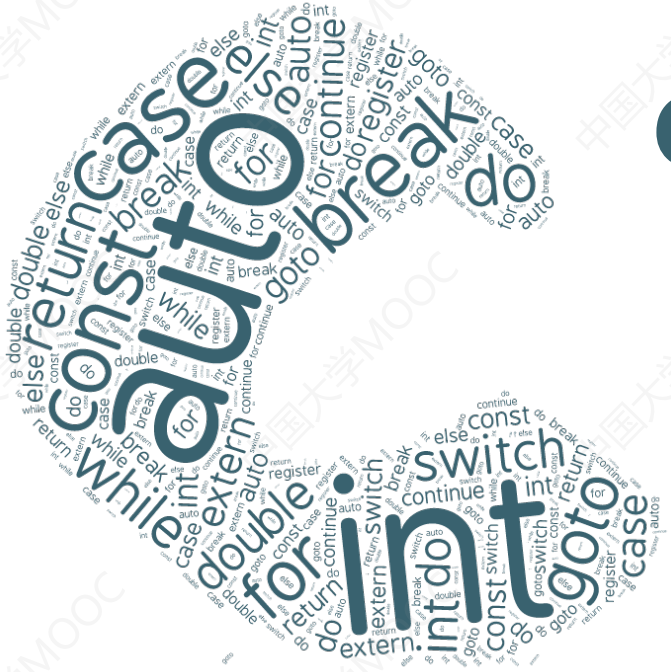

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

chapter 4

函数

目录
content

4.1 C语言程序结构及模块化设计

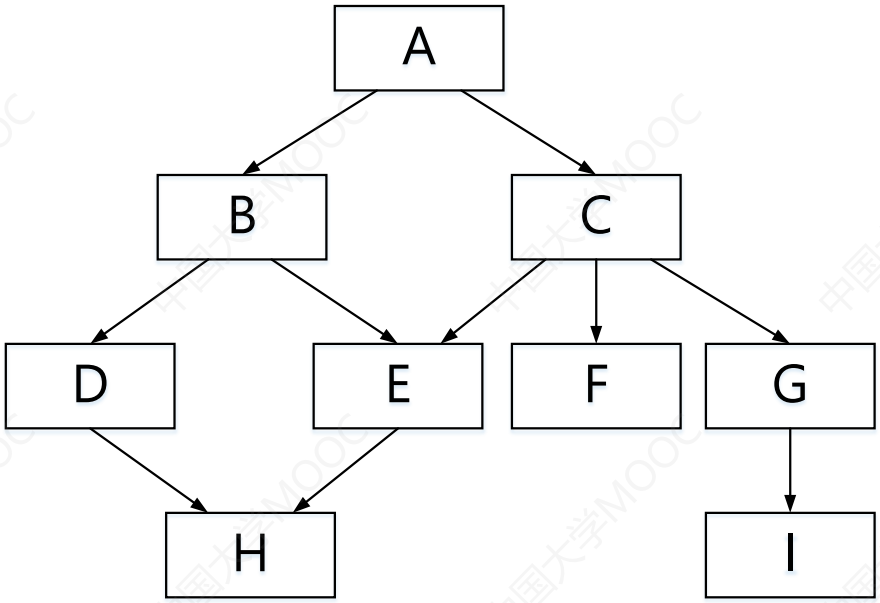
开发一些比较复杂的软件时，结构化的开发方法是常采用的方法，基本要点是：

- **自顶向下**
- **逐步求精**
- **模块化设计**

基本思想：把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围。

4.1.1 结构化软件及其优越性

模块化软件示意图



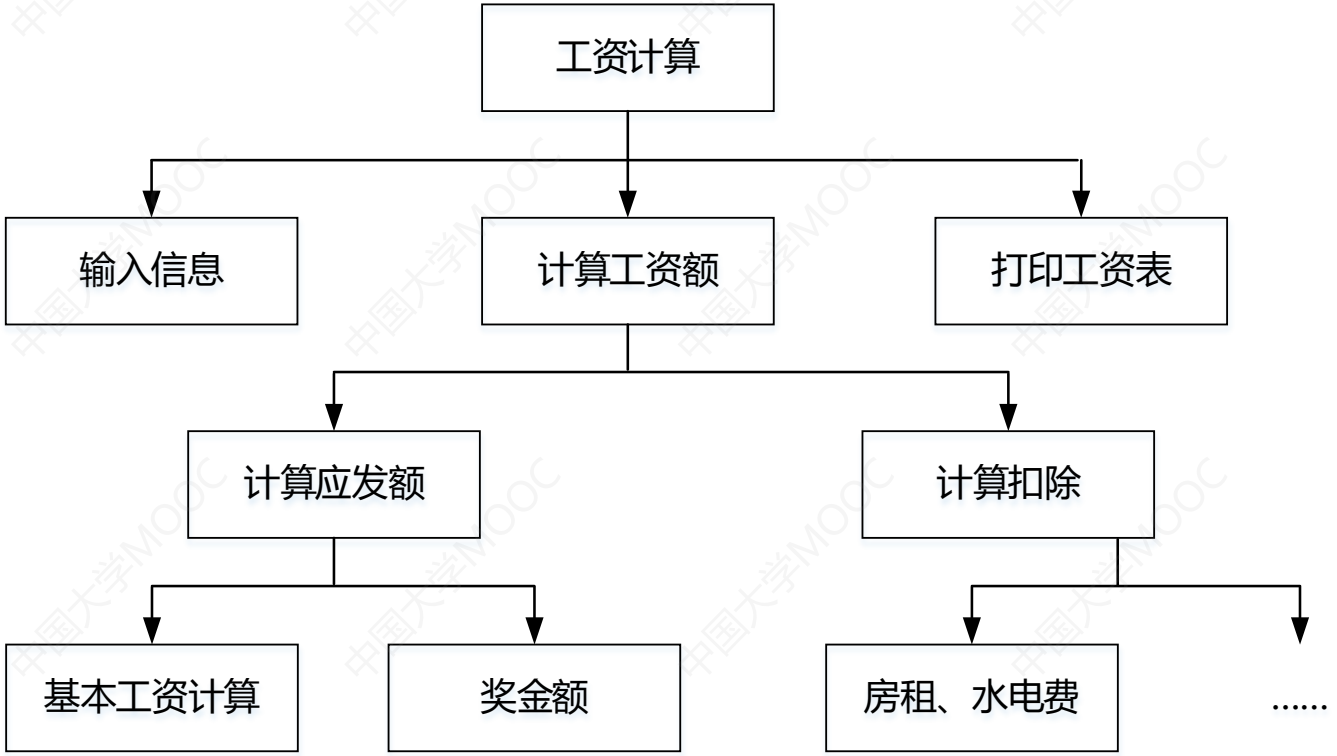
矩形框：功能模块

箭头：模块间的调用关系

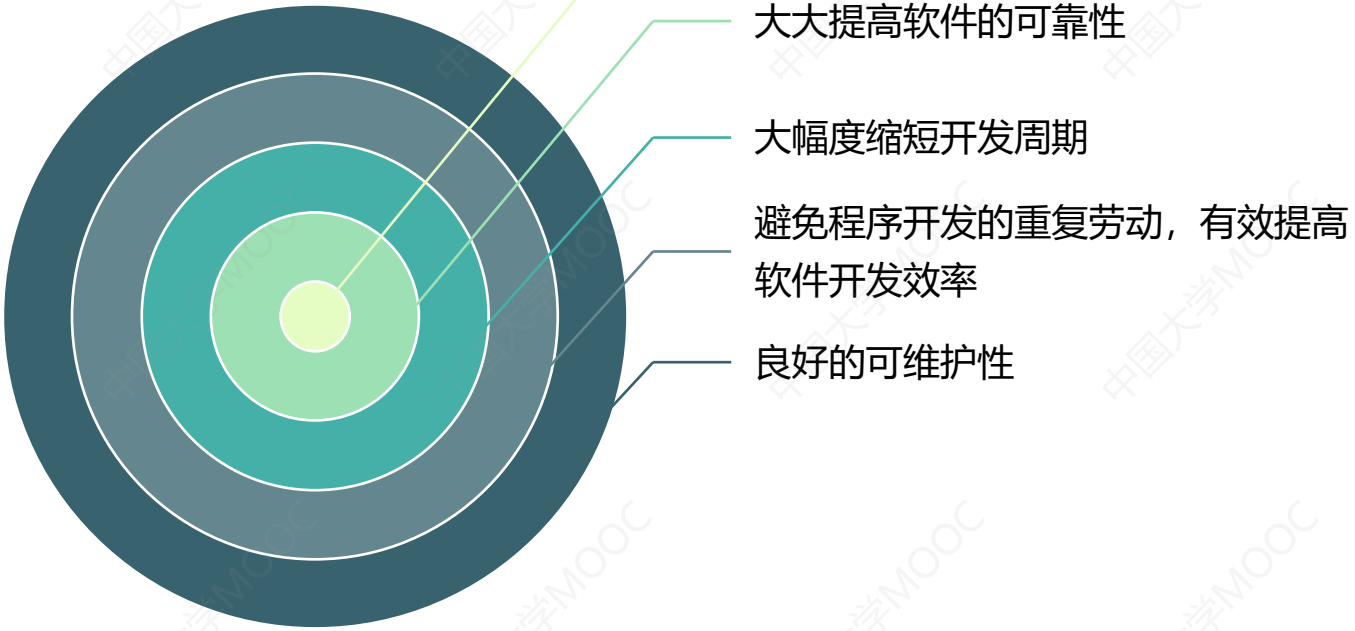
（箭头指向的是被调用模块）

4.1.1 结构化软件及其优越性

“工资计算程序”的自顶向下开发



4.1.1 结构化软件及其优越性



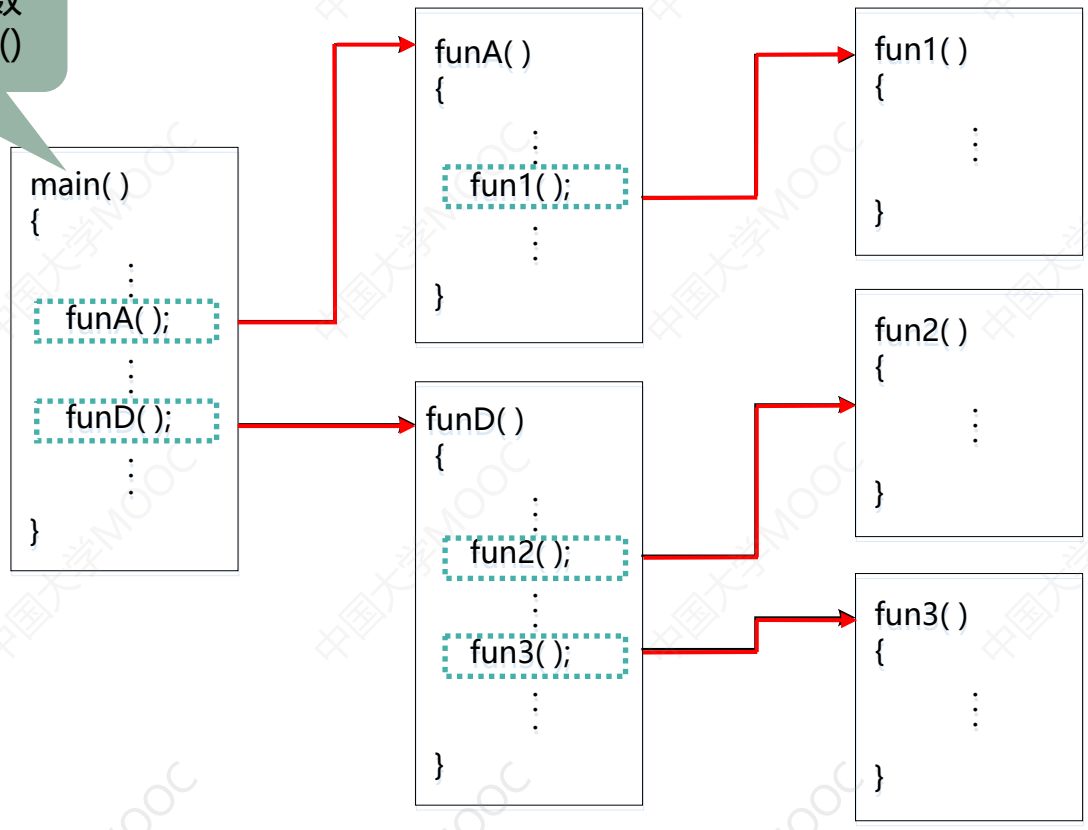
