Interactive Brainfuck

Brainfuck 是一种只由 8 种符号构成的，极小化，深奥 (esoteric) 的程序语言，由 Urban Müller 在 1993 年创造。Müller 的目标是创建一种简单的、可以用最小的编译器来实现的、符合图灵完全思想的编程语言。Brainfuck 这个名字已经暗示了它的代码很难读懂。在这次作业中，你将一步步编写一个能运行任意 Brainfuck 代码的程序，或者说，一个解释器。

程序 = 状态机？

“状态机”是一种表示有限个状态以及在这些状态之间的转移和动作等行为的数学计算模型。这个概念远远没有听起来那么抽象：你的电脑就可以是一个具有“开机”“关机”两个状态，可以通过开机按钮来在这两个状态之间转移的状态机；你也可以是一个具有“睡觉”、“吃饭”、“学习”、“娱乐”等状态，可以根据时间、心情等条件在这些状态间转移的状态机。

根据这种思想，任何计算机程序也能被看作是一个状态机：状态是计算机的内存（中的所有变量，这些变量变化的值就对应着不同的状态），而执行任意一条简单语句（任意一行最简单的代码）都会转移到一个新的状态。想象一下，你的 `main` 函数的第一行代码决定了程序的初始状态，之后执行的每一句代码都在转移到下一个状态，直到这个程序运行结束。

Brainfuck 就是在模拟状态机

根据这个思想，如果能模拟出一个进行运算过程的状态机（图灵机），就相当于具有了模拟出任何计算机程序的能力。人用纸和笔就能进行任何简单的运算：

- 在纸上写上或擦除某个符号；
- 把注意力从纸的一处移动到另一处；

而 Brainfuck 语言所做的也正是如此：

Brainfuck 为每个程序创建了一个长度为 30000 字节，已初始化为零的数组。此外，它还具有一个初始指向数组第一个字节（即下标为 0 的元素）的指针（假设它叫 `ptr`）。它的 8 种运算符分别对应以下 8 种操作：

运算符	含义	C语言的等价表达
`>`	指针前进到下一格（自增1）	`ptr++`
`<`	指针回退到上一格（自减1）	`ptr--`
`+`	指针所指字节的值加一	`(*ptr)++`
`-`	指针所指字节的值减一	`(*ptr)--`
`.`	输出所指字节的内容	`putchar(*ptr)`
`,`	向指针所指的字节输入内容	`*ptr = getchar()`
`[`	**(Part B内容)** 若指针所指字节的值为零，则向后跳转，跳转到其对应的 `]` 的下一个指令处	`while (*ptr != 0) {`

运算符	含义	C语言的等价表达
`]`	**(Part B内容)** 若指针所指字节的值不为零，则向前跳转，跳转到其对应的 `[` 的下一个指令处	`}`

Brainfuck 这种极端简化的深奥语言 (esolang) 被称为图灵焦油坑 (Turing Tarpit)。虽然理论上可以写出任何程序（图灵完备），但是即使只是写简单的程序也不现实。例如下面是 Brainfuck 的“Hello world”（`tests/A/hello_world.bf`）：

