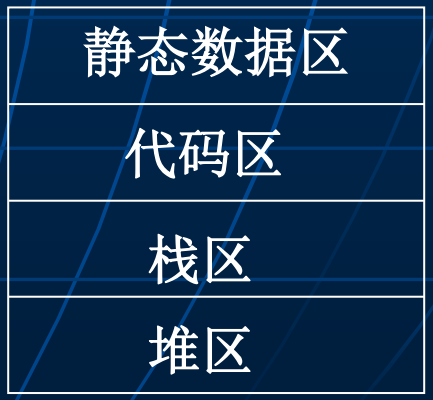
栈在函数调用中的作用

程序实体在内存中的安排

- 程序运行时，程序中的实体将存储在四个区域中：
  - 静态数据区：用于全局变量、static存储类的局部变量以及常量的内存分配 。如果没有显式初始化，系统将把它们初始化成0。
  - 代码区：用于存放程序的指令，对C++程序而言，代码区存放的是所有函数代码。
  - 栈区：用于自动存储类的局部变量、函数的形式参数以及函数调用时有关信息（如：函数返回地址等）的内存分配。
  - 堆区：用于动态变量的内存分配。
- 在上述区域中，栈区有着重要的作用：
  - 栈空间被各个函数共享，从而节省空间。

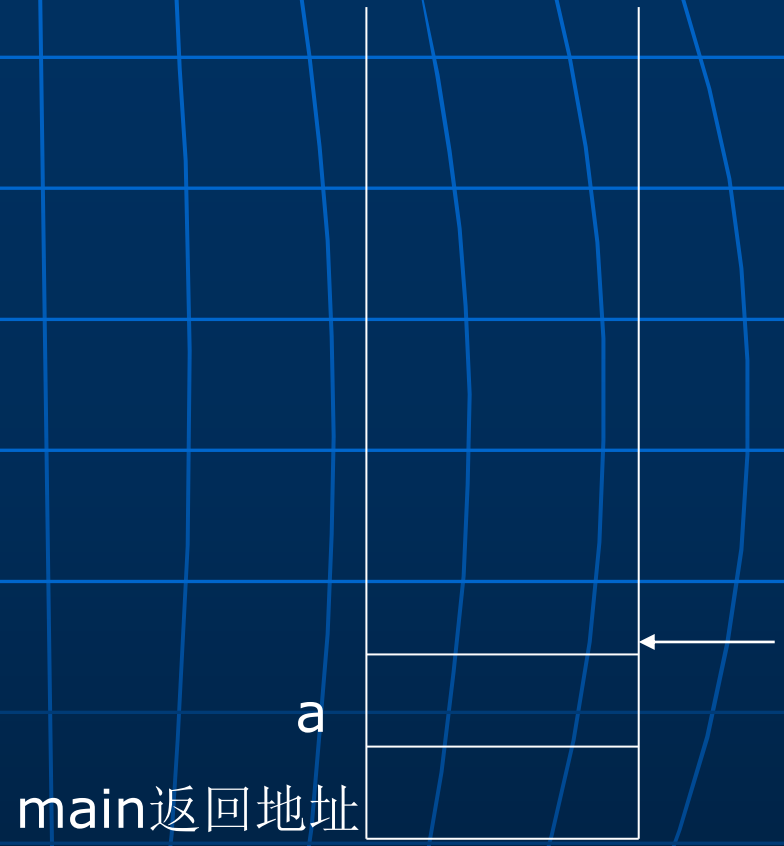


栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```

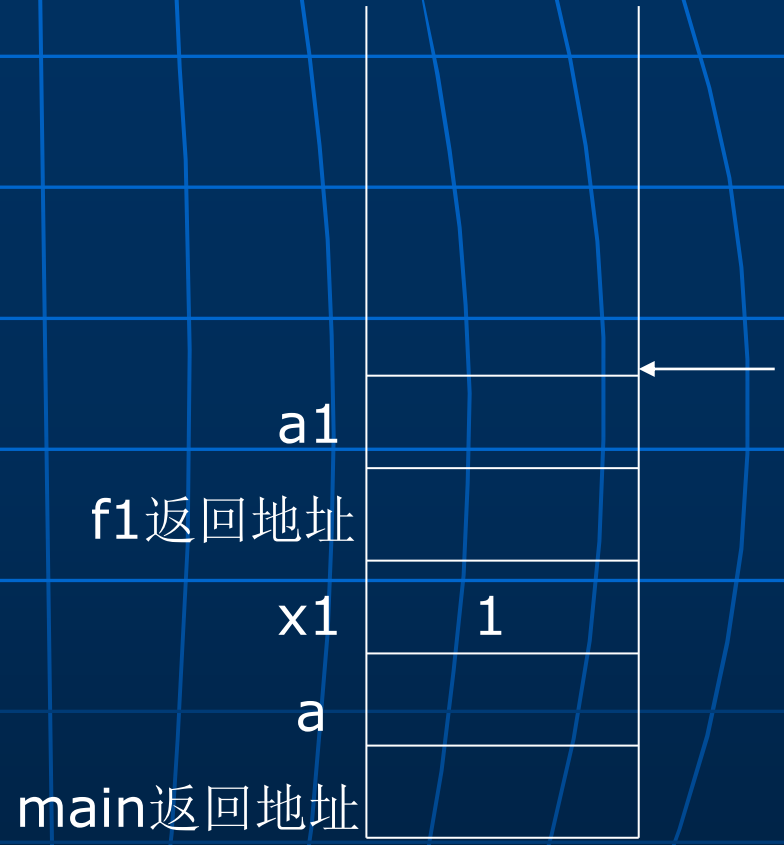