从软件工程上看，**可靠性、效率、可维护性**是软件质量的主要评价指标。因此，模块化软件能够成为高质量的软件。

4.1.2 C语言程序的结构

函数的调用关系

有且仅有一个函数称为主函数main()

函数调用语句



程序的执行总是从主函数开始。主函数中的所有语句按先后顺序执行完，则程序执行结束。

函数的定义：即编写完成函数功能的程序块。

函数定义的一般格式是：

```
<存储类型><数据类型>函数名(<形式参数及说明>)
{
    说明语句;
    执行语句;
}
```

函数头

函数体

函数的定义

函数定义的几点说明：

（1） 函数的存储类型说明只有static（静态）和extern（外部）两种 。存储类型为static时，对函数的调用局限于所在文件；存储类型为extern时，可省略，表示此函数是外部函数，可供其他文件调用。

（2） 函数的数据类型是函数返回值的数据类型，可以是各种基本数据类型（char、int、double等）和复杂数据类型，包括指针类型和结构体。当函数的数据类型为int时，可省略。当函数不需要返回值时，将函数的数据类型指定为void 。

（3）函数名与变量名一样也是标识符的一种，与变量名命名规则类似，最好“见名知意”。由圆括号包围的部分称为参数表，也称形式参数表，简称形参表。

（4） 由大括号括起来的程序部分称为函数体。

4.2.2 函数的调用

1、 函数的执行过程

在一个函数中调用另一个函数时，程序控制就从调用函数中转移到被调用函数，并且从被调用函数的函数体起始位置开始执行该函数的语句。在执行完函数体中的所有语句，或者遇到return语句时，程序控制就返回调用函数中原来的断点位置继续执行。

2、函数的原型（也称函数声明）

在一个函数中调用另外一个函数（即被调用的函数）需要具备下列条件：

（1）被调用的函数必须是已经存在的函数（是库函数或用户自己定义的函数）。

（2）如果使用库函数，在本文件的开头用#include命令将调用有关函数时所需要的信息“包含”到本文件中来；如果调用的是用户自己定义的函数，在主调函数所在文件中，对被调用函数进行声明。

在C语言中，函数原型的一般形式为：

（1）函数数据类型 函数名（参数类型1，参数类型2.....）；

（2）函数数据类型 函数名（参数类型1 参数名1，参数类型2 参数名2.....）；

函数的调用

3、函数调用时参数的使用

在调用一个函数时，必须使用具有实际量的值作为函数的参数，这时函数的参数称为实参数。

（1）实参数的个数和顺序必须与函数定义的形式参数保持一致;

