第8章 用函数实现模块化设计

§1 函数调用概念

例1: 编程打印如下图文信息

```
**********************
    How do you do!
**********************
```

```
#include <stdio.h>
printstar( )
 { printf("**********************\n"); }
 main( )
  { printstar( );    /*调用printstar函数打印星号线*/
    printf("  How do you do!\n");
    printstar( );    /*调用printstar函数打印星号线*/
  }
```

§1 函数调用概念

例2: 编程计算:

$$a = \frac{6.9 + y}{y + \sqrt{1 + 2y + 3y^2}}, b = \frac{\sin y}{y^2 + \sqrt{1 + 2y^2 + 3y^4}},$$

$$c = \frac{2y}{\sin^2 y + \sqrt{1 + 2\sin^2 y + 3\sin^4 y}}$$

```
main( )
{ float a, b, c, y;
   scanf("%f", &y);
   a=(6.9+y)/f(y); /*y是实际参数*/
   b=sin(y)/f(y*y); /*y2是实际参数*/
   c=2.*y/f(sin(y)*sin(y));
  printf("%f,%f,%f",a, b, c);
}
```

```
#include <math.h>
float f(x)
 float x;   /*x是形式参数*/
{  float z;
   z=x+sqrt(1+2.*x+3.*x*x);
   return(z);
}
```

§2 定义函数的方法

1. K&R 传统语法

[类型符] 函数名([形参名表])
[形参定义]
{ [内部变量定义]
[语句序列]
}

2.函数原型法——新ANSI标准

[类型符] 函数名([形参定义表])
{ [内部变量定义]
[语句序列]
}

内部变量定义语法：
类型名 变量名[,变量名]...

传统语法中形参定义：
类型名 变量名[,变量

函数原型中形参定义语法：
类型名 形参名[, 类型名 形参

§2 定义函数的方法

1. K&R 传统语法

[类型符] 函数名([形参名表])
[形参定义]
{ [内部变量定义]
[语句序列]
}

2.函数原型法——新ANSI标准

[类型符] 函数名([形参定义表])
{ [内部变量定义]
[语句序列]
}

```
eg1:float  f1( )
    {  float x,y;
       .......
    }
```
正确

```
eg2:char  f2( )
    {char c1,char c2;
       .......
    }
```
错误

```
eg3:double  f3( d1,d2 )
    double d1,d2;
    {  double e1,e2;
       .......
    }
```
正确

§2 定义函数的方法

1. K&R 传统语法

[类型符] 函数名([形参名表])
[形参定义]
{ [内部变量定义]
[语句序列]
}

2.函数原型法——新ANSI标准

[类型符] 函数名([形参定义表])
{ [内部变量定义]
[语句序列]
}

```
eg4:double  f4(double d1,d2 )
    {  double e1,e2;
       .......
    }
```

错误

```
double  f4(double d1,double d2 )
 {  double e1,e2;
    .......
 }
```

正确

§2 定义函数的方法

例1: 以下正确的函数首部定义形式是_____。

A. double fun(int x, int y)　　B. double fun(int x; int y)

C. double fun(int x, int y);　　D. double fun(int x, y);

例2: 以下正确的函数形式是_____。

A. double fun(int x,int y)
```
{ z=x+y; return z;
}
```

B. fun(int x,y)
```
{ int z;
  return z;}
```

C. fun(x,y)
```
{ int x,y; double z;
  z=x+y; return z;
}
```

D.double fun(int x,int y)
```
{ double z;
  z=x+y; return z;
}
```

§2 定义函数的方法

- 函数返回值:可将0个(控制)或1个数值带回调用点

  return([表达式]) ; 或 return [表达式] ;

return( ); 或 return ;

return(2*x+1); 或 return 2*x+1 ;

```
printstar( )
 {  printf("*****************\n");  return;  }
main( )
 {  printstar( ); /*调用printstar打印星号线*/
    printf("  How do you do!\n");
    printstar( ); /*调用printstar打印星号线*/
 }
```

