数据结构 A
作业 4 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：栈
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

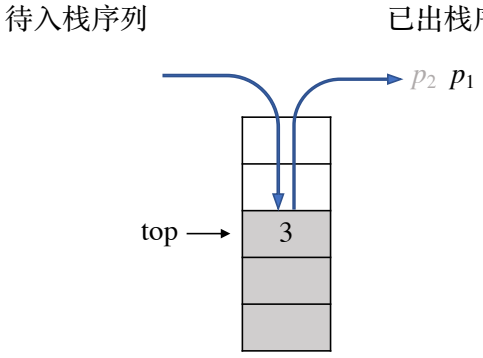
一个栈的入栈序列为 1，2，3，...，n，其出栈序列为 p_1，p_2，...，p_n，若 $p_2 = 3$，则 p_3 可能取值的个数是（　　）。

A. $n-1$
B. $n-2$
C. $n-3$
D. 无法确定

答案：A

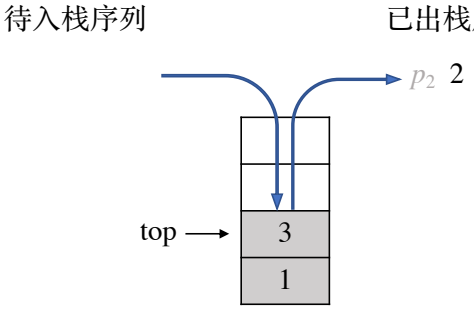
解析：在栈内，已知 $p_2 = 3$，说明**第二个出栈的元素是 3**。此时一定可以确定的是，**1，2 已经入栈过**，而 $4 \sim n$ 的入栈情况未知。

图中左侧待入栈序列中的数字一定 > 3。

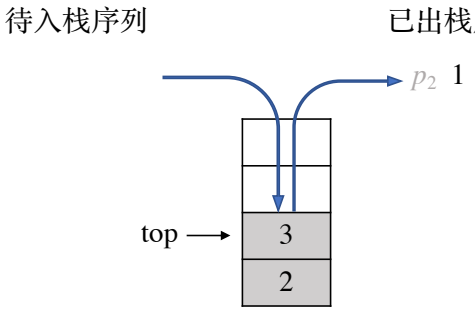


因此只要在 3 出栈后，放入任意个后面的数，都可以做到在 3 的下一个出栈，所以 $4 \sim n$ 都可能作为 p_3 出现。

此时再考虑 $1, 2$ 是否有可能作为 p_3。如果 2 作为 p_3，则为下图的情况，1 入栈、2 入栈、2 出栈。



如果 1 作为 p_3，则为下图的情况，1 入栈、2 入栈、2 出栈。



所以 $1 \sim 2, 4 \sim n$ 都有可能是 p_3。

习题 2 单选题

由两个栈共享一个数组空间的好处是（　　）。

A. 减少存取时间，降低上溢出发生的几率

B. 节省存储空间，降低上溢出发生的几率

C. 减少存取时间，降低下溢出发生的几率

D. 节省存储空间，降低下溢出发生的几率

答案：B

解析：上溢指的是栈满，共享栈空间表示两个栈分别占据一段空间的两端。因此可以互相调节存储空间，只有两栈都满时才会发生上溢，所以降低了上溢的发生几率。

习题 3 多选题

假如栈的入栈顺序是 a, b, c, d，下面 4 个选项中可能是它的出栈顺序的是（　　）。

A. a, c, b, d

B. b, c, d, a

C. c, d, b, a

D. d, c, a, b

答案：ABC

解析：参考本次习题 1 和例 3.6。一方面可以使用代码验证，另一方面可以自行画图验证。当我们有 $p_1, ..., p_{i-1}$ 时，要让下一个出栈的元素是 p_i，可能进行的操作是唯一的。

定义入栈序列为 a，出栈序列为 b，模拟一个栈。

若栈空，则无法操作出栈，这时将 a 的下一个元素入栈。

否则，检验 b 中下一个元素，相等时可以直接出栈，若不相等则只能继续往栈内加入元素。

当入栈序列已空，仍未匹配到出栈序列的下一个元素，则说明序列不可能出现。

A：a 入，a 出，b 入，c 入，c 出，b 出，d 入，d 出。

B：a 入，b 入，b 出，c 入，c 出，d 入，d 出，a 出。

C：a 入，b 入，c 入，c 出，d 入，d 出，b 出，a 出。

D：a 入，b 入，c 入，d 入，d 出，c 出，b 出**不来**，因此不选。

习题 4 填空题

把中缀表达式 `3+(5-2)*6` 转化为后缀表达式时，需要的顺序栈容量至少是（　　），得到的后缀表达式是（　　）（用 `#` 表示一个数字串结束）。

答案：3，`3#5#2#-6#*+`

解析：参考课本 3.1.7 部分，模拟入栈顺序。设 *postexp* 是后缀表达式。

遍历中缀表达式，遇到数字符，则将连续的数字符末尾加上“#”后添加到 *postexp* 后面；遇到“(”，将其进栈；遇到“)”，退栈运算符并添加到 *postexp*，直到

