一. 函数复习

1. 函数的基本概念：

①函数原型：告诉编译器相关函数的**签名。**

②函数调用：表明在这执行函数。

③函数定义：明确地指定了函数要做什么。

2. 函数签名（signature）：**函数的返回值和函数的参数被称为函数的签名。**

3. 函数原型的两种使用方法：①用逗号指明参数的类型和数量：void f（int a, int b）；（注意这里a，b是假名，可以被替换）②不创建变量void f(int, int);C99/11标准；

4. 函数定义的一些要点（主要是函数头）：①每个变量必须单独声明，同类型的不能像变量声明一样一起声明。②formal paramters之间用逗号分隔。

5. 返回值的使用：①使用return；直接终止函数。②没有返回值的函数类型应该定义为void；

二. 上个世纪留下来的老古董——函数原型

1. 实参和形参之间的关联：主调函数把**实参**根据其类型存在称为**栈**（zhan，第四声）的临时存储区。而形参则通过形式参数的**类型**读取值。

2. 函数原型存在的意义：利用函数原型，我们可以在实参存入栈之前将其进行类型转化，这样一来我们可以避免不少麻烦。

3. 无参数和未指定参数，在ANSI风格的c中，我们使用void f();来表示函数**未指定参数。**而通过void f(void)；来表明函数**没有参数。**（注意与C++区别）

4. 多个参数：很多函数允许有多个参数，如printf()和scanf(), 他们的函数原型（以printf为例）：int printf(const char *, ...); 这第一个char *代表一个字符串，后面的...代表未指定的参数，详见第　17章。

5. 函数原型的优点：可以让编译器捕获在使用函数时可能出现的许多错误或纰漏。

三. 递归

1. 递归的底层实现：每一层的递归都会创建一个新的变量，每个变量都属于本级递归私有。但函数的代码并没有被拷贝，被调用后又得从头开始执行函数的代码。

2. 尾递归：尾递归等价于循环，这个没什么好说的。

3. 递归与循环对比：递归效率上真的时不如循环。①空间上，每次递归都会创建一组变量，每次都把一组新的变量放在栈中（思考：栈是一种临时存放变量组的特殊结构）。②时间上，函数的调用更加耗费时间。

4. 使用递归的一些思考：

①由于位于递归调用之前的语句，均按顺序进行；后面的语句则恰恰相反。我们可以思考什么是可以在递归之前做好的，这个放在递归之前；什么是可以利用逆序的，这个放在递归之后。　　②递归的终止条件：递归的终止条件是参数相关的。

5. 利用递归算法的函数——打印十进制数对应的二进制数

```
void to_binary(unsigned long n){
  int r;
  r = n % 2;
  if (n > =2) to_binary(n / 2);
  putchar(r == 0 ? '0':'1');
  return;
  }
```

四. 同时编译多个源代码文件

1. 现在我有两个.c文件，请问如何将他们合并？答案是：利用编译器同时编译。**注意编译的同时会生成多个（而不是单个）相应的目标文件，这就告诉我们在每个源文件中应该包含规定的预处理指令**（如#include<stdio.h>），不管源文件之间有没有重复，不然都无法通过编译。

2. 头文件的使用：在c中我们一般将符号常量和函数声明放在头文件中，这样在我们以后修改时，就不需要再每个文件中单独地修改，只要统一地修改即可。

3. 总结一下：做一个项目要用到三个文件：两个源文件：一个包含main（），负责程序的主要逻辑；一个包含所有函数的定义；这两个文件要一起编译。一个头文件：包含符号常量和函数原型，这个头文件要被那两个源文件同时包含。

4. #include< “hotels.h” >中的双引号表明被包含的文件位于当前目录中。

五. &运算符和*运算符

1. 地址概念的引入：编写代码时可以认为变量有两个属性：名称和值；计算机编译和加载程序后，认为变量也有两个属性（地址和值）。***地址就是变量在计算机内部的名称***。很多语言中，地址都归计算机管，对程序员隐藏。

2. 指针的概念：指针用于**储存**变量的地址。

3. 获取变量的地址：使用一元运算符&；（是**普通变量**的**派生量**）

4. 获取指针（或地址常量）的对应的值：使用一元运算符*；（是**指针变量**的**派生量**）

5. 关于栈的拓展：在一个函数中，变量确实存储在栈当中，因为后声明的变量地址反而比先声明的变量地址小；

6. 其他的一些内容：

①在PC机上，地址都是用16进制表示，单位是**字节**。

②从编译器角度看：函数所谓的局部变量就是把变量放在一个比较遥远的地址中了。

注意在C语言中函数都是值传递，而没有引用传递（前提是不刻意传递地址）。

六. 指针简介

1. 定义：指针是一个值为内存地址的变量。

2. 声明指针变量时必须指定指针所指向变量的类型。因为①不同的变量占用不同的存储空间②哪怕所占空间相同（如long和float），存储数字的方式也不同。一些指针操作要求知道操作对象的大小（由此看来，**指针的值不仅仅是开头的地址，还蕴含着所占内存的大小，这就是为什么说指针是一个新的类型**）。

3. 指针的声明：

int * pi; char * pc; float * pf, * pg;

注意*和指针名之间的空格可有可无，但通常在**声明时使用**空格，在**解引用时省略**空格。　　另外不能将指针的声明看作是对解引用指针后变量的声明，这在函数头中可以得到体现。

七. 更改calling function中的变量

1. 问题的提出：大家都知道在C语言中只有变量值的引用，可见按照我们以前的知识在函数中是无法对全局变量进行修改的。不过还好我们现在有了指针这个方法。

2. 关于返回值：called function对calling function所做的贡献只来自于两处：①返回值；②对calling function中变量值的修改。事实上，由于C缺乏对多个函数返回值的支持，**我们可以通过修改全局变量来模拟有多个返回值的函数。**

3. 原理：之前提到过所谓局部变量只不过是换了一个比较偏远的地方存储而已，所以我们可以利用指针的方法来获得并修改相应的局部变量。

使用格式：

①在函数调用的过程中，我们应该传递那些我们想要修改变量的地址（现在你知道为什么scanf要传递地址了吧，如果不传递地址你的变量永远不可能传进来）。

如：interchange(&x, &y);

②函数原型和函数声明中函数头的格式：

如：void interchange(int * x, int * y); ——由此看来，应该把地址当作普通的指针变量来对待　　③在函数体当中：

```
int temp;
temp = *u;
*u = *v;
*v = temp;
```

可见在函数体中使用的就是指针本身了，而它基本上以指针解引用的形式出现。