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++.>
++++++++++++++++++++++++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++.>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++.>
++++++++++.>
```

我们的代码框架，以及，你需要做什么？

我们为你提供了两个文件：`ibf.h` 与 `ibf.c`。所有你需要理解的部分和需要完成的工作都在 `ibf.c` 文件中。头文件 `ibf.h` 中包含了我们提供的魔法 (magic!)，如交互输入输出、命令行参数读取等，尝试阅读或读懂这个文件都不是必要的。

我们已经为你设计好了代码的框架：用一个结构体 `struct brainfuck_state` 来储存上述的 Brainfuck 状态模型：里面包括一个字节数组 `memory_buffer` 和一个储存当前指针位置的数值 `memory_pointer_offset`。这个简单的结构体足以完成 part A 中的所有工作：实现除了 `[` 与 `]` 两个循环运算符以外的其余六种运算符操作。也就意味着，这时你就可以运行任何不包含 `[` 与 `]` 的 Brainfuck 程序！

简单地说，在本次作业中，你需要填写我们为你在 `ibf.c` 文件中留出的几个函数，实现相应的功能。在实现的过程中，你可能会需要定义新的函数或为结构体添加新的成员，你都可以自由地去做。

Part A

在这一部分里，你需要实现：

- Brainfuck 状态 `struct brainfuck_state` 的创建与删除
  - `brainfuck_state_new()`
  - `brainfuck_state_free()`
- 除 `[` 与 `]` 两个循环运算符以外的六种运算符
  - `brainfuck_execute_plus()`
  - `brainfuck_execute_minus()`
  - `brainfuck_execute_previous()`
  - `brainfuck_execute_next()`
  - `brainfuck_execute_input()`
  - `brainfuck_execute_output()`
- 执行一段仅含这六种运算符的 Brainfuck 程序
  - `brainfuck_main()`

关于这些函数的具体说明及注意事项，请阅读代码中对应函数前的注释。

如何使用

我们为你提供的代码是一个交互式的程序，也就是它的名字（Interactive BrainFuck, IBF）的由来。你可以如同在命令行使用 `python` 指令交互式运行 Python 一样交互式地运行 Brainfuck 代码。我们提供了样例程序，在终端中进入当前目录并输入 `.\ibf_std` (Windows) 或 `./ibf_std` (MacOS/Linux) 即可运行。建议你在开始写之前先试用以下以大致了解它在做什么。你也同样可以对比你自己写完的程序与样例程序在相同输入时的表现来检查你的程序中可能的问题。

若你直接运行程序而不传入任何命令行参数（例如，由 VSCode 的 code runner），你将会看到两行说明信息，以及一行以 `>>>` 开头的交互式输入框。

试着输入这个简单的程序：`,+++.`

对照上面的运算符含义，可以看出，这段程序先输入一个字符，再将它的值加 3（ASCII 码向前移 3 位），最后输出。输入这个程序后回车确认，此时程序会等待 `,`（输入运算符）的输入。再向它输入 `a` 你就会看到它给出的输出 `d`。

此时程序的状态仍然保留，你可以通过 `>>>` 交互式输入框输入代码并继续运行。试着再输入 `....`，将当前字节打印 4 次，你会得到 `dddd`。

我们还提供了很多（很厉害的）示例程序。IBF 也可以运行文件中的 Brainfuck 程序。你需要在命令行中运行你生成的可执行文件，为它提供 Brainfuck 程序文件的路径作为参数。

- Windows下，指令将类似 `.\ibf .\tests\hello_world.bf`。
- MacOS/Linux下，指令将类似 `./ibf ./tests/hello_world.bf`。

相同主名的文件为一组示例。程序、输入（部分程序无）、输出的扩展名分别为 `.bf`，`.in` 和 `.out`。

当提交至 OJ 时，你只需要提交 `ibf.c` 文件中的代码。

Part B

在这一部分里，你将需要实现最后两个运算符 `[` 与 `]`。

只靠一个字节数组与一个指针是做不到运行循环的。（想想为什么？）因此，你需要给结构体添加额外的成员以实现 `[` 与 `]` 两个循环运算符。

那么，需要添加哪些额外成员，用来做什么呢？我们鼓励你思考写出你自己的方案，但你也可以直接采用我们为你提供的一种方案。它被以注释的形式藏在了代码里，解除注释即可使用。我们添加的额外成员有：

- `char* loop_buffer;` 一个可以存放循环体的字符串。
- `size_t loop_buffer_size;` 一个可以保存循环体长度的变量。
- `size_t unmatched_depth;` 一个可以保存当前未匹配的嵌套循环层数的变量。

基于你的实现，你可能不会用到全部这些新成员，也可能会需要自己定义其他的成员变量或为需要的功能添加新的函数。这都是你的自由。

Part B 注意事项

- 循环可能嵌套，请确保你能够处理这样的情况。
- 每个 `[` 与 `]` 判断是否为零的字节与当前指针所指位置有关。
- 我们已经帮你处理了括号不匹配时的报错情况。换言之，你可以相信 `brainfuck_main()` 需要运行的代码中，括号是匹配的。
    - 有意思的小测试：当你运行交互式终端形式的 ibf 时，若你输入的一行代码并不完整而有未被匹配的 `[`，此时代码并不会被运行，而是等待你继续向终端中输入代码，直到括号匹配。这与命令行运行 Python 时可以将循环体分行输入类似。试着先输入一行 `,>++++[`，再输入 `<.>-]`，最后给里面的逗号 `,` 输入一个字符看看它会怎样运行！