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ......
}
void f2(int x2)
{ int a2;
   ......
   f1(1);
   ......
}
void f3(int x3, int x4)
{ int a3;
   ......
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



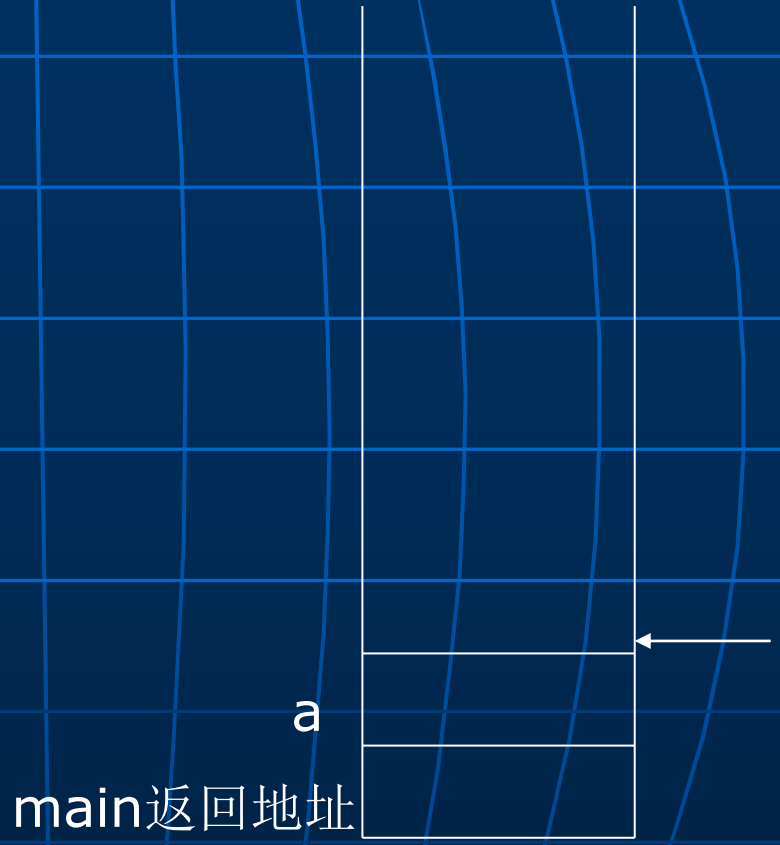
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ......
}
void f2(int x2)
{ int a2;
   ......
   f1(1);
   ......
}
void f3(int x3, int x4)
{ int a3;
   ......
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



