Assignment1

ID	2019K8009929039
name	周鹏宇

1. **编译器相对于解释器的优点是什么？解释器相对于编译器的优点是什么？**
   - 由编译器产生的机器语言目标程序通常比一个解释器要快很多，但其生成的目标程序（可执行程序）往往依赖于所在的系统和体系结构，比如Windows系统下的.exe格式的可执行文件就不能在Mac或者Linux上使用，尽管它们都是类Unix系统；
   - 解释器的错误诊断效果通常要比编译器更好，因为它逐个语句执行源程序，而且可移植性更好一些
2. **在一个语言处理系统中，编译器产生汇编语言而不是机器语言的好处是什么？**

   汇编语言比较容易输出和调试，例如

   ```
   000000000000 00000 000 00000 1110011 //ECALL
   ```

   相较于左侧的机器语言，右侧的汇编指令ECALL显然更有利于人类去理解和阅读
3. **对下图中的块结构的C代码，指出赋给w、x、y和z的值**

(1)

```
int w, x, y, z;
int i = 9; int j = 14;
{
    int j = 3;
    i = 7;
    w = i + j;
}
x = i - j;
{
    int i = j;
    y = i + j;
}
z = i + j;
```

(2)

```
int w, x, y, z;
int i = 7; int j = 6;
{
    int i = 5;
    w = i + j;
}
x = j - i;
{
    int j = 5;
    i = 4;
    y = i + j;
}
z = i + j;
```

- 对于（1），有w = 10（此时全局i为7，全局j为14）x = -7（此时i，j全局值不变）y = 28(此时i，j全局值不变) z = 21（此时i=7，j=14）

  $w = 10;\ x = -7;\ y = 28;\ z = 21$

```
/Users/zhoupengyu/Documents/chrpos/cmake-build-debug/chrpos
w:10  x:-7  y:28  z:21
进程已结束,退出代码0
```

- 对于（2），有w = 11(此时i，j全局值不变) x = -1(此时i，j全局值不变) y = 9(此时i全局值为4) z = 10(此时i = 4，j = 6)

  $w = 11;\ x = -1;\ y = 9;\ z = 10$

```
/Users/zhoupengyu/Documents/chrpos/cmake-build-debug/chrpos
w:11  x:-1  y:9 z:10
进程已结束,退出代码0
```

4. **下面的C代码的打印结果是什么？**

```
#include <stdio.h>
#define a x
int x = 6;

void b(){x = a; printf("%d\n", x);}
void c(){int x = 1; printf("%d\n", a*2);}
void main(){
  b();
  c();
}
//代码经过略微修改
```

   打印结果为：

```
zhoupengyu@zhoupengyudeMacBook-Pro  ~/Documents/2022春季学期资料  ./homo1
6
2
```

5. **有人把程序设计语言分为编译型和解释型两类，例如C是编译型，Python是解释型。这个分类是否合理?能否构建C语言的解释器，或者Python的静态编译器?谈谈你的看法。**

   - 关于编译型语言和解释型语言的定义：
     - 编译型语言：开发完成以后需要将所有的源代码都转换成可执行程序
     - 解释型语言：每次执行程序都需要一边转换一边执行，用到哪些源代码就将哪些源代码转换成机器码，用不到的不进行任何处理。

- 我认为这个分类是在长期的编程实践过程中人们约定俗成的习惯叫法，至少对于C语言而言，其可以归于解释器的定义的
- 关于构建C的解释器或者Python的静态编译器
    - 首先对于C的解释器，这个是已经有大量实践的，甚至可以在500-600行的C代码内写出一个简单的解释器；
    - 然后是Python的静态编译器，由于Python语言的动态性太强，虽说有部分编译器（LLVM后端的numba）能够编译Python语法的**子集**，但对整个Python进行静态编译似乎可行性不大