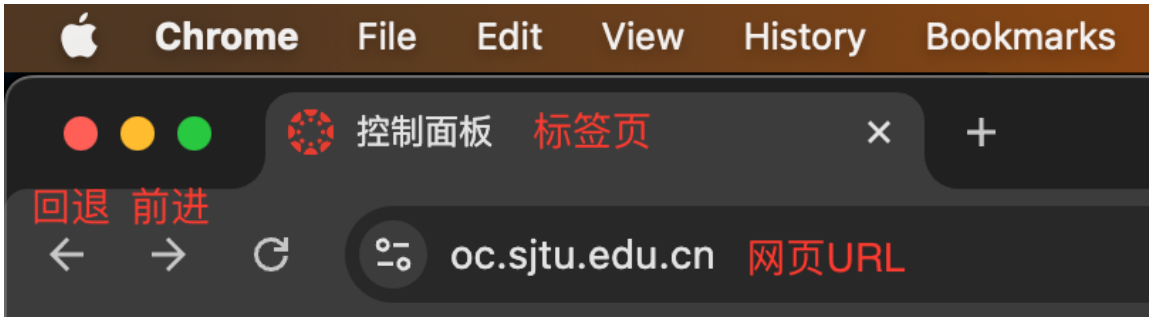
Assignment 4 ADT

1. 浏览器前进与回退

在现代网页浏览器（如 Google Chrome、Firefox、Edge）中，前进（Forward）和回退（Back）功能是最常用的导航机制之一。用户在浏览网页时，可能会频繁跳转页面，并希望能够回到上一页或前进到刚刚访问过的页面。



题目描述

本题需要同学们使用合适的ADT实现一个简单的**网页浏览器历史管理系统**，我们只要求支持在单个标签页下的操作：

1. **访问新网页**：`visit URL`（访问`URL`，并清空**回退历史**，即不能在新网页上点击前进）
2. **回退**：`back`（查询**访问历史**，返回访问的上一个网页，如果没有上一个网页，则忽略此命令）
3. **前进**：`forward`（如果之前回退过网页，则前进到回退前的网页，如果没有回退历史，则忽略此命令）
4. **查询当前网页**：`current`（输出当前访问的网页`URL`并换行，如果当前没有访问网页，则忽略此命令）

解释：访问历史为用户已经访问过的网页，比如下面三条指令执行完后访问历史可以表示为`{url1, url2, url3}`

```
visit url1
visit url2
visit url3
```

回退历史记录着之前回退过的（即执行`back`）的网页，比如在上面三条访问指令执行完后再执行如下指令，回退历史可以表示为`{url3, url2}`，意思就是第一个回退的网页为`url3`，第二个回退的网页是`url2`，那么如果在当前使用`current`命令，打印出的网页就是`url1`

```
back
back
```

初始时浏览器没有访问的网页，即访问历史和回退历史都是空的

具体例子

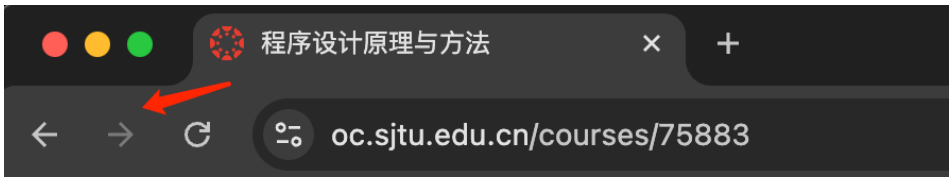
输入（第一行为指令数量）

```
10
visit google.com
visit oc.sjtu.edu.cn
current
back
current
forward
current
visit oc.sjtu.edu.cn/courses/75883
forward
current
```

输出

```
oc.sjtu.edu.cn
google.com
oc.sjtu.edu.cn
oc.sjtu.edu.cn/courses/75883
```

