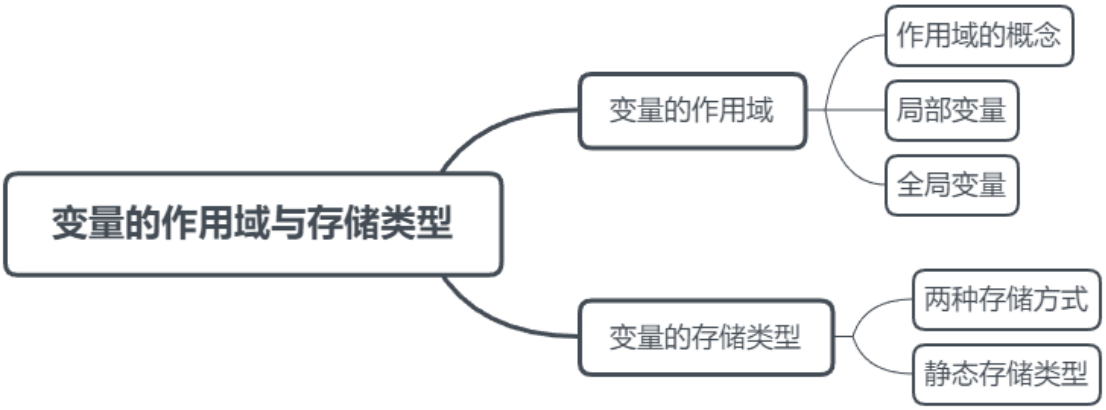
专题七 变量的作用域与静态存储类型

【内容预览】



【知识清单】

7.1、变量的作用域

一、变量的作用域定义

变量的作用域其实是一个范围，在这个范围中，被定义的变量是可用的，可以被系统识别，如果使用的变量超过这个范围，则编译系统会指出所使用的变量没有定义。变量的作用域依赖于变量的位置，有两种情况：

（1）函数内部定义的，称为局部变量；

（2）函数外部定义的，称为全局变量；

二、局部变量

局部变量是一个函数内定义的变量，只能在本函数中可以使用，在函数外不能使用该变量。还有一种局部变量是在复合语句块内部定义说明的变量，其作用域仅限于所在的块内，在变量的作用域以外该变量不能被引用。

如：

```
int main(){
    int x = 10;
    cout << "x=" << x << endl;    //x=10
    if(x > 0){
        int x;
        x = -10;
        cout << "x=" << x << endl; //x=-10
    }
    cout << "x=" << x << endl;     //x=10
    return 0;
}
```

复合语句

（1）主函数 main()中定义的变量也只有在 main()函数中有效。在主函数中也不能使用其它函数中定义的变量。

（2）不同函数中也可以使用相同名字的变量，它们代表不同的对象，互不干扰。

（3）函数的形式参数也是一种局部变量。

（4）在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效，离开本复合语句变量无效。这种复合语句称为“分程序”或“程序块”。

三、全局变量

全局变量是在函数外部定义的变量，其作用域是从定义位置开始到本源文件结束。它不属于哪一个函数而属于一个源程序文件。

说明：

（1）可以利用全局变量从被调函数中返回多个值。

（2）如果没有指定初值，则全局变量的初值是变量数据类型的系统默认值。**int 类型为 0，float 和 double 类型为 0.0，char 类型为‘\0’,指针类型为 NULL。**

（3）若函数中的局部变量与全局变量同名，则在该函数中全局变量不起作用。

（4）全局变量浪费内存空间，因为全局变量在程序的全部执行过程中都占用内存单元，而不是在需要时才分配。

（5）全局变量使函数的通用性降低，定义的全局变量在某一函数中发生改变，则该变量的值将影响到其它函数。如果将一个函数移到另一个文件中，还要将有关的全局变量，及其值一起移过去。但若该全局变量与其他文件的变量同名时，就会出现问题，降低了程序的可靠性和通用性。在程序设计中，在划分模块时要求模块的“内聚性”强，与其它模块的“耦合性”弱。即模块的功能要单一，与其它模块的互相影响要小，而用全局变量违背了这个准则。

（6）过多地使用全局变量不仅加大了系统开销，而且使程序的可读性降低。

7.2、变量的静态存储类型

一、静态存储方式和动态存储方式

从变量值存在于内存中的时间（即生存期）角度来分：

静态存储方式：指在程序运行期间分配固定的存储空间的方式；

动态存储方式：指在程序运行期间根据需要动态地分配存储空间的方式，是变量的默认存储方式。

（1）静态分配存储区——►静态变量：在整个程序结束以后才释放变量所占用的存储单元。

（2）动态分配存储区——►动态变量：在其所在函数执行结束时就释放变量所占用的存储单元。

二、变量的静态存储类型

static 和 extern 两个关键字都可以用来修饰声明静态变量，区别在于：

（1）用 static 声明局部变量

功能：有时候我们希望函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时该变量为上次函数调用结束时的值。这时我们可以使用 static 声明该局部变量。

