

第6章 函数

——变量的作用域

本节要讨论的主要问题

- 为什么要尽量避免使用全局变量？
- 编译器如何区分不同作用域的同名变量？



变量的作用域

- **变量的作用域（Scope）**
 - * 变量的作用（能被读写访问的）范围
 - * 取决于变量在源程序中被定义的位置
- **局部变量（Local Variable）**
 - * 在语句块内（函数、复合语句）定义的变量
- **全局变量（Global Variable）**
 - * 在所有函数之外定义的变量

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

```
#include <stdio.h>
int main()
{
    int sum = 0, m;
    for (int i=0; i<5; i++)
    {
        scanf("%d", &m);
        sum = sum + m;
    }
    printf("sum = %d", sum);
    return 0;
}
```

**C++和C99允许在for
循环的初始化部分定义
变量，但C89不支持**

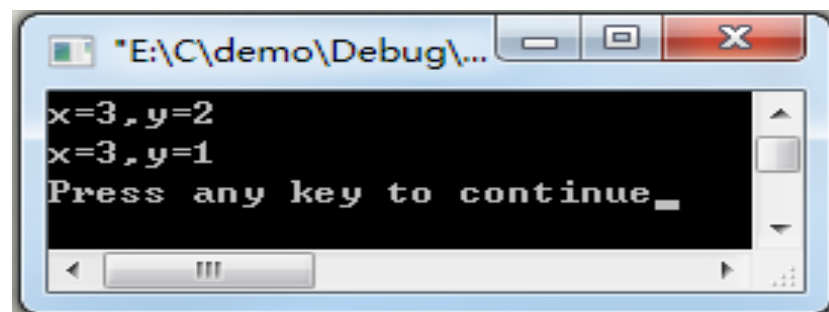
局部变量的作用域

- 作用域较小的局部变量隐藏作用域较大的局部变量

```
#include <stdio.h>
int main()
{
    int x = 1, y = 1;
    {
        int y = 2;
        x = 3;
        printf("x=%d,y=%d\n", x, y);
    }
    printf("x=%d,y=%d\n", x, y);
    return 0;
}
```