§2 定义函数的方法

- 函数返回值:可将0个(控制)或1个数值带回调用点
  return([表达式]) ; 或 return [表达式] ;
- C语言中， 函数的类型是指函数返回值的类型

```
#include <stdio.h>
main( )
{ float a, b, c, y;
   scanf("%f", &y);
   a=(6.9+y)/f(y);
   b=sin(y)/f(y*y);
   c=2.*y/f(sin(y)*sin(y));
  printf("%f,%f,%f",a, b, c);
}
```

```
#include <math.h>
float f(x)
 float x;   /*x是形式参数*/
{  float z;
   z=x+sqrt(1+2.*x+3.*x*x);
   return(z);
}
```

§2 定义函数的方法

- 函数返回值:可将0个(控制)或1个数值带回调用点
  return([表达式]) ; 或 return [表达式] ;
- C语言中，函数的类型是指函数返回值的类型
- 若缺省函数类型说明，表示该函数是整型函数
- 可用void定义无(空)类型函数，表明函数不返回任何数值

```
printstar( )
{ printf("**********\n"); }
main( )
{ int a;
  a=printstar( );
  printf("a=%d\n",a);
}
```

a=12

```
void printstar( )
  { printf("**********\n"); }
main( )
 {  int a;
     a=printstar( );
     printf("a=%d\n",a);
 }
```

编译时出错

防止采样无用数据

§2 定义函数的方法

- 函数返回值:可将0个(控制)或1个数值带回调用点
  return([表达式]) ; 或 return [表达式] ;
- C语言中， 函数的类型是指函数返回值的类型
- 若缺省函数类型说明， 表示该函数是整型函数
- 可用void定义无(空)类型函数， 表明函数不返回任何数值
- 若return 返回值类型与函数首部类型不一致,以函数类型为准

```
#include <stdio.h>
max( float x, float y)
{  float z;
   z=x>y?x:y;
   return z;
 }
```

```
main( )
{  float a, b;
   int c;
   scanf("%f,%f", &a, &b);
   c=max(a, b);
  printf("Max is %d\n", c);
 }
```

§2 定义函数的方法

例1：若调用一个函数，且函数中没有return语句，则正确的说法是 D 。

该函数A.没有返回值　　B.返回若干个系统默认值

C.能返回一个用户所希望的函数值

D.返回一个不确定的值

例2：C语言规定，函数返回值的类型是由 D 。

A.return语句中的表达式类型所决定

B.调用该函数时的主调函数类型所决定

C.调用该函数时系统临时决定

D.在定义该函数时所指定的函数类型所决定

§3 对函数的调用

一、函数调用语法

被调函数名（[实参列表]）

个数相等
赋值兼容

例1：被调函数定义：

```
void printstar( )
 { printf("**********\n"); }
```

调用语法：

```
printstar( )
```

例2：被调函数定义：

```
float f(float x)   /*x是形式参数*/
{  float z;
   z=1+2*x;
   return(z);
}
```

调用语法：

```
f(2.0)      ← 5.0
f(2)        ← 5.0
f('2')      ← 101.0
```

调用语法：

```
f(2.0,3.0)
f("2")
```

错误

§3 对函数的调用

二、函数调用的执行过程

```
#include<stdio.h>
main( )
 { int  i=2, p;
   p=f(i, ++i);
   printf("%d",p);
   }
int f(int a, int b)
{ int c;
  if(a>b) c=1;
  else if(a==b) c=0;
  else c=-1;
  return( c );  }
```

0

从右向左顺序计算各实参表达式之值

↓

将实参之值传入形参的内存单元——形实结合

↓

控制转被调函数并执行

↓

执行至return或}控制返回调用点

§3 对函数的调用

例1：下面函数调用语句中实参的个数为 B 。

```
func((exp1,exp2),(exp3,exp4,exp5));
```

A. 1　　B. 2　　C. 4　　D. 5

例2(考题)：下列程序的运行结果是 22 。

```
func( int a, int b)
{ return  a+b; }
main( )
{ int x=6, y=7, z;
  z=func(func(x++, y++), func(--x, --y));
  printf("%d", z);
}
```

§3 对函数的调用

三、对被调函数的声明

类型符 被调函数名( );

类型 被调函数(类型 形参, 类型 形参…);

```
main( )
{  float add( );
    float a, b, c;
    scanf(“%f,%f”, &a, &b);
    c=add(a, b);
    printf(“%f”, c);
}
float  add( float x, float y)
{ float z;
   z=x+y;
   return z;
 }
```

