CS 1502H 程序设计思想与方法 (C++)
Principles and Methodology in Programming
Chapter 6. 函数-2

陈东尧
上海交通大学
2025年秋季

内容回顾

```
void swap(int x, int y);
int main()
{
    int a(5),b(8);
    cout << "a="<<a<<", b="<<b<< endl;
    swap(a,b);
    cout << "a="<<a<<", b="<<b<< endl;
    return 0;
}
```

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

这个代码是错误的，swap中的操作对外界（主调函数）没有影响

本章大纲

- 函数的用途和定义
- 函数的使用
- 变量的作用域与存储类别
- 函数参数—数组和默认值
- 内联、重载、模版、递归

变量也存在着空间和时间的限制

- 空间限制：变量的作用域（scope）
- 时间限制：变量的生存期限（lifetime）

变量的作用域（Scope）

- 一个变量能被存取的程序部分，称为变量的作用域。
- 变量的作用域与程序块（block）有关。所谓的程序块是带有声明的复合语句。

局部变量 和 全局变量

- **局部变量（Local Variable）**
  - 在语句块内（函数、复合语句）定义的变量
  - main()函数中说明的变量也是局部变量
- **全局变量（Global Variable ）**
  - 在所有函数之外定义的变量
  - 作用范围：从定义位置到文件结束
  - 作用：方便函数间的数据传递
  - 如在作用范围外的函数要使用此变量:
    - 用关键词extern在函数内说明此全局变量

```
void Function();
int x = 1; int y = 2;

int main()
{   int i=0;
    cout <<x << “, ”<<y<<endl;
    Function();
    return 0;
}

void  Function()
 {   int a = 0;
     cout <<x << “, ”<<y<<endl;
 }
```

全局变量

局部变量

局部变量

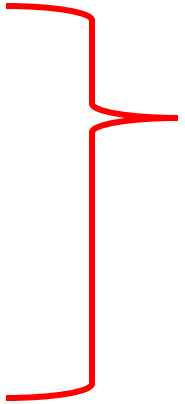
变量的作用域

- 在块（block）中说明的变量是局部的，仅能在本块中和内部的块中存取。
- 当内部块与外部块有同名变量时，在内部块中屏蔽外部块的同名变量。
- 在一个函数中，我们不能存取主调函数（caller）的变量，即使知道该变量的名字。

局部变量的作用域

- 仅能在定义它的语句块（包括其下级语句块）内访问

```
int main()
{   int sum = 0, m;
    for (int i=0; i<5; i++)
    {
        cout<< "input a nmuber:";
        cin>>m;
        sum = sum + m;
    }
    cout << "sum = "<< sum << endl;
    return 0;
}
```

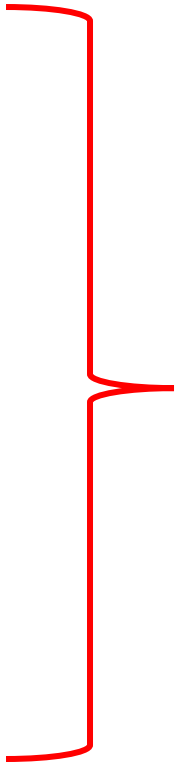


int i 的作用域

C++ 允许在for循环的初始部分定义变量。

全局变量的作用域

```
void Function();
int x = 1; int y = 2;

int main()
{    int i=0;
     cout <<x << “, ”<<y<<endl;
     Function();
     return 0;
}

void  Function()
 {    int a = 0;
      cout <<x << “, ”<<y<<endl;
 }
```



全局变量的作用范围：从定义变量的位置开始，到本程序结束.

变量名同名 ---局部变量与全局变量同名

```
void Function();
int x = 1; int y = 2;

int main()
{
    cout <<"x="<<x<<"  y=" <<y<<endl;
    Function();
    return 0;
}

void  Function()
{
    int x=2, y=1 ;
    cout <<"x="<<x<<"  y=" <<y<<endl;
}
```

局部变量隐藏全局变量,互不干扰

```
x=1   y=2
x=2   y=1
```

变量名同名 ---形式参数与全局变量可以同名

```
void Function(int x,  int y) ;
int x = 1; int y = 2;

int main()
{
    cout <<"x="<<x<<"  y=" <<y<<endl;
    Function(x,y);
    return 0;
}

void Function(int x,  int y)
 {
     x=2; y=1 ;
     cout <<"x="<<x<<"  y=" <<y<<endl;
 }
```