其中需要解释的是第四个current命令，我们可以在自己的浏览器上模拟上面的操作，可以看到在current前的forward命令是会被忽略的



所以第四个current的输出是`oc.sjtu.edu.cn/courses/75883`

输入输出格式

输入的第一行为`n`，表示有`n`条指令

```
n
command 1
command 2
...
command n
```

输出为遇到`current`命令时输出的网页`URL`

数据范围

对于100%的数据，`1 < n < 10000`

提示

- 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);
```

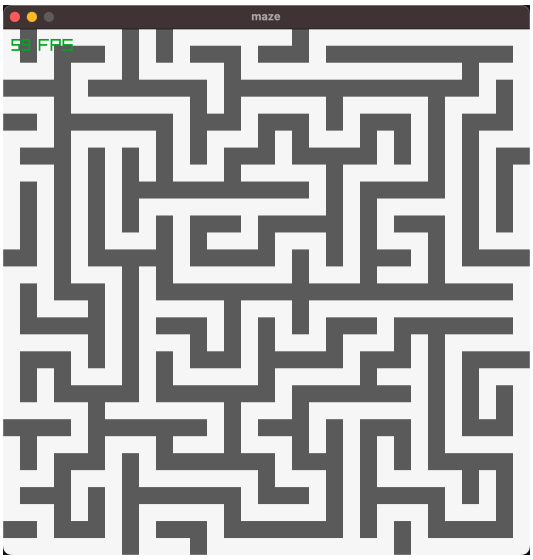
- 关于`cin`和`getline`交替使用时会出现换行符被留在输入流导致`getline`无法读出预期输入的问题

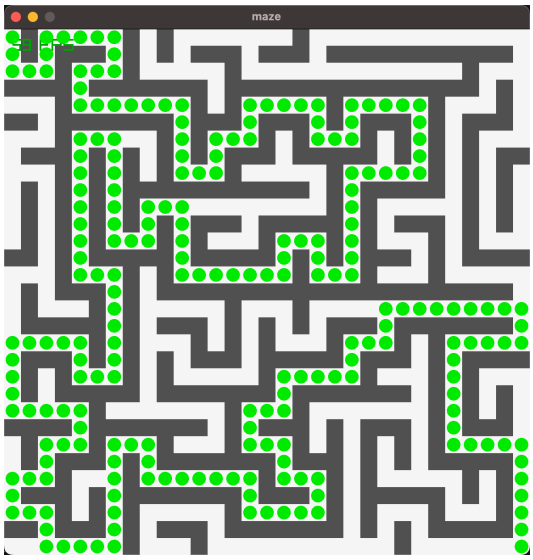
```
int n; string cmd;
cin >> n; // The '\n' would be left in the cin stream
getline(cin, cmd); // getline gets nothing because it encounters '\n'
```

一个解决方法是在`cin`后插入`cin.ignore()`跳过换行符