§3 对函数的调用

三、对被调函数的声明

类型符 被调函数名( );

类型 被调函数(类型 形参, 类型 形参…);

类型 被调函数(形参类型, …);

```
main( )
{  float add(float x,float y );
   float a, b, c;
   scanf("%f,%f", &a, &b);
   c=add(a, b);
   printf("%f", c);
}
float  add( float x, float y)
{ float z;
   z=x+y;
   return z;
 }
```

§3 对函数的调用

三、对被调函数的声明

类型符 被调函数名( );

类型 被调函数(类型 形参, 类型 形参…);

类型 被调函数(形参类型, …);

缺省声明的情形

- 被调函数定义在主调函数之前，可以缺省对被调函数的声明
- 在C程序源文件的开头处集中说明了被调函数

```
main( )
{  float add(float ,float  );
   float a, b, c;
   scanf("%f,%f", &a, &b);
   c=add(a, b);
   printf("%f", c);
}
float  add( float x, float y)
{ float z;
  z=x+y;
  return z;
 }
```

§3 对函数的调用

三、对被调函数的声明

类型符 被调函数名( );

类型 被调函数(类型 形参, 类型 形参...);

类型 被调函数(形参类型, ...);

缺省声明的情形

- 被调函数定义在主调函数之前，可以缺省对被调函数的声明
- 在C程序源文件的开头处集中说明了被调函数

```
char letter(char,char);
float f(float,float);
int m(int,int);
main( )
{...}
char letter(char c1,char c2)
{...}
float f(float x,float y)
{...}
int m(int j,int k)
{...}
```

§3 对函数的调用

例:下列程序有语法性错误,有关错误原因的正确说法是 C 。

```
main( )
{  int G=5,k;
   void prt_char( );
   ……
   k=prt_char(G);
   …… }
```

A.语句void prt_char();有错， 它是函数调用语句， 不能用void说明

B.变量名不能使用大写字母

C.函数说明和函数调用之间有矛盾

D.函数名不能使用下划线

§4 函数间数据传递

形实结合：主调函数对被调函数的通信

一、被调函数中形参变量的生存期有多长？

形参变量生存期等同于定义它的函数的执行期：在被调函数被调用执行时，为其形参动态分配主存单元，伴随被调函数执行结束而消失单元

§4 函数间数据传递

二、基本数据类型作实参

计算出实参表达式之值，送入形参内存单元中：形实结合方式是值传递方式，数据传输是单向的

例1(考题）：分析程序运行结果。

```
void swap( int a, int b)
{ int t;
   if(a>b) t=a, a=b, b=t; }
main( )
{ int x=15, y=12, z=20;
  if(x>y) swap(x, y);
  if(x>z) swap(x, z);
  if(y>z) swap(y, z);
  printf(‘%d,%d,%d”,x, y, z); }
```

15,12,20

§4 函数间数据传递

例2：C语言规定，简单变量做实参时，它和对应形参之间的数据传递是__B__。

A. 地址传递

B. 单向值传递

C. 由实参传给形参，再由形参传回给实参

D. 由用户指定传递方式

例3：在C语言中，以下不正确的说法是__B__。

A. 实参可以是常量、变量或表达式

B. 形参可以是常量、变量或表达式

C. 实参可以为任意类型

D. 形参应与对应的实参类型一致

§4 函数间数据传递

例4：C语言中，以下说法正确的是__A__。

A.实参和与其对应的形参占用各自独立的存储单元

B.实参和与其对应的形参共同占有一个存储单元

C.只有当实参和与其对应的形参同名时才占有共同单元

D.形参是虚拟的，不占用存储单元

例5：以下错误的描述是__D__。

函数调用可以

A.出现在执行语句中

B.出现在一个表达式中

C.作为一个函数的实参

D.作为一个函数的形参

§4 函数间数据传递

例6：写出程序运行结果

```
#include <stdio.h>
main( )
  { int x=1;
     void f1( ),f2( );
     f1( );      f2(x);
     printf("x=%d\n",x);
  }
  void f1(void)
   { int x=3;
     printf("x=%d\t",x); }
  void f2(x)
    int x;
  { printf("x=%d\t",++x); }
```