局部变量隐藏全局变量,互不干扰

```
x=1   y=2
x=2   y=1
```

变量名同名 ---实际参数与形式参数可以同名

```
void Function(int x, int y) ;

int main()
{
    int x = 1; int y = 2;
    cout <<"x="<<x<<"  y=" <<y<<endl;
    Function(x,y);
    return 0;
}

void Function(int x, int y)
 {
     x=2; y=1 ;
     cout <<"x="<<x<<"  y=" <<y<<endl;
 }
```

- 形参值改变不影响与其同名的实参值

```
x=1   y=2
x=2   y=1
```

变量名同名 ---形式参数不可以和局部变量同名

```
1   #include <iostream>
2   using namespace std;
3   void Function(int x,  int y) ;
4
5   int main()
6   {
7       int x = 1; int y = 2;
8       cout <<"x="<<x<<"  y=" <<y<<endl;
9       Function(x,y);
10      return 0;
11  }
12
13  void Function(int x,  int y)
14  {
15      int x=2;
16      int y=1 ;
17      cout <<"x="<<x<<"  y=" <<y<<endl;
18  }
```

- 形式参数不可以和局部变量同名
- 如果同名、编译器会报错

```
x=1   y=2
x=2   y=1
```

Line	Message
15	error: declaration of 'int x' shadows a parameter
16	error: declaration of 'int y' shadows a parameter

假如变量名同名…

- 问题：假如同名变量出现在同一个(级)作用域中？
  - 编译器将报错
  - 编译器只能区分不同作用域中的同名变量

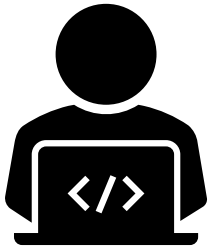
- 只要同名的变量出现在不同的作用域内
  - 二者互不干扰
  - 编译器有能力区分不同作用域中的同名变量

假如变量名同名 --- 全局变量与局部变量重名

- 如果目前的程序块中有局部变量与程序的全局变量重名，那么如何调用全局变量？
  - 解决方法：使用作用域限定符::

```
// C++ program to show that we can access a global
// variable using scope resolution operator :: when
// there is a local variable with same name
#include<iostream>
using namespace std;

// Global x
int x = 0;

int main()
{
        // Local x
        int x = 10;
        cout << "Value of global x is " << ::x;
        cout<< "\nValue of local x is " << x;
        return 0;
}
```

```
int p = 1, q = 5, r=3; //全局变量(global variable)
void f1(); void f2(); void f3();

int main()
{
	f3(); cout << "after f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f1(); cout << "after f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f2(); cout << "after f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	return 0;
}
void f1()
{
	int p = 3, r = 2;
	q=p+q+r; cout << "f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f2()
{
		p=p+q+r; cout << "f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f3()
{
	int q;
	r = 2*r; //直接操作全局变量会改变全局变量值!
	q=r+p;
	cout << "f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
```



代码演示

chapter6-local-global-variable-02.cpp

```
int p = 1, q = 5, r=3; //全局变量(global variable)
void f1(); void f2(); void f3();

int main()
{
	f3(); cout << "after f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f1(); cout << "after f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f2(); cout << "after f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	return 0;
}
void f1()
{
	int p = 3, r = 2;
	q=p+q+r; cout << "f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f2()
{
		p=p+q+r; cout << "f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f3()
{
	int q;
	r = 2*r; //直接操作全局变量会改变全局变量值!
	q=r+p;
	cout << "f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
```

f3: p=1 ,q=7, r=6

after f3: p=1 ,q=5, r=6

```
int p = 1, q = 5, r=3; //全局变量(global variable)
void f1(); void f2(); void f3();

int main()
{
	f3(); cout << "after f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f1(); cout << "after f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f2(); cout << "after f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	return 0;
}
void f1()
{
	int p = 3, r = 2;
	q=p+q+r; cout << "f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f2()
{
		p=p+q+r; cout << "f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f3()
{
	int q;
	r = 2*r; //直接操作全局变量会改变全局变量值!
	q=r+p;
	cout << "f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
```