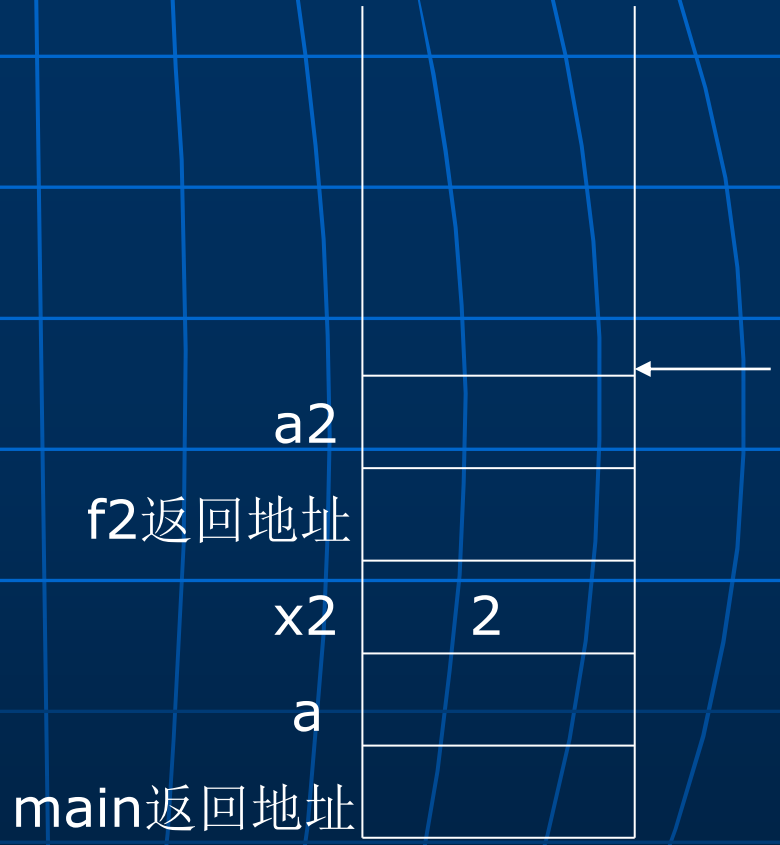
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



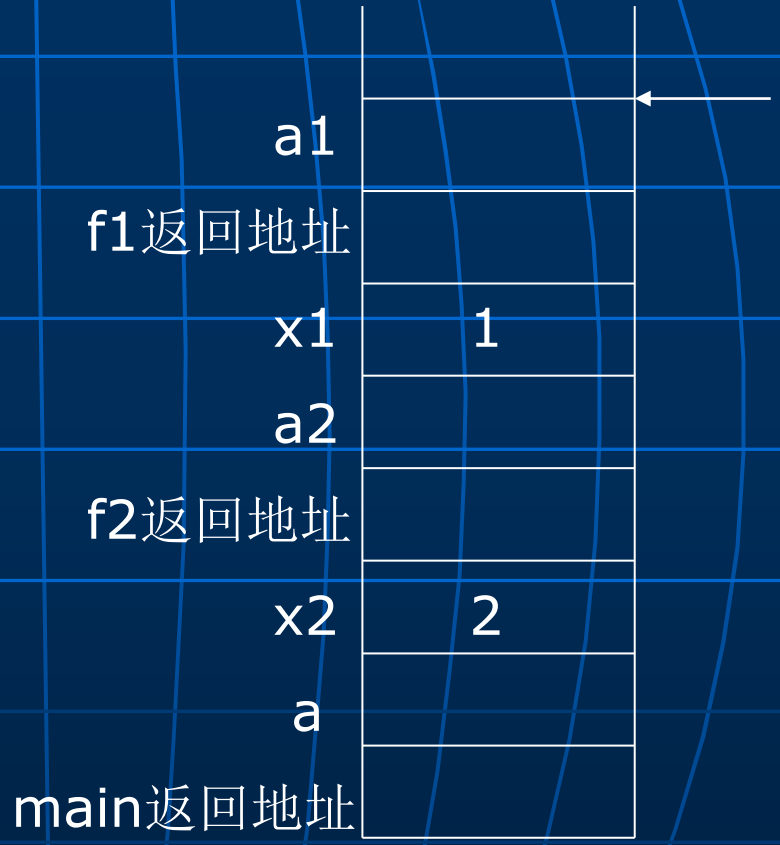
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



