分而治之与信息隐藏：

模块化程序的设计就是体现了这种“分而治之”的思想。就是从头将其分解为各个模块的过程。

信息隐藏是将信息隐藏起来，设计得当的函数可以把不必要的信息隐藏起来。使得程序更加安全和紧凑。

函数的分类：

函数是构成程序的基本模块。

程序的执行从main（）函数入口，中间经过函数的处理。各个函数各司其职。完成各自的功能。

函数分两大类：

标准库函数：

符合ANSIC标准的C语言编译器都要提供这种库函数。

使用尖括号进行包含

第三方库函数：

自定义函数：自己写的函数，调用可以用；

函数的定义处理：

函数的定义：

返回值类型 函数名(类型 形式参数1,类型 形式参数2）函数头

{

声明语句序列

可执行语句序列

}

函数名是函数的唯一标识,用于说明函数的功能。

函数名标识符的命名规则与交量的命名规则相同。

通常变量名用小写字母开头的单词组合而成,函数名则用大写字母开头的单词组合而成。

Windows应用程序的标识符通常采用“大小写”混排的单词组合而成。

函数名使用“动词”或者“动词+名词”(动宾词组)的形式,如函数名GetMax等。

而变量名使用“名词”或者”形容词+名词”的形式,

函数体内部定义的变量只能在函数体内访问,称为内部变量。函数头部参数表里的变量,称为形式,参数(Parameter,简称形参),也是内部变量,即只能在函数体内访问。

在函数定义的前面写上一段注释来描述函数的功能及其形参，是一个非常好的编程习惯。

函数的调用：

名称(参数)；

函数的返回值只能有一个，函数返回值的类型可以是除数组以外的任何的类型。

形参表是函数的人口。如果说函数名相当于说明运算的规则的话,那么形参表里的形参就相当于运算的操作数,而函数的返回值就是运算的结果。

若函数没有函数返回值,则需用void定义返回值的类型。若函数不需要人口参数,则需用void代替函数头部中形参表中的内容,它告诉编译器该函数不接收来自调用程序。的任何数据。

return可以有多个，但不表示函数可以有多个返回值。

函数原型：

long　fact(int a);

函数封装和防御性设计：

将函数的内容不给用户看到，而让用户只关注与函数的功能的过程称为函数的封装。

让函数遇到不正确的输入或者非法的数据的时候仍然可以保证自己不出错的能力，称为健壮性。

在函数中用一些代码来处理异常的技术，称为防御性程序设计。

函数的递归调用和递归函数：

递归函数

1>>>>自己要使用自己

2>>>>一定要有一个结束基线

也就是：

一般情况下要有一个参数，且必须拥有返回值

要有一个基线情况：基线情况下返回值

要有一个或者多个一般情况：一般情况下递归>>>>自己调用自己的情况

如一个循环加法：

```
#include<stdio.h>

int a(int n)
```

```
{
if(n == 1)              //基线，在到达此处时结果开始固定：：基线情况
return 1;               //必须要有返回值
else                    //通常情况下：：一般情况
return (n*a(n-1));      //递归的过程
}

int main()
{
int ac = 4;
ac = a(ac);
printf("%d\n",ac);
return 0;
}
```

每个迭代程序原则的基础上都可以转换成等价的递归程序，反之则不然。
如汉诺塔是一个只有使用递归才可以解决的问题。

C语言 作用域和生命周期

变量的作用域
被花括号括起来的区域叫做语句块。
变量的作用域的规则是：每个变量仅仅在定义它的语句块内有效，并拥有自己的存储空间。
不在任何的语句块内定义的变量，称为全局变量。
语句块内的变量称为局部变量。

