

# 《数据结构》上机报告

2018 年 10 月 28 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：\_\_\_\_\_

|        |   |            |
|--------|---|------------|
| 实验题目   | 栈   |            |
| 问题描述   | 栈是限制仅在表的一端插入和删除的线性表。栈的操作简单，重点掌握栈具有后进先出（LIFO）的特性。顺序栈是栈的顺序存储结构的实现。链栈是栈的链式存储结构的实现。   |            |
| 基本要求   | 1. (p1)实现栈的基本操作，本题练习顺序栈的基本操作，包括入栈、出栈、判栈空、判栈满、取栈顶元素、栈的遍历。(p2)实现对有序表（非递减）插入元素、删除元素、查找元素的功能<br>2. (p2)实现两个指定数制之间的转换<br>3. (p3)使用栈实现括号匹配的检验功能<br>4. (p4)使用栈完成中缀表达式的计算<br>5. (p5)使用栈完成火车进站问题 |            |
|        | 已完成基本内容（序号）：  | 1, 2, 3, 4 |
| 选做要求   |   |            |
|        | 已完成选做内容（序号）   | 5          |
| 数据结构设计 | <pre> typedef struct {     ElemType *base;     ElemType *top;     int    stacksize; } SqStack;                     </pre>   |            |

|                                      |   |
|--------------------------------------|---|
| 功<br>能<br>(<br>函<br>数<br>)<br>说<br>明 | <p>1、 栈的初始化</p> <p>分配 MAXSIZE 的空间。将栈的头指针和尾指针同时指向初始分配的空间，将栈的大小定义为初始分配的 MAXSIZE 大小。</p> <pre> Status InitStack(SqStack &amp;S) {     S.base = (ElemType*)malloc(MAXSIZE * sizeof(ElemType));     if (!S.base) {         exit(OVERFLOWED);     }     S.top = S.base;     S.stacksize = MAXSIZE;     return OK; } </pre> <p>2、 进栈操作</p> <p>如果栈顶栈底相距的大小大于等于栈的最大空间，则应再为栈分配 MAXSIZE 的空间。更新栈的大小以及栈顶的位置。</p> <p>将当前要压入栈的值赋值给栈顶指针指向的空间，将栈顶指针向上移动。</p> <pre> Status Push(SqStack &amp;S, ElemType &amp;e) {     if (S.top - S.base &gt;= S.stacksize) {         S.base = (ElemType *)realloc(S.base, (S.stacksize + MAXSIZE) * sizeof(ElemType));         if (!S.base) return false;         S.top = S.base + S.stacksize;         S.stacksize += MAXSIZE;     }     *S.top = e;     S.top++;     return OK; } </pre> <p>3、 出栈操作</p> <p>如果栈顶指针和栈底指针指向同一位置，则证明该时刻栈已空，不能进行出栈操作。否则获取当前栈顶元素的值，将栈顶的指针向下移动。</p> <pre> Status Pop(SqStack &amp;S) {     ElemType e;     if (S.top == S.base) {         return ERROR;     }     else {         e = *S.top;         S.top--;         return e;     } } </pre> |
|--------------------------------------|---|

```
}
```

```
}
```

#### 4、栈的判空

如果栈顶指针和栈底指针指向同一位置，则证明该时刻栈已空。

```
Status StackEmpty(SqStack S)
```

```
{
```

```
    if (S.top == S.base) {
```

```
        return TRUE;
```

```
    }
```

```
    else {
```

```
        return FALSE;
```

```
    }
```

```
}
```

#### 5、获取栈顶元素

如果栈顶指针和栈底指针指向同一位置，则证明该时刻栈已空，不能进行获取操作。否则获取当前栈顶元素的值。

```
Status GetTop(SqStack S)
```

```
{
```

```
    ElemType e;
```

```
    if (S.top == S.base)
```

```
        return FALSE;
```

```
    else {
```

```
        e = *(S.top - 1);
```

```
    }
```

```
    return e;
```

```
}
```

