Assignment 3 ADT

> 本次作业中建议大家使用`StanfordCppLib`中的ADT，接口的使用方法大家可以到StanfordCppLib网站上搜索。

> 本次作业中，使用judger可以不指定输入输出目录和源程序目录，使用命令如下（以第一题为例）：
>
> ```
> python judger_batch.py -T 1_merge
> ```
>
> 这条命令等价于：
>
> ```
> python judger_batch.py -T 1_merge -I data/1_merge -O data/1_merge -S source/1_merge
> ```
>
> `-I`和`-O`默认为`data/task_name`，`-S`默认为`source/task_name`；你也可以显示地指定这些路径。

> 如果你使用了`StanfordCppLib`，那么需要将编译`StanfordCppLib`产生的cs1604文件夹的**绝对路径**（可直接复制文件管理器上的路径）复制到`source/cs1604.txt`下（需要自己创建），以让`judger`成功编译你的程序。如果不使用`StanfordCppLib`，则不需要创建`cs1604.txt`。

> 在助教进行评测的时候，我们的judger会根据你提交文件夹的目录下是否有`cs1604.txt`这个文件来决定是否引入`StanfordCppLib`来编译你的程序，所以如果使用了`StanfordCppLib`，那么在你提交的文件夹的目录下需要有`cs1604.txt`这个文件（里面内容可以为空），来声明自己使用了`StanfordCppLib`。

> 本次作业一共有四道题目，总分为十分，其中前三道为必做题，分数共计十分；第四道为选做题，难度较大，可用于补救前三道题的错误，分数为两分，附加分数不会溢出总分。

1. 合并有序数组

合并两个有序数组为一个新的有序数组是实现归并排序算法的基础，本题会给定两个升序数组，需要你输出这个两个数组合并后的升序数组。

题目描述

给定两个升序的整数数组`v1`和`v2`，长度分别为`n`和`m`，返回它们合并后的长度为`n+m`的升序数组。

输入输出格式

输入第一行为两个数组的长度`n`和`m`，输入第二行是`n`个从小到大的整数，代表第一个数组，输入第三行是`m`个整数，代表第二个数组。整数范围为`int`范围。

输出长度为`n+m`的升序整数序列，代表合并后的数组。

输入

```
3 5
1 3 5
2 4 6 8 10
```

输出

```
1 2 3 4 5 6 8 10
```

数据范围

对于100%的数据，`0 < n,m <= 1e5`

2. 迷宫

在本题中，会给你一个二维迷宫以及多个入口，你的任务判断这几个入口是否能抵达迷宫的出口，你需要使用合适的`ADTs`去表示和求解迷宫。

题目描述

迷宫可以用一个二维向量或者说网格（Grid）来表示，每一个坐标是一个通道（用字符`.`来表示）或者一个墙壁（用字符`W`表示），迷宫的边界外可以视为都是墙壁。迷宫只能往上下左右方向探索。 迷宫的入口会通过输入来给定（保证在迷宫的范围内），迷宫的出口固定为迷宫的右下角，输入的迷宫中的出口一定是通道。对于每一个入口，如果这个入口能抵达出口，则输出`reachable`，否则输出`unreachable`，如果给定的入口已经是一个墙壁，则直接输出`unreachable`。一个7x7迷宫的例子如下：

```
.W..WWW
...W...
.W..W..
..WW...
W....W.
W.W.W..
W....W.
```

如果入口为迷宫的左上角（即`(0,0)`），上面的迷宫的一个求解路径可以表示为（用`x`表示路径）：

```
xW..WWW
x..W...
xW..W..
xxWWxxx
WxxxxWx
W.W.W.x
W....Wx
```

输入输出格式

输入的第一行为`n m k`，分别表示迷宫的行数，列数和入口的数目，第二行开始的`k`行输入`k`个入口的坐标，之后开始输入`n`行`m`列的迷宫。我们保证输入数据合法。输出为`k`行，第`i`行代表第`i`个入口是否可以到达迷宫出口

