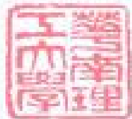


第3章　函数

3.1　函数的定义与调用

3.2　函数参数的传递

3.3　函数调用机制

3.4　函数指针

3.5　内联函数和重载函数

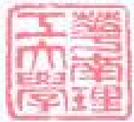
3.6　变量存储特性与标识符作用域

3.7　多文件结构程序

3.8　命名空间

3.9　终止程序执行

小结



第3章　函数

- 函数（Function）是功能抽象的模块

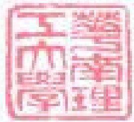
参数

输入流

函数

输出流

返回值



第3章　函数

- 函数（Function）是功能抽象的模块
- 函数作用 —— 任务划分；代码重用
- 函数是C++程序的重要组件

3.1 函数的定义和调用

- 函数定义由两部分组成：函数首部和函数操作描述
- 函数调用是通过表达式或语句激活并执行函数代码的过程

```
// 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
int main()
{  double  vol,  r,  h ;
    cin  >>  r >> h ;
    vol = volume ( r,  h ) ;
    cout << "Volume = " <<  vol  << endl ;
}
```

3.1 函数的定义和调用

- 函数定义由两部分组成：函数首部和函数操作描述
- 函数调用是通过表达式或语句激活并执行函数代码的过程

函数定义

```
// 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
int main()
{  double  vol,  r,  h ;
   cin  >>  r >> h ;
   vol = volume ( r,  h ) ;
   cout << "Volume = " <<  vol  << endl ;
}
```

3.1　函数的定义和调用

- 函数定义由两部分组成：函数首部和函数操作描述
- 函数调用是通过表达式或语句激活并执行函数代码的过程

```
// 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double radius,  double  height )
{   return   3.14 * radius * radius * height ;
int main()
{  double  vol,  r,  h ;
   cin  >>  r >> h ;
   vol = volume ( r,  h ) ;
   cout << "Volume = " <<  vol  << endl ;
}
```

函数调用

3.1.1 函数定义

函数定义形式

类型 函数名 （ 形式参数表 ）

{

语句序列

}

3.1.1 函数定义

函数定义形式

类型 函数名 （形式参数表）

{

语句序列

}

函数头——函数接口，包括：

3.1.1 函数定义

函数定义形式

类型 函数名 （形式参数表）

{

语句序列

}

函数头——函数接口，包括：

函数返回值类型 函数体中由 **return** 语句返回的值的类型。没有返回值其类型为**void**

3.1.1 函数定义

函数定义形式　　*类型* ***函数名*** *（形式参数表）*

```
{
    语句序列
}
```

函数头——函数接口，包括：

函数返回值类型　　函数体中由 **return** 语句返回的值的类型。没有返回值其类型为**void**

函数名　　用户定义标识符

3.1.1 函数定义

函数定义形式　　*类型 函数名 （**形式参数表**）*

{

语句序列

}

函数头——函数接口，包括：

函数返回值类型　　函数体中由 **return** 语句返回的值的类型。没有返回值其类型为**void**

函数名　　用户定义标识符

形式参数表　　逗号分隔的参数说明表列，缺省形式参数时不能省略圆括号。一般形式为：

类型 参数1 ，类型 参数2 ，… ，类型 参数n

3.1.1 函数定义

函数定义形式

类型 函数名 （ 形式参数表 ）

{

语句序列

}

函数头——函数接口

函数体——函数的实现代码。

3.1.1 函数定义

例3-1

```
void  printmessage ( )
   {  cout << "How do you do!" << endl ;
   }
```

3. 1. 1 函数定义

例3-1

```
void printmessage ( )
  { cout << "How do you do!" << endl ;
  }
```

函数返回值类型

无返回值

3.1.1 函数定义

例3-1

```
void printmessage ( )
  { cout << "How do you do!" << endl ;
  }
```

3.1.1 函数定义

例3-1

```
void  printmessage ( )
    {  cout << "How do you do!" << endl ;
    }
```

形式参数表

无参数

3.1.1 函数定义

例3-1

```
void  printmessage ( )
   {  cout << "How do you do!" << endl ;
   }
```

函数体

无 return 语句

3.1.1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
            return x ;
   else
            return y ;
}
```

3.1.1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
        return x ;
   else
        return y ;
}
```

函数返回值类型

3. 1. 1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
            return x ;
   else
            return y ;
}
```

函数名

3.1.1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
           return x ;
   else
           return y ;
}
```

形式参数表

3.1.1　函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
        return x ;
   else
        return y ;
}
```

3.1.1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
        return x ;
   else
        return y ;
}
```

return 语句形式：

　　return *表达式*

或　**return** （*表达式*）

作用：

- 返回函数值
- 不再执行后续语句，程序控制返回调用点
  一个函数体内可以有多个return 语句
- *表达式* 返回值的类型与函数类型不相同时，自动强制转换成函数的类型

3.1.1 函数定义

例3-2

```
double max ( double x , double y )
{  if ( x > y )
           return x ;
   else
           return y ;
}
```

```
Type  FunctionName ()
{ // statements
  return expression ;
}
void  FunctionName ()
{ // statements
  return ;          //可省略
}
```

3.1.2　函数调用

调用形式　　　*函数名（ 实际参数表 ）*

3.1.2 函数调用

调用形式 ***函数名*** （ *实际参数表* ）

函数名 函数的入口地址

3.1.2 函数调用

调用形式　　　*函数名*（ ***实际参数表*** ）

函数名　　　函数的入口地址

实际参数表　与形式参数必须在个数、类型、位置一一对应

3.1.2　函数调用

调用形式　　　*函数名（实际参数表）*

函数名　　　函数的入口地址

实际参数表　与形式参数必须在个数、类型、位置一一对应

用表达式或语句形式调用;

若函数返回值类型为void，则只能用语句调用

3.1.2 函数调用

例3-1

```
#include<iostream>
using namespace std ;
void  printmessage ()
   {  cout << "How do you do!" << endl ;
   }
int main()
{  printmessage() ;  }
```

3.1.2 函数调用

例3-1

```
#include<iostream>
using namespace std ;
void  printmessage ()
   {  cout << "How do you do!" << e
   }
int main()
{  printmessage() ;  }
```

函数调用语句

3. 1. 2　函数调用

例3-2

```
#include<iostream>
using namespace std ;
double max ( double x , double y )
  {  if ( x > y )
       return x ;
   else
       return y ;
  }
int main()
  { double a, b;
     cin >> a >> b ;
     double m = max( a, b );
     cout << max( m, 3.5 ) << endl ;
  }
```

3.1.2 函数调用

例3-2

```
#include<iostream>
using namespace std ;
double max ( double x , double y )
  {  if ( x > y )
      return x ;
    else
      return y ;
  }
int main()
  { double a, b;
      cin >> a >> b ;
      double m = max( a, b );
      cout << max( m, 3.5 ) << endl ;
  }
```

函数调用表达式

3.1.2 函数调用

例3-2

```
#include<iostream>
using namespace std ;
double max ( double x , double y )
  {  if ( x > y )
      return x ;
   else
      return y ;
  }
int main()
  { double a, b;
      cin >> a >> b ;
      cout << max( max( a, b ) ,  a+3.5 ) << endl ;
  }
```

实际参数是表达式

3.1.3 函数原型

- 函数原型的作用是告诉编译器有关函数的信息：

  函数的名字

  函数返回的数据类型

  函数要接受的参数个数、参数类型和参数的顺序

- 编译器根据函数原型检查函数调用的正确性
- 函数原型的形式：

  类型 函数名 （ 形式参数表 ）；

3.1.3　函数原型

- 函数原型的作用是告诉编译器有关函数的信息：

  函数的名字

  函数返回的数据类型

  函数要接受的参数个数、参数类型和参数的

- 编译器根据函数原型检查函数调用的正确性
- 函数原型的形式：

  类型　函数名（形式参数表）;