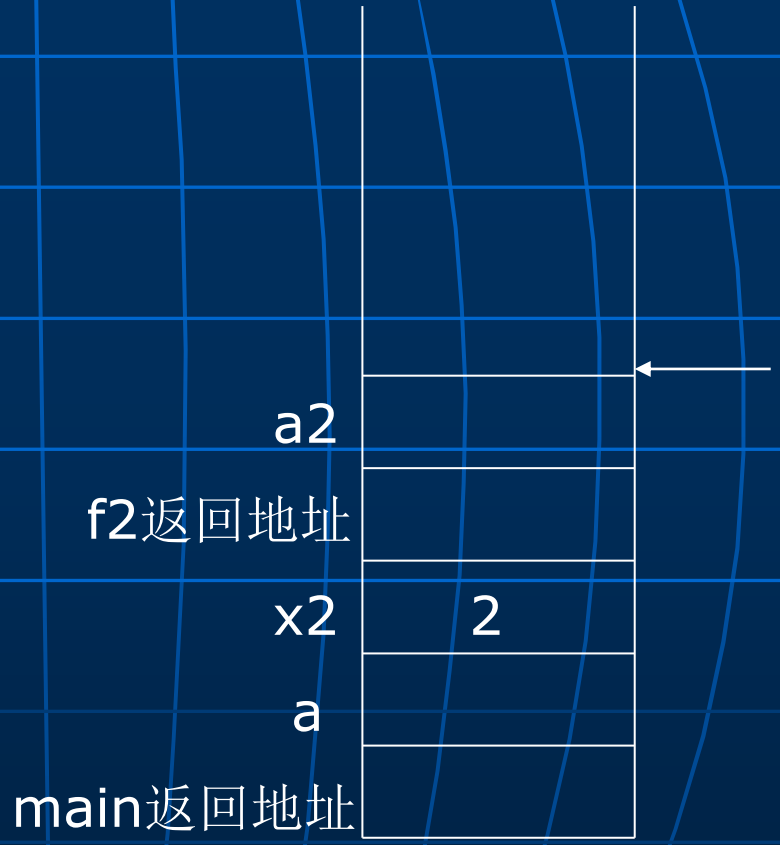
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ......
}
void f2(int x2)
{ int a2;
   ......
   f1(1);
   ......
}
void f3(int x3, int x4)
{ int a3;
   ......
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



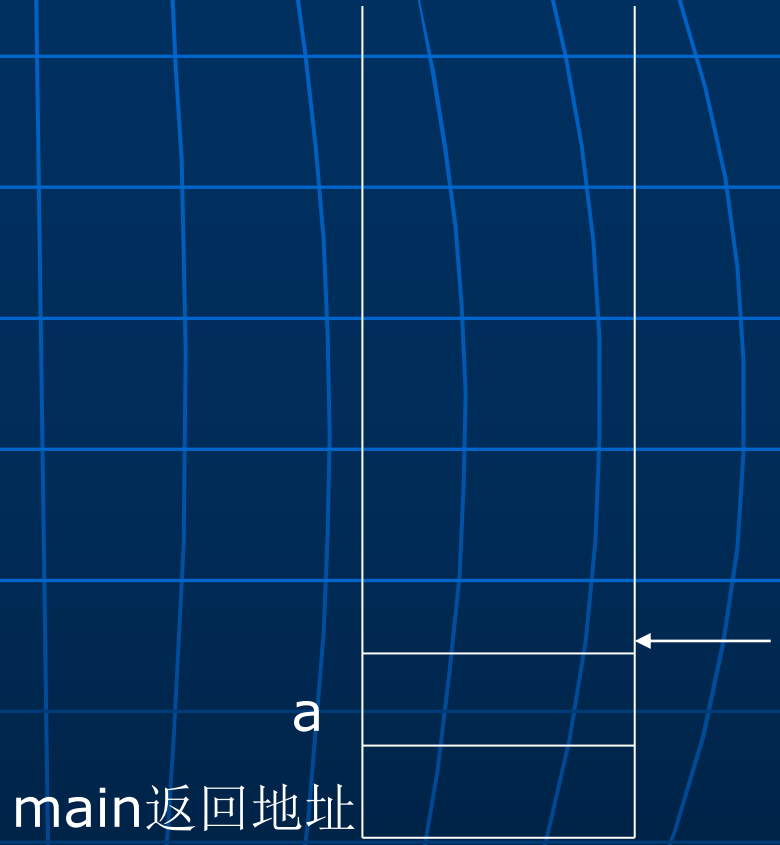
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



