C++程序设计

第二讲 4.6 存储类别和作用域

版本号：V2023.08

4.6 存储类别和作用域

变量的**作用域**：是指变量在程序中有效的空间范围，变量的作用域与定义变量的位置有关；
变量的**生存期**：指的是变量生成到变量空间释放的时间范围，变量的生存期与变量的存储类别有关。

C++可将作用域分为五种：块作用域、文件作用域、函数原型作用域、函数作用域和类作用域。

1. 块作用域

块：就是用花括号括起来的那部分程序。在块内阐明的标识符只能在该块内引用，即其作用域在块内。全部函数体内定义的变量都具有块作用域。

在一种函数内或块内定义的变量被称为局部变量；对于块中嵌套其他块的情况，假如嵌套块中有同名局部变量，服从局部优先原则，即在内层块中屏蔽外层块中的同名变量。

在**函数原型作用域** 和**函数作用域**内定义的变量 都属于局部变量；

【例4.12】局部变量的屏蔽。

```
#include <iostream.h>
void main()
{       int i=5;
        {       cout<<"i1="<<i<<'\n';           //A
                int i;
                i=7;
                cout<<"i2="<<i<<'\n';           //B
        }
        cout<<"i3="<<i<<'\n';                   //C
        return;
}
```

运营成果是：

```
i1=5
i2=7
i3=5
```

2. 文件作用域

在全部函数外定义的变量称为全局变量，全局变量的作用域称为文件的作用域；
当局部变量与全局变量同名时，局部变量（块作用域内的变量）优先。它与块作用域不同的是：在块作用域内可经过域运算符"：："来引用与局部变量同名的全局变量。

【例4.13】在块作用域内引用文件作用域中的同名全局变量。

```
#include <iostream.h>
int i=0;                        //A
void main()
{       int i=5;                //B
        {       cout<<i;        //C
                int i=7;        //D
                cout<< i; //E
                cout<<::i;//F
        }
}
```

程序运营后成果为：

570

4．6．2 变量的存储类别

操作系统为一种C++程序的运营所分配的内存分为4个区域，如图4.2所示。

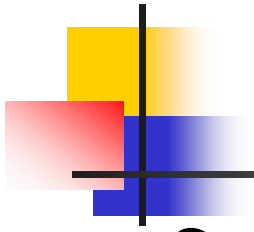
代码区：存储程序代码，即程序中各个函数的代码块

静态数据区：存储程序的全局数据和静态数据

栈区：存储程序中的局部变量，如函数中的局部变量等

堆区：存储动态分配的数据。

图4.2 程序在内存中的区域



C++使用四种阐明符auto(自动类型)、register(寄存器类型)、static(静态类型)和extern(外部类型)来拟定变量的存储类型。加上变量的作用域的不同，在C++中变量共有下列5种存储形式：

局部变量、全局变量、外部变量、静态变量、寄存器变量；

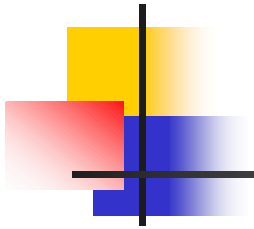
1、 自动存储变量（局部变量，属于auto型）

在函数或块内部阐明的变量称为局部变量(如形参变量)，因为建立和撤消这些变量是由系统自动完毕，所以此类变量又称为自动型变量。

注意：

a：局部变量仅由阐明它的函数或块的内部语句所访问；换言之，局部变量在定义自己的函数或块之外是不可见的。

b：局部变量在进入函数时生成，退出时消灭。所以局部变量的生存期仅存在于被定义的函数或块中。

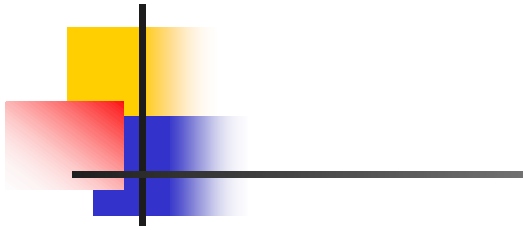


```
#include<iostream.h>
void func1( );
void func2( );
```

```
void func1( )
{       int  x;
        x = 10;
        cout<<x<<'\t';
}
```

```
void main(void )
{       int  x = 20;
        func1( );
        func2( );
        cout<< x;
}
```

```
void func2( )
{       int  x;
        x = -199;
        cout<<x<<'\t';
}
```

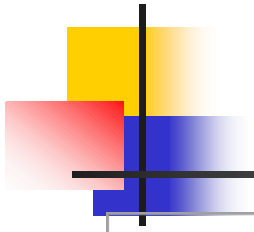


程序阐明：

整形变量x被阐明了3次，一次在func1中， 另一次在func2和main中, func1和func2中旳x互不有关，也就是说，局部变量旳值不能在两次调用之间保持。在C语言中，必须在函数或语句块旳开头先于其他任何语句阐明全部旳局部变量。但在C++中，只需在变量使用之前阐明就能够了。(成果为： 10 -199 20)

2、全局变量

与局部变量不同，全局变量在任何函数之外阐明，涉及main函数，而且可被任何一种函数使用。全局变量将在整个程序运营期间保持有效，同步可被函数内的任何体现式访问。阐明全局变量的最佳位置在程序的一开始。

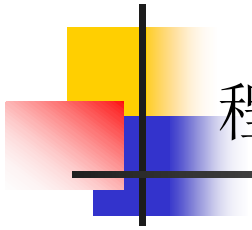


```
#include<iostream.h>
int count;
void func1( );
void func2( );
```

```
void func1( )
{  int  temp;
   temp = count;
   func2( );
   cout<<“count is: ” << temp;
}
```

```
void main(void)
{  count = 100;
   func1( );
   exit(0);
}
```

```
void func2( )
{  int  count;
 for(count=1;count<10;count++)
   cout.put(‘。’);
}
```



程序分析：

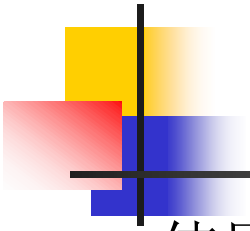
全局变量count既不在main( )中阐明，也不在func1( )中阐明，但能够在两者中被访问。函数func2中也阐明了一种同名的局部变量count，但func2访问count时，仅访问自己阐明的局部变量count，而不是那个全局变量count。

注意：

全局变量和某一函数的局部变量同名时，在函数内对该名的访问权仅针对局部变量，对全局变量无任何影响。

程序执行成果为：

。。。。。。。。。。count is 100　　// 9个“。”



使用全局变量应注意的几点：

- 不论是否需要，全局变量在整个程序执行期间均占用固定的内存空间。

- 在使用局部变量就能够到达其功能时，应防止使用全局变量，这么能够提升函数的通用性。

- 大量使用全局变量时，可能造成不期望的副作用。例如，变量值在程序其他地点的偶尔变化，将在程序的整个生存期内有效。

3、外部变量

因为C++语言支持分离编译，既一种完整旳C++语言程序能够由多种独立旳文件构成，每个文件能够分别单独编译，然后再由链接程序完毕程序组装。所以必须将程序需要旳全部全局变量经过某种措施告诉全部旳独立文件。措施就是在一种文件中阐明全部全局变量，在其他文件中用extern再次描述它们。因另外部变量定义是：**由extern阐明旳全局变量称为外部变量。**