天津大学

数据结构实验报告

实验名称：栈和队列

学院名称 智能与计算学部

专　　业 软件工程

学生姓名 陈昊昆

学　　号 3021001196

年　　级 2021 级

班　　级 软工 3 班

时　　间 2023 年 5 月 18 日

1. 实验内容

1. 实现无表头节点的链栈类 MyStack
   - 栈初始化，MyStack();//创建栈
   - 销毁栈,~MyStack();//清除栈内元素,销毁栈
   - 入栈，bool push(const int&);//入栈,成功返回 true,失败返回 false
   - 出栈，bool pop(int& item); //出栈,成功返回 true,失败返回 false,出栈元素放在 item
   - 返回栈顶元素 bool getTop(int& item);//获取栈顶,成功返回 true,失败返回 false,栈顶元素放在 item
   - 输出栈内元素,void printStack();//按照栈顶到栈底的顺序输出栈内所有元素

2. 实现顺序循环队列类 MyQueue
   - 队列初始化，MyQueue(const int& capacity);//创建队列,容量为 capacity
   - 销毁队列,~ MyQueue();//销毁队列
   - 入队,bool enQueue(const int&);//入队,成功返回 true,失败返回 false
   - 出队,bool deQueue(int& item); //出栈,成功返回 true,失败返回 false，出队元素放在 item
   - 返回队头元素,bool getFront(int& item); //获取队首,成功返回 true,失败返回 false,栈顶元素放在 item
   - 输出队列内元素，void printQueue();//按照队头到队尾的顺序输出栈内所有元素

3. 基于 MyStack 进行扩展改进(如改变栈内元素类型、对符号进行合理数值映射)，实现中缀表达式转后缀表达式和表达式求值，支持小括号、加减乘除四则运算,一位正整数(0-9)运算,不涉及多位数、负数及小数