函数原型是声明语句

3.1.3 函数原型

使用函数原型

```
#include <iostream>
using namespace std ;
double max( double, double ) ;          // 函数原型
int main()
{ double a, b, c, m1, m2 ;
   cout << "input a, b, c :\n" ;
   cin >> a >> b >> c ;
   m1 = max( a, b ) ;                   // 函数调用
   m2 = max( m1, c ) ;
  cout << "Maximum = " << m2 << endl ;
}
double max( double x, double y )        // 函数定义
{  if ( x > y )    return x ;
    else     return y ;
}
```

3.1.3 函数原型

使用函数原型

```
#include <iostream>
using namespace std ;
double max( double, double ) ;          // 函数原型
int main()
{ double a, b, c, m1, m2 ;
   cout << "input a, b, c :\n" ;
   cin >> a >> b >> c ;
   m1 = max( a, b ) ;                   // 函数调用
   m2 = max( m1, c ) ;
  cout << "Maximum = " << m2 << endl ;
}
double max( double x, double y )        // 函数定义
{  if ( x > y )    return x ;
   else       return y ;
}
```

函数原型的参数表
不需要参数名

3.1.3 函数原型

使用函数原型

```
#include <iostream>
using namespace std ;
double max( double, double ) ;            // 函数原型
int main()
{ double a, b, c, m1, m2 ;
   cout << "input a, b, c :\n" ;
   cin >> a >> b >> c ;
   m1 = max( a, b ) ;                     // 函数调用
   m2 = max( m1, c ) ;
  cout << "Maximum = " << m2 << endl ;
}
double max( double x, double y )          // 函数定义
{ if ( x > y )    return x ;
    else      return y ;
}
```

函数调用出现在定义之前
函数原型声明是必须的

3.1.3 函数原型

函数定义在调用之前

```
#include <iostream>
using namespace std ;
double max( double x, double y )        // 函数定义
{  if ( x > y )    return x ;
     else      return y ;
}
int main()
{ double a, b, c, m1, m2 ;
   cout << "input a, b, c :\n" ;
   cin >> a >> b >> c ;
   m1 = max( a, b ) ;                    // 函数调用
   m2 = max( m1, c ) ;
  cout << "Maximum = " << m2 << endl ;
}
```

3.1.3 函数原型

函数定义在调用之前

```
#include <iostream>
using namespace std ;
double max( double x, double y )          // 函数定义
{ if ( x > y )   return x ;
    else    return y ;
}
int main()
{ double a, b, c, m1, m2 ;
   cout << "input a, b, c :\n" ;
   cin >> a >> b >> c ;
   m1 = max( a, b ) ;                     // 函数调用
   m2 = max( m1, c ) ;
  cout << "Maximum = " << m2 << endl ;
}
```

函数定义出现在调用之前
无必要作函数原型声明

3.1.3 函数原型

cmath中几个常用的数学函数

函数原型	说明
int abs(int *n*);	*n*的绝对值
double cos(double *x*);	*x*（弧度）的余弦
double exp(double *x*);	指数函数，e^x
double fabs(double *x*);	*x*的绝对值
double fmod(double *x*, double *y*);	*x*/*y*的浮点余数
double log(double *x*);	*x*的自然对数（以e为底）
double log10(double *x*);	*x*的对数（以10为底）
double pow(double *x*, double *y*);	求幂，x^y
double sin(double *x*);	*x*（弧度）的正弦
double sqrt(double *x*);	*x*的平方根
double tan(double *x*);	*x*（弧度）的正切

3.1.3 函数原型

```
// 例3-3  用库函数求正弦和余弦值
#include <iostream>
#include <cmath>
using namespace std ;
int main()
{  double PI = 3.1415926535;
   double x, y;
   x = PI/ 2;
   y = sin( x );
   cout << "sin( " << x << " ) = " << y << endl ;
   y = cos( x );
   cout << "cos( " << x << " ) = " << y << endl ;
}
```

3.1.3 函数原型

```
// 例3-3  用库函数求正弦和余弦值
#include <iostream>
#include <cmath>
using namespace std ;
int main()
{  double PI = 3.1415926535;
   double x, y;
   x = PI / 2;
   y = sin( x );
   cout << "sin( " << x << " ) = " << y << endl ;
   y = cos( x );
   cout << "cos( " << x << " ) = " << y << endl ;
}
```

包含头文件

3.1.3　函数原型

```
// 例3-3  用库函数求正弦和余弦值
#include <iostream>
#include <cmath>
using namespace std ;
int main()
{  double PI = 3.1415926535;
   double x, y;
   x = PI / 2;
   y = sin( x );
   cout << "sin( " << x << " ) = " << y << endl ;
   y = cos( x );
   cout << "cos( " << x << " ) = " << y << endl ;
}
```

调用库函数

3.2　　函数参数的传递

C++有三种参数传递机制：

值传递

指针传递

引用传递

3.2.1 传值参数

- 调用函数时，实参表达式的值被复制到相应形参标识的对象中，并按形参类型强制转换
- 函数内对形参的访问、修改，都在形参的标识对象进行
- 函数返回时，形参对象被撤消，不影响实参的值
- 值传送的实参可以是常量、有确定值的变量或表达式
- 函数返回值通过匿名对象传递

1. 值传递机制

```
// 例3-4  强制类型转换
#include<iostream>
using namespace std ;
int main ( )
{ double  add1 ( double ,  double ) ;     // 函数原型
   double  add2 ( int , int ) ;           // 函数原型
   double  a ,  b,  c ;
   cin >> a >> b ;
   c = add1 ( a , b ) ;          cout << "c1=" << c << endl ;
   c = add2 ( 1/a , 1/b) ;       cout << "c2=" << c << endl ;
}
double  add1 ( double  x ,  double  y )
{ return ( x + y ) ; }

double  add2 ( int  i , int  j )
{  return ( i + j ) ; }
```

1. 值传递机制

```
// 例3-4  强制类型转换
#include<iostream>
using namespace std ;
int main ( )
{ double  add1 ( double ,  double ) ;                    // 函数原型
  double  add2 ( int , int ) ;                    // 函数原型
  double  a ,  b,  c ;
  cin >> a >> b ;
  c = add1 ( a , b ) ;          cout << "c1=" << c << endl ;
  c = add2 ( 1/a , 1/b) ;       cout << "c2=" << c << endl ;
}
double  add1 ( double  x ,  double  y )
{ return ( x + y ) ; }

double  add2 ( int  i , int  j )
{  return ( i + j ) ; }
```