x=3　　x=2　　x=1

§4 函数间数据传递

三、数组名作实参

1. 形参可以是数组名，且形参数组和实参数组的类型必须一致

2. 数组名作实参，不是“值传递”而是“地址传递”。其效果是形参数组和实参数组同一片内存单元

3.由于“地址传递”，产生了“数值的双向传递”效果

4.形参数组长度可缺省说明，即使说明了长度，系统也不予以检查

5. 形、实数组长度可以不等，但形参数组长度不能大于实参数组长度

go

§4 函数间数据传递

例1:阅读下列程序的运行结果

```
#include<stdio.h>
void swap( int a1, int b1)
 { int t;
    t=a1; a1=b1; b1=t;
  }
 main( )
{ int a, b;
   a=2,b=3;
   printf("a=%d,b=%d\n",a,b);
   swap(a, b);
   printf("a=%d,b=%d\n",a,b);
}
```

```
#include<stdio.h>
void swap(int ar[2] )
 { int t;
   t=ar[0];ar[0]=ar[1];ar[1]=t;
  }
main( )
{int a[2]={2,3};
  printf("%d, %d\n",a[0],a[1]);
  swap(a);
  printf("%d, %d\n",a[0],a[1]);
 }
```

§4 函数间数据传递

例2:数组score中有10个成绩，用函数average求平均成绩

```
#include <stdio.h>
main( )
{ int i;
   float score[10], aver;
   float average( float array[10]);
   for( i=0; i<10; i++)
    scanf("%f",&score[i]);
    aver=average(score);
    printf("Average is %f",aver);
 }
```

```
float average(float array[10])
{ int j;
   float av, sum=0.0;
   for(j=0; j<10; j++)
    sum+=array[j];
    av=sum/10;
     return(av);
 }
```

§4 函数间数据传递

例2:数组score中有10个成绩，用函数average求平均成绩

```
#include <stdio.h>
main( )
{ int i;
   float score[10], aver;
   float average( float array[    ]);
   for( i=0; i<10; i++)
    scanf("%f",&score[i]);
    aver=average(score);
    printf("Average is %f",aver);
 }
```

```
float average(float array[    ])
{ int j;
   float av, sum=0.0;
   for(j=0; j<10; j++)
    sum+=array[j];
    av=sum/10;
     return(av);
 }
```

Back

§4 函数间数据传递

例2:数组score中有10个成绩，用函数average求平均成绩

```
#include <stdio.h>
main( )
{int i,num;
  float score[10], aver;
  float average( float array[ ],int n);
   for( i=0; i<10; i++)
    scanf("%f",&score[i]);
   scanf("%d",&num);
    aver=average(score, num);
    printf("Average is %f",aver);
 }
```

```
float average(array, n)
float array[ ];
int n;
{ int j;
   float av, sum=0.0;
   for(j=0; j<n; j++)
    sum+=array[j];
    av=sum/n;
     return(av);
 }
```