f3: p=1 ,q=7, r=6
after f3: p=1 ,q=5, r=6
f1: p=3 ,q=10, r=2
after f1: p=1 ,q=10, r=6

```
int p = 1, q = 5, r=3; //全局变量(global variable)
void f1(); void f2(); void f3();

int main()
{
	f3(); cout << "after f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f1(); cout << "after f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f2(); cout << "after f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	return 0;
}
void f1()
{
	int p = 3, r = 2;
	q=p+q+r; cout << "f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f2()
{
		p=p+q+r; cout << "f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f3()
{
	int q;
	r = 2*r; //直接操作全局变量会改变全局变量值!
	q=r+p;
	cout << "f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
```

f3: p=1 ,q=7, r=6
after f3: p=1 ,q=5, r=6
f1: p=3 ,q=10, r=2
after f1: p=1 ,q=10, r=6
f2: p=17 ,q=10, r=6
after f2: p=17 ,q=10, r=6

注意事项⚠：全局变量的使用须知

- **作用：**
  - 当多个函数必须共享同一个变量时；
  - 当少数几个函数必须共享大量数据时；
- 全局变量破坏了模块化，建议尽量少使用！
- 当全局变量和局部变量同名时，在局部变量的作用范围中全局变量被屏蔽。如果要使用全局变量，可以使用作用域运算符::
- 全局变量的使用将在模块化设计中详细介绍

课堂练习

C++程序中，下列说法不正确的是（ ）

A. 不同函数中可以使用相同名字的变量

B. 函数中形式参数是局部变量

C. 在一个函数内定义的变量只在本函数范围内有效

D. 在一个函数内的复合语句中定义的变量在本函数范围内有效

课堂练习

C++程序中，下列说法不正确的是（ ）

A. 不同函数中可以使用相同名字的变量

B. 函数中形式参数是局部变量

C. 在一个函数内定义的变量只在本函数范围内有效

D. 在一个函数内的复合语句中定义的变量在~~本函数范围内有效~~复合语句中有效

```
#include <iostream>

int main()
{
std::cout<<"Hello World";
{
int a = 0;
}
std::cout << a;

return 0;
}
```

C++中变量的生存时限
(Variable Lifetime)

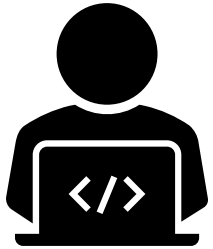
C++中的静态（static）变量

- 静态变量的特点在于变量的**生命周期**：在修饰变量的时候，static修饰的静态局部变量只执行初始化一次，而且延长了局部变量的生命周期，直到程序运行结束以后才释放，且不改变作用域。

【扩展阅读】C/C++中static的用法全局变量与局部变量：
https://www.cnblogs.com/33debug/p/7223869.html

例子：C++中的静态（static）变量

- 猜猜输出是什么？



代码演示

chapter6-static-variable-trivia.cpp

0 1 2 3 4

去掉static 会有何结果？

0 0 0 0 0

```
// C++ program to demonstrate
// the use of static Static
// variables in a Function
#include <iostream>
#include <string>
using namespace std;

void demo()
{
        // static variable
        static int count = 0;
        cout << count << " ";
        // value is updated and
        // will be carried to next
        // function calls
        count++;
}

int main()
{
        for (int i=0; i<5; i++)
                demo();
        return 0;
}
```