退栈的是“(”为止（该左括号不添加到 *postexp* 中）；

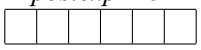
遇到运算符 op_2，将其跟栈顶运算符 op_1 的优先级进行比较，只有当 op_2 的优先级高于 op_1 的优先级时才直接将 op_2 进栈，否则将栈中“(”（如果有）之前[1]的优先级等于或高于 op_2 的运算符均退栈并添加到 *postexp*，再将 op_2 进栈。

模拟这一过程，栈内元素如下所示：

1. 遇到数字 **3**，直接添加到后缀表达式 *postexp*

原始串：**3**+(5-2)*6

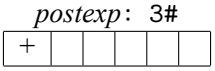
postexp：3#



2. 遇到运算符 +，发现栈内没有符号，可以进栈

原始串：3+(5-2)*6

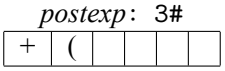
postexp：3#



3. 遇到左括号 **(**，将其进栈

原始串：3+**(**5-2)*6

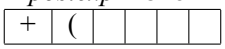
postexp：3#



4. 遇到数字 **5**，直接添加到后缀表达式 *postexp*

原始串：3+(**5**-2)*6

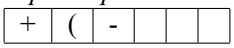
postexp：3#5#



5. 遇到运算符 -，发现栈顶是左括号，直接入栈

原始串：3+(5-2)*6

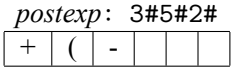
postexp：3#5#



6. 遇到数字 **2**，直接添加到后缀表达式 *postexp*

原始串：3+(5-**2**)*6

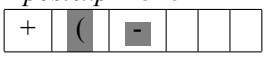
postexp：3#5#2#



7. 遇到右括号**)**，退栈，将运算符添加到后缀表达式，直到下一个左括号

原始串：3+(5-2)*6

postexp：3#5#2#-



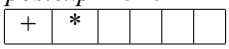
栈内元素数量最多为 3，后缀表达式：3#5#2#-6#*+

[1]这里的“之前”指遇到括号之前，也就是在栈顶操作

8. 遇到运算符 *，发现比栈顶的 + （op_1）优先级高，将运算符 * 入栈

原始串：3+(5-2)*6

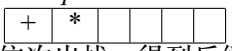
postexp：3#5#2#-



9. 遇到数字 **6**，直接添加到后缀表达式

原始串：3+(5-2)*6

postexp：3#5#2#-6#



10. 最后依次出栈，得到后缀表达式

postexp：3#5#2#-6#*+

习题 5 逆波兰表达式求值

【问题描述】

根据逆波兰表示法，求表达式的值。有效的运算符包括 +，-，*，/。每个运算对象可以是整数，也可以是另一个逆波兰表达式。假设给定逆波兰表达式总是有效的，换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。其中整数除法只保留整数部分。

【输入形式】

每个样例是一行，为有效的表达式，每个数字和运算符号之间用“,”隔开

【输出形式】

表达式的计算结果。

【样例输入】

2,1,+,3,*

【样例输出】

9

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<stack>
using namespace std;
```

```
stack<int> s;
int main()
{
    ifstream in("in.txt");
    char ch;
    while(in>>ch)
    {
        if(ch>='0'&&ch<='9')//此时读入的是数字
        {
            int x=ch-'0';//将数字存入x
            in>>ch;
            while(ch!=',')//数字结束后会有逗号承接下一个内容
            {
                x=x*10+ch-'0';//读入x的下一位
                in>>ch;
            }
            s.push(x);
        }
        else//输入的最后一个内容一定是运算符
        {
            if(ch=='+')
            {
                int x=s.top();//取出栈顶两个元素
                s.pop();
                int y=s.top();
                s.pop();
                s.push(x+y);
            }
            else if(ch=='-')
            {
                int x=s.top();
                s.pop();
                int y=s.top();
                s.pop();
                s.push(y-x);//注意减数关系
            }
            else if(ch=='*')
            {
                int x=s.top();
                s.pop();
                int y=s.top();
                s.pop();
```