#### 6、读取栈内所有元素的值

如果栈顶指针和栈底指针指向同一位置，则证明该时刻栈已空，不能进行读取操作，输出栈已空。否则获取当前栈顶元素的值，打印该值，将栈顶指针下移直到栈底。

```
Status ReadAll(SqStack S)
```

```
{
```

```
    if (S.base == NULL) {
```

```
        return ERROR;
```

```
    }
```

```
    if (S.top == S.base) {
```

```
        cout << "Stack is Empty" << endl;
```

```
        return ERROR;
```

```
    }
```

```
    ElemType *p;
```

```
    p = S.top;
```

```

while (p > S.base){
    p--;
    cout << *p << " ";
}
cout << endl;
return OK;
}

```

## 7、数制转换

使用栈实现数制转换的功能，并且可在两个指定的数制之间进行转换。首先将输入的数使用各位数乘以  $n$  次方的方法全部转化为 10 进制数，然后使用除  $n$  取余法，将余数压进栈内，最后输出栈得到结果。

可能出现的问题在于，由于 16 进制中会出现字符型数据，所以如果采用全字符型的输入，可能导致输入的两位以上（包括两位）的数字以单个各位数的情况出现。本文采用了一个 bool 值进行判断，如果是持续输入数字就不断地乘 10 相加。也可以采用 atoi 函数进行整体转换。

```

Status Trans(SqStack &S, int be, int af, char *str)
{
    int i, len, temp=0;
    i = 0;
    len = getStrLength(str)-1;
    while (str[i]) {
        int value = 0;
        if (str[i] >= '0' && str[i] < '9') {
            value = str[i] - 48;
        }
        if (str[i] >= 'A' && str[i] <= 'F') {
            value = str[i] - 65 + 10;
        }
        if (str[i] >= 'a' && str[i] <= 'f') {
            value = str[i] - 97 + 10;
        }
        temp+= int(value*pow(be, len));
        len--;
        i++;
    }
    while (temp) {
        int value = 0;
        value = temp % af;
        char ch;
        if (value>=0 && value<=9) {
            ch = value + 48;
        }
        if (value >= 10 && value <= 15) {

```

```

        ch = value + 65-10;
    }
    Push(S, ch);
    temp /= af;
}
return OK;
}

```

## 8、括号匹配

使用栈完成括号匹配功能。本文一开始采用的是一次性输入所有字符，然后将所有字符统一入栈，但本题只要求找到第一个不完全匹配的括号。所以一次性入栈会造成多个不必要的元素进入到栈内。

进而，本文采用一次输入一个元素，立刻进行检验的办法。如果输入的是左括号，直接入栈；如果输入的是右括号：首先检验栈是否为空，如果为空则已找到需要匹配的右括号；如果不为空，检验当前栈顶的括号是否与它匹配；若匹配，则将栈顶元素弹出；否则，将该值暂存，成为可能没有匹配的括号。如此往复。

最终，如果栈不空，则说明仍有左括号没有找到与之匹配的右括号，栈顶的左括号即为答案；否则，暂存的右括号就是答案。如果没有暂存的右括号，则说明括号完全匹配。

该题无需考虑是否是第一个不匹配的括号，如果要求的话，仍需考虑左括号的进栈顺序。

```

void Match()
{
    char ch,res,e;
    bool find = false;
    SqStack S;
    InitStack(S);
    while ((ch = getchar()) != EOF) {
        if (ch == '(' || ch == '{' || ch == '[') {
            Push(S, ch);
            continue;
        }
        if (ch == ')') {
            if (StackEmpty(S)) {
                e = GetTop(S);
                if (e == '(') {
                    Pop(S);
                    continue;
                }
            }
            else {
                if (!find) {
                    res = e;
                    find = true;
                    continue;
                }
            }
        }
    }
}

```

```

        }
    }
}
else {
    if (!find) {
        res = ch;
        find = true;
        continue;
    }
}
}
if (ch == ']') {
    if (StackEmpty(S)) {
        e = GetTop(S);
        if (e == '[') {
            Pop(S);
            continue;
        }
        else {
            if (!find) {
                res = e;
                find = true;
                continue;
            }
        }
    }
    else {
        if (!find) {
            res = ch;
            find = true;
            continue;
        }
    }
}
if (ch == '}') {
    if (StackEmpty(S)) {
        e = GetTop(S);
        if (e == '{') {
            Pop(S);
            continue;
        }
        else {
            if (!find) {
                res = e;