（如此有用的）静态变量在C++中是如何实现的？

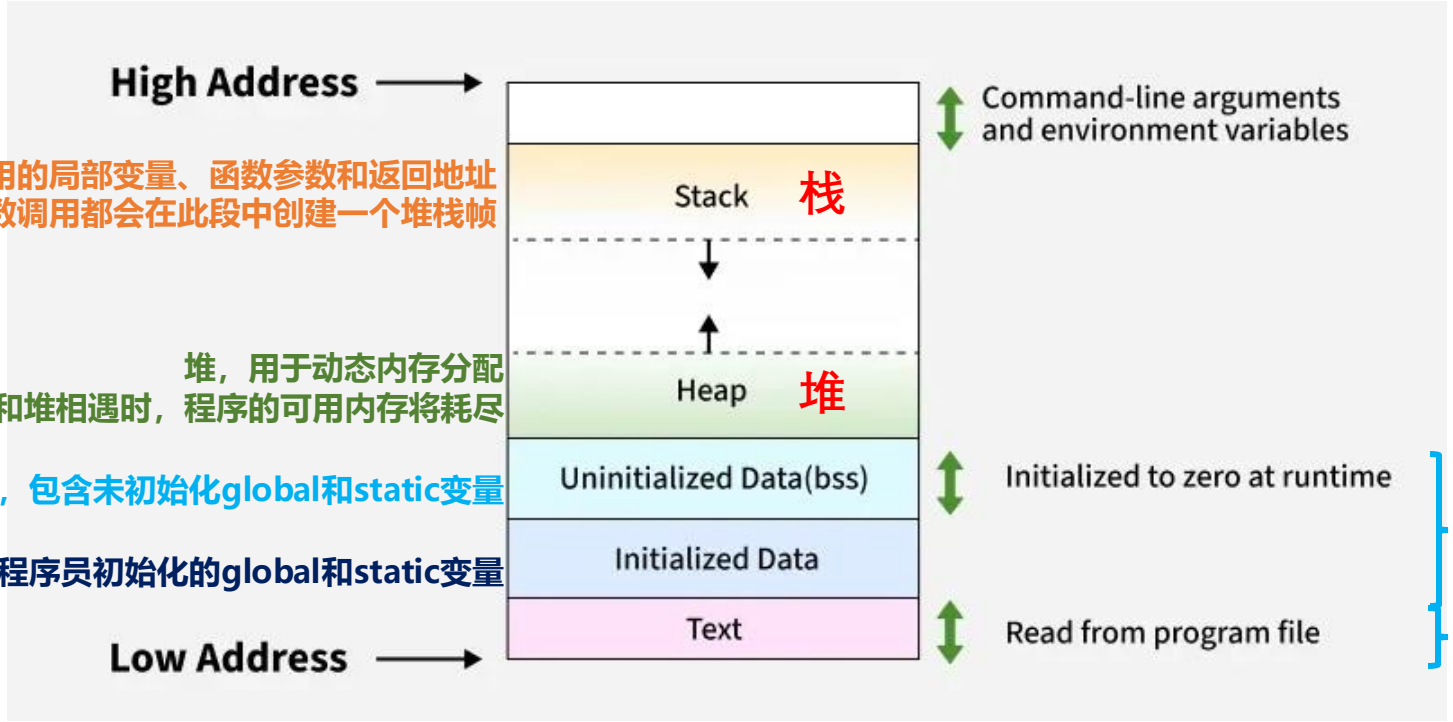
存储类别

- 在C++语言中， 每个变量有两个属性：
  - 数据类型：变量所存储的数据类型
  - 存储类别：变量的存储位置和期限
- 标准的变量定义：

  存储类别　数据类型　变量名；
- 存储类别：
  - 自动变量：auto
  - 静态变量：static
  - 寄存器变量：register
  - 外部变量：extern

深入理解存储类别：C++中的内存布局（Memory Layout）

- 知道内存布局对调试程序非常有帮助，可以知道程序执行时，到底做了什么



一个程序的虚拟内存地址空间

【扩展阅读】内存布局中的堆和栈区别:
https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

```cpp
#include <iostream>
using namespace std;

long Factorial(int n);

int main()
{
	int i, n;
	cout << "Input n:";
	cin >> n;
	for (i = 1; i <= n; i++){
		cout <<"Factorial("<<i<<")=" <<
		Factorial(i)<<endl;
	}
	return 0;
}

long Factorial(int n){
	static long p = 1;
	p = p * n;
	return p;
}
```

```cpp
#include <iostream>
using namespace std;

long Factorial(int n);

int main()
{
	int i, n;
	cout << "Input n:";
	cin >> n;
	for (i = 1; i <= n; i++){
		cout <<"Factorial("<<i<<")=" <<
		Factorial(i)<<endl;
	}
	return 0;
}

long Factorial(int n){
	static long p = 1;
	p = p * n;
	return p;
}
```

```
Input n:10
Func(1)=1
Func(2)=2
Func(3)=6
Func(4)=24
Func(5)=120
Func(6)=720
Func(7)=5040
Func(8)=40320
Func(9)=362880
Func(10)=3628800

Process returned 0 (0x0)
Press any key to continue.
```

```cpp
#include <iostream>
using namespace std;

long Factorial(int n);

int main()
{
	int i, n;
	cout << "Input n:";
	cin >> n;
	for (i = 1; i <= n; i++){
		cout <<"Factorial("<<i<<")=" <<
		Factorial(i)<<endl;
	}
	return 0;
}

long Factorial(int n){
	long p = 1;
	p = p * n;
	return p;
}
```