全局变量与程序共生死。
局部变量只在指定的时期存在，如函数体内部。

变量的存储类型：

自动变量也称为动态局部变量。
如果没有指定变量的存储类型，那么变量的存储类型就缺省为auto。
如果不希望形参值在函数内被修改，那么只要将关键字const放在形参前面，将形参声明为常量即可。
//如果不希望形参值在函数内被修改，那么只要将关键字const放在形参前面，将形参声明为常量即可。

```
#include <stdio.h>
#include <stdlib.h>
int fun(const int m,int n)
{
int sum = 0;
m = 7;//试图修改m的值，报错，编译不通过
```

```
---编译失败---
D:\Users\wangzewei\Documents\1.c: In function 'fun': -gcc
D:\Users\wangzewei\Documents\1.c: In function 'fun': -gcc
D:\Users\wangzewei\Documents\1.c:7:1: error: assignment of read-only parameter 'm' -gcc
```

```
sum = m+n;
return sum;
}

int main()
{
    int n = 5;
    int m = 3;
    int sum;
    sum = fun(m,n);
    printf("hello,%d\n",sum);
    system("pause");
    return 0;
}
```

变量的存储方式和类型：

auto 自动变量自动变量不被自动初始化，默认值为随机数。自动变量在退出函数后，其分配的内存立刻被释放。

static 静态变量：static局部变量不初始化，默认值为0。离开{}，static局部变量不会释放，只有整个程序结束才释放。static全局变量只能在定义所在的文件中使用此变量（内部链接）。

extern 外部变量保存在静态存储区内，在程序的运行期间分配固定的存储单元，其的生存周期是整个程序的运行期。没有显式初始化的将会默认为0；

register 寄存器变量：是放在CPU里面了。高速程序。现代编译器一般可以自己进行优化，所以这一项不用管。

并列的语句块内部可以定义同名变量，不会相互干扰，因为他们各自占据着不同的内存单元，并且有着不同的作用域。

形参无法实现数据的交换处理。

a.普通局部变量
属于某个{}，在{}外部不能使用此变量，在{}内部是可以使用。执行到普通局部变量定义语句，才会分配空间，离开{}，自动释放。普通局部变量不被自动初始化，默认值为随机数。自动变量在退出函数后，其分配的内存立刻被释放。

b.static局部变量
属于某个{}，在{}外部不能使用此变量，在{}内部是可以使用。在编译阶段就已经分配空间，初始化只能使用常量。static局部变量不初始化，默认值为0。离开{}，static局部变量不会释放，只有整个程序结束才释放。
注意：静态局部变量的作用域属于某个{},但是它的生命周期却是从编译阶段到整个程序结束。

c.普通全局变量
在编译阶段分配空间，只有整个程序结束才释放。普通全局变量只要定义了，任何地方都能使用，使用前需要声明所有的.c文件，只能定义一次普通全局变量，但是可以声明多次（外部链接）。
注意：全局变量的作用域是全局范围，但是在某个文件中使用时，必须先声明。

d.static全局变量
在编译阶段分配空间，只有整个程序结束才释放。static全局变量只能在定义所在的文件中使用此变量（内部链接）。不同的.c文件，可以定义一次static全局变量。

extern函数和static函数的区别（作用域）
extern函数所有文件都是调用，所有文件中只能定义一次extern函数。
static函数只能在定义所在的文件中调用此函数，不同文件，可以定义一次static函数。

模块化程序设计

高内聚，低耦合，保证每个模块的相对独立性。

抽象是处理复杂问题的工具，逐步求精就是一种具体的抽象技术。

自底向上，方法是先编写基础程序段，再进行扩大，补充和升级。

自顶向下是先设计主程序表达问题，在逐步的细化细节的过程。

逐步求精技术可以理解为是一种不断由底向上修正所补充的自顶向下的程序设计方法。

优点如下：1，得到良好结构的程序。2，可以简化程序的正确性策略。

断言

<assert.h>

断言仅用于调试程序，不能作为程序的功能。

使用场景如下：

1，检查程序中的各种假设的正确性。

2，证实或者测试某种不可能发生的状况确实不会发生。

assert宏的原型定义在<assert.h>中，其作用是先计算表达式 expression ，如果expression的值为假（即为0），那么它先向stderr打印一条出错信息，然后通过调用abort 来终止程序运行。就会打印错误信息：Assertion failed。

方便调试，使代码更稳定、不易于出错。在很多项目中，都通过断言来控制代码是否使用。