```
int n; string cmd;
cin >> n;
cin.ignore(); // skip '\n'
getline(cin, cmd);
```

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（注意出口固定为迷宫的右下角）

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，而且还需要考虑到程序的效率，比如我们提供的第5个测试样例是一个比较大的样例，如果你用的方法不够高效，程序的运行时间会过长（judger设置最长运行时间为20s）。

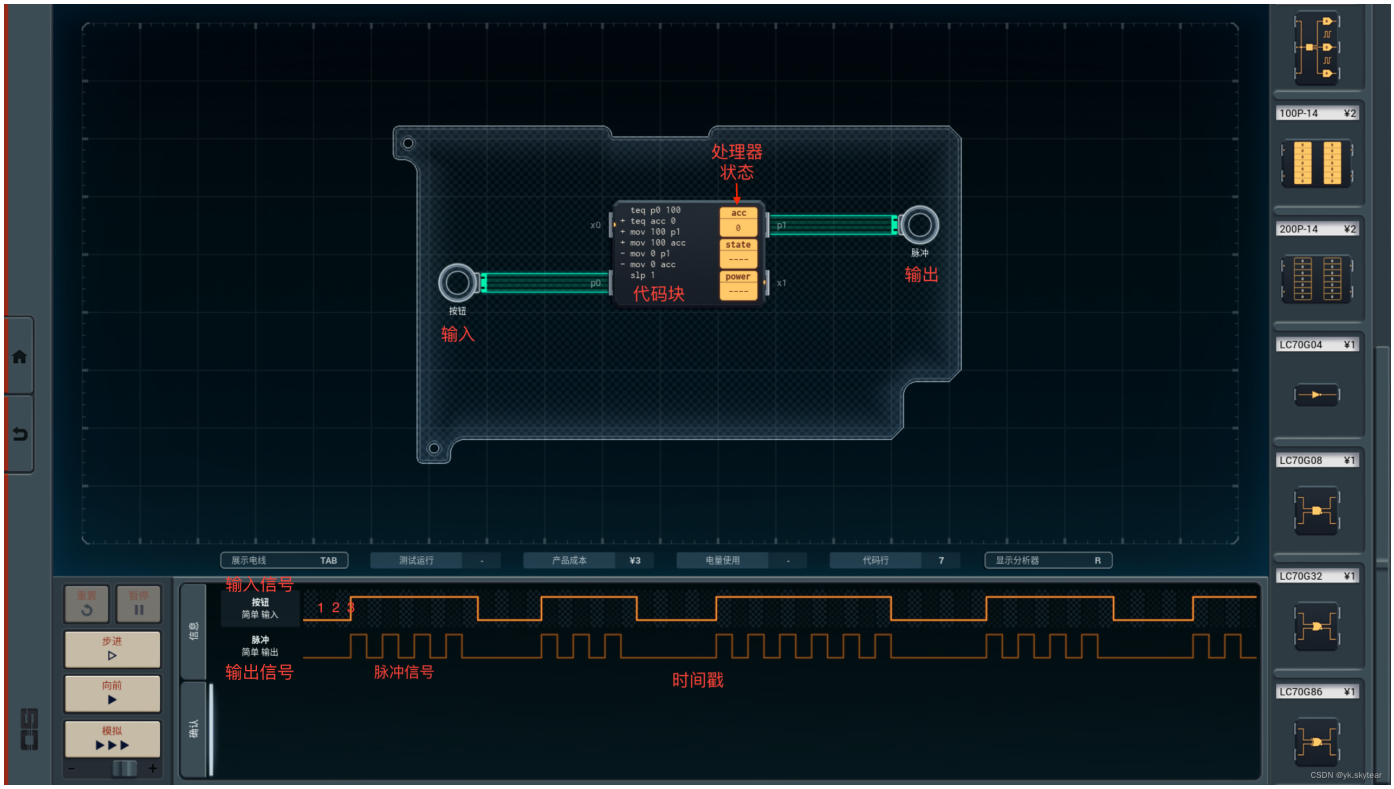
在本题中你可能会用到`pair`来表示一个点，下面是`pair`的一些基本的初始化方法，详细可以参考C++中pair的用法。

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

3. ASM-I/O模拟器

背景：SHENZHEN I/O

SHENZHEN I/O是一款编程游戏，玩家扮演一名移居中国深圳的电气工程师角色，在游戏中构建电路，并编写程序来运行电路，以完成游戏中客户提出的任务。下图展示了一个游戏前期关卡中设计的一个根据输入信号来输出一定规律的脉冲信号的电路。

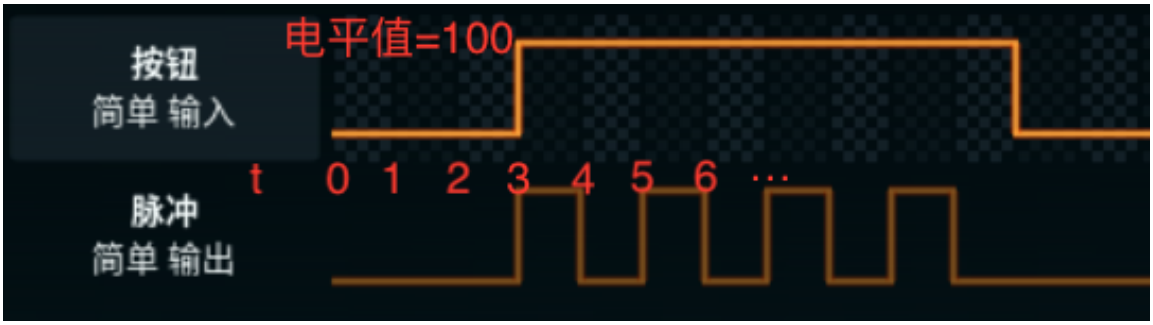


具体来说，输入信号是一段随着时间变化的电平波形，比如上图中的输入信号在第0-2周期为0电平，在第3个周期从0电平变为了100电平，以此类推。

游戏玩法：玩家在给定的处理器（CPU）上编写汇编程序（Assembly），处理器会读取输入信号并根据玩家的程序将输入信号进行处理并发出输出信号。玩家的任务是保证程序能根据输入信号和具体的任务发出正确的输出信号，上图中玩家需要在输入电平变为100的时候输出脉冲信号（即一个周期为高电平，下一个周期为低电平），当玩家设计的电路的输出信号与要求的输出信号一致时，玩家即可通关。

题目描述

在本题中，你的任务是设计一个ASM-I/O模拟器去模拟游戏中处理器，具体任务是解析所提供的汇编程序，根据汇编指令模拟处理器的运行，并发出输出信号，我们会根据模拟器的输出信号来判断你的程序是否正确。



比如说上图中的输入信号我们用下面的电平值序列来表示，第`t`个值表示第`t`个周期的输入信号，这里输入信号有12个周期