（2）实参数据类型必须和相应的形式参数相同。可以这样认为：实参的值是形参初始化的初值。

函数的调用

4、函数调用的方式

凡是已定义的函数就可以用如下格式直接调用它：

```
函数名（实参表）;
```

源程序中可出现以下三种函数调用方式：

（1） 函数调用语句

函数原型： void delay(unsigned t);

调用语句： delay(6000);

（2） 函数表达式

```
v = volume(3, 4, 5);
c = 2 * max(a, b);
```

（3） 作为函数的实参

```
m = max(a, max(b, c));
```

其中max(b, c)是第一次函数调用，它的返回值作为max( )第二次调用的实参。m的值取a, b, c中最大值。

4.2.3 参数数目可变的函数

C语言中可以定义参数数目可变的函数。定义参数数目可变的函数时，必须至少明确说明一个形参；在列出的最后一个形参后面用省略符（.....）来说明该函数的参数数目可变。

例如：

函数原型：int printf(char *format,.....);

调用形式：printf(格式化字符串，输出参数1，输出参数2.，....，输出参数n);

4.3 变量的存储类型

C语言中的变量具有两种属性：

（1）根据变量所持有数据的性质不同而分为各种数据类型。

（2）根据变量的存储方式不同而分为各种存储类型。

变量的数据类型决定了该变量所占内存单元的大小及形式；变量的存储类型规定了该变量所在的存储区域，因而规定了该变量作用时间的长短，即寿命的长短，这种性质又称为"存在性"。

变量在程序中说明的位置决定了该变量的作用域，即在什么范围内可以引用该变量，"可引用"又称为"可见"，所以这种性质又称为"可见性"。

4.3.1 ——几个基本概念

- **作用域**：是该变量可访问的程序部分。
- **变量的生存期**：变量从定义开始到它所占有的存储空间被系统收回为止的这段时间。
- **变量的可见性**：在某个程序区域，可以对变量进行访问(或称存取)操作，则称该变量在该区域为可见的，否则为不可见的。
- **全局变量和局部变量：**在一个函数内部或复合语句内部定义的变量叫内部变量，又称为"局部变量"。在函数外定义的变量称为外部变量，又称为"全局变量"。

4.3.1 ——几个基本概念

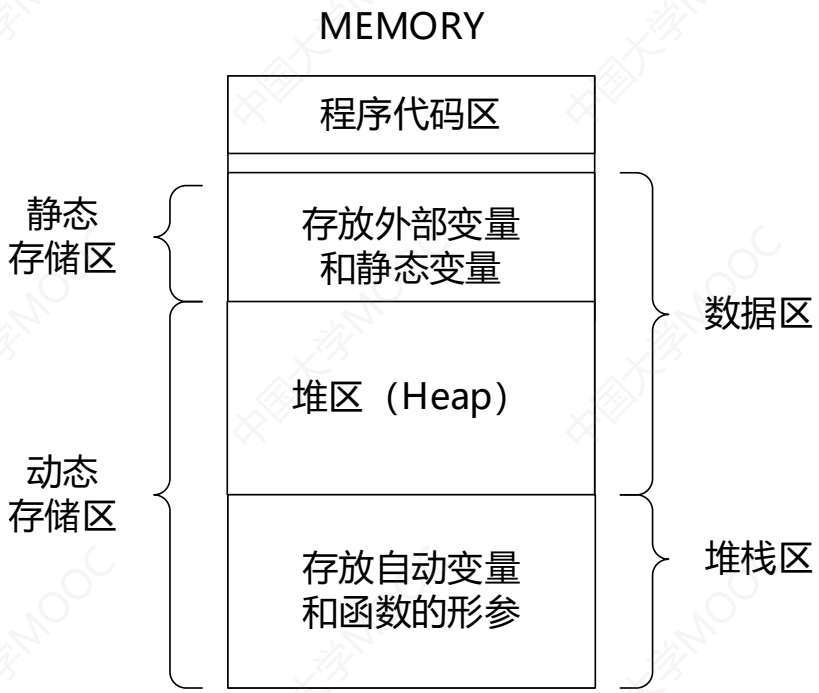
- **动态存储变量和静态存储变量**

在程序运行期间，所有的变量均需占有内存，有的是临时占用内存，有的是整个程序运行过程中从头到尾占用内存。

对于在程序运行期间根据需要进行临时动态分配存储空间的变量称为“动态存储变量”，对于在程序运行期间永久性占用内存的变量称为“静态存储变量"。

4.3.1 ——使用内存的情况

- 一个正在运行的程序可将其使用内存的情况分为如下三类:
- **程序代码区**：程序的指令代码存放在程序代码区。
- **静态存储区**：静态存储变量存放区，包括全局变量。
- **动态存储区**：存放局部自动变量，函数的形参以及函数调用时的现场保护和返回地址等。



4.3.1 ——变量定义与存储类型

- **变量定义的一般形式为:**

  <存储类型> 数据类型 变量名表;

- **存储类型包括:**

auto	自动型
register	寄存器型
extern	外部参照型
static	静态型

4.3.2 自动型变量

1. 自动变量用关键字auto进行存储类型声明

(1)

```
void main
{
        auto int x, y;
        auto float z;
        ……..
}
```