Back

§4 函数间数据传递

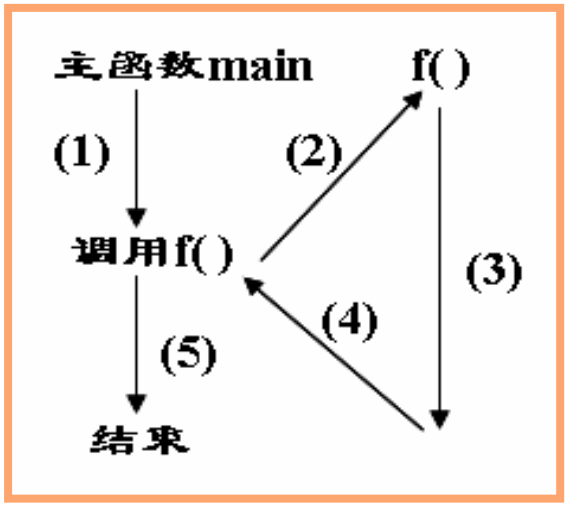
例3:说出下列程序的功能及结果

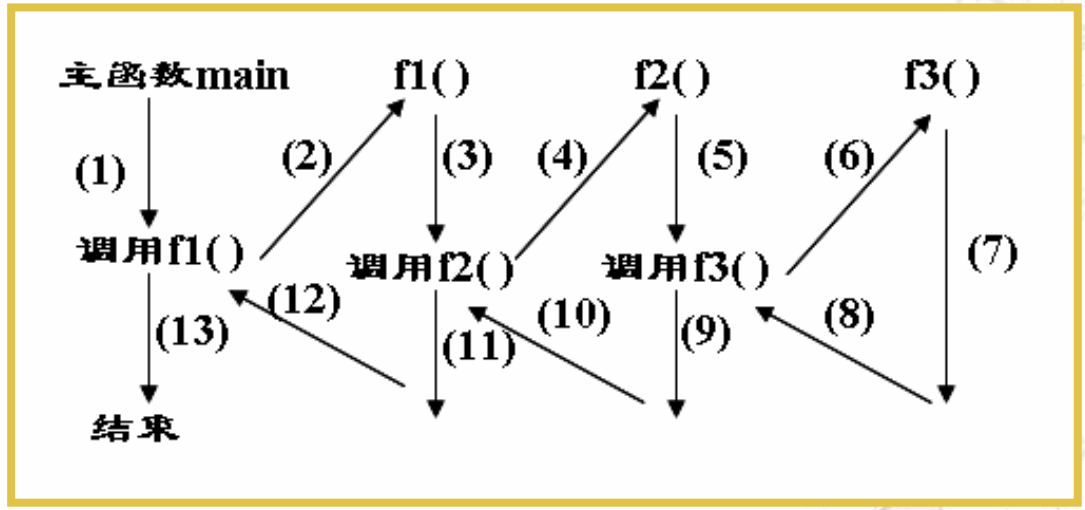
```
max_value(int arr[ ][4])
{ int i,j,k,max;
  max=arr[0][0];
  for(i=0;i<3;i++)
  for(j=0;j<4;j++)
   if(arr[i][j]>max) max=arr[i][j];
  return max;
}
main( )
{int a[3][4]={1,3,5,7,2,4,6,8,15,17,33,12};
 printf("max value is %d\n",max_value(a)); }
```

max value is 33

多维数组作形参，只可以省略第一维的大小说明，其它高维说明不可缺省

§5 函数嵌套调用





例1：在C语言中，以下正确的描述是 B 。

A.函数的定义可以嵌套，但函数的调用不可以嵌套

B.函数的定义不可以嵌套，但函数的调用可以嵌套

C.函数的定义和调用均不可以嵌套

D.函数的定义和调用均可以嵌套

§5 函数嵌套调用

例2:函数的嵌套调用： 求圆环的面积

```
#include <math.h>
#define PI  3.1415926
float area_ring(float,float);
float area(float);
main( )
{float r,r1;
  printf(“input two radius:\n);
  scanf(“%f,%f”&r,&r1);
  printf(“area_ring is %f”,area_ring(r,r1));
}
float area_ring(float x,float y)
{ float c;
   c=fabs(area(x)-area(y));
   return c; }
```

```
float area(float r)
{
  return PI*r*r;
}
```

§5 函数嵌套调用

例3:编写函数mysum用以求: $\sum_{i=0}^{n} f(i)$,其中f(i)=i+5

```
#include <stdio.h>
int mysum(int);
int f(int);
main( )
{ int n;
  printf("Input data:");
  scanf("%d",&n);
  printf("%d,%d\n",n,mysum(n));
}
```

```
int mysum(int n)
{
  int i,sum=0;
  for(i=0;i<=n;i++)
     sum=sum+f(i);
  return sum;
}
int f(int i)
{ return i+5;
}
```

§5 函数嵌套调用

例4:求$1^k+2^k+3^k+...+n^k$的值，假设k为4，n为6

```
#include <stdio.h>
int power(int m,int k)
{ int j,p=1;
  for(j=1;j<=k;j++) p=p*m;
  return p; }
```

```
int add(int n,int k)
{ int i,s=0;
  for(i=1;i<=n;i++) s+=power(i,k);
  return s;
}
```

```
void main( )
{ int sum,n=6,k=4;
  sum=add(n,k);
  printf("1^4+2^4+3^4+4^4+5^4+6^4=%d\n",sum);
}
```