输入

```
7 7 1
0 0
.W..WWW
...W...
.W..W..
..WW...
W....W.
W.W.W..
W....W.
```

输出

```
reachable
```

数据范围

对于100%的数据，`2 < n,m < 500`，`0 < k < 10`

提示

你可以使用图算法**广度优先搜索（Breadth First Search）**来求解迷宫，这个算法可以用`queue`来实现，下面的伪代码是该题的一种解法：

```
for each entrance:
  create an empty queue;
  add this entrance to the queue;
  while (the queue is not empty):
    dequeue a position from the front of the queue;
    if this position is the exit:
      break
    for each point adjacent to the position in a cardinal direction:
      if this position is not a wall:
        add that point to the queue;
  output the reachability of the exit;
```

注意：所有搜索过的坐标不需要再搜索，否则会导致死循环，如何在上述算法中实现该功能?

在本题中你可能会用到`pair`来表示一个点，下面是`pair`的一些基本的初始化方法，详细可以参考C++中pair的用法。

```
typedef pair<int,int> point;
...
point p1(1,1);
point p2;
p2.first=2;
p2.second=2;
```

3. ASM虚拟机

在道题中，你将会实现一个简单的"ASM"语言（Assembly）的虚拟机。虚拟机也是一个计算机程序，它可以模拟真实的计算机系统，而在这次作业中要实现的虚拟机功能是模拟一段程序的执行，不过这个程序不是用C, Java, 和Python等语言写的，而是用一个更加简单，更加接近计算机底层的语言——"ASM"

我们要实现的虚拟机的核心部件正是大家学过的`Stack`，基于栈的虚拟机称为**Stack Machine**[1]，例如课上介绍的RPN。实际中Stack Machine的例子有Java虚拟机JVM[2]和Python的字节码解释器CPython[3]等。

除此之外，另一种常见的计算模型是**Register Machine**[4]，它是一种更加符合现代计算机体系结构的模型。

栈机器

我们的虚拟机通过输入一段程序，对这段程序进行解析，并模拟运行，最终输出结果。

Stack Machine的核心是它的计算模型。

计算模型

Stack Machine的计算模型包含三部分：

- A program counter(pc)：指向现在执行的语句，值为整数，从0开始
- A state: 存储变量到其值的映射，变量的类型是`string`，值的类型是`int`
- An evaluation stack: 存储操作数，操作数都是`int`类型
- Language: 该栈机器的执行语言

语言

我们的机器输入的是一个小型的程序语言"ASM"，它是一种更接近计算机底层的语言，也可以称为指令。

具体我们要实现的指令有：

- `Add`: 将栈顶两个元素出栈，两者相加，并将结果放入栈顶
- `Sub`: 将栈顶两个元素出栈，第二个出栈的元素减去第一个出栈的元素，并将结果放入栈顶
- `Mul`: 乘法，与`Add`类似
- `Div`: 整数除法，与`Sub`类似
- `Assign x`: 将栈顶元素a出栈，并更新state将变量x映射到a，其中x是一个类型为string的变量名
- `Var x`: 将state中变量x的值放入栈顶
- `Jmp n`: 跳转到pc=n处
- `JmpEq n`: 将栈顶两个元素出栈，若两者相等，则跳转到pc=n处
- `JmpGt n`: 将栈顶两个元素出栈，若第二个出栈的元素大于第一个出栈的元素，则跳转到pc=n处
- `JmpLt n`: 小于关系，执行方式与`JmpGt`类似
- `Const n`: 将整数n放入栈顶
- `Print x`: 打印变量x的值并换行

- `Halt`: 程序结束

程序执行

从pc=0开始，程序每一步会读取当前`pc`所指向的指令，并执行指令，每执行一条非跳转（不是`Jmp`类型）指令，`pc=pc+1`。若遇到`Jmp n, JmpEq n, JmpGt n, JmpLt n`这类的指令，则根据执行结果选择是否跳转到`pc=n`处，如果不跳转，则`pc=pc+1`。更新`pc`后重复这一过程，直到遇到`pc`指向`Halt`或者程序运行错误。