- 删除static关键字
- 静态局部变量改成自动变量，即局部变量

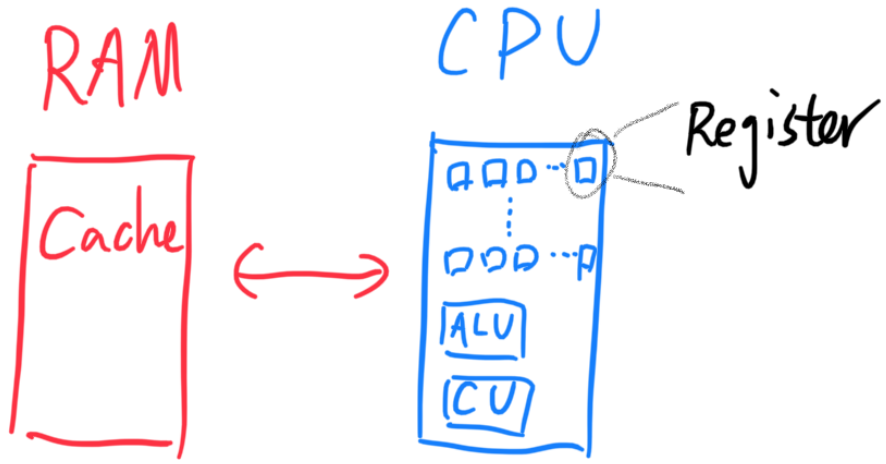
```
Input n:10
Func(1)=1
Func(2)=2
Func(3)=3
Func(4)=4
Func(5)=5
Func(6)=6
Func(7)=7
Func(8)=8
Func(9)=9
Func(10)=10

Process returned 0 (0x0)
Press any key to continue.
```

注意事项⚠： 静态变量的使用须知

1. 未被程序员初始化的静态变量都由系统初始化为0
2. 局部静态变量的初值是编译时赋的。当运行时重复调用此函数时，不重复赋初值
3. 虽然局部静态变量在函数调用结束后仍然存在，但其他函数不能引用它
4. 局部的静态变量在程序执行结束时消亡
5. 用static还可以用在函数定义或说明中。该函数只能被用于本源文件中，其他源文件不能调用此函数 → 达到了隐藏的作用

寄存变量（register variable）

- 存储在寄存器中，代替自动变量或形参，可以提高变量的访问速度
- 如无合适的寄存器可用，则编译器把它设为自动变量
- 现在的编译器具有一定的智能，能够识别频繁使用的变量并进行优化
- 注意：只有局部自动变量才能定义为寄存器变量，全局变量和静态局部变量不能定义




```
register int i = 10;
```

extern 外部变量

- 声明一个不在本模块作用范围内的全局变量。如：

  extern int num;

  num为一个全局变量

- 用途：
  - 在某函数中引用了一个声明在本函数后的全局变量时，需要在函数内用extern声明此全局变量。
  - 当一个程序有多个源文件组成时，用extern可引用另一文件中的全局变量。

extern 的示例

```cpp
//chapter6-extern-variable-file1.cpp
#include <iostream>
using namespace std;

void f();
extern int x; //外部变量的声明

int main()
{
f();
cout << "in main(): x= "<< x << endl;
return 0;
}
```

```cpp
//chapter6-extern-variable-file2.cpp
#include <iostream>
using namespace std;

int x = 314; //全局变量的定义

void f()
{
cout << "in f(): x= "<< x << endl;
}
```

g++ **chapter6-extern-variable-file1.cpp chapter6-extern-variable-file2.cpp** -o **extern**

完成linkage所需文件

可执行文件

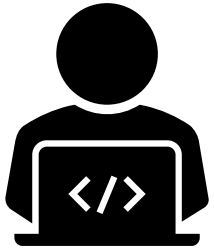
代码演示

chapter6-extern-variable-file1.cpp
chapter6-extern-variable-file2.cpp

```
void a();      void b();
void c();      void d(int);
int main()
{ int x=6;
  cout <<“x in main is “ << x << endl;
  a(); b(); c(); d(x);  x++;
  a(); b(); c(); d(x);
  cout << “x in main is ” << x << endl; return 0;
}
void a()
{ int x=25; cout << “x in a is ”<< x << endl;
}
void b()
{ static int x=50;
  cout << “x in b is ”<<  x++ << endl;
}
void c()
{ extern int x;
   x*=10; cout << “x in c is ”<< x << endl;
}
int x=2;
void d(int x)
{ cout << “x in d is ” << ++x << endl;
}
```