```

```

        find = true;
        continue;
    }
}
}
else {
    if (!find) {
        res = ch;
        find = true;
        continue;
    }
}
}
}

if (StackEmpty(S)) {
    res = GetTop(S);
    cout << "no" << endl;
    cout << res << "期待右括号" << endl;
}
else {
    if (find) {
        cout << "no" << endl;
        cout << res << "期待左括号" << endl;
    }
    else {
        cout << "yes" << endl;
    }
}
return;
}

```

## 9、中缀表达式计算

本文采用将中缀表达式转化为后缀表达式进行运算。利用栈进行转换。在这个过程中需要一个栈用来保存操作符，需要一个数组用来保存后缀表达式，然后从头到尾扫描表达式如果遇到操作符，则跟符号栈的栈顶操作符比较优先级，如果大于栈顶操作符的优先级，则入栈，否则不断取栈顶操作符加到后缀表达式的末尾，直到栈顶操作符优先级低于该操作符，然后将该操作符入栈；遇到操作数，直接加到后缀表达式的末尾；遇到左括号，入栈；遇到右括号，则依次弹出栈顶操作符加到后缀表达式的末尾，直到遇到左括号，然后将左括号出栈。由于后缀表达式本身不需要括号来限制哪个运算该先进行，因此可以直接利用栈来模拟计算：遇到操作数直接压栈，碰到操作符直接取栈顶的 2 个操作数进行计算（注意第一次取出的是右操作数），然后再把计算结果压栈，如此循环下去。最后栈中剩下的唯一一个元素便是整个表达式的值

```

Status Trans(char str[], int arr2[], int arr1[], SqStack &S)
{
    int len = strlen(str);
    int i, t, tx, len1 = 0;
    bool flag = true;
    for (i = 0; i < len - 1; i++) {
        tx = str[i];
        if (tx >= '0' && tx <= '9')
            if (i > 0 && str[i - 1] >= '0' && str[i - 1] <= '9') {
                arr2[len1 - 1] = arr2[len1 - 1] * 10 + str[i] - '0';
            }
            else {
                arr2[len1++] = str[i] - '0';
            }
        else if (tx != '+' && tx != '-' && tx != '*' && tx != '/' && tx != '(' && tx != ')') {
            printf("ERROR");
            flag = false;
            break;
        }
        else {
            if (Count(S) == 0 || tx == '(') Push(S, tx, 1000000);
            else if (tx == ')') {
                t = 0;
                while (t != '(') {
                    Pop(S, t);
                    if (t != '(') {
                        arr1[len1] = 1;
                        arr2[len1++] = t;
                    }
                }
            }
            else {
                t = 0;
                while (1) {
                    GetTop(S, t);
                    if (t == '(' || compare(tx, t) || Count(S) == 0) {
                        Push(S, tx, 1000000);
                        break;
                    }
                }
                else {
                    Pop(S, t);
                    arr1[len1] = 1;
                    arr2[len1++] = t;
                }
            }
        }
    }
}

```



```

    }
    }
}

}

for (i = 0; i <= Count(S); i++) {
    Pop(S, t);
    arr1[len1] = 1;
    arr2[len1++] = t;
}

int sum = 0, x, y;
for (i = 0; i < len1; i++) {
    if (flag == false) break;
    if (arr1[i] == 0) {
        Push(S, arr2[i], 100000);
    }
    else {
        Pop(S, y);
        Pop(S, x);
        switch (arr2[i]) {
            case '+':
                sum = x + y;
                Push(S, sum, 100000);
                break;
            case '-':
                sum = x - y;
                Push(S, sum, 100000);
                break;
            case '*':
                sum = x * y;
                Push(S, sum, 100000);
                break;
            case '/':
                if (y == 0) {
                    cout << "ERROR" << endl;
                    flag = false;
                }
                else {
                    sum = x / y;
                    Push(S, sum, 100000);
                }
                break;
        }
    }
}

```

```

        default: break;
    }

}

}

Pop(S, sum);
if (flag == true) {
    cout << sum << endl;
}
}

```

#### 10、 列车进站问题

列车进站问题的检验实际上就是对每一个输入的序列进行一次模拟。如果该序列最后的所有输入都可以出栈，则该序列为有效序列。否则不能出栈。

首先判断输入的字符串长度是否和总的列车数相等，如果不相等则不能有效输出。其次，使用两个字符数组都从0处开始计算。如果输出数组的字符等于输入位置的字符，则正确，弹出该字符。否则将该字符压入持续到直到找到该字符。