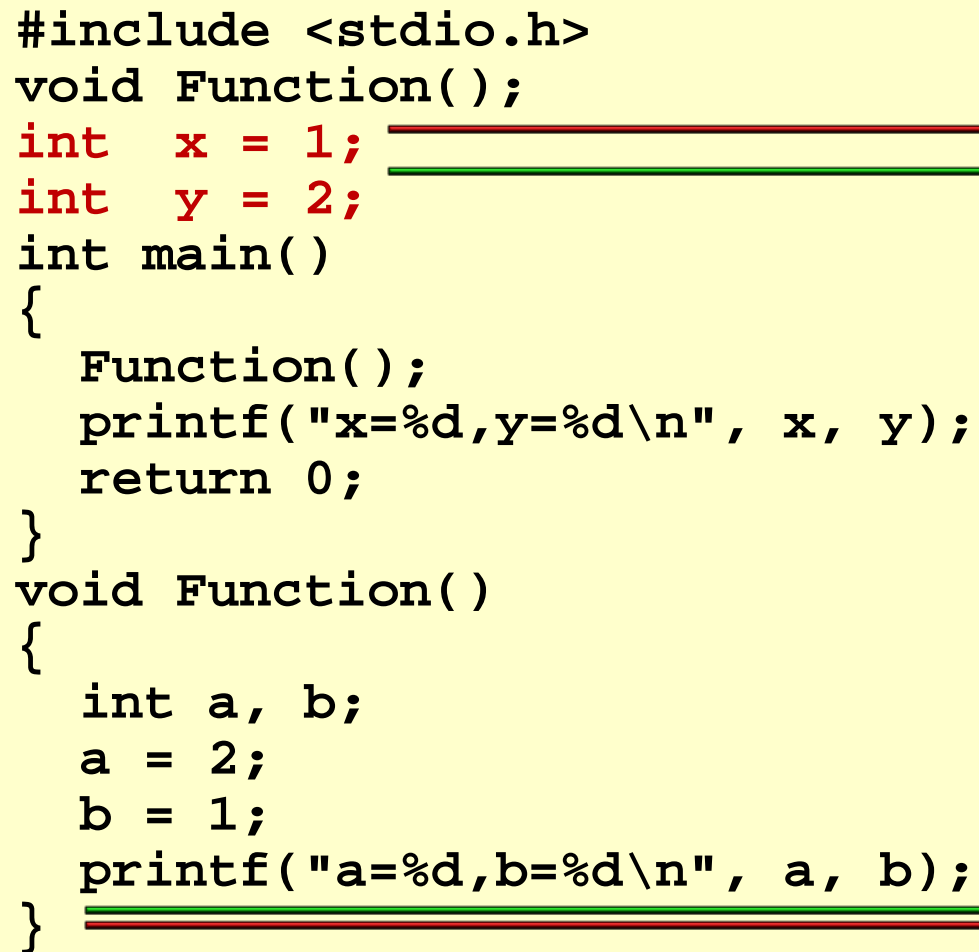
作用范围：函数体内

作用范围：复合语句内



全局变量的作用域

```
#include <stdio.h>
void Function();
int x = 1;
int y = 2;
int main()
{
    Function();
    printf("x=%d,y=%d\n", x, y);
    return 0;
}
void Function()
{
    int a, b;
    a = 2;
    b = 1;
    printf("a=%d,b=%d\n", a, b);
}
```

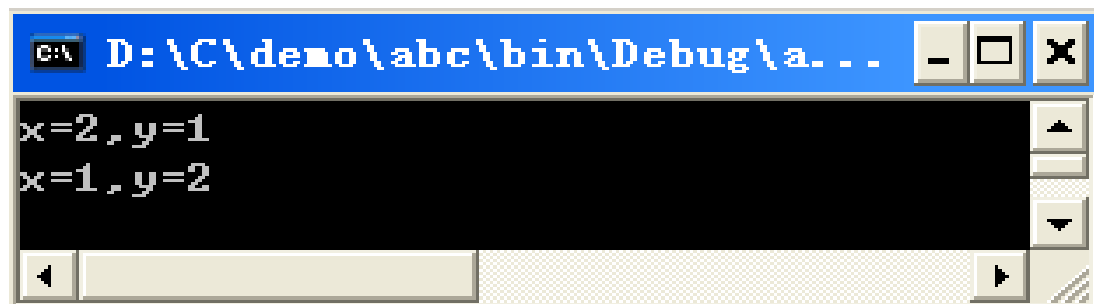


全局变量的作用范围：
从定义变量的位置开始，
到本程序结束

问题：假如变量名同名...

- **局部变量**与全局变量同名
 - * 局部变量隐藏全局变量, 互不干扰

```
#include <stdio.h>
void Function();
int x = 1;
int y = 2;
int main()
{
    Function();
    printf("x=%d,y=%d\n", x, y);
    return 0;
}
```



```
D:\C\demo\abc\bin\Debug\a...
x=2,y=1
x=1,y=2
```

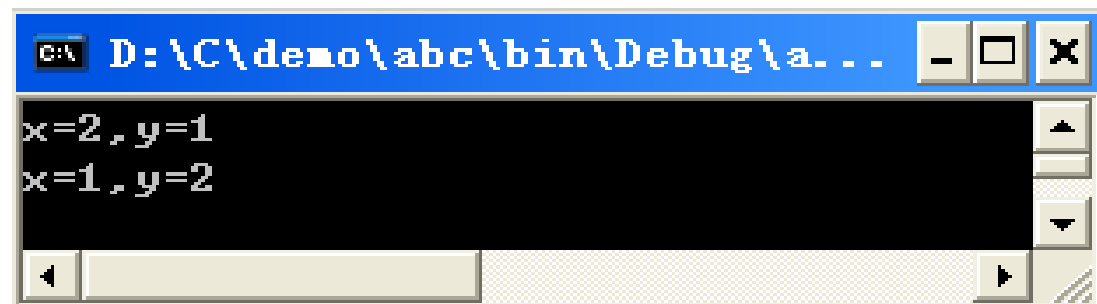
```
void Function()
{
    int x = 2;
    int y = 1;
    printf("x=%d,y=%d\n", x, y);
}
```

问题：假如变量名同名...

■ 形参与全局变量同名

* 局部变量隐藏全局变量, 互不干扰

```
#include <stdio.h>
void Function();
int x = 1;
int y = 2;
int main()
{
    Function(x, y);
    printf("x=%d,y=%d\n", x, y);
    return 0;
}
```



```
void Function(int x, int y)
{
    x = 2;
    y = 1;
    printf("x=%d,y=%d\n", x, y);
}
```


问题：假如变量名同名...