类型强制转换

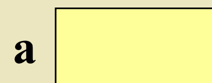
截取整数部分传送给形参

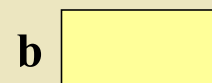
1．值传递机制

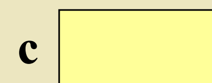
```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```

1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```

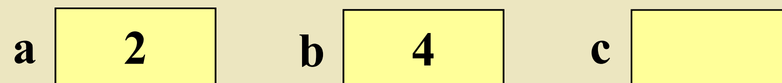
a 

b 

c 

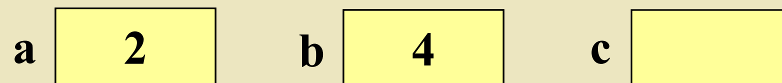
1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



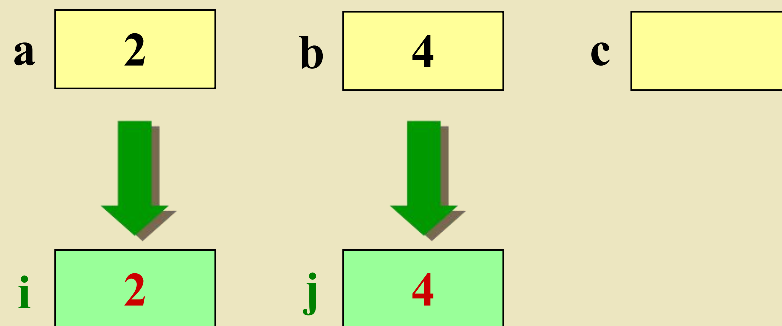
1. 值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



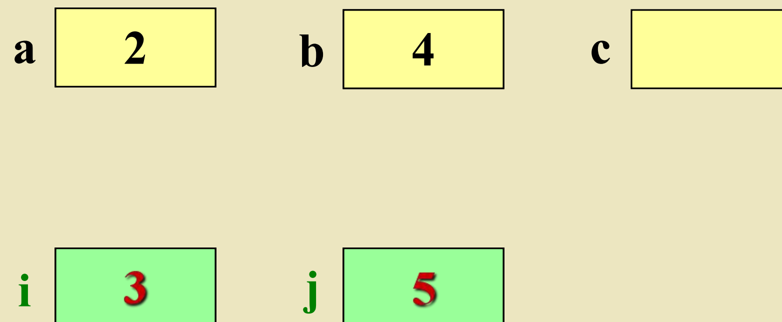
1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



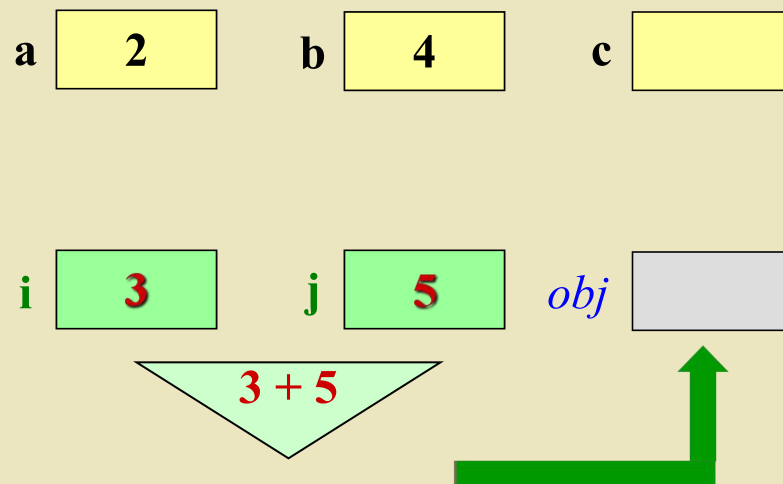
1. 值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i++; j++;
   return ( i + j );
}
```



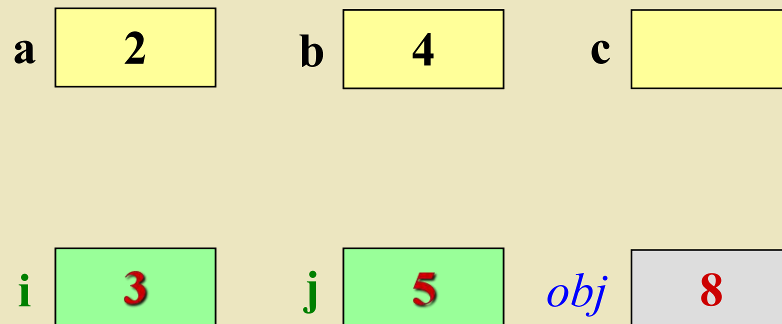
1. 值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



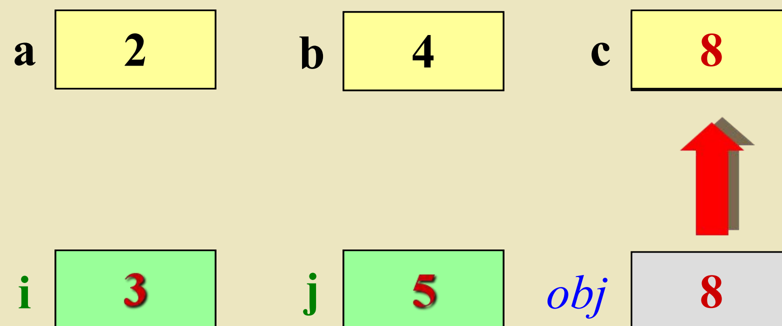
1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



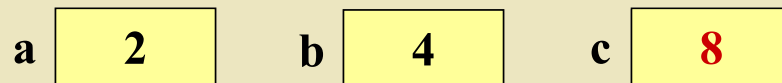
1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



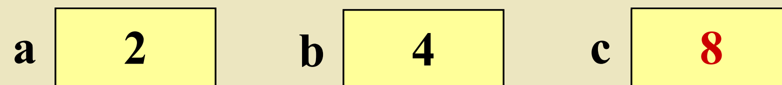
1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i + + ;  j + + ;
   return ( i + j );
}
```



1．值传递机制

```
// 例3-5  值参传递
#include<iostream>
using namespace std ;
int add(int ,  int ) ;
int main()
{
  int a, b, c ;
  cin >> a >> b;
  c = add(a,b) ;
  cout << "c = " << c << endl ;
}
int add(int i, int j )
{  i++;  j++;
   return ( i + j );
}
```



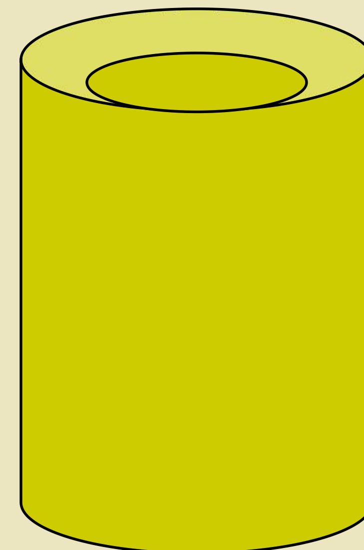
输出

c = 8

例3-6 计算圆筒的体积

分析：

分别定义函数求不同几何体的体积



- 圆柱体的体积 = $\pi r^2 h$

  double CylinderVolume(double **r**, double **h**) **;**

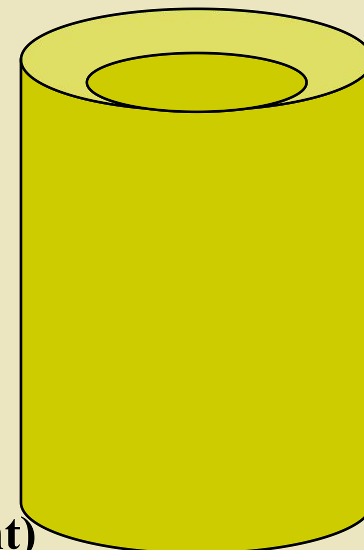
- 圆筒的体积 = 外圆柱体的体积 –内圆柱体的体积

  double **DonutSize**(double **Outer**, double **Inner**, double **Height**) **;**

例3-6 计算圆筒的体积

```
// 通过参数半径 r 和高度 h，返回圆柱体体积
double CylinderVolume( double r, double h )
{  const  double  PI = 3.1415;
   return  PI * r * r * h;
}

// 通过参数外圆柱体半径 Outer，内圆柱体半径 Inner
// 和圆柱体高度 Height 返回圆筒体积
double DonutSize(double Outer, double Inner, double Height)
{     double OuterSize = CylinderVolume(Outer, Height);
      double HoleSize = CylinderVolume(Inner, Height);
      return OuterSize - HoleSize;
}
```



例3-6 计算圆筒的体积

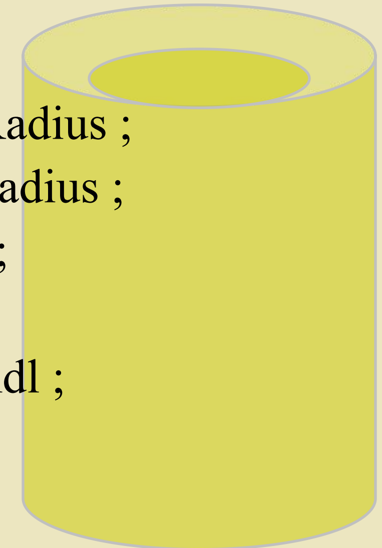
```
// 通过参数半径 r 和高度 h，返回圆柱体体积
double CylinderVolume( double r, double h )
{  const  double  PI = 3.1415;
   return  PI * r * r * h;
}
```

