第 1 章 函数

1.1 函数的概念

函数是 c 语言的功能单位，实现一个功能可以封装一个函数来实现。
定义函数的时候一切以功能为目的，根据功能去定函数的参数和返回值。

1.2 函数的分类

1、从定义角度分类（即函数是谁实现的）
1.库函数 (c 库实现的)
2.自定义函数 （程序员自己实现的函数）
3.系统调用 (操作系统实现的函数)

2、从参数角度分类
1.有参函数
函数有形参，可以是一个，或者多个，参数的类型随便
完全取决于函数的功能

```
int    fun(int a,float b,double c)
{
}

int max(int x,int y)
{
}
```

2.无参函数
函数没有参数,在形参列表的位置写个 void 或什么都不写

```
int fun(void)
{
}

int fun()
{
}
```

3、从返回值角度分类
(1).带返回值的函数
在定义函数的时候，必须带着返回值类型，在函数体里，必须有 return
如果没有返回值类型，默认返回整型。
例 1 :

```
char fun()//定义了一个返回字符数据的函数
{
    char   b='a';
    return b;
}
```

例 2：

```
fun()
{
    return 1;
}
```

如果把函数的返回值类型省略了，默认返回整型

注：在定义函数的时候，函数的返回值类型，到底是什么类型的，取决于函数的功能。

(2).没返回值的函数

在定义函数的时候，函数名字前面加 void

```
void fun(形参表)
{
    ;
    ;
    return ;
    ;
}
```

在函数里不需要 return

如果想结束函数，返回到被调用的地方， return ;什么都不返回就可以了

例 3：

```
#include <stdio.h>
int max(int x,int y)
{
    int z;
    if(x>y)
        z=x;
    else
        z=y;
    return z;
}
void help(void)
{
    printf("**********\n");
```

```
        printf("********帮助信息*****\n");
        printf("**********************\n");
    }
    int main(int argc, char *argv[])
    {
        int num;
        help();
        num = max(10,10+5);
        printf("num=%d\n",num);
        return 0;
     }
```

1.3 函数的定义

什么叫做函数的定义呢？即函数的实现

1、函数的定义方法

返回值类型　函数名字(形参列表)

{

//函数体，函数的功能在函数体里实现

}

例 4：

```
int max(int x, int y)
{
    int z;
    if(x>y)
        z=x;
    else
        z=y;
    return z;
}
```

注：形参必须带类型，而且以逗号分隔

函数的定义不能嵌套，即不能在一个函数体内定义另外一个函数，

所有的函数的定义是平行的。

例 5：

```
void fun(void)
{
    ;
    ;
```

```
        ;
void fun2(void)
        {
            ;
        }
}
```

这个程序是错误的，不能再 fun 的函数体中，定义 fun2 函数。

例 6：

```
 void fun(void)
{
    ;
    ;

    ;
}
void fun2(void)
{
    ;
}
```

这个程序是正确的，fun 和 fun2 是平行结构

注：在一个程序中，函数只能定义一次

给函数起名字的时候，尽量的见名知意，符合 c 语言的命名规则

1.4 函数的声明

1、概念

对已经定义的函数，进行说明

函数的声明可以声明多次。

2、为什么要声明

有些情况下，如果不对函数进行声明，编译器在编译的时候，可能不认识这个函数，

因为编译器在编译 c 程序的时候，从上往下编译的。

3、声明的方法

什么时候需要声明

1）主调函数和被调函数在同一个.c 文件中的时候

1] 被调函数在上，主调函数在下

例 7：

```
void fun(void)
```

```
{
    printf("hello world\n");
}
int main()
{
    fun();
}
```

这种情况下不需要声明

2] 被调函数在下，主调函数在上

例 8：

```
int main()
{
    fun();
}
void fun(void)
{
    printf("hello world\n");
}
```

编译器从上往下编译，在 main 函数（主调函数），不认识 fun，需要声明

怎么声明 呢？

1] 直接声明法

将被调用的函数的第一行拷贝过去，后面加分号

例 9：

```
void fun(void);
int main()
{
    fun();
}
void fun(void)
{
    printf("hello world\n");
}
```

2] 间接声明法

将函数的声明放在头文件中，.c 程序包含头文件即可

例 10：

a.c

```
#include" a.h"
int main()
```

```
{
    fun();
}
void fun(void)
{
    printf("hello world\n");
}

a.h
    extern void fun(void);
```

2）主调函数和被调函数不在同一个.c 文件中的时候

一定要声明

声明的方法：

直接声明法

将被调用的函数的第一行拷贝过去，后面加分号，前面加 extern

间接声明法

将函数的声明放在头文件中，.c 程序包含头文件即可

1.5 函数的调用

函数的调用方法

变量= 函数名(实参列表);//带返回值的

函数名(实参列表);//不带返回值的

1、有无返回值

1).有返回值的，根据返回值的类型，需要在主调函数中定义一个对应类型的变量，接返回值

```
例 11：
int   max(int x,int y)// x、y 形参，是个变量
{

}
int main()
{
    int num;//需要定义一个 num 接收 max 函数的返回值
    num=max(4,8);//4 和 8 就是实参
}
```