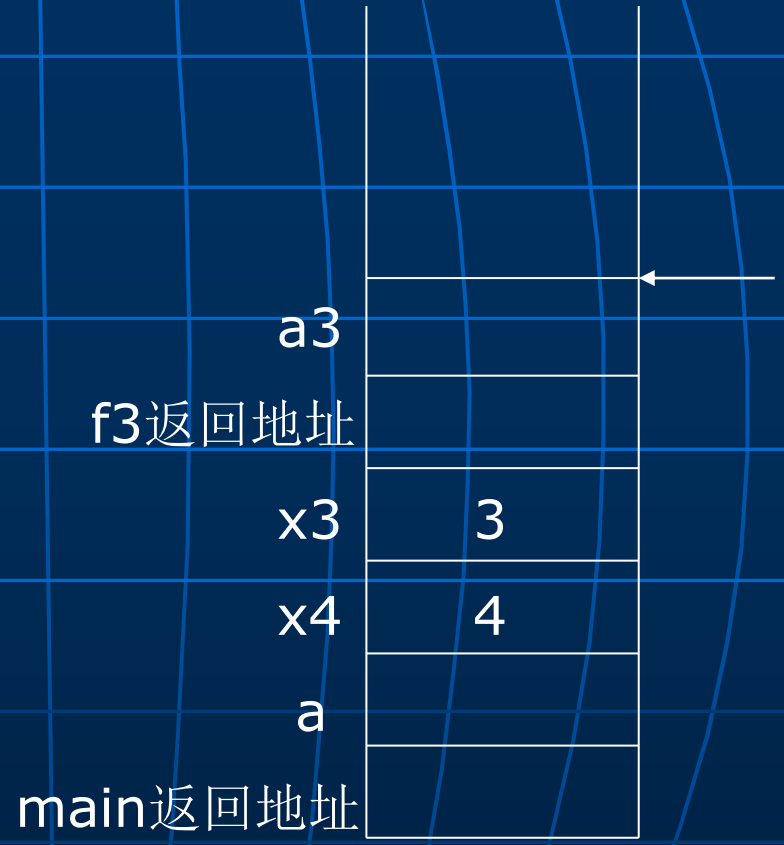
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



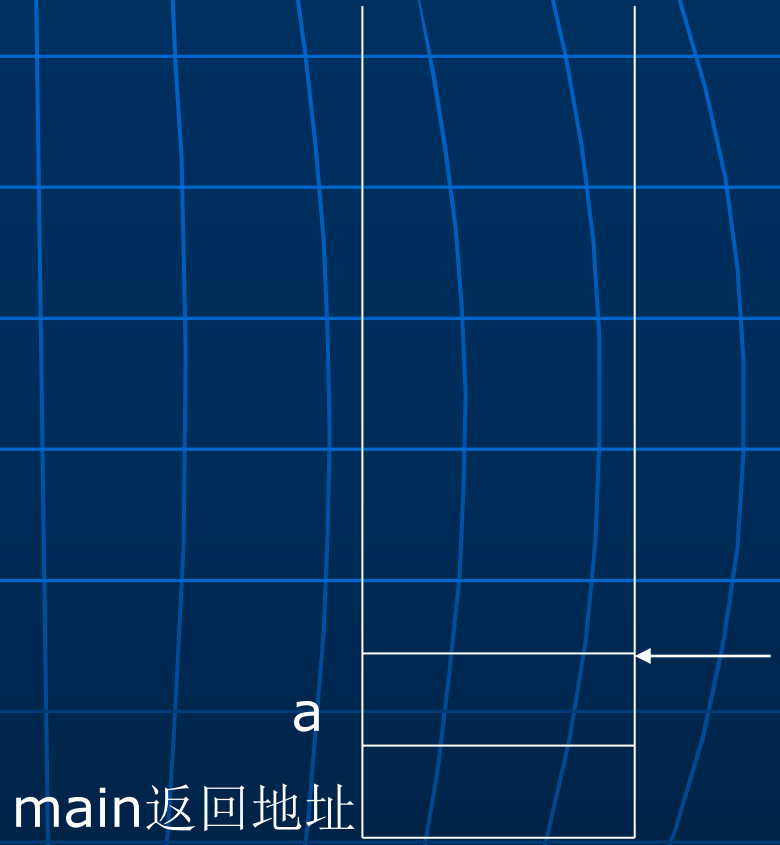
栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```



