data struct homework4

头文件包已经在群里发出来了，封装了(类似泛型的数组的)栈和队列以及一个链表排序，为了方便同学们使用，故意将链表排序实现放在.h文件中，更新了下头文件，其中链表的部分实验性使用

压缩包中有两个.c的示例

需要注意：

- 引用自己的头文件需要使用 `#include"dsdiy.h"`
- 栈和队列将结构体作为内容元素时需要将内容结构体tydef下
- 栈和队列使用前需要定义，对于每个栈和队列其在首次使用前都要在函数内使用flush刷新
- 链表排序主要就是看.c的demo里面的函数定义方法和注意事项

T1 栈操作（栈-基本题）

没有技巧，只有板子

这里千万不要用给你们的包，不然这题就无意义了

建议：好好封装成函数，留着备用

T2 C程序括号匹配检查

括号匹配建议:

1. 用fgets(字符串地址, 最长读入限制, 读入文件指针), 标准输入是stdin
   也就是fgets(s, 500, stdin)是限制最多读500的gets(s)
   文件就fgets(s, 500, filein)
   主要是方便//跳过一行，和记录行数
2. 过滤字符和注释考虑
   - 遇到//跳出当前行
   - 遇到/*找*/，*/后正常判定
   - 遇到 ' 或者 " 就找它自己

     ps: 注意/* xxxxx **/注释和/*/xxxx*/注释，极端下还能考虑'c/*ab(s*/'（单引号多内容）
3. 编写结构建议
   - 最重要的，利用出栈，一定要利用出栈把前面配对过的取出来，便于下一个判断
   - 对于整体结构，可以尝试一个while循环读入，里面一个while进行单行判定处理的二重循环

     内部处理循环前半部分先过滤字符、注释，后半部分写匹配逻辑，分开写更清晰
     对于过滤字符、注释，除了//直接跳出内部while，其它的考虑设置一个状态量"check"
     设定 `check=0`为正常判定模式 `check=1`为/*注释模式
     `check=2`为字符模式 `check=3`为字符串模式

具体参考以下不太伪的伪代码，明显发现过滤一二部分很像，思考下为什么分开，能不能合起来写(写代码我个人追求情况的一致性)

```
while 有输入
   读入一行

   while 这行没结束(即是当前字符不是'\n'也不是'\0')
        # part 1
        if check为/*注释模式 then
            if 当前字符和下一字符是*/ then
                回复check到正常判定模式,并跳到*/后，(使用continue)回到内部while处判断
        if check为字符模式或者字符串模式 then
            if 当前字符和其对应 then
                回复check到正常判定模式,并跳到当前字符后，(使用continue)回到内部while处判断
        if 当前字符和下一个字符是// then
            跳过当前行，(break)跳出内部while

        if 查到'\''或者'\"'或者"/*" then
            check进入对应模式(以便下次过滤)

        # part 2
        这里写你的正常处理部分，包括括号识别，判定逻辑

        # end handle
        跳到下一个字符
   end while

end while
```

下面说明为什么要用出栈特性(栈特性是后入先出)
对于(){(()}, 可能你会写一个标记数组，标记是否匹配过，然后去循环查找没配的来匹配
这你就不得不考虑(}),{(}这些情况乃至更恶心的情况

```
利用栈的特性:
读入(，直接入栈                (
读入)，栈顶(，匹配，弹出       空
读入{入栈，空栈直接入          {
读入(，直接入栈                {(
读入)，栈顶(，匹配，弹出       {
读入}，栈顶{，匹配，弹出       空
完成，空栈，匹配正确
```

如果你问怎么最后输出，那么你不妨考虑新开一个数组无脑读入所有过滤后的大小括号

T3 计算器（表达式计算-后缀表达式实现，结果为浮点）

这一部分引导大家适度分割函数，以下采用的是中途处理计算的策略，不是先换为后缀表达式再解析计算后缀表达式

1. 再次建议使用fgets()读入，便于过滤空格，字符串转数字可以使用atoi函数也可以考虑自己写个函数(这个功能很明确且不容易出错，建议写成一个函数)
2. 编写结构建议：
    - 输入和预处理部分

先用fgets读入，然后用前几次作业中的某个填空题的代码将其中的空格过滤掉(没必要封装成函数，就用一次，但是也可以封装，留着以后用?)

- 处理部分

  建议设置操作数栈和操作符栈，不写在一起，简化思路

  计算逻辑采用符号优先级的逻辑，给(、)、+-、*/附上优先级，例如设置一个 `int priority[48];` 预先赋值

  `priority['('] = priority[')'] = 0, priority['+'] = priority['-'] = 1, priority['*'] = priority['/'] = 3`，当读入一个操作数时，只需要将其入操作数栈，而操作数栈设立的目的是保留一部分操作，比如2+3*5，那么我们就该暂时保留+，而优先处理*，也就是说操作数符号是用来暂存低优先级操作的

  当读入一个操作符时，需要判断目前栈顶符号的优先级，如果栈顶符号优先级高于新读入符号，那么肯定就应该将它算完，新符号才能入栈，计算后的结果重新入操作数栈就好，同时你必须将所有优先大于新读入操作符的栈内操作算完，即是说栈内操作符号优先级应该递增

  那么是否改严格递增，也就是上述判断"优先级大于新输入操作"时判不判等于? 这题按道理应该严格递增，因为+-*/同级情况下是自左向右计算，考虑*6/3*2+1，如果你不判等，那么你会在6/3*2入栈后，读入+，计算3*2，然后计算6/6~~大喜~~