```
0 0 0 100 100 100 100 100 100 100 100 0 0
```

我们的模拟器需要根据汇编程序来输出如下所示的脉冲信号，长度也是12个周期

```
0 0 0 100 0 100 0 100 0 100 0 0
```

处理器模型：栈机器

我们要实现的处理器的核心计算部件是大家学过的`Stack`，基于栈的机器称为**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。

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
- Program: 该栈机器的程序，其中程序用下面的语言编写

执行语言

我们提供了一个简单的汇编语言（Assembly Language）指令集，它是一种更接近计算机底层的语言，也可以称为指令，可以分为睡眠指令，计算指令，控制指令和信号输入输出指令四种。

睡眠指令：

- `Sleep n`：处理器睡眠`n`个周期，这`n`个周期内输出信号保持不变

计算指令：

- `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
- `Const n`: 将整数`n`放入栈顶

控制指令：

- `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

信号输入输出指令：

- `Input`: 将当前周期的输入信号的电平值（类型为`int`）放入栈顶
- `Output x`： 将当前周期的输出信号的电平值设置为`x`的值，这里`x`可以是一个`int`类型的常数，也可以是一个变量，变量的值为state中`x`所映射到的值

程序执行

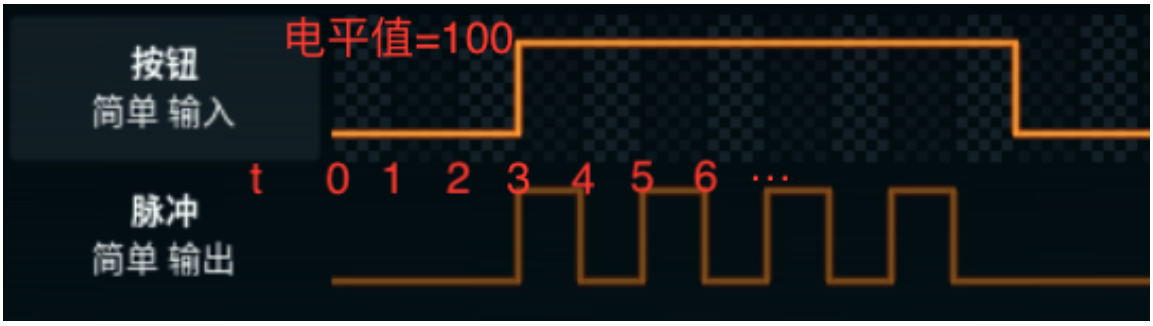
从pc=0开始，程序每一步会读取当前`pc`所指向的指令，并执行指令，每执行一条非跳转（不是`Jmp`类型）指令，`pc=pc+1`。若遇到`Jmp n, JmpEq n, JmpGt n, JmpLt n`这类的指令，则根据执行结果选择是否跳转到`pc=n`处，如果不跳转，则`pc=pc+1`。更新`pc`后重复这一过程，当执行完最后一条指令时，`pc`会设置为0，重新运行程序。程序的执行伴随着**stack**和**state**的变化，在你实现程序中，应该使用两个**ADT**来分别表示它们。

模拟器需要模拟时序，从`t=0`时间戳（或周期）开始，在每遇到`Sleep n`指令时（保证n大于0），更新为`t=t+n`，表示过了`n`个周期，在这`n`个周期内的输出信号为最近一次的`Output`指令提供的电平值。**注意**我们假设除`Sleep`以外的指令都是**瞬间完成**的，不会消耗周期。

一开始输出信号的电平默认为0，模拟器需要记录每个周期输出信号的电平值，并在模拟结束后打印，我们通过比较你的输出信号与正确的输出信号来判断模拟器是否正确。

示例程序1：脉冲信号

我们以前面所提到的根据输入信号来发出脉冲信号的例子来解释模拟器的运行，下图展示了输入信号和正确的输出信号，并标出时间戳



我们可以写一个下面这样的汇编程序来输出脉冲信号，为了方便理解，我们标出了每条指令的位置以及用注释来解释指令的作用，测试的程序中不会有这些额外输入

