            //输出一条路径

            mg[i][j]=0;             //让出口变为其他路径可走方块

            top--;                  //出口退栈,即回退一个方块

            i=St[top].i;j=St[top].j;

            di=St[top].di;          //让栈顶方块变为当前方块

        }

        find=false;                 //找下一个可走方块(i,1j1)

        while (di<4 && !find)

        {   di++;

            switch(di)

            {

            case 0:i1=i-1; j1=j;   break;

            case 1:i1=i;   j1=j+1; break;

            case 2:i1=i+1; j1=j;   break;

            case 3:i1=i,   j1=j-1; break;

            }

            if (mg[i1][j1]==0) find=true;

        }

        if (find)                   //找到了下一个可走方块(i1,j1)

        {   St[top].di=di;          //修改原栈顶元素的di值

            top++;St[top].i=i1;St[top].j=j1;

            St[top].di=-1;          //下一个可走方块(i1,j1)进栈

            mg[i1][j1]=-1;          //避免重复走到该方块-1

        }

        else                        //没有路径可走,则退栈(i,j)方块

        {

        //  mg[i][j]=0;             //让该位置变为其他路径可走方块-1

            top--;

        }

    }

    dispminpath();                  //输出最短路径

}

*int* main()

{

*int* i,j;

    printf("迷宫所有路径如下:\n");

    mgpath(1,1,M,N);

    for(i=0;i<M+2;i++)

    {

        for(j=0;j<N+2;j++)

            printf("%3d ",mg[i][j]);

            printf("\n");

    }

    return 1;

}

实验4 . 表达式的求解。

求解表达式(2+3)\*6+8\*5+4

要求：给出原表达式对应的字符数组。以及后缀表达式及其对应的字符数组。

给出表达式求解算法源代码含：1）表达式转换为后缀表达式实现算法源码。2）后缀表达式求解算法源码。并给出调试结果。

#include <iostream>

#include <stack>

using *namespace* std;

*void* trans(*char* \**exp*, *char* *postexp*[])

{

*char* e;

    stack<*char*> Optr;    //定义运算符栈（顺序栈）指针

*int* i = 0;           //i作为postexp的下标

    while (\**exp* != '\0') //exp表达式未扫描完时循环

    {

        switch (\**exp*) //根据元素类型,执行相应操作

        {

        case '(':

            Optr.push('('); //判定为左括号,入运算符栈,继续扫描其它字符

*exp*++;          //继续扫描其他字符

            break;

        case ')': //判定为右括号,则运算符出栈依次放入postexp字符数组中

            e = Optr.top();

            Optr.pop();

            while (e != '(') //不为'('时循环,直到碰的'('结束,左括弧不存入

            {

*postexp*[i++] = e; //将（运算符）e存放到postexp中,

                e = Optr.top();

                Optr.pop();

            }

*exp*++; //继续扫描其他字符

            break;

        case '+': //为加或减号时,栈里的运算符存入postexp,直到栈空或者

        case '-':

            while (!Optr.empty()) //栈空时退出循环

            {

                e = Optr.top();

                if (e != '(') //e不是'('

                {

*postexp*[i++] = e; //将(运算符）e存放到postexp中

                    e = Optr.top();

                    Optr.pop();

                }

                else

                    break; //碰到左括弧时,退出循环

            }

            Optr.push(\**exp*);

*exp*++; //继续扫描其他字符

            break;

        case '\*': //判定为'\*'或'/'号,栈里\*和/运算符全部存入postexp

        case '/':

            while (!Optr.empty()) //栈不空循环

            {

                e = Optr.top();

                if (e == '\*' || e == '/') //e为'\*'或'/'运算符

                {

*postexp*[i++] = e; //将('\*'或'/'运算符)e存放到postexp中

                    e = Optr.top();

                    Optr.pop();

                }

                else //e为非'\*'或'/'运算符时退出循环

                    break;

            }

            Optr.push(\**exp*);

*exp*++; //继续扫描其他字符

            break;

        default:                               //处理数字字符

            while (\**exp* >= '0' && \**exp* <= '9') //转存数字字符,

            {

*postexp*[i++] = \**exp*; //直接将数字字符存储字符数组postexp中

*exp*++;

            }                   //直到出现非数字字符时结束,此时再增加一个#

*postexp*[i++] = '#'; //增加#或者非数字字符,标识一个数值串结束

        }

    }

    while (!Optr.empty()) //此时exp扫描完毕,栈元素全部放入postexp

    {

        e = Optr.top();

        Optr.pop();

*postexp*[i++] = e; //将e存放到postexp中

    }

*postexp*[i] = '\0'; //给postexp表达式添加字符数组元素结束标识

}

//计算后缀表达式(已存储在postexp数组中）的值。

*double* compvalue(*char* \**postexp*)

{

*double* d, a, b, c, e;

    stack<*char*> Opnd;        //定义操作数栈

    while (\**postexp* != '\0') //postexp字符串未扫描完时循环

    {

        switch (\**postexp*)

        {

        case '+': //判定为'+'号

            a = Opnd.top();

            Opnd.pop();

            b = Opnd.top();

            Opnd.pop();

            c = b + a; //计算c

            Opnd.push(c);

            break;

        case '-': //判定为'-'号

            a = Opnd.top();

            Opnd.pop();

            b = Opnd.top();

            Opnd.pop();

            c = b - a;    //计算c

            Opnd.push(c); //将计算结果c进操作数栈Opnd

            break;

        case '\*': //判定为'\*'号

            a = Opnd.top();

            Opnd.pop();

            b = Opnd.top();

            Opnd.pop();

            c = b \* a;    //计算c

            Opnd.push(c); //将计算结果c进操作数栈Opnd

            break;

        case '/': //判定为'/'号

            a = Opnd.top();

            Opnd.pop();

            b = Opnd.top();

            Opnd.pop();

            if (a != 0)

            {

                c = b / a;    //计算c

                Opnd.push(c); //将计算结果c进操作数栈Opnd

                break;

            }

            else

            {

                printf("\n\t除零错误!\n");

                exit(0); //异常退出

            }

            break;

        default:                                       //处理数字字符'0'-'9'

            d = 0;                                     //转换成对应的数值存放到d中

            while ((\**postexp* >= '0') && (\**postexp* <= '9')) //碰到非数字字符时结束

            {

                d = 10 \* d + \**postexp* - '0'; //连续数字串转换为数值,参与运算

*postexp*++;                   //非数字字符'#'正好被废弃了-妙-

            }

            Opnd.push(d); //将连续数字字符串转换后的数值d进操作数栈Opnd

            break;        //等待参与运算

        }

*postexp*++; //继续处理其他字符

    }

    e = Opnd.top(); //取操作数栈Opnd栈顶元素（存放运算结果）,赋值给e返回

    return e;       //返回e （结果）

}

//建立如下主函数调用上述算法实现算术表达式的求解

*int* main()

{

*char* exp[] = "(2+3)\*6+8\*5+4"; //可将exp改为键盘输入

    //char exp[50];

    //gets(str);//输入表达式字符串

    //puts(str);//输出表达式字符串

*char* postexp[100];

    trans(exp, postexp);

    cout<<"中缀表达式:"<<exp<<endl;

    cout<<"后缀表达式:"<<postexp<<endl;

    cout<<"表达式的值:"<<compvalue(postexp)<<endl;

    return 0;

}