一、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。

函数就是讲特定功能的代码封装在一个函数内部，当要使用这些代码时，只需要通过函数名就可以使用，这样操作起来更加方便

二、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：

```c
#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;
```

```
}
```

三、函数的定义

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

}

注意：

函数名字是标识符，所以需要满足标识符的命名规则

形参：可以有，也可以没有，也可以有多个，但是即使没有，函数名字后面也必须加括号

函数体上下位置必须有大括号

如果要返回函数执行的结果，也就是返回值，则return后面跟的变量或者值，必须与函数名左边的返回值类型一致

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。

在一个程序中，相同的函数名只能出现一次

```c
//定义一个没有参数也没有返回值的函数
void myfun1()
{
 printf("hello world\n");
 printf("nihao beijing\n");
 printf("welcome to 1000phone\n");

 return ;
}

//定义一个有参数的函数
void myfun2(int a, int b)
{
 int sum;
 sum = a + b;

 printf("%d + %d = %d\n", a, b, sum);
```

```
}

//定义一个有返回值的函数
int myfun3(int a, int b)
{
 int sum;
 sum = a + b;

 return sum;
}
```

四、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

```
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

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

```
5  void fun(void)
6  {
7  printf("hello world\n");
8  }
```

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（常用）

将被调用的函数的第一行拷贝过去，后面加分号

```
1  #include <stdio.h>
2
3  //函数的声明：一般当子函数在主函数的下方时，需要在主函数的上方进行声明
4  void myfun1();
5  void myfun2(int a, int b);
6  int myfun3(int a, int b);
7  int main(int argc, char *argv[])
8  {
9   myfun1();
10   return 0;
11  }
12
13  void myfun1()
14  {
15   printf("hello world\n");
16   printf("nihao beijing\n");
17   printf("welcome to 1000phone\n");
18
19   return ;
20  }
21
22  void myfun2(int a, int b)
23  {
24   int sum;
25   sum = a + b;
26
27   printf("%d + %d = %d\n", a, b, sum);
28  }
29
30  int myfun3(int a, int b)
31  {
32   int sum;
```

```
 sum = a + b;

 return sum;
}
```

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

```
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（常用）

将函数的声明放在头文件中，.c程序包含头文件即可

```
myfun.c
#include "myfun.h" //此时包含的头文件要使用双引号，在当前目录下去找对应头文件

void myfun1()
{
 printf("hello world\n");
 printf("nihao beijing\n");
 printf("welcome to 1000phone\n");

 return ;
```

```
}

myfun.h
#ifndef MYFUN_H
#define MYFUN_H

//函数的声明
void myfun1();

#endif // MYFUN_H

main.c
#include <stdio.h>
#include "myfun.h"

int main(int argc, char *argv[])
{
 myfun1();

 return 0;
}
```

五、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

```
#include <stdio.h>

void myfun1();
void myfun2(int a, int b);
int myfun3(int a, int b);
int main(int argc, char *argv[])
{
 //函数的调用
 //没有参数也没有返回值
  //直接写函数名，并且要在后面加括号
 myfun1();

 printf("**********************\n");
```

```c

 //有参数，没有返回值
 //需要在函数名右边括号中传入参数，参数可以是常量表达式，也可以是变量表达式
 myfun2(100, 90);

 int x = 10, y = 20;
 //x、y：实参，实际参数，本质就是在函数调用的时候将实参的值传递给形参
 myfun2(x, y);

 printf("**********************\n");

 //有参数也有返回值
 //可以使用一个变量接收函数执行结果（返回值），或者直接输出也可以
 int n;
 n = myfun3(100, 90);
 printf("n = %d\n", n);

 printf("sum = %d\n", myfun3(90, 66));

 return 0;
}

void myfun1()
{
 printf("hello world\n");
 printf("nihao beijing\n");
 printf("welcome to 1000phone\n");

 return ;
}

//a、b：形参，形式参数，主要用于保存实参传递的值，本质跟实参没有任何关系，只是值传递
void myfun2(int a, int b)
{
 int sum;
 sum = a + b;

 printf("%d + %d = %d\n", a, b, sum);
}

```

```
54 int myfun3(int a, int b)
55 {
56  int sum;
57  sum = a + b;
58
59  return sum;
60 }
```