```
return CylinderVolume(Outer, Height) - CylinderVolume(Inner, Height) ;
```

```
double DonutSize(double Outer, double t Inner, double Height)
{     double OuterSize = CylinderVolume(Outer, Height);
      double HoleSize = CylinderVolume(Inner, Height);
      return OuterSize - HoleSize;
}
```

例3-6 计算圆筒的体积

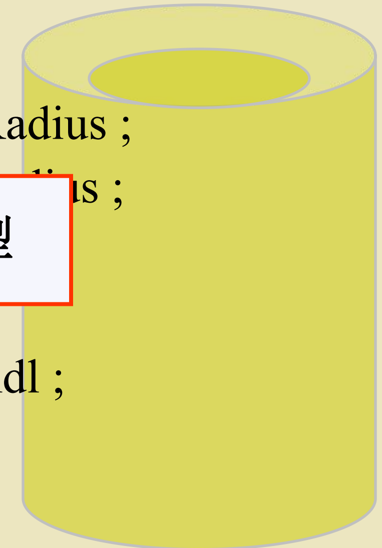
```
#include<iostream>
using namespace std ;
double CylinderVolume( double r, double h );
double DonutSize(double Outer, double Inner, double Height);
int main()
{ double OuterRadius, InnerRadius, Height;
  cout << "the radius of the outer cylinder : " ;    cin >>  OuterRadius ;
  cout << "the radius of the inner cylinder : " ;    cin >>  InnerRadius ;
  cout << "the height of the cylinder : " ;          cin >> Height ;
  cout << "size of the donut is : "
         << DonutSize( OuterRadius, InnerRadius, Height) << endl ;
}
double CylinderVolume( double r, double h )
{ const  double PI = 3.1415;
  return  PI * r * r * h;
}
double DonutSize(double Outer, double Inner, double Height)
{ double OuterSize = CylinderVolume(Outer, Height);
  double HoleSize = CylinderVolume(Inner, Height);
  return OuterSize - HoleSize;
}
```



例3-6 计算圆筒的体积

```
#include<iostream>
using namespace std ;
double CylinderVolume( double r, double h );
double DonutSize(double Outer, double Inner, double Height);
int main()
{ double OuterRadius, InnerRadius, Height;
  cout << "the radius of the outer cylinder : " ;    cin >>  OuterRadius ;
  cout << "the radius of the inner cylinder : " ;    cin >>  InnerRadius ;
  cout << "the height of the cylinder : " ;          cin >>  Height ;
  cout << "size of the donut is : "
       << DonutSize( OuterRadius, InnerRadius, Height) << endl ;
}
double CylinderVolume( double r, double h )
{ const  double PI = 3.1415;
  return  PI * r * r * h;
}
double DonutSize(double Outer, double Inner, double Height)
{ double OuterSize = CylinderVolume(Outer, Height);
  double HoleSize = CylinderVolume(Inner, Height);
  return OuterSize - HoleSize;
}
```

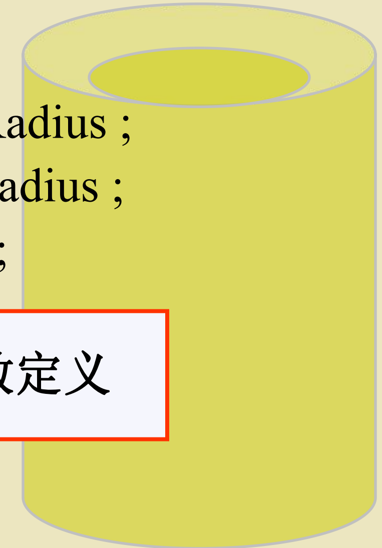
函数原型



例3-6 计算圆筒的体积

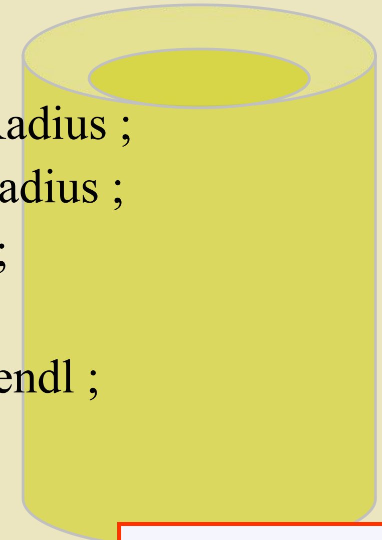
```
#include<iostream>
using namespace std ;
double CylinderVolume( double r, double h );
double DonutSize(double Outer, double Inner, double Height);
int main()
{ double OuterRadius, InnerRadius, Height;
  cout << "the radius of the outer cylinder : " ;    cin >>  OuterRadius ;
  cout << "the radius of the inner cylinder : " ;    cin >>  InnerRadius ;
  cout << "the height of the cylinder : " ;          cin >> Height ;
  cout << "size of the donut is : "
        << DonutSize( OuterRadius, InnerRadius, Height
}
double CylinderVolume( double r, double h )
{ const  double PI = 3.1415;
  return  PI* r * r * h;
}
double DonutSize(double Outer, double Inner, double Height)
{ double OuterSize = CylinderVolume(Outer, Height);
  double HoleSize = CylinderVolume(Inner, Height);
  return OuterSize - HoleSize;
}
```

函数定义



例3-6 计算圆筒的体积

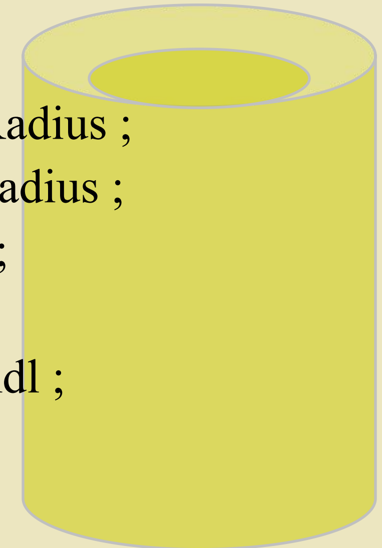
```
#include<iostream>
using namespace std ;
double CylinderVolume( double r, double h );
double DonutSize(double Outer, double Inner, double Height);
int main()
{ double OuterRadius, InnerRadius, Height;
  cout << "the radius of the outer cylinder : " ;    cin >>  OuterRadius ;
  cout << "the radius of the inner cylinder : " ;    cin >>  InnerRadius ;
  cout << "the height of the cylinder : " ;          cin >> Height ;
  cout << "size of the donut is : "
        << DonutSize( OuterRadius, InnerRadius, Height) << endl ;
}
double CylinderVolume( double r, double h )
{ const  double PI = 3.1415;
  return  PI * r * r * h;
}
double DonutSize(double Outer, double Inner, double Height)
{ double OuterSize = CylinderVolume(Outer, Height);
  double HoleSize = CylinderVolume(Inner, Height);
  return OuterSize - HoleSize;
}
```



调用函数

例3-6 计算圆筒的体积

```
#include<iostream>
using namespace std ;
double CylinderVolume( double r, double h );
double DonutSize(double Outer, double Inner, double Height);
int main()
{ double OuterRadius, InnerRadius, Height;
   cout << "the radius of the outer cylinder : " ;    cin >>  OuterRadius ;
   cout << "the radius of the inner cylinder : " ;    cin >>  InnerRadius ;
   cout << "the height of the cylinder : " ;          cin >> Height ;
   cout << "size of the donut is : "
          << DonutSize( OuterRadius, InnerRadius, Height) << endl ;
}
double CylinderVolume( double r, double h )
{ const  double PI = 3.1
   return  PI * r * r * h;
}
double DonutSize(doub
{ double OuterSize = C
   double HoleSize = Cy
   return OuterSize - Ho
}
```



C:\WINDOWS\system32\cmd.exe