§6 变量存储类

一、变量存储类别含义

变量生存期

指变量内存单元存在时间有多长

变量作用域

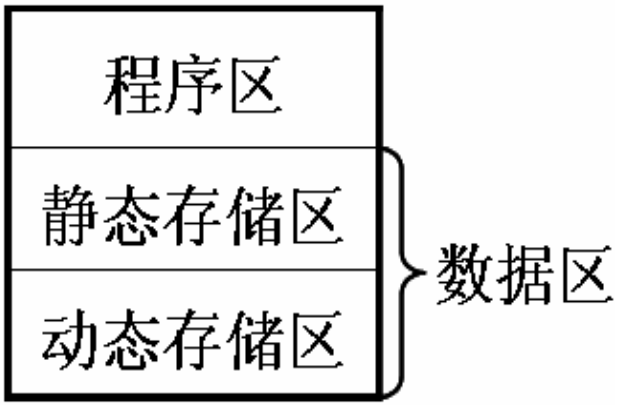
指变量数值可被存取的范围有多大

存储类型

- 局部变量—auto(定义)
- 全局变量—extern (声明)
- 静态变量—static (定义)
- 寄存器变量—register (定义)

§6 变量存储类

二、程序的用户存储区



静态存储区： 变量在整个程序执行期间占据固定存储单元

动态存储区： 在程序执行期间根据函数调用动态为变量分配存储单元

§6 变量存储类

三、局部变量

- 局部变量是指函数体内定义的变量及形式参数
  - —生存期：同定义它的函数执行期，为自动动态分配
  - —作用域：只限于定义该变量的函数内访问

```
eg1: main( )
     { int i,j,k;
       ……
     }
     float fun(float f1)
     { int i,j,k,m;
      ……
     }
```

```
eg1: main( )
     { auto int i,j,k;
       ……
     }
     float fun(auto float f1 )
     { auto int i,j,k,m;
      ……
     }
```

§6 变量存储类

三、局部变量

- 局部变量是指函数体内定义的变量及形式参数
  - —生存期：同定义它的函数执行期，为自动动态分配
  - —作用域：只限于定义该变量的函数内访问
  - — 块：定义有变量的复合语句

```
eg2: main( )
     { int m;
       …
       f(……);
     }

f(……)
 { float m;
     …
 }
```

```
eg3: main( )
     { int m,n;
       …
       f(…);
     }

f(…)
 {
     …
   m++;
 }
```

错误

§6 变量存储类

三、局部变量

eg4: 说出下列程序的运行结果

```
main( )
{ int i=1;
   { int i=2;
      { int i=3;
         printf("%d,",i);
       }
      printf("%d,",i);
    }
   printf("%d",i);
}
```

3,2,1

§6 变量存储类

四、全局变量

- 含义：在任何函数体外定义的变量，也称为外部变量
- 生存期：在静态存储区分配，等同于整个程序的执行期
- 作用域：隐含从定义点到源文件结束

  —在定义点前加以声明即可访问： extern 类型符 外部变量名;

```
int p=1,q=5;
float f1(int a)
{ …}
char c1,c2;
char f2(int x,inty)
{ …}
main( )
 { …}
```

p,q的作用域

c1,c2的作用域

```
extern char c1,c2;
int p=1,q=5;
float f1(int a)
{ …}
char c1,c2;
char f2(int x,inty)
{ …}
main( )
 { …}
```

p,q和c1,c2的作用域

§6 变量存储类

四、全局变量

- 含义：在任何函数体外定义的变量，也称为外部变量
- 生存期：在静态存储区分配，等同于整个程序的执行期
- 作用域：隐含从定义点到源文件结束
  - —在定义点前加以声明即可访问： extern 类型符 外部变量名;
  - —在同一程序的其它源文件中：用extern 声明后，即可访问

```
int p=1,q=5;
extern char c1,c2;
float f1(int a)
{ …}
char c1,c2;
char f2(int x,inty)
{ …}
main( )
 { …}
```

