西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C++程序设计
Programming in C++

1011018

主讲：魏英，计算机学院

作用域、生命期和程序的组织结构

- 1、局部变量和全局变量
- 2、作用域
- 3、生命期

- 在函数内部或复合语句中（简称区域）定义的变量，称为局部变量（local variable），又称为内部变量。

- 下列变量是局部变量：
  - ① 在一个函数内部定义的变量
  - ② 函数的形式参数
  - ③ 在某个复合语句中定义的变量

```
1    int f1(int x,int y) // f1函数
2    {
3        int a,b,m=100;
4        if (x>y) {
5            int a,t;
                ⋮
20       }
         ⋮
35   }
36   int main() // main函数
37   {
38       int a=15,b=10,c,n=200;
39       c = f1(a,b);
            ⋮
60   }
```

a,t有效

a,b,m有效

x,y有效

a,b,c有效

- 局部变量的说明。
- （1）局部变量只能在定义它的区域及其子区域中使用。
- （2）在同一个区域中不能定义相同名字的变量。
- （3）在不同区域中允许定义相同名字的变量，但本质上它们是不同的变量
- （4）如果一个变量所处区域的子区域中有同名的变量，则该变量在子区域无效，有效的是子区域的变量，称为定义屏蔽。

```
1    int f1(int x,int y) // f1函数
2    {
3        int a,b,m=100;
4        if (x>y) {
5            int a,t;
             a=100;
20       }
         ⋮
35   }
36   int main() // main函数
37   {
38       int a=15,b=10,c,n=200;
39       c = f1(a,b);
           ⋮
60   }
```

a,t有效

a,b,m有效

x,y有效

a,b,c有效

- 在源文件中，但在函数外部定义的变量，称为全局变量（global variable），全局变量的有效区域是从定义变量的位置开始到源文件结束。

12.1 局部变量和全局变量

```
1    int m=10,n=5;
2    int f1(int x,int y) // f1函数
3    {
4        int m=100;
            ⋮
10       x = m + n;
            ⋮
20   }
21   int a=8,b=4;
22   int main() // main函数
23   {
24       int a=15,n=200,x;
25       x = a + b + m + n;
            ⋮
30   }
        ⋮
        ⋮
```

x,y,m 有效

a,n,x 有效

a,b有效

m,n有效

文件结束行

- 函数之间数据传递尽管可以利用全局变量，但这样一来也导致两个函数彼此分不开，违背模块化的原则，所以结构化程序设计

- 提倡少用或不用全局变量。

【例12.1】编写一个函数swap用于交换两个整数的值——方法1

```
#include <iostream>
using namespace std;
void swap(int x,int y)
{  int t;     t=x; x=y; y=t;  }
int main( )
{
    int a,b;
    cin>>a>>b;  swap(a,b);
    cout<<"a="<<a<<",b="<<b<<endl;
    return 0;
}
```

运行结果：

```
3  4↙
a=3, b=4
```

【例12.1】编写一个函数swap用于交换两个整数的值——方法2

```
#include <iostream>
using namespace std;
int x,y;
void swap()
{  int t;     t=x; x=y; y=t;  }
int main( )
{
    int a,b;  cin>>a>>b;
    swap(a,b); cout<<"a="<<a<<",b="<<b<<endl;
    return 0;
}
```

运行结果：

```
3 4 ↙
a=4, b=3
```

- C++的实体通常有三类：
- ①变量或对象。例如变量、数组等；
- ②函数；
- ③类型。包含结构体类型、共用体类型、类类型。

- 作用域是程序中的一段区域。在同一个作用域上，C++程序中每个名字都与唯一的实体对应；如果在不同的作用域上，程序中可以多次使用同一个名字，对应不同作用域中的不同实体。

- 作用域分有：
- （1）文件作用域（file scope）
- （2）函数作用域（function scope）
- （3）块作用域（block scope）
- （4）类型声明作用域（declaration scope）
- （5）函数原型作用域（function prototype scope）

- 实体在作用域内可以使用称为可见（visible），又称有效。可见的含义是指实体在作用域上可以使用。

- 下面给出C++实体可见规则。

- （1）规则一。同一个作用域内不允许有相同名字的实体，不同作用域的实体可以有相同名字。

- （2）规则二。实体在包含它的作用域内，从定义或声明的位置开始，按文件行的顺序往后（往下）直到该作用域结束均是可见的，包含作用域内的所有子区域及其嵌套。

- （3）规则三。若实体A在包含它的作用域内的子区域中出现了相同名字的实体B，则实体A被屏蔽。

```
int main()
{  int a,b,c;
    c=2;
     {  int c;
        .....
        c=a+b;
        .....
     }
    c=c+1;
    return 0;
}
```

- （4）规则四。可以使用extern声明将变量或函数实体的可见区域往前延伸，称为前置声明（forward declaration）。
- extern声明变量实体的形式为：

```
extern 类型 变量名, ......
```

- extern声明函数原型的形式为：

```
extern 返回类型 函数名(类型1 参数名1, ......);
extern 返回类型 函数名(类型1,......);
```