```
the radius of the outer cylinder : 10.25
the radius of the inner cylinder : 7.5
the height of the cylinder : 6.12
size of the donut is : 938.468
请按任意键继续. . .
```

2．实际参数求值的副作用

- C++没有规定在函数调用时实际参数的求值顺序
- 若实际参数表达式之间有求值关联，同一个程序在不同编译器可能产生不同的运行结果

2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

从右向左计算实参表

x	4	y	6	z	

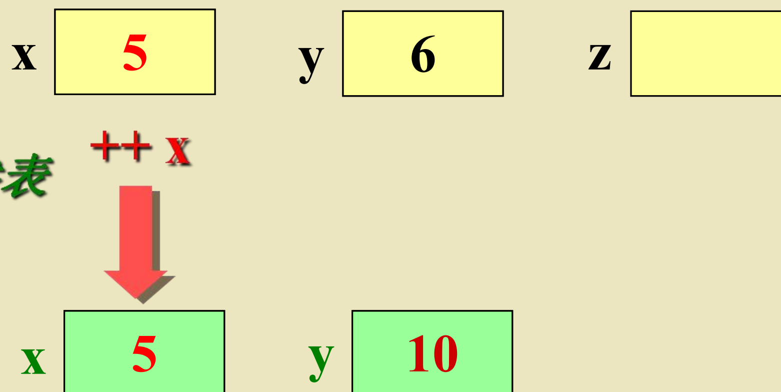
x+y

x		y	10

2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

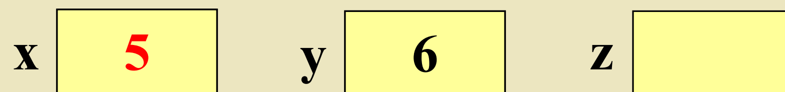
从右向左计算实参表

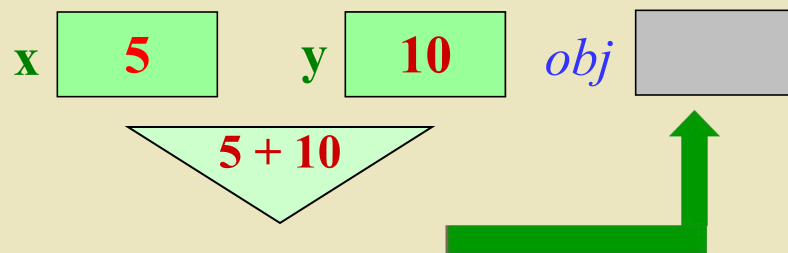


2．实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

从右向左计算实参表

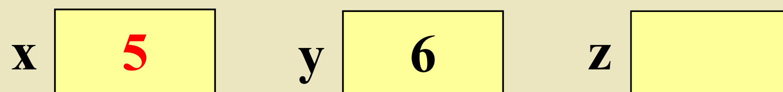


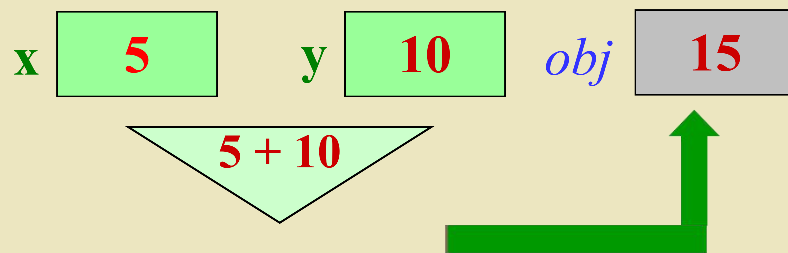


2．实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

从右向左计算实参表

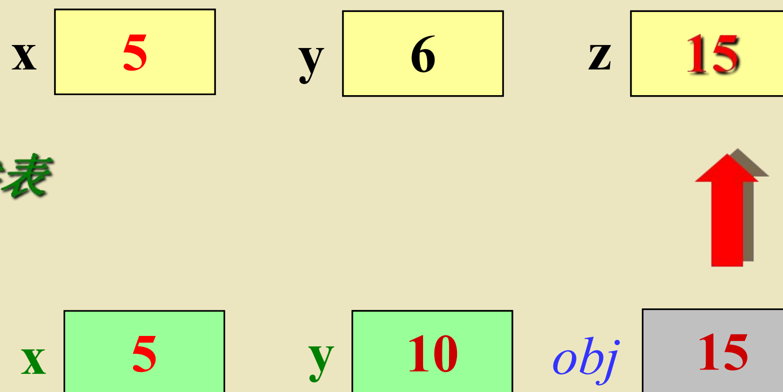




2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?!\n" ;
}
```

从右向左计算实参表



2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  int  z = add ( ++ x , x + y ) ;
  cout << " 5 + 11 = " << z << " ?
}
```

x 5　y 6　z 15

VC6.0

```
5 + 11 = 15 ?!
Press any key to continue
```

VC2010

```
5 + 11 = 16 ?!
请按任意键继续. . .
```

2. 实际参数求值的副作用

```
#include<iostream>
using namespace std ;
int  add ( int  x ,  int  y )
{ return  x + y ; }
int main ( )
{ int  x = 4 ,  y = 6 ;
  ++ x ;
  int  z = add ( x , x + y ) ;
  cout << " 5 + 11 = " << z << "\n" ;
}
```

VC6

```
"C:\test\Debug\Cpp1.exe"
 5 + 11 = 16
Press any key to continue
```

VC2010

```
C:\WINDOWS\system32\cmd.exe
 5 + 11 = 16
请按任意键继续. . .
```

3. 默认参数

- C++允许指定传值参数的默认值。当函数调用中省略默认参数时，默认值自动传递给被调用函数
- 默认参数在函数原型定义
- 默认参数放在一般参数之后

```
// 例3-7  使用默认参数
#include<iostream>
using namespace std ;
double  power ( double  real,  int n = 2  ) ;
int main ( )
{ double  r = 3.0 ;
   cout  <<  power ( r )  <<  endl ;
   cout  <<  power ( r, 3 )  <<  endl ;
}
double  power ( double  real ,   int  n )
{ if  ( n == 0 )
        return  1.0 ;
   double  result = real ;
   for ( int  i = 2 ;  i <= n ;  i ++ )
        result *= real ;
  return  result;
}
```

```
// 例3-7  使用默认参数
#include<iostream>
using namespace std ;
double  power ( double  real, int n = 2 ) ;
int main ( )
{ double  r = 3.0 ;
  cout  <<  power ( r )  <<  endl ;
  cout  <<  power ( r, 3 )  <<  endl ;
}
double  power ( double  real ,   int  n )
{ if  ( n == 0 )
      return  1.0 ;
  double  result = real ;
  for ( int  i = 2 ;  i <= n ;  i ++ )
      result *= real ;
 return  result;
}
```

定义默认参数

```
// 例3-7  使用默认参数
#include<iostream>
using namespace std ;
double  power ( double  real,  int n = 2  ) ;
int main ( )
{ double  r = 3.0 ;
   cout  <<  power ( r ) << endl ;
   cout  <<  power ( r, 3 )  <<  endl ;
}
double  power ( double  real ,   int  n )
{ if  ( n == 0 )
        return  1.0 ;
   double  result = real ;
   for ( int  i = 2 ;  i <= n ;  i ++ )
        result *= real ;
  return  result;
}
```

使用默认参数
power (r， 2)

```
// 例3-7  使用默认参数
#include<iostream>
using namespace std ;
double  power ( double  real,  int n = 2  ) ;
int main ( )
{ double  r = 3.0 ;
  cout  <<  power ( r )  <<  endl ;
  cout  <<  power ( r, 3 ) <<  endl ;
}
double  power ( double  real ,   int  n )
{ if  ( n == 0 )
      return  1.0 ;
  double  result = real ;
  for ( int  i = 2 ;  i <= n ;  i ++ )
      result *= real ;
 return  result;
}
```

不使用默认参数

3. 默认参数