程序的执行伴随着**stack**和**state**的变化，在你实现Stack Machine的程序中，应该使用两个**ADT**来分别表示它们。

程序结果的正确性通过`Print x`指令的输出以及错误处理来判断，我们的测试会比较你的输出与正确的输出。

错误处理

用这个语言写的程序并不保证安全，它有可能出现各种各样的错误，本题需要你处理四种错误（其他错误不考察）：除零，`pc`跳转到了程序之外，操作数缺失和输出未定义变量。在程序运行的时候，如果出现了这些错误，你需要输出`Error`并终止程序（注意这个错误并不会影响之前运行输出的数据）

示例: factorial

计算10的阶乘，输入的第一行代表指令的数量。

Input

这里为了方便解释把指令所在的位置标出来，实际输入中不会有这些值。

```
18
0: Const 10
1: Assign z
2: Const 1
3: Assign y
4: Var z
5: Const 0
6: JmpEq 16
7: Var y
8: Var z
9: Mul
10: Assign y
11: Var z
12: Const 1
13: Sub
14: Assign z
15: Jmp 4
16: Print y
17: Halt
```

Output

```
3628800
```

其对应的C++代码如下，注释中标出每条语句对应输入的哪些指令

```
//C++ like language
//Every statement corresponds to serval commands
z = 10; //pc from 0 to 1
y = 1; //pc from 2 to 3
while(z != 0) { //pc from 4 to 6
      y = y * z; // pc from 7 to 10
      z = z - 1; // pc from 11 to 14
} // pc 15
cout<<y<<endl; // pc 16
```

其中`pc=4`到`pc=6`是判断`z`的值是否为`0`，如果是，则跳转到`pc=16`输出阶乘的结果，不是则进入循环。`pc=15`是执行完循环体并跳回到循环条件判断语句，也就是`pc=4`

输入输出格式

第一行`n`表示命令数量，接下来n行输入n条命令，我们保证输入格式都是合法的。

Input format

```
n
command 0
command 1
...
command n-1
```

Output

对于运行中遇到的`Print x`指令都要输出一行数据或者在运行错误时输出`Error`并终止程序

数据范围

对于100%的数据，`0<n<100`

提示

- 关于指令的读取，这里推荐一种方法：你可以通过`getline`方法读取一行指令，比如`"Const 10"`。接下来你需要将它分解（split），根据空格来将指令分成一个个的`token`，这里的`token`分别为`"Const"`, `"10"`，然后将之存储到`Vector`中。关于如何根据空格来分解`string`，这里贴出网上提供的解决方法How do I iterate over the words of a string?
- 关于如何将`string`转换为`int`，可以使用`stoi(str)`函数

```
string str = "10";
int a = stoi(str);
```

4. 文本编辑器（选做）

文本编辑器（text editor）是一种计算机软件，主要用于用来编写和查看文本文件。程序员写程序也是通过编辑器来编写程序，如今的各种集成开发环境（IDE）都会自带文本编辑器。比较著名的文本编辑器包括Vim[5],GNU Emacs[6],notepad++以及现在很热门的VSCode。

文本编辑器的设计需要考虑性能，用户体验，功能性，可扩展性等等，大多数时候这些性质是互相冲突的，所以要设计出一个好用的编辑器并不简单[7]。不过编辑器的基本功能并不复杂，在本题中，我们的目标是实现一个通过命令行来进行操作的文本编辑器，它具有最简单的编辑功能（如插入，删除，撤回等）。

题目描述

我们的编辑器会存储文本并记录当前光标（cursor）的位置，文本可以看成是字符串序列，每一个字符串代表着一行中的内容，光标是一个二维坐标`(x,y)`，指向第x行第y个字符之后，注意文本行数从1开始，每行的字符坐标从1开始（在这个语境下，“第0个字符之后”的意思就是行首）。编辑器通过命令来修改文本和光标，命令的规范如下：