(2)

```
if (x!=y)
{
    int i ;
    for (i = 0 ; i < 10 ; i++)
    {
        int    j ;
        ……
     }
}
```

在主函数内定义了自动型int变量x，y和自动型float 变量z，在函数内或复合语句中定义自动型变量时auto可缺省

在条件判断后的那个复合语句中定义了一个自动型int变量i，在for循环后的那个复合语句中定义了一个自动型int变量 j

2. 作用域及寿命

由于自动型变量只能作内部变量，所以自动变量只在定义它的函数或复合语句内有效，即“局部可见”。

变量的作用域是指该程序中可以使用该变量名字的范围。对于在函数开头声明的自动变量来说，其作用域是声明该变量的函数。不同函数中声明的具有相同名字的各个局部变量之间没有任何关系。函数的参数也是这样的，实际上可以将它看作是局部变量。

4.3.2 自动型变量

例4.1 自动型变量的作用域

```
#include <stdio.h>
 void  main( )
{
    int x=5;                          //auto缺省……(1)
    printf("x=%d\t",x);
    if(x>0)
    {
      int x=10;                       //auto缺省……(2)
      printf("x=%d\t",x);
    }
    printf("x=%d\n",x+2);
 }
```

运行结果：

x=5　x=10　x=7

自动型变量

例4.2 下面的例子说明了自动变量的特性

```
#include<stdio.h>
 void func( );
 void func( )
{
        auto int a = 0;
        printf(" a of func( ) = %d\n",++a);
 }
void main( )
{
         int a = 10 ;
        func( );
        printf(" a of main( ) = %d\n",++a);
        func( );
        func( );
 }
```

调用func( )函数

运行结果：

```
a of func( )=1
a of main( )=11
a of func( )=1
a of func( )=1
```

4.3.2 自动型变量

例4.3 下面的程序说明自动变量的初始化和作用域

```
#include<stdio.h>
int n;
void show( );
void show( )
{
  auto int i=3;
  n++;
  i++;
  printf("input the value: n=%d  i=%d\n", n, i);
  {
    auto int  i=10;
    i++;
    printf("now the value i=%d \n",i);
   }
  printf("then the value i=%d\n",i);
}
```

```
void main( )
{
    auto int i;
    auto int n=1;
    printf("at first n=%d\n",n);
    for(i=1 ; i<3 ; i++)
    {
        show( );
     }
    printf("at last n=%d",n);
}
```

运行结果：

```
at first n=1
input the value: n=1  i=4
now the value i=11
then the value i=4
input the value: n=2  i=4
now the value i=11
then the value i=4
at last n=1
```

4.3.3 寄存器型变量

1. 定义

寄存器型变量在函数内或复合语句内定义，例如:

```
void main( )
{
    register int i;
    for (i=0 ;i=<100 ;i++)
   {
      ……
    }
 }
```

寄存器型变量存储在CPU的通用寄存器中，因为数据在寄存器中操作比在内存中快得多，因此通常把程序中使用频率最高的少数几个变量定义为register型，目的是提高运行速度，从而节省了大量的时间，大大加快了程序的运行速度。

2. 分配寄存器的条件

（1）有空闲的寄存器；

（2）变量所表示的数据的长度不超过寄存器的位长；

3. 作用域和寿命

作用域和寿命同auto类型，也是在定义它的函数或复合语句内有效，即"局部可见"。

4.3.3 寄存器型变量

例4.4 用寄存器变量提高程序执行速度

```
#include<stdio.h>
#define T 10000
void delay1( );
void delay2( );
void delay1( )
{
    register unsigned i=0 ;
    for ( ; i<T ; i++)
    {
    }
}
void delay2( )
{
    unsigned i ;
    for (i=1 ; i<T ; i++)
    {
    }
}
```

函数的形参也可以指定为寄存器变量，一个函数一般以拥有2个寄存器变量为宜

```
void main( )
{
    unsigned int i;
    printf("\a调用delay1( )第一次延时!\n");
    for ( i=0 ; i<60000 ; i++)
    {
      delay1();
    }
    printf("\a第1次延时结束!\n调用delay2( )第2次延时!\n");
    for ( i=0 ; i<60000 ; i++)
    {
      delay2();
    }
    printf("\a第2次延时结束!\n");
}
```

运行结果：

```
调用delay1( )第一次延时!
第1次延时结束!
调用delay2( )第2次延时!
第2次延时结束!
```

4.3.4 外部参照型变量[extern]

1. 定义:

extern型变量一般用于在程序的多个编译单位之传送数据，在这种情况下指定为extern型的变量是其它编译单位的源程序中定义的，它的存储空间在静态数据区，在程序执行过程中长期占用空间。要访问另一个文件中定义的跨文件作用域的全局变量，必须进行extern说明。

2. 作用域及寿命:

「 全局存在，全局可见 」

4.3.4 外部参照型变量[extern]

```
/*file1.c*/
extern int x;
void main( )
{
  x++;
  fun1();
  fun2();
 }
```

```
/*file2.c*/
extern int x;
void fun1 ( )
{
  x+=3;
 }
```

```
/*file3.c*/
int x=0;
void fun2( )
{
  printf("%d",x);
 }
```

file1.c和file2.c中的extern int x； 告诉编译程序X是外部参照变量，应在本文件之外去寻找它的定义。所以上面的x虽在两个源文件中，但它们是同一个变量。在文件之外的file3.c中，定义了int x=0，即为它们调用的变量。

4.3.4 外部参照型变量[extern]

例4.5 说明了外部变量的特性

```
#include <stdio.h>
int n = 100;
void hanshu( );
void hanshu(void)
{
  n-=20 ;
}
int main(void)
{
   printf("n=%d\n",n);
   for( ; n>=60 ; )
    {
        hanshu( );
        printf("n=%d\n",n);
    }
   return 0 ;
}
```

运行结果：

n=100
n=80
n=60
n=40

4.3.4 外部参照型变量[extern]

例4.6 用extern声明外部变量

文件file1.c中的内容为:

```
#include <stdio.h>
int a;
int m;
int power( );
void main( )
{
   int b=3,c,d;
  printf("input the number a and its   \
          power m:\n");
  scanf("%d,%d",&a,&m);
  c = a*b;
  printf("%d*%d=%d\n",a,b,c);
  d = power();
  printf("%d**%d=%d",a,m,d);
}
```

文件file2.c中的内容为:

```
extern int a;
extern int m;
int power( );
{
    int i,y=1;
    for ( i=1 ; i<=m ; i++)
    {
        y*=a;
     }
    return(y);
}
```

运行结果:

```
input the number a and its power m:
5,4                    //输入
5*3=15                 //输出
5**4=625
```

本程序的作用是给定b的值，输入a和m，求a*b，和a的值。

4.3.5 静态型变量[static]

1. 定义

静态型变量既可以在函数或复合语句内进行，也可以在所有函数之外进行。在函数或复合语句内部定义的静态变量称为局部静态变量，在函数外定义的静态变量称为全局静态变量。

```
static float x;          /*定义x全局静态变量*/
void main( )
{
    static int y;        /*定义y局部静态变量*/
    ……
}
```

2. 作用域和寿命

static类型变量	**全局寿命**
全局static变量	**全局可见**
局部static变量	**局部可见**

4.3.5 静态型变量[static]

例4.7 考察静态变量的值

```
#include <stdio.h>
int a ;
int f( );
int f( )
{
    auto int b=0;
    static int c=3;
    b++;a++;
    c++;
    return(a+b+c) ;
 }
void main( )
{
    int i;
    for(i=0;i<3;i++)
    {
        printf("%d\t",f());
     }
 }
```

运行结果：

6 8 10

4.3.5 静态型变量[static]

例4.8 说明外部静态变量和外部变量的区别

文件file1.c如下:

```
#include<stdio.h>
static float x;
float y ;
float f2( );
float f1( );
float f1( )
{
    return(x*x);
}
void main( )
{
     x=500;
     y=100;
     printf("f1=%f,\n f2=%f\n", f1( ), f2( ));
}
```

文件file2.c如下:

```
extern float y;
float f2( )
{
   return(y*y);
 }
```

运行结果:

fl=250000.000000 ,

f2=10000.000000

4.3.5 静态型变量[static]

例4.8 局部静态变量与自动变量的区别

```
#include<stdio.h>
void value( );
void value( )
{
    int au=0;
    static int st=0;
    printf("au_variable=%d, st_variable=%d\n",au,st);
    au++;
    st++;
}
void main( )
{
    int i;
    for(i=0;i<3;i++)
    {
        value( );
    }
}
```

运行结果：

au_variable=0, st_variable=0
au_variable=0, st_variable=1
au_variable=0, st_variable=2

4.4 函数间的数据传递

C程序是由若干个相对独立的函数组成的，但是各个函数处理的往往是同一批数据。所以说程序中的函数虽然是离散的，但被处理的数据却是连续的（数据常常贯穿若干函数中连续流动）。因此，在程序运行期间，函数之间必然存在着数据的相互传递过程。

函数间数据传递实现的方法：

- 使用函数参数在函数间传递数据
- 使用返回值传递数据
- 使用全局变量传递数据

4.4.1 使用函数参数在函数间传递数据

在一个函数中调用另一个函数时，实参数的值传递到形式参数中，这样就实现了把数据由调用函数传递给被调用函数。在使用参数传递数据时，可以采用两种不同的方式：**值传递** 和 **地址传递**。

1、函数调用的值传递

使用值传递方式调用时，实参可以是常量、已经赋值的变量或表达式值，甚至是另一个函数，只要它们有一个确定的值，被调用函数的形参就可以使用变量来接收实参的值。

采用这种方式时，每个参数只能传递一个数据。所以，当需要传递的数据较多时，一般不采以这种方式，而采用地址传递方式。

4.4.1 使用函数参数在函数间传递数据

例4.10 比较两个整数的大小

```
#include <stdio.h>
int comp(int x, int y);
void main()
{
  int a = 10, b = 20;
  printf("%d\n", comp(a, b));
  printf("%d\n", comp(30, b));
}
int comp(int x, int y)
 {
  if (x > y)
    return 1;
  else if (x < y)
    return -1;
  else
    return 0;
}
```