```
int f ( ) ;
……
void delay ( int k , int time = f ( ) ) ;
```

3. 默认参数

```
int  f ( ) ;
……
void  delay ( int k ,  int time = f ( ) ) ;

void  ferror1 ( int x ,  int y = 1 ,  int  z ) ;
```

3. 默认参数

```
int  f ( ) ;
……
void  delay ( int k ,  int time = f ( ) ) ;

void  ferror1 ( int x ,  int y = 1 ,  int  z ) ;

void  ferror2 ( int x ,  int y = 0 ) ;
void  ferror2 ( int x ) ;
ferror2 ( 3 ) ;        // 调用哪个函数？
```

3.2.2　指针参数

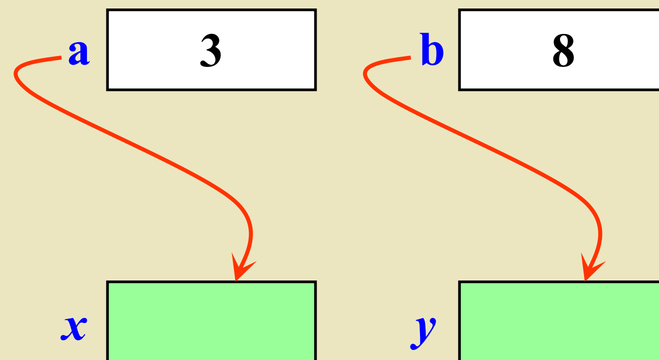
- 形参指针对应的实际参数是地址表达式，即对象的指针
- 实际参数把对象的地址值赋给形式参数名标识的指针变量
- 被调用函数通过形参指针间接访问实参所指对象

3.2.2 指针参数

```
// 例3-8  交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int * ,   int *  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( &a , &b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  * x ,  int * y )
{  int  temp = * x ;
   * x  =  * y  ;
   * y =  temp ;
}
```

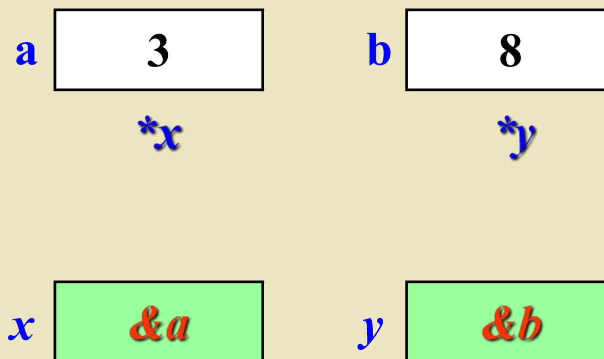
3.2.2 指针参数

```
// 例3-8 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int * ,   int *  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( &a , &b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  * x ,  int * y )
{  int  temp = * x ;
   * x  =  * y  ;
   * y =  temp ;
}
```



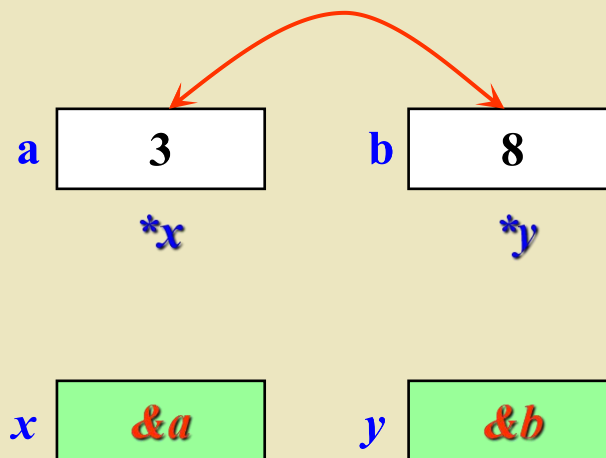
3.2.2 指针参数

```
// 例3-8  交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int * ,   int *  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( &a , &b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  * x , int * y )
{  int  temp = * x ;
   * x  =  * y  ;
   * y =  temp ;
}
```

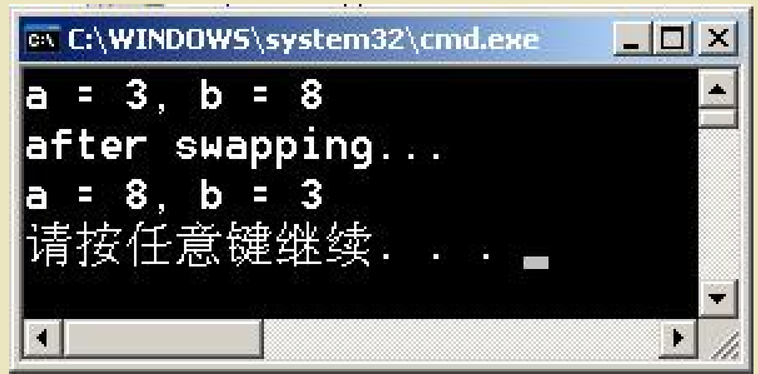


3.2.2 指针参数

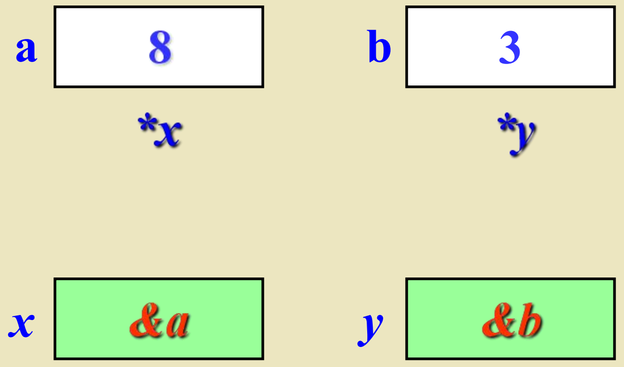
```
// 例3-8  交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int * ,  int * ) ;
int main ()
{  int  a = 3 ,  b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( &a , &b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  * x , int * y )
{  int  temp = * x ;
   * x  =  * y  ;
   * y =  temp ;
}
```



3.2.2　指针参数




```
// 例3-8  交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int * ,   int * ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( &a , &b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  * x , int * y )
{  int  temp = * x ;
   * x  =  * y  ;
   * y =  temp ;
}
```



3.2.2 指针参数

用 const 约束指针参数

无约束

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x 20

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

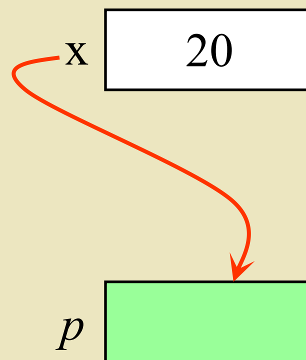
x 20

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

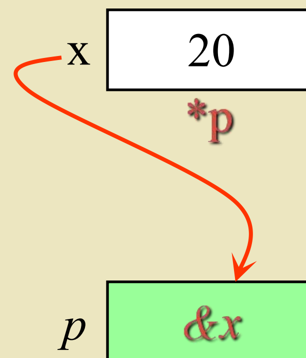


3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

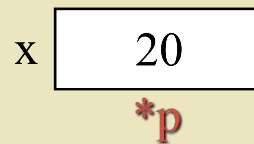
无约束



3.2.2 指针参数

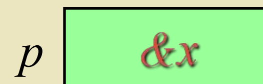
```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

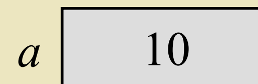
用 const 约束指针参数



x 20

*p

p 

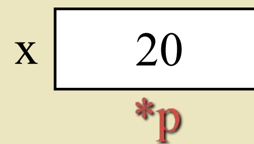
a 

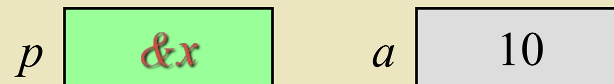
3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束



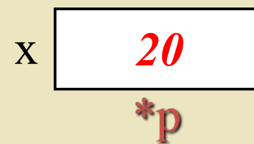


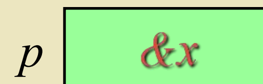
3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x 

