$Chapter\ 1 - HW01$

2015K8009929049 冯吕

2018 年 7 月 9 日

1.1.2 解：

- 编译器相对解释器的优点：编译器把源代码编译成目标代码，生成的程序执行时不再需要编译器。由编译器编译产生的目标程序通常运行速度比解释器要快很多。但由编译器产生的机器代码依赖于体系结构，移植到其他平台需要重新编译源程序；
- 解释器相对编译器的优点：解释器逐条语句的执行源程序，执行过程清晰直观，因此解释器的错误诊断效果通常比编译器要好。另外，依赖解释执行的程序可移植性一般更好，移植到其他平台时可直接执行或修改较少代码。

1.1.3 解：汇编指令是机器指令的助记符，通常情况下，一条汇编指令就对应一条机器指令。生成汇编指令比直接生成机器语言会更方便进行调试、优化和输出。

1.6.1 解：

```
int w, x, y, Z;
int i = 4; int j = 5;
{
  int j = 7;
  i = 6;
  w = i + j;
}
x = i + j;
{
  int i = 8;
  y = i + j;
}
z = i + j;
```

- 在第二行声明了变量 i, j，它们具有全局作用域。而在第一个大括号内，重新声明了一个变量 j，因此，覆盖了前面声明的变量 j 的作用域，而 i 没有重新声明，所以，和前面的是同一个变量，只是重新赋值为了 6，所以 $w = i + j = 6 + 7 = 13$；
- 离开第一个大括号之后，大括号内声明的 j 消失，所以 $x = i + j = 6 + 5 = 11$；
- 进入第二个大括号之后，同理，重新声明的 i 覆盖了外面的 i 的作用域，所以 $y = i + j = 8 + 5 = 13$；
- 离开大括号后，大括号内的 i 消失，$z = i + j = 6 + 5 = 11$。

1.6.2 解：和上题同样分析，可知：

- $w = i + j = 5 + 4 = 9$;
- $x = i + j = 3 + 4 = 7$;
- $y = i + j = 7 + 6 = 13$;
- $z = i + j = 7 + 4 = 11$;

1.6.4 解:

```
#define a (x+1)
int x = 2;
void b() {
  x = a;
  printf ( "%d\n", x );
  }
  void c() {
        int x = 1;
        printf ( "%d", a );
        }
        void main(){
          b();
          c():
        }
```

在函数外面，定义了一个宏，声明了一个 *int* 型变量 x 并初始化为 2，所以，该变量具有全局作用域。在函数 b 中，没有重新声明 x，因此 $x = a = (x + 1) = 2 + 1 = 3$；而在函数 c 内，重新声明了一个局部变量 x 并初始化为 1，因此 $a = (x + 1) = 1 + 1 = 2$。所以，打印结果为 $3, 2$。