- （5）规则五。在全局作用域中，变量或函数实体若使用static修饰，则该实体对于其他源文件是屏蔽的，称为私有的（private）。
- static修饰变量实体的形式为：

```
static 类型 变量名[=初值], ......
```

- static修饰函数原型的形式为：

```
static 返回类型 函数名(类型1 参数名1, ......);
static 返回类型 函数名(类型1, ......) ;
```

【例12.2】作用域举例。

```
1  // FILE1.CPP 全局作用域
2  int a=1 , b=2;      //全局变量
3  int c=10 , d=11;     //全局变量
4  void f1(int n,int m)     //f1函数作用域
5  {
6       int x=21, y=22,z=23;  //f1局部变量
7       extern int h,k;//正确, h=60 k=61 规则四
8       n = n + t ;      //错误, t 违反规则二
9       if (n>100) {  //块作用域
10           int x=31,t=20; //复合语句局部变量
11           n=x+y; //正确, n=31+22 规则二 规则三
12           if (m>10) {  //嵌套块作用域
13            int y=41;//嵌套的复合语句局部变量
14            n=x+y;//正确, n=31+41 规则二 规则三
15           }
16      }
```

```
17          n = a + x ;       //正确，n=1+21   规则二
18          m = e + f ;       //错误，e,f 违反规则二
19          n = h + k ;       //正确，n=60+61  规则四
20      }
21      int e=50 , f=51;       //全局变量
22      int h=60 , k=61;       //全局变量
23      void f2(int n,int m)      //f2函数作用域
24      {
25          n=a+b+e+f;   //正确，n=1+2+50+51 规则二
26          m=z;         //错误，z 违反规则一
27      }
28      int f3(int n,int m)      //f3函数作用域
29      {
30          return n+m;      //正确，规则二
31      }
```

```
32  int f4(int n,int m)     //f4函数作用域
33  {
34      return n-m;     //正确，规则二
35  }
36  // FILE1.CPP 文件结束
```

```
1   // FILE2.CPP 全局作用域
2   int a=201 , b=202;
    //错误，连接时与FILE1.CPP的同名 违反规则一
3   void f1(int n,int m)
    //错误，连接时与FILE1.CPP的同名 违反规则一
4   {
5       n=n*m;
6   }
7   static int c=210 , d=212;    //正确，规则五
8   static void f2(int n,int m) //正确，规则五
9   {
10      n=n/m;
11  }
```

```
12  extern int h , k;      //正确，规则四
13  extern int f4(int n,int m); //正确，规则四
14  int main()
15  {
16      int p,q,r;//main函数局部变量
17      p = c + d;//正确，p=210+212 规则二
18      f2(-1,-2);//正确，不是FILE1.CPP的 规则二
19      q=e+f;//错误，试图使用FILE1.CPP违反规则一
20      f3(-10,-12);
21      r = h + k;      //正确，r=60+61 规则四
22      f4(-20,-22);      //正确，规则四
23      return 0;
24  }
25  // FILE2.CPP 文件结束
```

- C++中，每个名字都有作用域，即可以使用名字的区域，而每个对象都有生命期（lifetimes），即在程序执行过程中对象存在的时间。

- 1. 动态存储
- 动态存储（dynamic storage duration）是指在程序运行期间，系统为对象动态地分配存储空间。动态存储的特点是存储空间的分配和释放是动态的，要么由函数调用来自动分配释放，要么由程序指令来人工分配释放，这个生命期是整个程序运行期的一部分。
- 动态存储的优点是对象不持久地占有存储空间，释放后让出空闲空间给其他对象的分配。

- 动态存储在分配和释放的形式有两种，一种是由函数调用来自动完成的，称为自动存储（automatic storage），一种是由程序员通过指令的方式来人工完成的，称为自由存储（free storage）。

- 2. 静态存储
- 静态存储（static storage duration）是指对象在整个程序运行期持久占有存储空间，其生命期与程序运行期相同。静态存储的特点是对象的数据可以在程序运行期始终保持直到修改为止，或者程序结束为止，静态存储的分配和释放在编译完成时就决定好了。

- 现代程序设计的观点是，除非有必要尽量少地使用静态存储。

- 3. 自动对象
- 默认情况下，函数或复合语句中的对象（包含形参）称为自动对象（automatic objects），其存储方式是自动存储，程序中大多数对象是自动存储。

```
auto 类型 变量名[=初值], ......
```

- 4. 寄存器变量
- C++语言允许用CPU的寄存器来存放局部变量，称为寄存器变量。在局部变量前加上register存储类别修饰来定义的，其形式为：

```
register 类型 变量名[=初值], ......
```

- 5）静态局部对象
- 在局部对象的前面加上static存储类别修饰用来指明对象是静态局部对象（static local object），一般形式为：

```
static 类型 变量名[=初值] , ......
```

【例12.3】局部静态变量举例。

```
#include <iostream>
using namespace std;
int fun()
{
  static int cnt=0; //静态局部变量会保持其值
  cnt++;
  return cnt;
}
int main()
{
  int i,c;
  for (i=1;i<=10;i++) c=fun();
  cout<<c<<endl;
  return 0;
}
```

C程序设计