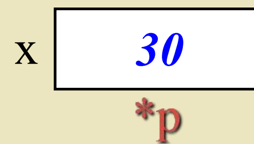
p 

a

3.2.2 指针参数

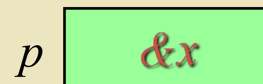
```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数



x 30

*p

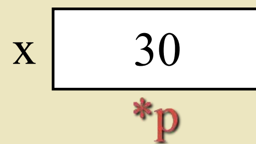
p 

a

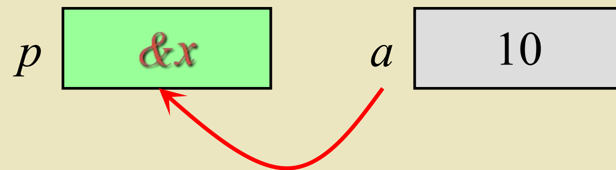
3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数



x 30
*p

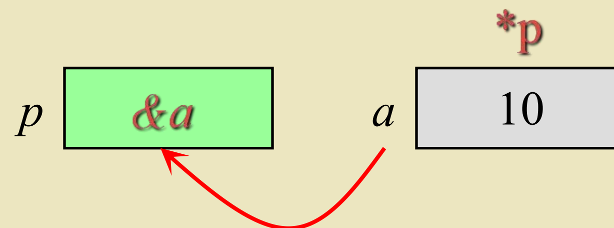


3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

x 30



3.2.2　指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x: 30

p: &a　　*p → a: 10

3.2.2　指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x　30

p　&*a*　　*a*　10　（*p）

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x: 30

p: *&a*

a: *20* (*p)

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

无约束

x 30

*p

p &a   *a* 20

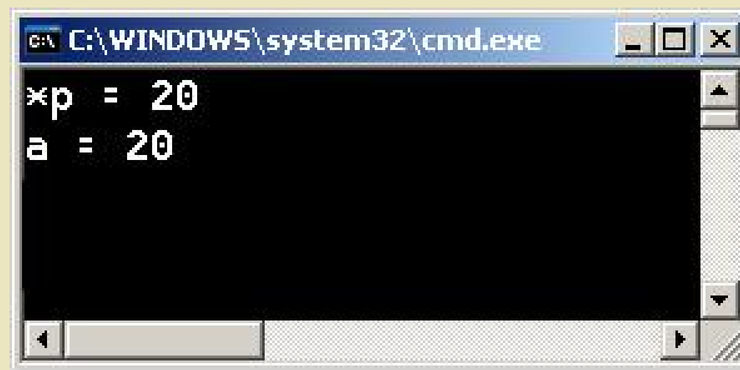
C:\WINDOWS\system32\cmd.exe

```
*p = 20
a = 20
```

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

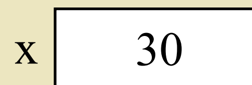
用 const 约束指针参数

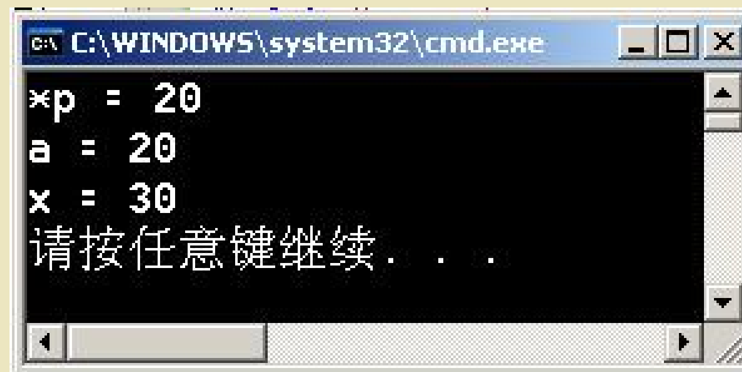


3. 2. 2　指针参数

```
#include <iostream>
using namespace std ;
void func ( int * p ) ;
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

x 



3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( const int * p )
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( const int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

约束形参对象

3.2.2 指针参数

用 const 约束指针参数

```
#include <iostream>
using namespace std ;
void func ( const int * p )
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( const int * p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

约束形参对象

不能修改常对象

3.2.2 指针参数

```
#include <iostream>
using namespace std ;
void func ( int * const p )
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * const p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

用 const 约束指针参数

约束形参指针

3.2.2 指针参数

用 const 约束指针参数

```
#include <iostream>
using namespace std ;
void func ( int * const p )
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * const p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

约束形参指针

不能修改指针常量

3.2.2 指针参数

用 const 约束指针参数

```
#include <iostream>
using namespace std ;
void func ( int * const p )
int main()
{ int x = 20 ;
  func( &x ) ;
  cout << "x = " << x << endl ;
}
void func ( int * const p )
{ int a = 10 ;
  *p += a ;
  p = &a;
  *p = *p + a ;
  cout << "*p = " << *p << endl ;
  cout << "a = " << a << endl ;
}
```

使用 const限 定指针

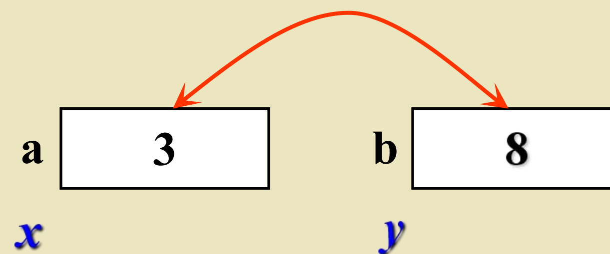
可以保护实参对象

3.2.3　引用参数

- 当形参为引用参数，调用函数时，形参是实参的别名
- 执行函数时，通过别名在实参上操作
- 函数返回，实参的别名取消。

3.2.3 引用参数

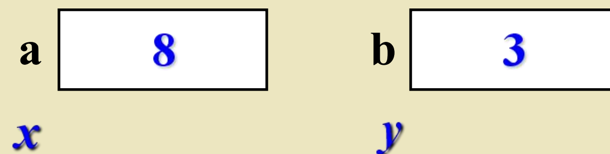
```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int & ,   int &   ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap (int  & x ,  int & y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```



3.2.3 引用参数

比较

```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int & ,   int &  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap (int  & x ,  int & y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```



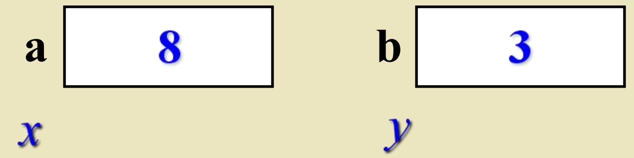
在实参对象上操作

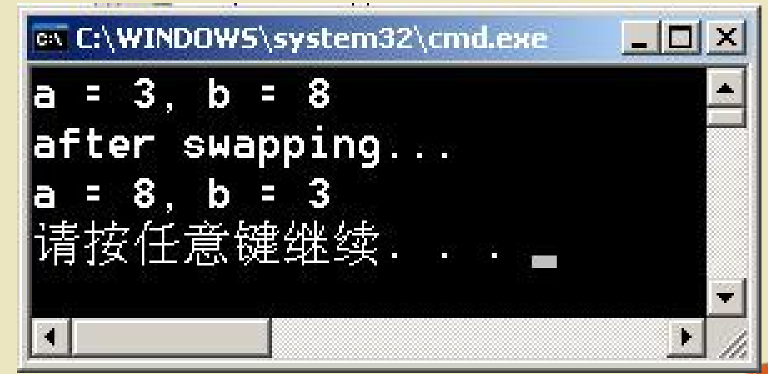
3.2.3 引用参数

```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int & ,   int &  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap (int  & x ,  int & y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```

执行swap函数时

x，y分别是a，b的别名

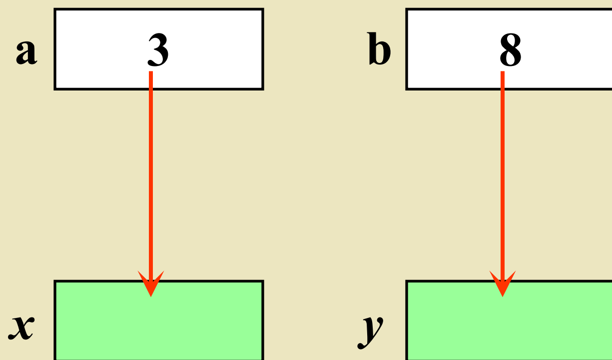




3.2.3 引用参数

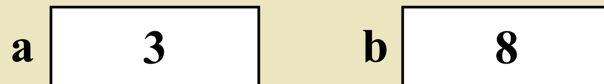
```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int ,  int  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  x ,  int  y )
{  int  temp =  x ;
    x  =  y  ;
    y =  temp ;
}
```