结果：

x in main is 6	无须解释
x in a is 25	局部变量
x in b is 50	静态局部变量
x in c is 20	外部变量（引用了在本函数外的全局变量）
x in d is 7	main中变量的值传递
x in a is 25	局部变量
x in b is 51	静态局部变量，在代码生存周期中不会第二次初始化
x in c 200	外部变量
x in d is 8	main中变量的值传递
x in main is 7	main中变量



代码演示

chapter6-final-example.cpp

```
int p = 1, q = 5, r=3; //全局变量(global variable)
void f1(); void f2(); void f3();

int main()
{
	f3(); cout << "after f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f1(); cout << "after f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	f2(); cout << "after f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
	return 0;
}
void f1()
{
	int p = 3, r = 2;
	q=p+q+r; cout << "f1: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f2()
{
		p=p+q+r; cout << "f2: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
void f3()
{
	int q;
	r = 2*r; //直接操作全局变量会改变全局变量值!
	q=r+p;
	cout << "f3: p="<<p<<" ,q="<<q<<", r="<<r<<endl;
}
```

f3: p=1 ,q=7, r=6

after f3: p=1 ,q=5, r=6

总结：变量的作用域和存储类型

- 全局变量
  - 静态全局变量 （只限本文件使用）
  - 外部变量 （非静态外部变量允许其他文件引用）
- 局部变量
  - 静态局部变量（离开函数，值仍保留）
  - 动态局部变量（自动变量）（离开函数，值就消失）

变量作用域—问题回顾

- 自动变量和静态局部变量有什么相同之处和不同之处？
- 静态变量和全局变量有什么相同之处和不同之处？
- 静态全局变量和静态局部变量有什么相同之处和不同之处？

局部变量和静态局部变量有什么相同之处和不同之处?

- 静态局部变量属于静态存储类别，在静态存储区内分配存储分配单元。在**程序整个运行期间**都不释放。而自动变量（即动态局部变量）属于动态存储类别，占动态存储区空间而不占静态存储空间，函数调用结束后立即释放。
- 对静态局部变量是在编译时赋初值的，**即只赋值一次**，在程序运行时它已有初值。以后每次调用函数时不再重新赋初值而只是**保留**上次函数调用结束时的值。而对自动变量赋初值，**不是在编译时**进行的，而是在函数调用时进行，每调用一次函数重新给一次初值，相当于执行一次赋值语句。
- 如在定义局部变量时不赋初值的话，则对静态局部变量来说，编译时自动**赋初值0**或空字符。而对自动变量来说，如果不赋初值则它的值是一个不确定的值。这是由于每次函数调用结束后存储单元已释放，下次调用时又重新另分配存储单元，而所分配单元的值**不确定**。
- 虽然静态局部变量在函数调用结束后仍然存在，但**其他函数不能引用它**。

静态变量和全局变量有什么相同之处和不同之处?

- 全局变量、静态局部变量、静态全局变量都在静态存储区分配空间，生命周期是从程序开始到程序结束。
- 全局变量具有全局作用域。全局变量只需在一个源文件中定义，就可以作用于所有的源文件。当然，其他不包含全局变量定义的源文件需要用extern关键字再次**声明**这个全局变量。
- 静态局部变量具有局部作用域，它只被初始化一次，自从第一次被初始化直到程序运行结束都一直存在，它和全局变量的区别在于全局变量对所有的函数都是可见的，而静态局部变量**只对定义自己的函数体**始终可见。
- 静态全局变量也具有全局作用域，它与全局变量的区别在于如果程序包含多个文件的话，它作用于定义它的文件里，**不能作用到其它文件里**，即被static关键字修饰过的变量具有文件作用域。这样即使两个不同的源文件都定义了相同名字的静态全局变量，它们也是不同的变量。

本章大纲

- 函数的用途和定义
- 函数的使用
- 变量的作用域与存储类别
- 函数参数—数组和默认值
- 内联、重载、模版、递归

数组作为函数的参数

- 设计一函数，统计10位同学的平均成绩，返回值是平均成绩。
- 设计考虑：如何传递参数？
  - 参数是10位同学的考试成绩，可以用10个整型数来表示，所以有10个整型的形式参数；
  - 一组同类数据可以用一个数组来描述，所以参数也可以是一个10个元素的整型数组；
- 第二种方法更加简练；