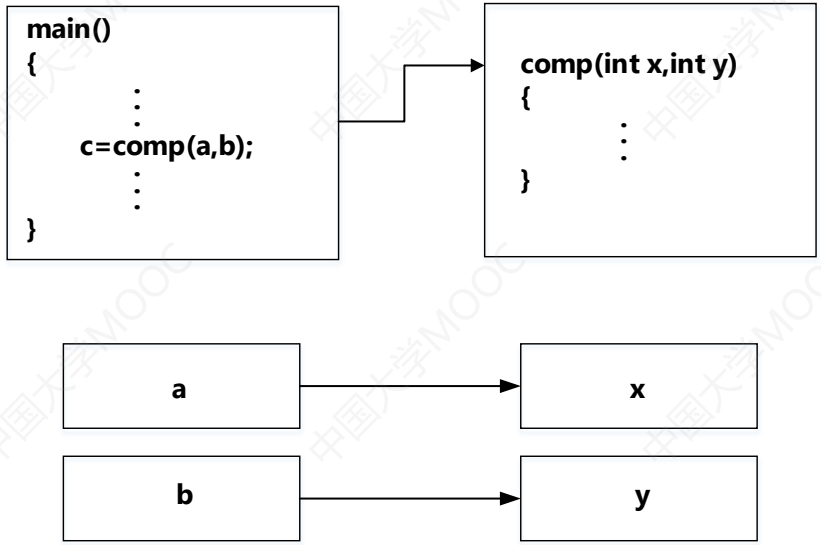
函数原型的声明

采用值传递方式在函数间传递数据

运行结果：

-1

1



数据复制方式传递数据的特点：

数据在传递方和被传递方占用不同的内存空间，被传递数据在被调用的函数中无论如何变化，都不会影响该数据在调用函数中的值。

4.4.1 使用函数参数在函数间传递数据

例 实现两个整数的交换

```
#include"stdio.h"
void swap(int,int);
void main （）
{
    int a =3， b=5;
    printf （"a=%d,  b=%d\n",a,b);
    swap( a, b)；
    printf （"a=%d,  b=%d\n",a,b);
}

void swap(int a,int b)
{
    int temp;
    temp =a;
    a = b;
    b = temp;
}
```

交换a，b的值

运行结果：

a=3, b=5

a=3, b=5

2、地址传送方式传递数据

- 参数传递的不是数据本身，而是数据的存储地址。
- 变量的地址作为参数调用一个函数，而被调用函数的形式参数必须是可以接收地址值的指针变量，并且它的数据类型必须与被传递数据的数据类型相同。
- 把变量的地址传递给被调函数，被调函数通过这个地址找到该变量的存放位置，直接对该地址中存放的变量的内容进行存取操作。因此，在被调函数中若修改了地址中的内容，实际上修改了实参的值。
- 如果想让形参的改变影响实参，即函数返回时需要获得几个结果值，应采用地址传递方式。

4.4.1 使用函数参数在函数间传递数据

例4.11 对三个整数a,b,c 进行从小到大的排序

```
#include <stdio.h>
void   swap(int *x, int *y);
void   main()
{
  int a, b, c;
  scanf("%d%d%d", &a, &b, &c);
  if (a > b)
        swap(&a, &b);
  if (a > c)
        swap(&a, &c);
   if (b > c)
        swap(&b, &c);
   printf("%d\t%d\t%d\n", a, b, c);
}
```

采取传地址的方式传递数据

```
void    swap(int *x, int *y)
 {
   int  temp;
   temp = *x;
  *x = *y;
   *y = temp;
    return;
 }
```

运行结果：

5,4,8　　//输入

4 5 8　　//输出

4.4.2 使用返回值传递数据

函数被调用后可以向调用它的函数返回一个返回值。返回值通过函数中使用return语句实现：

```
return （表达式）；
```

功能：把程序控制从被调用函数返回调用函数中，同时把返回值带给调用函数。

- return语句只能把一个返回值传递给调用函数，当要求返回的值多于一个时不能使用返回值传递。
- 当返回值是数值，调用函数需要使用和返回值具有相同数据类型的变量接收该返回值；而当返回值是地址值，应使用指针接收。

4.4.2 使用返回值传递数据

例4.13 幂函数的使用

```
double power(double x, int n)
{
  double p;
  for(p=1;n>0;--n)
    p = p * x;
  return(p);
}
```

幂函数的功能是计算x的n次方。该函数用形式参数接收x和n的值，计算结果使用return语句传递给调用它的函数中

使用返回值传递数据

在return语句可以不带表达式部分，即：return;

这种情况下，它仅实现程序控制的转移，程序从被调函数返回主调函数，而不传递任何返回值。

C语言的函数中不是必需要有return语句，没有return语句的C函数，程序控制到达包围函数的下面大括号}时，自动返回调用函数。

4.4.2 使用返回值传递数据

例4.14 符号函数的使用

```
int sign(int x)
{
   if(x==0)
         return(0);
   else if (x>0)
         return(1);
   else
         return(-1);
}
```