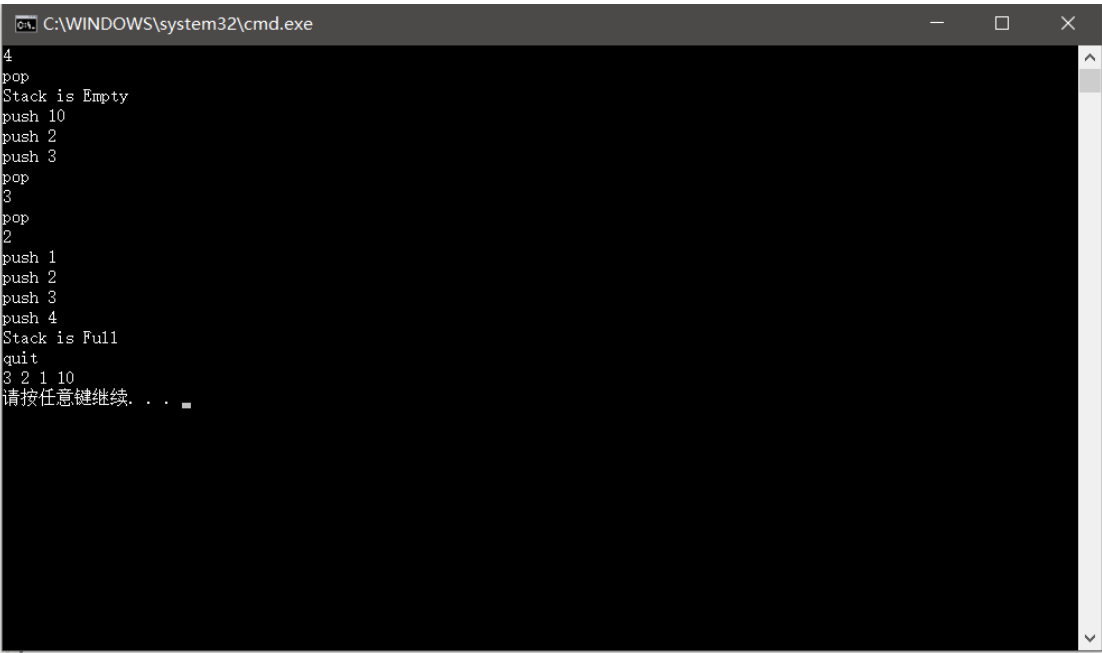
```

void Station(char *instr, char *outstr)
{
    if (getStrLength(outstr) != getStrLength(instr)) {
        cout << "no" << endl;
        return;
    }

    int i, j=0;
    ElemType e=0;
    char input[MAXSIZE] = { 0 }, out[MAXSIZE] = { 0 };
    SqStack S;
    InitStack(S);

    for (i = 0; i<getStrLength(outstr); i++) {
        if (!StackEmpty(S)) {
            e = GetTop(S);
            if (e == outstr[i]) {
                Pop(S);
                continue;
            }
        }
        while (instr[j] != outstr[i]) {
            if (j >= getStrLength(instr)) {
                break;
            }
            Push(S, instr[j]);
            j++;
        }
    }
}

```

|      |   |
|------|---|
|      | <pre>    }<br/>    j++;<br/>}<br/>if (i == j &amp;&amp; StackEmpty(S)){<br/>    cout &lt;&lt; "yes" &lt;&lt; endl;<br/>}<br/>else{<br/>    cout &lt;&lt; "no" &lt;&lt; endl;<br/>}<br/>while (!StackEmpty(S)){<br/>    Pop(S);<br/>}<br/>return ;<br/>}</pre> |
| 开发环境 | Visual studio 2017  |
| 调试分析 |    |

```
C:\WINDOWS\system32\cmd.exe
16 10
aD14
44308
请按任意键继续. . .
```