f1.c

```
extern char c1,c2;
extern int p,q;
float f3(float x,float y)
{ …}
void getl( )
{ …}
char *copy(char c)
 { …}
```

f2.c

§6 变量存储类

例1:有成绩数组score[10],编子函数求出平均、最高和最低成绩

```
float Max,Min;
float average(float arr[ ],int n)
{int i;
 float aver,sum=arr[0];
 Max=Min=arr[0];
for(i=1;i<n;i++)
{ sum+=arr[i];
 if(arr[i]>Max) Max=arr[i];
 else if(arr[i]<Min) Min=arr[i]; }
aver=sum/n; return aver;
}
```

```
main( )
{ float aver,score[10];
   int i;
   for(i=0;i<10;i++)
     scanf("%f",&score[i]);
   aver=average(score,10);
   printf("Max=%.2f,",Max);
   printf("Min=%.2f,",Min);
   printf("Ave=%.2f",aver);
}
```

§6 变量存储类

例2:阅读下列程序运行结果

```
int a=13,b=-8;
main( )
{ int a=8;
  printf("%d",max(a,b);
}
max(int a,int b)
{ int c;
  c=a>b?a:b;
  return c;
}
```

若外部变量与内部变量同名，则在内部变量的作用域内，外部变量失去定义

8

§6 变量存储类

例3(考题):程序输出第一行是_____，第二行是_____，第三行是_____。

```
int i,j=2;
void p(void)
{ for(i=0;i<4;i++)
   { printf("%d",++j);
     if((i+1)%2==0) printf("\n"); }
}
main( )
{ for(i=0;i<3;i++)  p( );
   printf("%d\n",i);
}
```

§6 变量存储类

五、静态变量

在变量类型定义前加static，即定义了静态变量

静态局部变量:局部变量定义前加static定义

静态全局变量:全局变量定义前加static定义

静态全局变量

- **生存期**：分配在静态存储区，同程序执行期
- **作用域**：仅限于定义该变量的源文件内各函数可访问

§6 变量存储类

五、静态变量

在变量类型定义前加static，即定义了静态变量

静态局部变量:局部变量定义前加static定义

静态全局变量:全局变量定义前加static定义

静态局部变量

- 作用域：仅限于定义它的函数内
- 生存期：同整个程序执行期
- 在多次函数调用中，保持值的连续性

§6 变量存储类

例1:阅读下列程序运行结果

```
void v( )
{ int a=0;
  static int s=0;
  printf("a=%d,s=%d\n",a,s);
  ++a;  ++s;
}
main( )
{ int i;
  for(i=0;i<5;i++) v( );
}
```

a=0,s=0
a=0,s=1
a=0,s=2
a=0,s=3
a=0,s=4

§6 变量存储类

例2：编程求：$sum = \sum_{i=1}^{5} i!$

```
#include <stdio.h>
int fac(int n)
{ static int t=1;
   t=t*n;   /*将t中值由(n-1)!变为n!*/
   return(t);
}
main( )
{ int i,sum=0;
  for(i=1;i<=5;i++)
    sum+=fac(i);
 printf("sum=%d\n",sum);
}
```

§6 变量存储类

例3（考题）：下列程序运行后，第一行 输出 是____，第二行 输出 是____。

```
int f(int x)
{ int y=1;
   static int z=1;
   z+=z+y++;
   return z+x; }
main( )
{ printf("%d\n",f(3));
  printf("%d\n",f(3));
}
```

§6 变量存储类

例4（考题）：下列程序运行结果是____。

```
main( )
{ int a=1,b=2,x1,x2,x3;
   x1=add(a,b);
   x2=add(add(a,b),b);
   x3=add(a,b);
   printf("%d\n%d\n%d\n",x1,x2,x3); }
int add(int x,int y)
{ static int z=1;
  z=x+y+z;
  return z; }
```

§6 变量存储类

六、寄存器变量

在变量类型定义前加register，即定义了寄存器变量

寄存器变量只适用于局部变量，不适用于外部变量和静态变量

寄存器变量之值存放在CPU通用寄存器中

register仅起建议作用，实际分配由编译系统决定

```
eg: int i;              /*i是局部变量*/
    register int j;     /*j是寄存器变量*/
```