静态局部变量在编译时被赋初值，即只赋一次初值；而对自动变量 auto 赋初值是在函数调用时进行，每调用一次函数重新赋值一次。

如：

```
#include<iostream>
using namespace std;
int f(int a){
   static int c = 3;
   c = c + 1;
   return c;
}
int main(){
   int a = 2, i;
```

```
    for(i = 0;i < 3;i++)
        cout << f(a) << ' ';
}                                    //结果为：4 5 6
```

（2）用 extern 声明外部变量

一般，我们在定义外部变量时省略类型标识符 extern，但有时需要用 extern 对该变量声明，以扩展外部变量的作用域。

a)在一个文件内声明外部变量

外部变量的作用域只限于从定义处到文件结束。如果在定义之前的函数想引用该外部变量，则**应该在引用之前用 extern 对该变量声明**，表示该变量是一个已经定义的外部变量。

如：

```
#include<iostream>
using namespace std;
int max(int x,int y){
    ……
}
int main(){
    extern A,B;
    ……
    return 0;
}
int A = 13, B = -8;          //外部变量
```

b)在多文件的程序中声明外部变量

如果一个程序包含两个文件，在两个文件中都要用到同一个外部变量“A”，不能分别在两个文件中各自定义一个外部变量 A，否则在程序链接时会出现“重复定义”的错误。**正确的做法是：在任一个文件中定义外部变量 A，而在另一个文件中用 extern 对 A 作外部变量声明，即“extern A；”**。在编译和链接时，系统会把在一个文件中定义的外部变量的作用域扩展到本文件，在本文件中也可以合法地引用外部变量 A。

如：

```
//file1.cpp
#include<iostream>
using namespace std;
int A;   //此处省略 extern
int main(){
    ……
}
//file2.cpp;
extern A;       //声明 A 为一个已定义的外部变量
int power(int n){
    ……
}
```

【解题技巧】

例 7-1 有以下程序

```
#include<iostream>
using namespace std;
int main(){
    int x = 5;
```

```
    cout << "x=" << x << '\t';
    if(x > 0)  {
        int x = 10;
        cout << "x=" << x << '\t';
     }
    cout << "x=" << x + 2;
    return 0;
    }
```

程序运行的结果为_____

正解：x=5　x=10　x=7

分析：第一个 printf 语句中的 x 是在主函数一开始定义的，所以 x=5. 第二个 printf 语句中的 x 是在 if（）里面定义的，此时应看作另一个 x，记为 x'，其值为 10。第三个 printf 语句中的变量 x 是在一开始说明的，因为 x' 已经释放，所以，结果为 x=7；

【精选习题】 答案 P117

基础篇

1、以下程序运行的结果为_____

```
#include<iostream>
using namespace std;
void f(){
   int a = 0;
   cout << ++a << '\t';
   }
int main(){
   int a = 10;
   f();
   cout << a++ << '\t';
   f();
   return 0;
}
```

提高篇

1、以下程序运行的结果为_____

```
#include <iostream>
using namespace std;
int f(int n){
   static int k=1;
   k=k*n;
   return k;
}
int main(){
   int i,sum=0;
   for(i=1;i<=5;i++)
      sum=sum+f(i);
   cout << "sum=" << sum << endl;
```

```
    return 0;
}
```

2、有以下程序

```
#include <iostream>
using namespace std;
int fun(){
    static int x=1;
    x*=2;
    return x;
}
int main(){
    int i,s=1;
    for(i=1;i<=3;i++) s*=fun();
    cout << s << endl;
    return 0;
}
```

程序运行的结果为_____

3、以下程序运行的结果为_____

```
#include<iostream>
using namespace std;
int a=1;
void func(){
    static int x=1;
    int y=2;
    x=x+1;
    a=a+2;
    y=y+a;
    cout << "func:x=" << x << ",y=" << y << ",a=" << a << endl;
}
int main(){
    static int x=2;
    int y;
    y=a;
    cout << "main:x=" << x << ",y=" << y << ",a=" << a << endl;
    func();
    cout << "main:x=" << x << ",y=" << y << ",a=" << a << endl;
    func();
    return 0;
}
```