---

```
C:\WINDOWS\system32\cmd.exe
int main(){
for(int i=0;i<8;i++){
a[i]=(2+3/1;
}
}
no
(期待右括号)
请按任意键继续. . .
```

|      |   |
|------|---|
|      |    |
| 心得体会 | <p>1、 栈的思想是很多算法的基本原理</p> <p>除了题目中给出的表达式求值，括号匹配，列车进站等问题之外。栈还可以灵活的运用于树的运算，递归算法的核心思想也是与栈相关。所以深入理解栈的内涵，后进先出的思想在计算机中十分有用。</p> <p>2、 计算中缀表达式有两个方法，一个是将中缀表达式转化为后缀表达式。因为后缀表达式不需要考虑括号的问题，可以直接进行运算。本文采用的也是这种方法，也可以直接计算中缀表达式，使用判断优先级的算法去判断各个符号之间的优先级，使用两个栈，数据栈和符号栈进行计算。以下附上算法：</p> <pre> int Piroty(char ch) {     int e=0;     switch (ch) { </pre> |

```

        case '+': e = 1; break;
        case '-': e = 1; break;
        case '*': e = 2; break;
        case '/': e = 2; break;
        case '(': e = 0; break;
        case ')': e = 0; break;
    }

    return e;
}

int cal_Single(int m, int n, char ch)
{

    int e=0;
    if (ch == '/') {
        if (n == 0) {
            cout << "ERROR" << endl;
            return ERROR;
        }
    }

    switch (ch) {
        case '+': e = m+n; break;
        case '-': e = m - n; break;
        case '*': e = m*n; break;
        case '/': e = m/n; break;
    }

    return e;
}

int calculate(NuStack &S1, SqStack &S2)
{
    string str;
    char ch;
    int i = 0;
    cin >> str;
    bool sign = false, num = false;
    int neg = 0;
    int j, e;
    int size = str.size();
    char tmp_operation;
    string tmp_num;
    while (i < size) {
        if (str[i] == '=') {

```

```

        break;
    }
    if (str[i] >= '0' && str[i] <= '9') {
        num = true;
        sign = false;
        j = i;
        while (j < size && str[j] >= '0' && str[j] <= '9') { j++; }
        tmp_num = str.substr(i, j - i);
        int num = atoi(tmp_num.c_str());
        if (neg) {
            while (neg) {
                num *= -1;
                neg--;
            }
        }
        Push(S1, num);
        i = j;
    }
    else if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/') {
        if (sign) {
            if ((str[i] == '*' || str[i] == '/')) {
                cout << "ERROR" << endl;
                return ERROR;
            }
            if (str[i] == '-') {
                neg ++;
                i++;
                continue;
            }
        }
        sign = true;
        num = false;
        if (StackEmpty(S2)) {
            Push(S2, str[i]);
        }
        else {
            while (!StackEmpty(S2)) {
                tmp_operation = GetTop(S2);
                if (Priority(tmp_operation) >= Priority(str[i])) {
                    int m, n;
                    n = GetTop(S1);
                    Pop(S1);
                    m = GetTop(S1);
                    Pop(S1);

```

```

        ch = GetTop(S2);
        e=cal_Single(m, n, ch);
        if (e == ERROR) {
            return ERROR;
        }
        Push(S1, e);
        Pop(S2);
    }
    else {
        break;
    }
}
Push(S2, str[i]);
}
i++;
}
else {
    if (str[i] == '(') {
        Push(S2, str[i]);
    }
    else if(str[i] == ')'){
        int count = 0;
        if (count == 0) {
            if (GetTop(S2) == '(') {
                cout << "ERROR" << endl;
                return ERROR;
            }
            count++;
        }
        while (GetTop(S2) != '(') {
            tmp_operation = GetTop(S2);
            int m, n;
            n = GetTop(S1);
            Pop(S1);
            m = GetTop(S1);
            Pop(S1);
            ch = GetTop(S2);
            e = cal_Single(m, n, ch);
            if (e == ERROR) {
                return ERROR;
            }
            Push(S1, e);
            Pop(S2);
        }
    }
}

```



```

        Pop(S2);
    }
    else {
        cout << "ERROR" << endl;
        return ERROR;
    }
    i++;
}

}

while (!StackEmpty(S2)) {
    tmp_operation = GetTop(S2);
    int m, n;
    n = GetTop(S1);
    Pop(S1);
    m = GetTop(S1);
    Pop(S1);
    ch = GetTop(S2);
    e = cal_Single(m, n, ch);
    if (e == ERROR) {
        return ERROR;
    }
    Push(S1, e);
    Pop(S2);
}
return GetTop(S1);
}

```