4. 基于 MyQueue，实现输出杨辉三角
   - void yanghui(const int&);//输出杨辉三角

2. 程序实现

```
#include "stack_queue.h"
#include <bits/stdc++.h>
using namespace std;
```

```
listNode::listNode(){

}

// 栈初始化，MyStack();//创建栈
MyStack::MyStack(){
    stackSize = 0;
    topNode = NULL;
}

// 销毁栈,~MyStack();//清除栈内元素,销毁栈
MyStack::~MyStack(){
    stackSize = 0;
    listNode* p = topNode;

    while(p != NULL){
        listNode* q = p;
        delete q;
        p = p->next;

    }
}

// 入栈，bool push(const int&);//入栈,成功返回 true,失败返回 false
bool MyStack::push(const int& n){
    listNode* p = new listNode;
    if(p == NULL){
        return false;
    }
    stackSize++;

    p->data = n;
    p->next = topNode;
    topNode = p;

    return true;
}

// 出栈，bool pop(int& item); //出栈,成功返回 true,失败返回 false,出栈元素放
在 item
bool MyStack::pop(int& item){
    if (topNode == NULL){
        return false;
    }
```

```
    item = topNode->data;
    listNode* p = topNode;
    topNode = topNode->next;
    delete p;
    return true;
}

// 返回栈顶元素 bool getTop(int& item);//获取栈顶,成功返回 true,失败返回
false,栈顶元素放在 item
bool MyStack::getTop(int& item){
    if (topNode == NULL){
        return false;
    }

    item = topNode->data;
    return true;
}

void MyStack::printStack(){
    listNode* p = topNode;

    while(p != NULL){
        if(p->next == NULL)
            cout << p->data << endl;
        else
            cout << p->data << ",";

        p = p->next;

    }
}

//////////////////////////////////////////////////////////////////////

listNode1::listNode1(){

}

// 栈初始化, MyStack();//创建栈
MyStack1::MyStack1(){
    stackSize = 0;
    topNode = NULL;
}
```

```
// 销毁栈,~MyStack();//清除栈内元素,销毁栈
MyStack1::~MyStack1(){
    stackSize = 0;
    listNode1* p = topNode;

    while(p != NULL){
        listNode1* q = p;
        delete q;
        p = p->next;

    }
}

bool MyStack1::isempty(){
    if(stackSize == 0) return true;
    else return false;
}

// 入栈, bool push(const int&);//入栈,成功返回 true,失败返回 false
bool MyStack1::push(const char& n){
    listNode1* p = new listNode1;
    if(p == NULL){
        return false;
    }
    stackSize++;

    p->data = n;
    p->next = topNode;
    topNode = p;

    return true;
}

// 出栈, bool pop(int& item); //出栈,成功返回 true,失败返回 false,出栈元素放
在 item
bool MyStack1::pop(char& item){
    if (topNode == NULL){
        return false;
    }

    item = topNode->data;
    listNode1* p = topNode;
    topNode = topNode->next;
    delete p;
    return true;
```

```
}

// 返回栈顶元素 bool getTop(int& item);//获取栈顶,成功返回 true,失败返回
false,栈顶元素放在 item
bool MyStack1::getTop(char& item){
    if (topNode == NULL){
        return false;
    }

    item = topNode->data;
    return true;
}

void MyStack1::printStack(){
    listNode1* p = topNode;

    while(p != NULL){
        if(p->next == NULL)
            cout << p->data << endl;
        else
            cout << p->data << ",";

        p = p->next;

    }
}

////////////////////////////////////////////////////////////////////

// 队列初始化, MyQueue(const int& capacity);//创建队列,容量为 capacity
MyQueue::MyQueue(const int& n){
    capacity = n;
    queue = new int [capacity];
    front = 0;
    rear = 0;
}

// 销毁队列,~ MyQueue();//销毁队列
MyQueue::~MyQueue(){
    delete queue;
}

// 入队,bool enQueue(const int&);//入队,成功返回 true,失败返回 false
bool MyQueue::enQueue(const int& n){
    if((rear + 1) % capacity == front){
```

```
        return false;
    }

    queue[rear] = n;
    rear = (rear + 1) % capacity;
    return true;
}

// 出队,bool deQueue(int& item); //出栈,成功返回 true,失败返回 false, 出队元素放在 item
bool MyQueue::deQueue(int& item){
    if (rear == front){
        return false;
    }

    item = queue[front];
    front = (front + 1) % capacity;
    return true;
}

// 返回队头元素,bool getFront(int& item); //获取队首,成功返回 true,失败返回 false,栈顶元素放在 item
bool MyQueue::getFront(int& item){
    if (rear == front){
        return false;
    }

    item = queue[front];
    return true;
}

// 判断是否队满
bool MyQueue::isFull(){
    if ((rear + 1) % capacity == front){
        return true;
    }
    else {
        return false;
    }
}

void MyQueue::printQueue(){
    int cursor = front;

    while(cursor != rear)
```

```
    {
        if((cursor + 1) % capacity == rear)
            cout << queue[cursor] << endl;
        else
            cout << queue[cursor] << ",";

        cursor = (cursor + 1) % capacity;
    }
}

// 判断优先级 a 优先级高于 b 返回 true  a 优先级低于或等于 b 返回 false
bool priority(char a, char b){
    int ap, bp;
    if (a == '+' || a == '-') ap = 0;
    else ap = 1;

    if (b == '+' || b == '-') bp = 0;
    else bp = 1;

    if(ap > bp) return true;
    else return false;
}

//中缀表达式转后缀表达式,转化成功返回 true, 后缀表达式放在 result 中,表达式错误返回 false, 错误字符串 Expression is wrong!放在 result 中
bool postfixExpression(const string& s, string& result){
    int len = s.length();
    char* input  = new char [len];
    int len2 = len;
    strcpy(input,s.c_str());
    for(int i = 0; i < len; i++){
        if(input[i] == '(' || input[i] == ')'){
            len2--;
        }
    }
    char* output  = new char [len2];

    int outptr = 0;
    char ch;
    int flag1 = 0; // 统计数字个数
    int flag2 = 0; // 统计+-*/个数
    int flag3 = 0; // 统计左右括号数量是否一致
    MyStack1 ms1;

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

```
        if(input[i] >= '0' && input[i] <= '9'){
            output[outptr] = input[i];
            outptr++;
            flag1++;
        }
        else {
            if (input[i] == '+' || input[i] == '-' || input[i] == '*' ||
input[i] == '/'){
                flag2++;
                ms1.getTop(ch);
                if(ch == '('){
                    ms1.push(input[i]);
                    continue;
                }

                if(ms1.getTop(ch) == false){
                    ms1.push(input[i]);
                }
                else if(priority(input[i], ch)){
                    ms1.push(input[i]);
                }
                else {
                    ms1.pop(output[outptr]);
                    outptr++;
                    while(1){
                        if(ms1.getTop(ch) == false || priority(input[i],
ch) || ch == '('){
                        ms1.push(input[i]);
                        break;
                        }
                        else{
                            ms1.pop(output[outptr]);
                            outptr++;
                        }
                    }