执行结果

```
Starting C:\Users\lzx\Desktop\src\build-04_fu
Debug\debug\04_fun_call.exe...
hello world
nihao beijing
welcome to 1000phone
**********
100 + 90 = 190
10 + 20 = 30
**********
n = 190
sum = 156
C:\Users\lzx\Desktop\src\build-04_fun_call-De
```

六、函数总结

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。

当编写函数的时候，一开始不要想着函数如何传参和函数的返回值应该是什么

而是当在编写代码的途中，要使用某些值，但是当前函数中不存在，此时就需要进行传参，这时候考虑怎么传参就是合适的时机

当函数编写完毕后，考虑是否要将某些结果返回给其他函数去使用，此时需要考虑返回值

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

七、变量的存储类别

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。

在32位操作系统中，虚拟内存被分为两个部分，3G的用户空间和1G内核空间，其中用户空间是当前进程所私有的，内核空间，是一个系统中所有的进程所公有的

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。

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

```c
#include <stdio.h>

//定义一个普通全局变量
//只要是在main函数外（也在子函数外）的变量，就是全局变量
//如果全局变量没有进行初始化，则系统自动将其初始化为0
int num;

//全局变量可以在程序的任意一个位置进行对其的操作
 void myfun()
 {
 num = 888;
 }

int main(int argc, char *argv[])
{
 printf("num = %d\n", num);

 myfun();

 printf("num = %d\n", num);

 return 0;
}
```

执行结果

```
Starting C:\Users\lzx\Desktop\s
Debug\debug\05_var_gloabl.exe..
num = 0
num = 888
C:\Users\lzx\Desktop\src\build-
\05_var_gloabl.exe exited with
```

7.3 静态全局变量

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

```c
#include <stdio.h>

//定义一个静态全局变量
//静态全局变量只能在其定义的.c文件中任意位置使用，不能跨文件使用
static int num;

void myfun()
{
 num++;
}

int main(int argc, char *argv[])
{
 printf("num = %d\n", num);

 myfun();

 printf("num = %d\n", num);

 return 0;
}
```

执行结果

```
Starting C:\Users\lzx\Desktop\src\
Desktop_Qt_5_8_0_MinGW_32bit-Debug
num = 0
num = 1
C:\Users\lzx\Desktop\src\build-06_
Debug\debug\06_var_global_static.e
```

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。

```c
#include <stdio.h>

//定义一个局部变量
//在函数内部定义的，不加任何修饰的变量都是局部变量
void myfun()
{
 int num = 100;
 num++;

 printf("num = %d\n", num);

 return ;
}

int main(int argc, char *argv[])
{
 //局部变量只能在定义的函数内部使用，声明周期相对较短，函数结束，局部变量就会释放
 //printf("num = %d\n", num);
 myfun();
 myfun();
 myfun();

 return 0;
}
```

执行结果

```
Starting C:\Users\lzx\Desktop\
Debug\debug\07_var_local.exe..
num = 101
num = 101
num = 101
C:\Users\lzx\Desktop\src\build
\07_var_local.exe exited with
```

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。

```c
#include <stdio.h>

//定义一个静态局部变量
//在函数内部定义的使用static修饰的变量就是静态局部变量

void myfun()
{
 //如果普通局部变量不进行初始化，则默认是随机值
 //如果静态局部变量不进行初始化，则默认是0
  int a; //普通局部变量
  static int num; //静态局部变量

  printf("a = %d\n", a);
  printf("num = %d\n", num);
}

void myfun1()
{
 //静态局部变量不会随着当前函数执行结束而释放空间，下次使用的函数之前的空间
```

```
20   //静态局部变量只会初始化一次
21   static int num1 = 100;
22   num1++;
23
24   printf("num1 = %d\n", num1);
25  }
26
27  int main(int argc, char *argv[])
28  {
29   myfun();
30
31   myfun1();
32   myfun1();
33   myfun1();
34
35   return 0;
36  }
```

执行结果

```
Desktop_Qt_5_8_0_MinGW_
a = 4200443
num = 0
num1 = 101
num1 = 102
num1 = 103
C:\Users\lzx\Desktop\src
```

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

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。

在分文件编程中，只需要将函数的实现过程写在指定的.c文件中，然后将其声明写在指定的.h文件中，其他文件只要包含了头文件，就可以使用外部函数

7.7 内部函数

内部函数也称之为静态函数，就是用static修饰的函数

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）

```c
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;
}
```