```
47                  s.push(x*y);
48              }
49              else if(ch=='/')
50              {
51                  int x=s.top();
52                  s.pop();
53                  int y=s.top();
54                  s.pop();
55                  s.push(y/x);//保留整数
56              }
57              else if(ch!=',')//如果是逗号，说明还有内容，否则没有
58                  break;
59          }
60      }
61      cout<<s.top();//最后剩余一个栈内元素
62      return 0;
63  }
```

解析：逆波兰表达式求值的过程是模拟栈的运算过程，遇到数字直接入栈，遇到运算符则将栈顶两个元素出栈进行运算，将结果入栈。

需要注意的是，当运算符是减法和除法时，栈顶的元素是被减数/被除数。

本答案利用了对字符的读入，将字符转化为数字，再将数字入栈。在读入数字时，需要注意数字可能是多位数，因此需要循环读入直到遇到逗号。

习题 6 合并栈操作

【问题描述】

栈是一种具有后进先出的数据结构。可合并栈是支持“merge”操作的栈。三种操作的说明如下：

1. push A x：将 x 插入栈 A 中。
2. pop A：删除栈 A 的顶部元素。
3. merge A B：合并栈 A 和 B。

其中，“merge A B”操作后栈 A 包含 A 和 B 之前的所有元素，B 变为空，新栈中的元素根据先前的进栈时间重新排列，就像在一个栈中重复“push”操作一样。给定两个可合并栈 A 和 B，请执行上述操作。

【输入形式】

测试用例的第一行包含一个整数 $n(0 < n \leq 10^5)$ 表示操作个数，接下来的 n 行每行包含一条指令 push、pop 或 merge，栈元素是 32 位整数。A 和 B 最初都是空的，并且保证不会对空栈执行 pop 操作。以 $n = 0$ 表示输入结束。

【输出形式】

对于每个 pop 操作，在一行中输出对应的出栈元素。

【样例输入】

```
9
push A 0
push A 1
push B 3
pop A
push A 2
merge A B
pop A
pop A
pop A
```

【样例输出】

```
1
2
3
0
```

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 5 个测试用例，每通过一个测试用例，得 20 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<stack>
using namespace std;
stack<pair<int,int>> A,B;
int main()
{
    ifstream in("in.txt");
    string op;
    int n;
    in>>n;
    for(int i=1;i<=n;i++)
    {
        in>>op;
        if(op=="push")//需要两个输入
        {
            string s;
            int x;
            in>>s>>x;
            if(s=="A")//放入时间戳和元素
                A.push(make_pair(i,x));
            else
                B.push(make_pair(i,x));
        }
        else if(op=="pop")
        {
            string s;
            in>>s;
            if(s=="A")
            {
                cout<<A.top().second<<endl;
                A.pop();//注意元素存储在second里
            }
            else
            {
                cout<<B.top().second<<endl;
                B.pop();
            }
        }
        else//merge
```

```
41          {
42              string x,y;
43              in>>x>>y;
44              stack<pair<int,int>> C;//新栈，放入时是倒序
45              while(!A.empty()&&!B.empty())
46              {
47                  //倒序，所以时间戳大的先入C
48                  if(A.top().first<B.top().first)
49                  {
50                      C.push(B.top());
51                      B.pop();
52                  }
53                  else
54                  {
55                      C.push(A.top());
56                      A.pop();
57                  }
58              }
59              while(!A.empty())//归并的剩余部分
60              {
61                  C.push(A.top());
62                  A.pop();
63              }
64              while(!B.empty())
65              {
66                  C.push(B.top());
67                  B.pop();
68              }
69
70              while(!C.empty())//把C倒回A
71              {
72                  if(x=="A")//B变空
73                      A.push(C.top());
74                  else
75                      B.push(C.top());
76                  C.pop();
77              }
78          }
79      }
80      return 0;
81  }
```

解析：栈是先进后出的，但是同一时刻在栈内的元素，一定满足越靠近栈底，入栈时间越早。因此可以利用之前学过的**归并**进行 `merge` 操作。

所以使用 `pair` 存储栈内元素，`pair` 的第一个元素是入栈时间，第二个元素是栈内元素。在合并栈时，将两个栈的元素合并到一个新的栈中，按照入栈时间排序即可。

注：`pair` 是一种 STL 标准库自带的“二元结构体”。它的两个内部变量分别叫 `first` 和 `second`。

总结

题目：栈

日期：2024 年 4 月 19 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：可以考虑自行赋值 $p_2 = 3$，逐个尝试 p_3 检验是否可能出现相应的值。错的同学还是有一些，建议自行画图验证。

习题 2：按书上例 3.7 理解。注意上溢下溢的概念区分。

习题 3：手动模拟和代码验证都可以尝试，考试时鼓励使用代码验证与手动模拟互相核对。

习题 4：仍然是对书上例题的理解，强烈建议手动模拟计算过程。如果觉得麻烦可以标注每个变量的变化过程。

习题 5：难点在读入多位数，可以参考答案练习不同类型的数据输入（习题 6 也涉及字符串的读入）。

习题 6：入栈时间戳是关键思想。作业中一般对大家复杂度没有太高要求，但是可以时常联想归并的做法。归并是常用的合并有序数列的低复杂度做法。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。