上机测试四

2019 年 11 月 8 日

1 题目描述

实现一个 OperationSequence 类,使之能够解析一个 OperationSequence (以下缩写为 OpSeq)，并完成对应的计算功能。

1.1 OpSeq 和 Step

对于 OpSeq ，我们规定如下:

1. 一个 OpSeq 由若干个 Step 组成;
2. 每个 Step 都接受一个整数列表，并在进行一定的计算（由对应的 Operation 定义) 后将计算结果传递到下一个 Step ；同时，每个 Step 内还保存一个计数器，用来记录当前已处理的所有列表（进入该 Step 前) 的长度之和。

1.2 Operation

关于 Operation ，我们规定如下:

1. 每个 Operation 定义了对整数列表的一种操作，且其操作之后的结果仍然是一个整数列表。
2. Operation 分为一元 Operation 和二元 Operation ,其中二元 Operation 的第一个参数会提前指定。

Step 内支持的 Operation 有如下 4 种:

1. init：一元 Operation 。返回一个列表的头部，其中头部指包含除了最后一个元素之外所有的值的列表。例: 将 init 应用在 [2,3,4] 则返回 [2,3]，将 init 应用在 [] 上则返回 [] （[] 为空列表）。

2. tail：一元 Operation 。返回一个列表的尾部，其中尾部指包含除了第一个元素之外所有的值的列表。例: 将 tail 应用在 [2,3,4] 则返回 [3,4]，将 tail 应用在 [] 上则返回 [] 。

3. add: 二元 Operation。返回对列表中的每个元素加上第一个参数的值之后的结果。例: 将 add 5 应用在 [2,3,4] 则返回 [7,8,9] 。注意，我们规定 add x 作用在 [] 上仍然返回 [] 。

4. reverse：一元 Operation 。返回列表倒序之后的结果。例: 将 reverse 应用在 [2,5,1] 上则返回 [1,5,2] 。

在本次测试中，OpSeq 中每个 Step 对应的 Operation 由以下两例所示的方式指定:

Example 1.1 *tail*

Example 1.2 *init . tail . add 1*

其中，Operation 之间以 "." 分隔，"." 表示函数复合，例如若 f, g 为两个 Operation, x 为一个列表，则把 f . g 应用在 x 上将返回 f(g(x)) ；二元 Operation 的第一个参数也会在上述字符串中给出。

1.3 接口说明

你需要定义 OpSeq 类，自行设计其成员变量，以支持如下接口:

```
class List{
    public:
    List(int* a, int len);
```

```
    void show();
}
class OpSeq{
   public:
      OpSeq(string s);
      List forward_computing(List input);
      List backward_computing(List input);
      int gather_value();
}
```

其中各接口说明如下：

1. `List(int* a,int len)`，接受一个整形数组以及其对应的长度作为参数，构造一个 `List` 对象；当 `a` 为 `nullptr` 且 `len` 等于 0 时，代表当前列表为空。
2. `show()`，依次打印出 `List` 中所有元素，元素之间以空格分隔，以换行符结尾；若 `List` 所表示的列表为空，则打印出 `EmptyList` 。
3. `OpSeq(string s)`，构造函数，`s` 为待解析的 `Operation` 序列。
4. `List forward_computing(List input)` 接受一个整数列表，正向地使用当前 `OpSeq` 中的 `Step` 处理该列表，并返回最终的结果。例：若 `s` 中定义的 `Operation` 序列为 `init . reverse . add 1`，`input` 为 `[2 5 3]`，则运算过程为 `init(reverse(add(1,[2,5,3])))` 返回列表 `[4,6]`
5. `List backward_computing(List input)`，接受一个整数列表，逆向地使用当前 `OpSeq` 中的 `Step` 处理该列表，并返回最终的结果。例：若 `s` 中定义的 `Operation` 序列为 `init . reverse . add 1`，`input` 为 `[2,5,3]`，则运算过程为 `add(1,reverse(init([2,5,3])))`，返回列表 `[6,3]`
6. `int gather_value()`，返回 `OpSeq` 中各 `Step` 内计数器的值之和。

2 示例调用

```cpp
// in OpSeq.cpp
#include "OpSeq.h"
int main()
{
    OpSeq seq("init . reverse . add 1");
    int a[3] = {2,5,3};
    List x(a,3);
    List ret(nullptr,0);


    ret = seq.forward_computing(x);
    ret.show();
    cout << seq.gather_value() << endl;

    ret = seq.backward_computing(x);
    ret.show();
    cout << seq.gather_value() << endl;
}
```

输出为:

```
4 6
9
6 3
16
```

3 注意事项

1. 所有输入的函数序列皆为合法。

2. 在提交时，请将 `OpSeq.h` ， `OpSeq.cpp` 压缩为 `OpSeq.zip` 提交