栈空间被各个函数共享

```
void f1(int x1)
{ int a1;
   ……
}
void f2(int x2)
{ int a2;
   ……
   f1(1);
   ……
}
void f3(int x3, int x4)
{ int a3;
   ……
}
int main()
{ int a;
   f1(1);
   f2(2);
   f3(3,4);
   return 0;
}
```

- 函数返回值的存储：
  - 如果函数的返回值为简单数据类型，则返回值通常通过CPU的某个寄存器返回；
  - 否则，返回值将存储到一块临时内存空间中，这个临时内存空间位于调用者的栈空间中，在函数调用时，调用者把这块空间的地址传给被调用者，被调用者通过这个地址存储返回值。
- 一般情况下，程序中不必关心上述栈的具体分配情况，但是，如果要进行混合语言编程，则需要考虑不同语言在栈使用上的差别。

递归函数执行中栈的使用情况

- 可以把一个递归函数看成多个同名的函数（多个实例，Instance），然后按函数的嵌套调用来理解递归调用过程。
- 注意：对递归函数的每一次递归调用都将产生一组新的局部变量（包括形参），虽然它们的名字相同，但它们是不同的变量（属于不同的实例），拥有不同的栈空间。

```
void f(int n)
{ int x=n+1;
   ... x,n ...
   if (n>0) f(n-1);
   ... x,n ...
}
......
f(2);
```

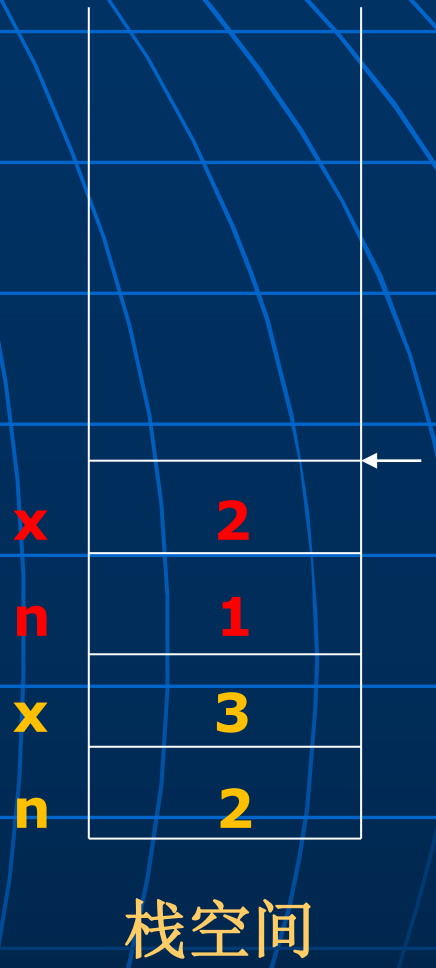
栈空间

```
void f(int n)
{ int x=n+1;
  ... x,n ... //3,2
  if (n>0) f(n-1);
  ... x,n ...
}
.....
f(2);
```

不同的实例：

```
void f(int n)
{ int x=n+1;
   ... x,n ... //3,2
  if (n>0) f(n-1);
   ... x,n ...
}
.....
f(2);
```

```
void f(int n)
{ int x=n+1;
  ... x,n ... //2,1
   if (n>0) f(n-1);
   ... x,n ...
}
```