- **并列语句块**内的局部变量同名
 - * 互不干扰
 - * 形参值改变不影响与其同名的实参值

```
#include <stdio.h>
void Function(int x, int y);
int main()
{
    int x = 1;
    int y = 2;
    Function(x, y);
    printf("x=%d,y=%d\n", x, y);
    return 0;
}
```



```
void Function(int x, int y)
{
    x = 2;
    y = 1;
    printf("x=%d,y=%d\n", x, y);
}
```

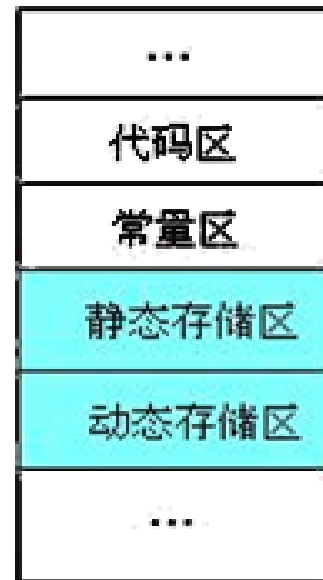
问题：假如变量名同名...

- 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
- 问题：假如同名变量出现在同一个作用域中？
 - * 编译器也将束手无策
 - * 编译器只能区分不同作用域中的同名变量

变量的作用域

- 问题：编译器如何区分不同作用域的同名变量？
 - * 编译器通过将同名变量映射到不同的内存地址来实现作用域的划分
 - * 局部变量和全局变量被分配的内存区域不同，因而内存地址也不同
 - * 形参和实参的作用域、内存地址不同，所以形参值的改变不会影响实参

RAM

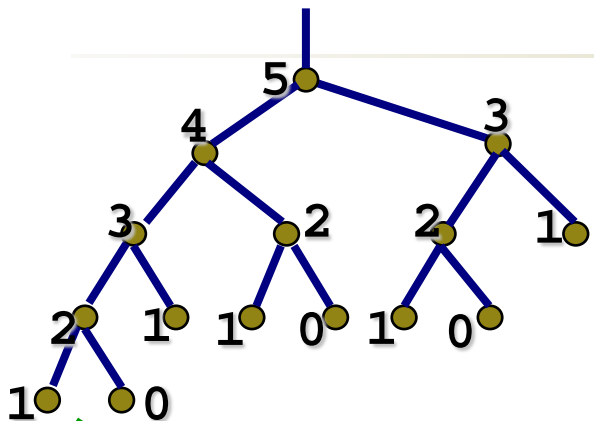


问题：全局变量有什么用？

- 全局变量在某些场合下很有用
 - * 当多个函数必须共享同一个固定类型的变量时
 - * 当少数几个函数必须共享大量数据时



打印计算Fibonacci数列第n项时所需的递归调用次数



计算Fib(5)共需15次Fib调用

```
long Fib(int n)
{
    count++;
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return (Fib(n-1) + Fib(n-2));
}
```

Input n:5 ✓

Fib(5)=5, count=15

```
#include <stdio.h>
long Fib(int a);
int count;
int main()
```

全局变量在未初始化时自动初始化为0

```
{
    int n, i, x;
    printf("Input n:");
    scanf("%d", &n);
    count = 0;
    x = Fib(i);
    printf("Fib(%d)=%d, count=%d\n", i, x, count);
    return 0;
}
```

□ 打印计算Fibonacci数列**每一项**时所需的递归调用次数

Input n:10 ✓

```
Fib(1)=1, count=1
Fib(2)=1, count=3
Fib(3)=2, count=5
Fib(4)=3, count=9
Fib(5)=5, count=15
Fib(6)=8, count=25
Fib(7)=13, count=41
Fib(8)=21, count=67
Fib(9)=34, count=109
Fib(10)=55, count=177
```

```
long Fib(int n)
{
    count++;
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return (Fib(n-1) + Fib(n-2));
}
```

```
#include <stdio.h>
long Fib(int a);
int count;
int main()
{
    int n, i, x;
    printf("Input n:");
    scanf("%d", &n);
    for (i=1; i<=n; i++)
    {
        count = 0;
        x = Fib(i);
        printf("Fib(%d)=%d, count=%d\n", i, x, count);
    }
    return 0;
}
```

不能省略

全部变量的副作用

- 破坏了函数的封装性，不能实现信息隐藏
 - * 谁都可改写它，很难确定谁改写了它
- 依赖全局变量的函数很难在其他程序中复用
 - * 依赖全局变量的函数不是“独立”的
- 对于使用全局变量的程序，维护比较困难
- 建议在可以不用时尽量不用
 - * 多数情况下，通过形参和返回值进行数据交流比共享全局变量的方法更好

讨论

- 你认为全局变量是利大于弊，还是弊大于利？最好举例说明。