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

Average函数的使用

```
int main()
{   int i, score[10];
    cout << "请输入10个成绩：" << endl;
    for ( i = 0; i < 10; i++)  cin >> score[i];
    cout << "平均成绩是：" << average(score)
         << endl;
    return 0;
}
```

注意：形式参数是数组，实际参数也是一个数组

一个有趣的现象

```
int main()
{   int i, score[10];
    cout << "请输入10个成绩：" <<
   endl;
  for ( i = 0; i < 10; i++)  cin >>
   score[i];
    cout << “平均成绩是：” <<
          average(score) << endl;
    cout << score[3];
    return 0;
}
```

```
int average (int array[10])
{   int i, sum = 0;
    for (i = 0; i < 10; ++i)
       sum += array[i];
    array[3] = 90;
    return sum / 10;
 }
```

程序运行结束后，score[3]中是什么？

注意事项⚠：数组参数的传递机制

- C++语言规定，数组名是数组的起始地址；
- 参数传递时，实际参数是数组名，形式参数也是数组名；
- 按照值传递，当用实际参数score调用函数 average时，是用score 初始化形式参数数组array。如 score 的首地址为1000，在函数中形参数组array的首地址也为1000。
- 形式参数和实际参数是同一数组！

注意事项⚠：数组参数的传递机制（续）

- 在函数中并没有定义新的数组
- 对形式参数数组指定规模是没有意义的，形式参数数组不需要指定大小，所以方括号中为空
- 函数如何知道数组的规模？用另一个整型参数表示
- 总结：数组传递需要两个参数
  - 数组名
  - 数组规模

默认参数

- 对于某些函数，程序往往会用一些固定的值去调用它。例如对于以某种数制输出整型数的函数print：

  void print(int value, int base);

  在大多数情况下都是以十进制输出，因此base的值总是为10。

- C++在定义或声明函数时可以为函数的某个参数指定默认值。
- 当调用函数时没有为它指定实际参数时，系统自动将默认值赋给形式参数。例如，可以将print函数声明为

  void print(int value, int base=10);

  调用print(20) 等价于 print(20, 10)

带有默认参数的函数的使用

C++在说明函数原型时，可以为一个或多个参数指定缺省值。调用此函数时，若缺省某一参数，C++自动以缺省值作为此参数的值。如：

```
int special(int x=2, float y=1.5);
```

调用时可用：

```
special(5, 3.2);   //x=5; y=3.2
special(6);          //x=6; y=1.5
special( );          //x=2; y=1.5
```

注意事项⚠：带有默认参数的函数使用注意事项

- 缺省参数无论有几个，都必须放在参数序列的最后；
- 在函数调用时，若某个参数省略，则其后的参数皆应省略而取其缺省值。
- 对参数默认值的指定只有在函数声明处有意义。因为函数的默认值是提供给调用者使用的。
- 因此编译器禁止声明和定义时同时定义缺省参数值,可以利用声明来随意更改默认参数的值。
- 在不同的源文件中，可以对函数的参数指定不同的默认值。
  - 例如对于上面的print函数，如果输出的大多是十进制数，那么在对应的源文件中可以指定base的默认值为10。
  - 经常要以二进制输出，那么在对应的源文件中可以指定默认值是2。

课堂练习

- 下列说法错误的是（ ）。

  A. 变量的作用域是指变量的作用范围，即在程序中可以被读写访问的区域，它取决于变量被定义的位置。

  B. 局部变量与全局变量同名时，全局变量隐藏局部变量，即全局变量起作用，局部变量不起作用。

  C. 形参也是局部变量，形参变量和实参变量的作用域是不同的，因此形参变量和实参变量同名时，二者互不干扰。

  D. 只要同名的变量出现在不同的作用域内，二者互不干扰，编译器有能力区分不同作用域中的同名变量

课堂练习

有如下的函数定义：

```
int Xfun(int x)
{
    int y = x;
    {int x = 10; y += x;}
    return x + y;
}
```

通过表达式Xfun(5)调用该函数，则得到的返回值是：（ ）

A. 20　B. 10　C. 15　D. 25