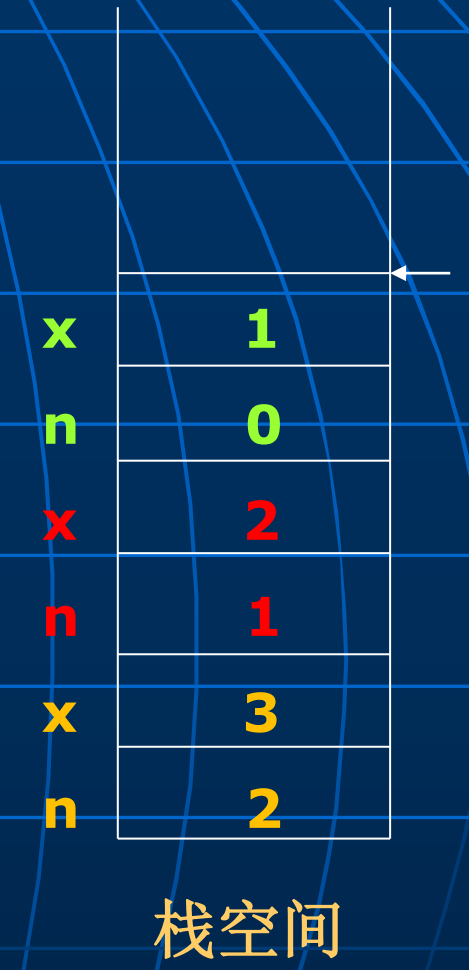


不同的实例：

```
void f(int n)
{ int x=n+1;
   ... x,n ... //3,2
  if (n>0) f(n-1);
   ... x,n ...
}
.....
f(2);
```

```
void f(int n)
{ int x=n+1;
   ... x,n ... //2,1
  if (n>0) f(n-1);
   ... x,n ...
}
```

```
void f(int n)
{ int x=n+1;
  ... x,n ... //1,0
   if (n>0) f(n-1);
   ... x,n ...
}
```

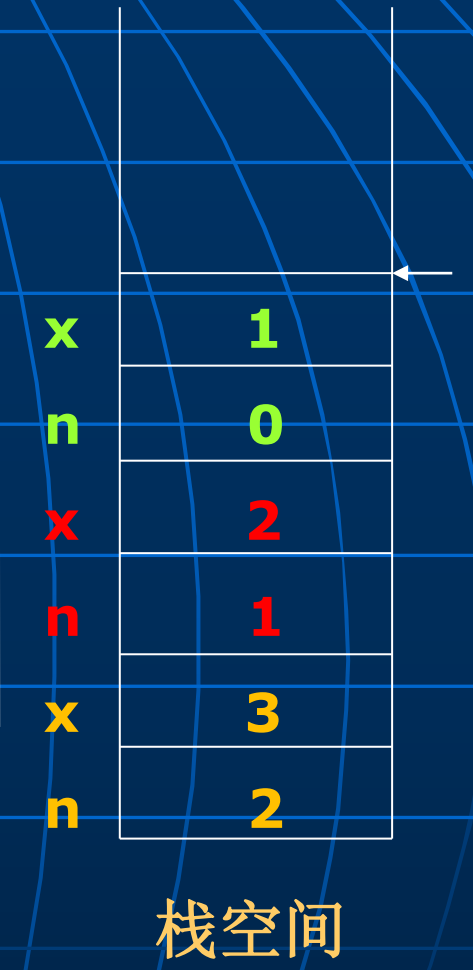


不同的实例：

```
void f(int n)
{ int x=n+1;
   ... x,n ... //3,2
  if (n>0) f(n-1);
   ... x,n ...
}
.....
f(2);
```

```
void f(int n)
{ int x=n+1;
   ... x,n ... //2,1
  if (n>0) f(n-1);
   ... x,n ...
}
```

```
void f(int n)
{ int x=n+1;
   ... x,n ... //1,0
  if (n>0) f(n-1);
   ... x,n ...
}
```