```
0: Const 100 // 将100推入栈
1: Input // 将当前周期的输入信号推入栈
2: JmpEq 5 // 如果输入电平为100，则跳到pc=5
3: Sleep 1 // 维持当前输出电平睡眠一个周期
4: Jmp 0 // 回到程序开始
5: Output 100 // 将输出电平更新为100
6: Sleep 1
7: Output 0 // 将输出电平更新为0，实现脉冲信号
8: Sleep 1 // 执行完这条指令后会回到程序开始
```

根据上图，如果我们的输入信号为如下的电平序列（t从0到12），每个时间戳的电平值通过空格分隔

```
0 0 0 100 100 100 100 100 100 100 100 0 0
```

那么输出信号为同样长度的电平序列

```
0 0 0 100 0 100 0 100 0 100 0 0 0
```

这里需要解释的是在第四个周期（t=3）的时候，第二行指令会执行然后跳转到pc=5，然后输出电平更新为100，之后睡眠一个周期，与图中所示一致。

示例程序2：factorial

我们可以用这个汇编语言来实现如下的计算阶乘的程序，在每个周期输出当前计算到的结果：

```
0: Input // 计算输入信号电平值的阶乘值
1: Assign z // 将输入电平赋值到z变量
2: Const 1
3: Assign y
4: Var z
5: Const 0
6: JmpEq 18 // 如果z==0，即计算结束，那么跳转到pc=18
7: Var y
8: Var z
9: Mul
10: Assign y // y = y*z
11: Output y // 将输出电平设置为y的值
12: Sleep 1
13: Var z
14: Const 1
15: Sub
16: Assign z // z = z-1
17: Jmp 4
18: Output 0 // 重置输出电平
19: Sleep 1
```

如果输入电平恒定为10，持续15个周期

```
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

那么输出电平的前10个周期为每次阶乘计算的中间结果（其中最后两个周期的结果为10的阶乘值），阶乘计算结束后电平重置为0并睡眠一个周期，之后周期性地重复这个输出

```
10 90 720 5040 30240 151200 604800 1814400 3628800 3628800 0 10 90 720 5040
```

该程序可以对应到如下的C++代码，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
while (cin >> z) { // 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
        cout << y << " "; // pc from 11 to 12
        z = z - 1; // pc from 13 to 16
  } // pc 17
  cout << 0 << " "; // pc from 18 to 19
}
```

输入输出格式

第一行`n`表示命令数量，`t`表示输入信号会有`t`个周期。接下来n行输入n条命令，最后一行输入为输入信号，有`t`个电平值，每个电平值通过空格分隔。 我们保证输入都是合法的，也不会出现如下错误：

- 运行时错误，比如除0，出栈时栈空，跳转到程序外，访问不存在的变量等错误
- 时间戳不更新（即没有`Sleep`）导致程序空转
- `Output`的参数既不是常量也不是定义过的变量

输入格式

```
n t
command 0
command 1
...
command n-1
n1 n2 n3 ... nt
```

输出

输出为`t`个周期的输出信号，每个电平值通过空格分隔

```
n1 n2 n3 ... nt
```

数据范围

对于100%的数据，`0<n, t<=100`

提示

- 关于指令的读取与存储，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?
- 判断一个`key`是否在`map`中可以使用`count`方法，比如想要判断`x`是否在`state`里面，可以用如下代码

```
if (state.count("x") != 0){
  // "x" is in state
}
```

- 为了方便理解，我们提供的5个测试样例的功能如下：
  - `1.in/out`：示例程序1 脉冲信号
  - `2.in/out`：示例程序2 计算阶乘
  - `3.in/out`：计时器，每个周期输出当前的周期
  - `4.in/out`：每2个周期将输入信号减半
  - `5.in/out`：欧几里德算法求解最大公约数

提交格式

每道题目的目录下面包含着`utilities.h`方便你对`ADT`进行调试，在提交的时候如果你使用到了`utilities.h`记得将其包含在文件目录下，你提交的文件结构应该类似如下形式：

```
<your student number>.zip
|- 1_browser
|   |- main.cpp
|   |- utilities.h
|
|- 2_maze
|   |- main.cpp
|   |- utilities.h
|
|- 3_asm
|   |- main.cpp
|   |- utilities.h
```

---

1. https://en.wikipedia.org/wiki/Stack_machine ↩
2. https://en.wikipedia.org/wiki/Java_virtual_machine ↩
3. https://en.wikipedia.org/wiki/CPython ↩