函数中可以根据需要设置多个return语句

在编写函数时，常常要求把函数运行的状态，比如：是否顺利地执行函数的功能，在执行过程中是否出错、溢出等状态返回给调用函数。在这种情况下，使用返回值返回状态的标志值。

4.4.3 使用全局变量传递数据

例4.15 求1+1/2+1/3+....1/n的值

```
#include <stdio.h>
int   n;
float  s;
void   count();
void main()
{
  scanf("%d", &n);
  count();
  printf("s=%f\n", s);
}
```

使用全局变量进行数据传递

```
void    count()
{
 int   i;
 if (n <= 0)
 {
   printf("the %d is invalid\n", n);
    s = 0;
 }
 else
 {
   for (i = 1; i <= n; i++)
   s += 1.0 / i;
 }
 return;
}
```

程序中全局变量的使用增加了函数之间的联系，但是降低了函数作为一个程序模块的相对独立性。在模块化软件设计方法中不提倡使用全局变量。

运行结果：

(1) 0
the 0 is invalid
s=0.000000

(2) 9
s=2. 828969

递归函数

递归函数 称为自调用函数。

特点：在函数内部直接或间接地自己调用自己。

在递归函数中，由于存在着自调用的过程，故程序控制将反复地进入它的函数体。为了防止自调用过程无休止地继续下去，在函数内必须设置某种条件。当条件成立时终止调用过程，并使程序控制逐步从函数中返回。

整数n的阶乘

数学公式

$$n! = 1 \times 2 \times 3 \times \ldots \times n$$

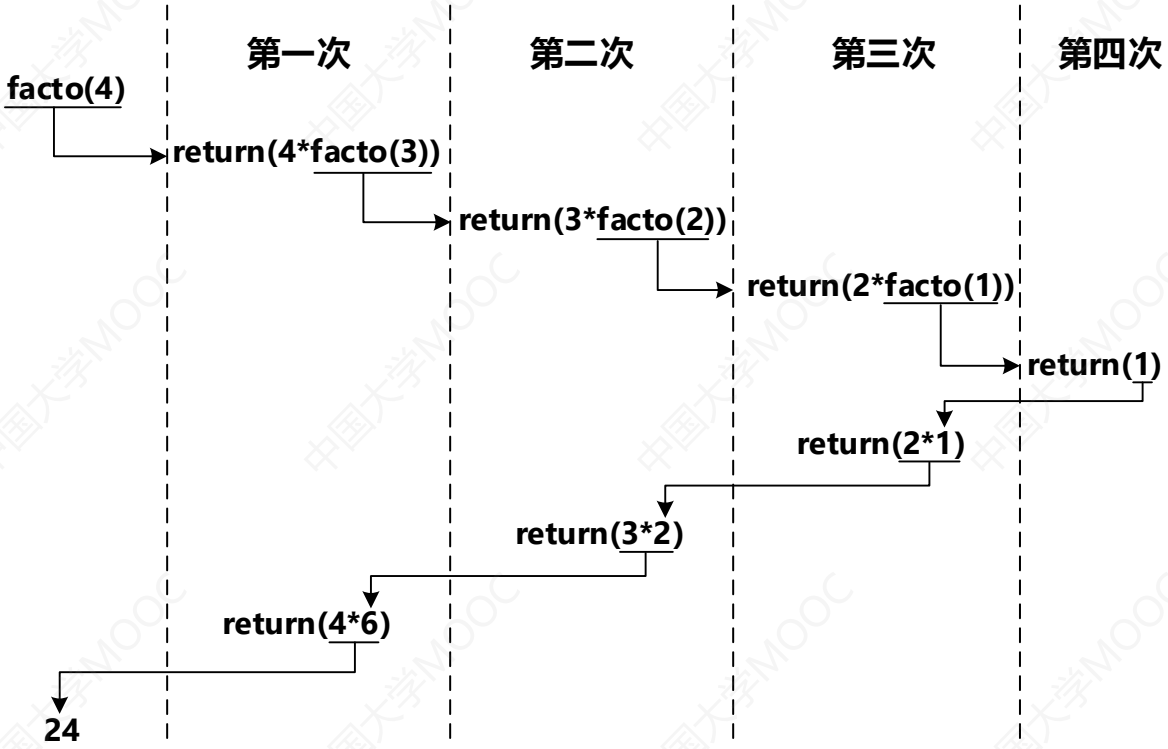
递归算法

$$n! = n \times (n-1)!$$

$$1! = 1$$

例4.16 阶乘的递归函数

```
facto(int x)
 {
      if(x == 1||x==0)
        return(1);
      else
        return(x*facto(x - 1));
}
```



本章小结

从本章开始，函数将是教学的核心和主要内容，本章主要是函数的初步知识，后面各章将进一步学习函数的使用和应用，本章需重点掌握的内容有：

- 了解C语言程序的结构特点与其优越性
- 熟练掌握函数定义与调用的方法
- 掌握变量的存储类型作用何使用方法
- 熟练掌握函数间数据传递的方法
- 了解递归函数的概念和使用方法

END