2).没有返回值的函数，不需要接收返回值。

```
例 12：
void fun(void)
{
    printf("hello world\n");
}

int main()
{
    fun();
}
```

2、有无形参

函数名(实参列表);//带形参的

函数名();//没有形参的

注意：实参，可以常量，可以是变量，或者是表达式

形参是变量，是被调函数的局部变量。

1.6 函数总结

在定义函数的时候，关于函数的参数和返回值是什么情况，完全取决于函数的功能。

使用函数的好处？

1、定义一次，可以多次调用，减少代码的冗余度。

2、使咱们代码，模块化更好，方便调试程序，而且阅读方便

1.7 变量的存储类别

1.7.1 内存的分区：

1、内存：物理内存、虚拟内存

物理内存：实实在在存在的存储设备

虚拟内存：操作系统虚拟出来的内存。

操作系统会在物理内存和虚拟内存之间做映射。

在 32 位系统下，每个进程的寻址范围是 4G,0x00 00 00 00 ~0xff ff ff ff

在写应用程序的，咱们看到的都是虚拟地址。

2、在运行程序的时候，操作系统会将 虚拟内存进行分区。

1).堆

在动态申请内存的时候，在堆里开辟内存。

2).栈

主要存放局部变量。

3).静态全局区

1：未初始化的静态全局区

静态变量（定义变量的时候，前面加 static 修饰），或全局变量 ，没有初始化的，存在此区

2：初始化的静态全局区

全局变量、静态变量，赋过初值的，存放在此区

4).代码区

存放咱们的程序代码

5).文字常量区

存放常量的。

1.7.2 普通的全局变量

概念：

在函数外部定义的变量

```
int num=100;//num 就是一个全局变量
int main()
{
    return 0;
}
```

作用范围：

全局变量的作用范围，是程序的所有地方。

只不过用之前需要声明。声明方法 extern int num;

注意声明的时候，不要赋值。

生命周期：

程序运行的整个过程，一直存在，直到程序结束。

注意：定义普通的全局变量的时候，如果不赋初值，它的值默认为 0

1.7.3 静态全局变量 static

概念：

定义全局变量的时候，前面用 static 修饰。

```
static int num=100;//num 就是一个静态全局变量
int main()
{
    return 0;
```

```
}
```

作用范围：

static 限定了静态全局变量的作用范围

只能在它定义的.c（源文件）中有效

生命周期：

在程序的整个运行过程中，一直存在。

注意：定义静态全局变量的时候，如果不赋初值，它的值默认为 0

1.7.4 普通的局部变量

概念：

在函数内部定义的，或者复合语句中定义的变量

```
int main()
{
    int num;//局部变量

    {
        int a;//局部变量
    }
}
```

作用范围：

在函数中定义的变量，在函数中有效

在复合语句中定义的，在复合语句中有效。

生命周期：

在函数调用之前，局部变量不占用空间，调用函数的时候，

才为局部变量开辟空间，函数结束了，局部变量就释放了。

在复合语句中定义的亦如此。

```
#include<stdio.h>
void fun()
{
    int num=3;
    num++;
    printf("num=%d\n",num);
}
int main()
```

```
{
    fun();
    fun();
    fun();
    return 0;
}
```

1.7.5 静态的局部变量

概念：

定义局部变量的时候，前面加 static 修饰

作用范围：

在它定义的函数或复合语句中有效。

生命周期：

第一次调用函数的时候，开辟空间赋值，函数结束后，不释放，
以后再调用函数的时候，就不再为其开辟空间，也不赋初值，
用的是以前的那个变量。

```
void fun()
{
    static int num=3;
    num++;
    printf("num=%d\n",num);
}
int main()
{
    fun();
    fun();
    fun();
}
```

注意：

1：

定义普通局部变量，如果不赋初值，它的值是随机的。
定义静态局部变量，如果不赋初值，它的值是 0

2：普通全局变量，和静态全局变量如果不赋初值，它的值为 0

1.7.6 外部函数

咱们定义的普通函数，都是外部函数。
即函数可以在程序的任何一个文件中调用。

1.7.7 内部函数

在定义函数的时候，返回值类型前面加 static 修饰。这样的函数被称为内部函数。

static 限定了函数的作用范围，在定义的.c 中有效。

内部函数和外部函数的区别：

外部函数，在所有地方都可以调用，

内部函数，只能在所定义的.c 中的函数调用。

扩展：

在同一作用范围内，不允许变量重名。

作用范围不同的可以重名。

局部范围内，重名的全局变量不起作用。（就近原则）

```
int num = 100; //全局变量
int main()
{
    //如果出现可以重名的情况，使用的时候满足向上就近原则
    int num = 999; //局部变量

    return 0;
}
```