                }
            }
            else if (input[i] == '('){
                ms1.push(input[i]);
                flag3++;
            }
            else {
                ms1.getTop(ch);
```

```
                while((!(ch == '(')) && !(ms1.isempty())){
                    ms1.pop(output[outptr]);
                    outptr++;
                    ms1.getTop(ch);
                }
                ms1.pop(ch);
                flag3--;
            }
        }
    }

    while(ms1.getTop(ch)){
        ms1.pop(output[outptr]);
        outptr++;

    }

    char* put = new char[len2];
    for(int u = 0; u < len2; u++){
        put[u] = output[u];
    }
    put[len2] = 0;

    if(((flag1 - 1 == flag2) && flag3 == 0) || (flag1 != 0 &&flag2 == 0
&& flag3 == 0)){
        result = put;
        return true;
    }
    else {
        result = "Expression is wrong!";
        return false;
    }


}



//表达式求值,表达式正确成功计算返回 true,结果放在 result 中,表达式错误计算不能
完成返回 false, result 中放 0
bool expressionVal(const string& s, int& result){
    string res;
    if (!postfixExpression(s, res)){
        result = 0;
        return false;
```

```
    }
    int len = res.length();
    char* charr  = new char [len];
    strcpy(charr,res.c_str());
    int* arr = new int [len];

    for(int i = 0; i < len; i++){
        if(charr[i] >= '0' && charr[i] <= '9') {
            arr[i] = charr[i] - '0';
        }
        else{
            switch (charr[i])
            {
            case '+':
                arr[i] = 60000;
                break;
            case '-':
                arr[i] = 60001;
                break;
            case '*':
                arr[i] = 60002;
                break;
            case '/':
                arr[i] = 60003;
                break;
            }
        }
    }


    for(int i = 0; i < len; i++){
        if(arr[i] >= 0 && arr[i] <= 9) {
            continue;
        }
        else {
            len = len - 2;
            switch (arr[i])
            {
            case 60000:
                arr[i-2] = arr[i-2] + arr[i-1];
                break;
            case 60001:
                arr[i-2] = arr[i-2] - arr[i-1];
                break;
            case 60002:
```

```
                arr[i-2] = arr[i-2] * arr[i-1];
                break;
            case 60003:
                if(arr[i-1] == 0){
                    result = 0;
                    return false;
                }
                arr[i-2] = arr[i-2] / arr[i-1];
                break;
            }
            for(int j = i - 1; j <= len-1; j++){
                arr[j] = arr[j+2];
            }
            i = i - 2;
        }

    }

    result = arr[0];
    return true;

}


void yanghui(const int& n){
    int lastone = 0;
    int thisone;


    MyQueue mq(n+2);
    mq.enQueue(1);
    mq.enQueue(1);


    for(int i = 2; i < n+2; i++){
        int j = i;
        while(j > 1){
            mq.deQueue(thisone);
            cout << thisone <<',';
            mq.enQueue(thisone + lastone);
            lastone = thisone ;
            j--;
        }
        mq.deQueue(thisone);
        cout << thisone << endl;
        mq.enQueue(thisone + lastone);
```

```
        lastone = 0;
        mq.enQueue(1);
    }

}
```

3. 实验结果

栈

```
3
4,0,146,-3,3
4
0,146,-3,3
0 0,146,-3,3
0
```

队列

```
3,-1,4,2,0,203
0
4,2,0,203
-1
4
0
```

四则运算

```
937*5-2/+18*+
25
Expression is wrong!
```

杨辉三角

```
1,1
1,2,1
1,3,3,1
1,4,6,4,1
1,5,10,10,5,1
```

4. 实验中遇到的问题及解决方法

理解了栈和队列的存储结构和操作原理。栈采用后入先出的原则,适用于实现递归和表达式求值等。队列采用先入先出的原则,适用于实现广度优先搜索等。

实现了中缀表达式到后缀表达式的转换。这是一个典型的应用栈的问题,通过比较运算符的优先级,使用栈实现转换。

实现了后缀表达式的求值。这也是一个典型的应用栈的问题,通过入栈出栈的方式,依次取出后缀表达式的运算符和运算数,进行相应的运算,最后结果入栈,栈顶元素即为表达式的值。

理解了栈在表达式求值中的重要作用。表达式求值实际上是通过压栈出栈来实现逆波兰表达式的计算过程。

实现了杨辉三角的输出。这个也算是一个使用队列的小应用。通过使用两个队列控制上一行和下一行的元素,实现每一行的输出。