- `c n m`：将光标移动到第`n`行第`m`个字符后，如果没有这个位置，则忽略此命令
- `i s`：在当前光标后插入字符串`s`（`s`不会含有换行符），插入结束后光标移动到`s`末尾
- `d k`：从当前光标开始，从后往前删除`min(k,m)`个字符（`m`表示当前光标位置到当前行行首的字符数目），删除结束后光标移动到被删除字符串的开始位置
- `ENT`：在当前光标下换行，光标后的文本转移到新行，光标位置移动到新行的行首（该操作与平时在编辑器上按下回车相似）
- `u`：undo操作，撤回最近一次**可撤回**的操作，如果没有这种操作，则忽略此命令（Windows上类似于在VSCode中按下ctrl+z，Mac上是cmd+z）。**可撤回**的操作有`c`,`i`,`d`,`ENT`和`r`且这个操作没有被编译器忽略，一个命令不能被undo两次
- `r`：redo操作，撤回最近一次没有被redo过的undo操作，如果没有这种undo操作，则忽略这次操作（Windows上类似于按下ctrl+y，Mac上是cmd+shift+z）。**注意：**为了实现的方便，undo和redo操作之间不会有`c`,`i`,`d`,`ENT`这些操作（即不会出现`i s1 => u => i s2 => r`这种情况）
- `p`：打印当前编辑器的文本内容，然后打印一个换行符（该指令不能被撤回）

初始状态下，光标位于第1行第0个字符后，也就是第一行空行的行首。

> 鉴于undo和redo实现的难度较大，我们的测试数据有40%不会出现undo和redo，70%不会出现redo。让同学们可以先着手实现基本功能再去考虑进阶功能。

输入输出格式

我们保证输入格式都是合法的

输入的第一行为`k`，表示有k个命令

接下来的k行每行一条命令，我们保证输入都是合法的。

```
k
command 0
command 1
...
command k-1
```

输出：对于运行中遇到的`p`，输出对应的文本

示例

输入
这里为了方便解释把指令所在的位置标出来，实际输入中不会有这些值。

```
20
1: i #include<iostream>
2: ENT
3: i using namespace std
4: i ;
5: ENT
6: i int main(){
7: c 3 10
8: ENT
9: c 4 1
10: ENT
11: i cout<<"Hello World!";
12: d 14
13: u
14: u
15: r
16: ENT
17: i return 0; }
18: c 5 19
19: i \n
20: p
```

输出

```
#include<iostream>
using namespace std;
int main()
{
cout<<"Hello World!\n";
return 0; }
```

解释

在执行完第11条命令的时候编辑器的文本如下（带行号标记）：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5: cout<<"Hello World!";
```

这个时候光标在第5行的末尾。第12条命令删除了14个字符，文本变为：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5: cout<<"
```

接下来第13条命令undo了这个操作，文本变为：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5: cout<<"Hello World!";
```

第14条命令是undo操作，因为第12条的删除命令已经被第13条命令undo过了，所以这个undo操作会undo第11条命令，也就是插入命令，这之后文本变为：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5:
```

然后第15条的redo命令会撤回14条的undo命令，所以文本变为：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5: cout<<"Hello World!";
```

执行完第16到第19条命令后文本变为：

```
1: #include<iostream>
2: using namespace std;
3: int main()
4: {
5: cout<<"Hello World!\n";
6: return 0; }
```

其中第18和第19条命令会把光标移动到第5行第19个字符之后，也就是`!`的后面，然后插入`\n`

数据范围

对于40%的数据，不会出现undo和redo操作

对于70%的数据，不会出现redo操作

对于80%的数据，`0<k<100`；对于100%的数据，`0<k<200000`

提示

- 可以使用`Vector<string>`来表示文本，插入和删除操作可以利用`string`提供的各种方法
- undo和redo操作可以使用两个栈来表示

提交格式

你提交的文件结构应该类似如下形式：

```
<your student number>.zip
|- 1_merge
|   |- main.cpp
|
|- 2_maze
|   |- main.cpp
|
|- 3_asm_vm
|   |- main.cpp
|
|- 4_editor
|   |- main.cpp
|
|- cs1604.txt (include it if you use StanfordCppLib)
```

---

1. https://en.wikipedia.org/wiki/Stack_machine ↩
2. https://en.wikipedia.org/wiki/Java_virtual_machine ↩
3. https://en.wikipedia.org/wiki/CPython ↩
4. https://en.wikipedia.org/wiki/Register_machine ↩
5. https://github.com/vim/vim ↩
6. https://www.gnu.org/software/emacs/ ↩
7. 怎么去实现一个简单文本编辑器？. 知乎 ↩