  特殊情况考虑: 因为此题架构问题，'('成为了一个尴尬的符号，细心的同学已经发现了，'('优先级是0，也就是说它会将前面全部的符号算完，但是事实上，读入'('后应该特判直接入栈，这里设计为优先级0，只是因为在其它除了')'的符号读入时'('可以雷打不动待在栈内，')'读入时将一直算到'('，但是它不操作符进栈(这里也许也需要特判)

处理逻辑框架参考:

```
while 字符串未处理完

    get(操作数)
    stack_pushback(操作数)

    get(新操作符c)
    if c == '(' 或者 stack_empty(op_stack) then
        c入栈op_stack并跳过后续步骤直接到下一个循环

    while priority[c] <= priority[op_stack栈顶]
        取操作数计算然后结果入栈，op_stack栈顶弹出
    end while

    if c != ')' then
        新操作符c入栈op_stack

end while
```

注意结尾的 = 应该将前面的全部出栈计算，这里怎么设计(是设计在处理逻辑内，还是处理逻辑后&输入之前)看各位同学自己来

先翻译为后缀表达式也可以参考上诉的优先级思路，此外还有分符号优先级的多函数相互调用递归的解法，有兴趣自行了解

T4 文本编辑操作模拟（简）

对于每一条新操作(前n条之后指令)，可以在输入之后就立刻操作文本，也可以存起来，等着操作输入完再一口气操作，当然由于有撤销操作，所以推荐后者，而不管选择哪一种做法，都避不开用栈和队列的思想，因为撤销是从最后的指令开始撤销，而操作是从最初的指令开始操作

一个简单的思路是，对于前n条操作，预先将倒着遍历一遍逆向操作(1删除2增加)，然后从n+1条开始记录新的指令，1、2入栈，3出栈栈顶操作，最后再从栈底(从头)开始执行

如果选用实时操作后面遇到3再撤回，那么你在删除时不仅要入栈，还要记录在哪里删除了什么，这样才能保证恢复的时候信息是完整的

T5 银行排队模拟（生产者-消费者模拟） - 分类别

和上一题一样是一个模拟，不过这里直接使用队列模拟就好，上个题要用栈和队两种思想。

问题是，是一个队列还是n个队列?
根据生活常识，显然是一个队列，真实情况下，大家都知道，银行医院等都是摇号排队，一旦有窗口，就呼叫序号最小的人，和此题从1开始编号如出一辙

更进一步，为什么选择一个队列，因为按照题意，开关窗口是看的等待人数(而不包含正在办理人数)，而且如果预先分好，在增删窗口时就会改变分配，逻辑十分的复杂

所以我们就确定了模拟的模型，一个排队队伍(队列)，一些乘客(结构体，包含排队号码，处理时间，入队时间)，一次次的处理周期(大循环)

关于增删窗口逻辑
只在周期最初(新增人之后)考虑增加窗口，在周期最末(前面处理完成之后，新的人上去处理，队伍人数减少后)考虑减少窗口

考虑这么一个问题
要是缩减窗口之后，有人在对公窗口办理业务怎么办，根据实践，将其赶到对私窗口是能过的，那么如果用常规数组去模拟窗口，那么会变得麻烦。

- 第一种方法是5个数组存放当前办理时间模拟正在办理逻辑，这会遇到上面的问题，需要不停转移用户，保证都在对私窗口
- 第二种是没有实际的窗口，有一个处理队列指针，将客户信息设计为用户节点，队列也是链表队列，每次就查询完成业务的客户出处理链表，当链表长度小于窗口数目是出队入链
- 第三种是抽象的窗口，换个脑子，第一种是5个窗口记时间，这边3个时间记窗口数目就好了，每次高时间窗口向低的转化，最低的就是办完的，三者总数就是正在办理的窗口数目

强力推荐方法2、3，arong写的第一种，遇到颇多问题

关于坑点

- 何时结束模拟的大循环：循环次数比用户波次大并且无人等待(有人办理无所谓，我们只输出等待时间)，这就牵扯到另一个坑点只在获取新客户（不管到达新客户数是否为0）时，才按策略调整服务窗口数，也就是没新人来了(循

环次数大于用户波次之后就不再考虑窗口增加了)，至于有客户去接受服务（即等待客户减少），银行将根据策略及时减少服务窗口，每次大循环结束前都判就好了，有人办理自然会减，没人去办理减也减不了

- 末尾(或者某段时间)客户少于3，部分做法会超出队列范围
- 增减窗口看的是等待人数，正在办理者不予考虑

（模型三）参考框架如下

```
while i<用户波次 and 空等待队列
    # 新人入队等待逻辑
    if i<用户波次 then
        初始化用户信息并进入等待队列最后考虑加窗口

    # 办理业务逻辑
    time[0] = time[1] # 办理时间2的变1(顺带清空时间1的窗口数)
    time[1] = time[2] # 办理时间3的变2(顺带清空时间2的窗口数)
    time[2] = 0       # 清空办理时间3的窗口数

    # 新人出队办理逻辑
    while 开放窗口数 > time[0]+time[1]+time[2] and 队列没空
        出队并将对应时间的窗口数目加一(time[x]++)，还要在这里输出
    end while

    # 末尾减少窗口逻辑
    考虑减少窗口

end while
```

函数调用关系（选做，不计分）

暂无