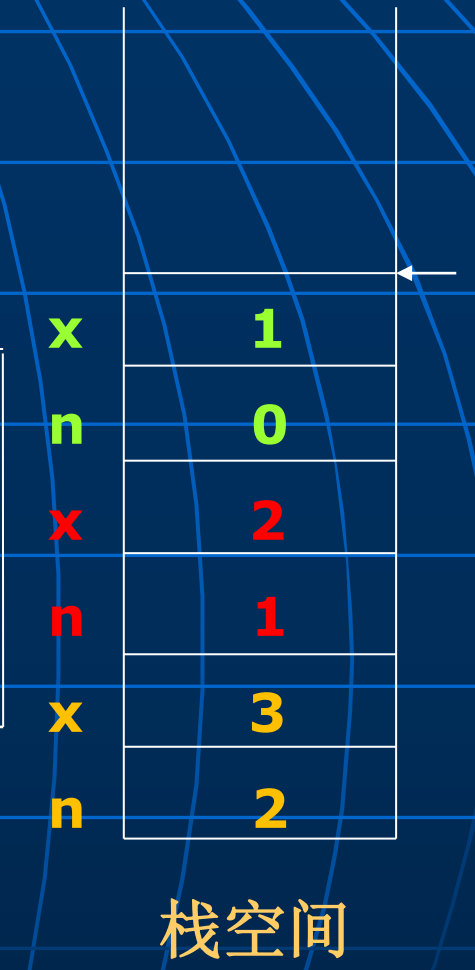


不同的实例：

```
void f(int n)
{ int x=n+1;
   ... x,n ... //3,2
  if (n>0) f(n-1);
   ... x,n ...
}
.....
f(2);
```

```
void f(int n)
{ int x=n+1;
   ... x,n ... //2,1
  if (n>0) f(n-1);
   ... x,n ...
}
```

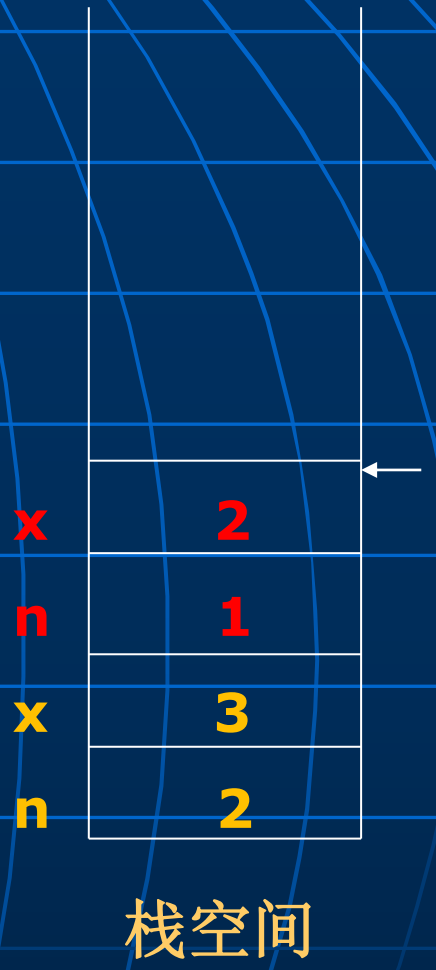
```
void f(int n)
{ int x=n+1;
   ... x,n ... //1,0
   if (n>0) f(n-1);
  ... x,n ... //1,0
}
```



不同的实例：

```
void f(int n)
{ int x=n+1;
   … x,n … //3,2
  if (n>0) f(n-1);
   … x,n …
}
……
f(2);
```

```
void f(int n)
{ int x=n+1;
   … x,n … //2,1
   if (n>0) f(n-1);
  … x,n … //2,1
}
```



```
void f(int n)
{ int x=n+1;
   ... x,n ... //3,2
   if (n>0) f(n-1);
   ... x,n ... //3,2
}
......
f(2);
```

```
void f(int n)
{ int x=n+1;
   ... x,n ...
   if (n>0) f(n-1);
   ... x,n ...
}
......
f(2);
```

栈空间

- 栈空间一方面被各个函数共享，另一方面由于受到栈空间的限制，它也对函数调用的深度（嵌套和递归调用）有所限制：
  - 过深的函数嵌套或递归调用会造成栈空间不足，出现“栈溢出”（stack overflow）错误，从而引起程序的异常终止。
- 另外，在程序设计时也需要很好地处置形式参数和局部变量的个数和大小，例如，
  - 不应把大的结构按值传递给函数。
  - 不应定义很大的局部数组变量。