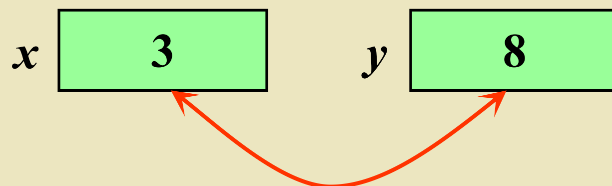
再比较



3.2.3 引用参数

```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int ,   int  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  x ,  int  y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```





3.2.3 引用参数

```
// 交换对象的值
#include<iostream>
using namespace std ;
void  swap ( int ,   int  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  x ,  int  y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```

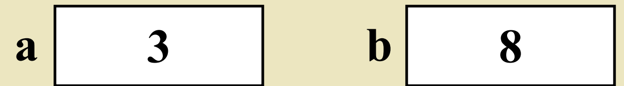
再比较

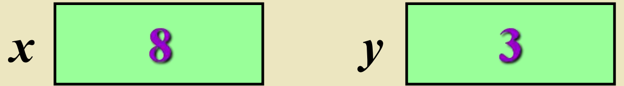
a	3	b	8
x	8	*y*	3

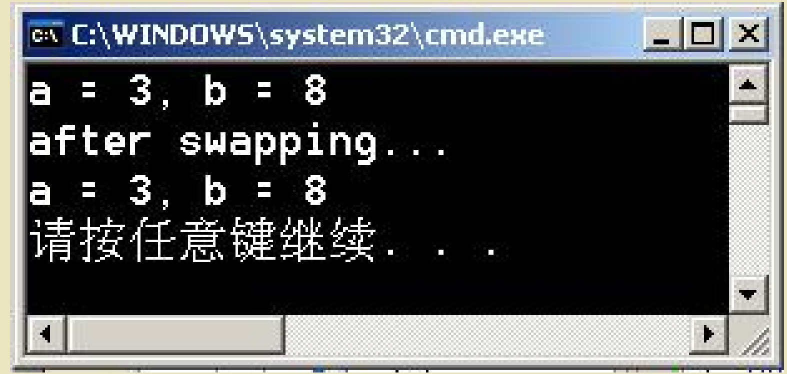
这是传值操作

3.2.3 引用参数

```
// 交换对象的值
#include<iostream>using namespace std ;
void  swap ( int ,  int  ) ;
int main ()
{  int  a = 3 ,   b = 8 ;
   cout << "a = " << a << ", b = " << b << endl ;
   swap ( a , b ) ;
   cout <<"after swapping... \n" ;
   cout << "a = " << a << ", b = " << b << endl ;
}
void  swap ( int  x ,  int  y )
{  int  temp =  x ;
   x  =  y  ;
   y =  temp ;
}
```







3.2.3 引用参数

用 const 约束引用参数

```
// 例3-9  不同数制输出正整数
#include<iostream>
using namespace std ;
#include<iomanip>
void display( const int & rk )
{ cout << dec << rk << " :\n"
          << "dec : " << rk << endl
          << "oct : " << oct << rk << endl
          << "hex : " << hex << rk << endl;
}
int main()
{  int  x ;
   cout << "number : " ;
   cin >> x ;
   display( x ) ;
   display( 4589 ) ;
}
```

3.2.3 引用参数

用 const 约束引用参数

```
// 例3-9  不同数制输出正整数
#include<iostream>
using namespace std ;
#include<iomanip>
void display( const int & rk )
{ cout << dec << rk << " :\n"
        << "dec : " << rk << endl
        << "oct : " << oct << rk << endl
        << "hex : " << hex << rk << endl;
}
int main()
{  int  x ;
   cout << "number : " ;
   cin >> x ;
   display( x ) ;
   display( 4589 ) ;
}
```

在实参对象上只读访问

3.2.3　引用参数

```
// 例3-9  不同数制输出正整数
#include<iostream>
using namespace std ;
#include<iomanip>
void display( const int & rk )
{ cout << dec << rk << " :\n"
        << "dec : " << rk << endl
        << "oct : " << oct << rk << endl
        << "hex : " << hex << rk << endl;
}
int main()
{  int  x ;
   cout << "number : " ;
   cin >> x ;
   display( x ) ;
   display( 4589 ) ;
}
```

用 const 约束引用参数

- 只有常引用对应的实参可以是常量或表达式
- 非约束的引用参数对应的实参必须是对象名

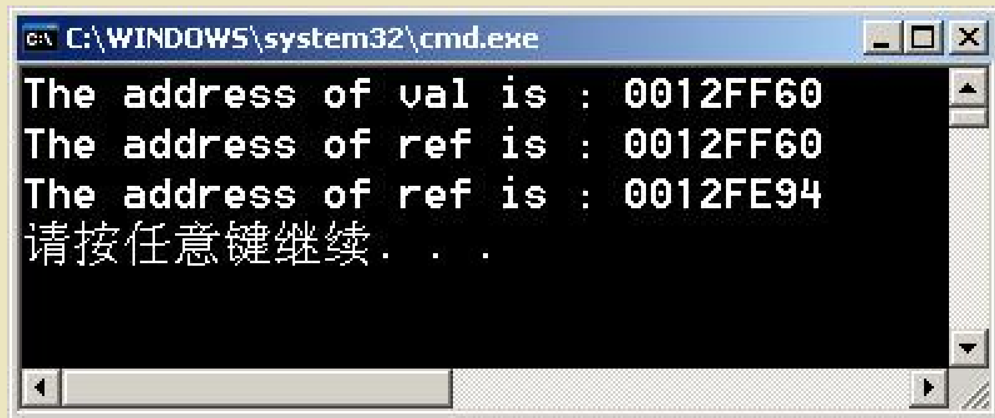
3.2.3 引用参数

用 const 约束引用参数

```
// 例3-9  不同数制输出正整数
#include<iostream>
using namespace std ;
#include<iomanip>
void display( const int & rk )
{ cout << dec << rk << " :\n"
        << "dec : " << rk << endl
        << "oct : " << oct << rk << endl
        << "hex : " << hex << rk << endl;
}
int main()
{  int  x ;
   cout << "number : " ;
   cin >> x ;
   display( x ) ;
   display( 4589 ) ;
}
```

3.2.3 引用参数

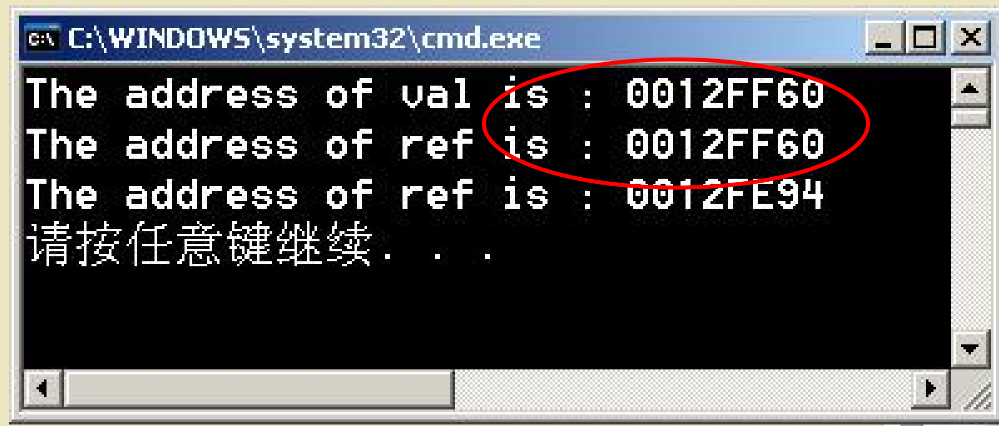
```
// 例3-10 常引用参数的匿名对象测试
#include<iostream>
using namespace std ;
void anonym ( const int & ref )
{ cout << "The address of ref is : " << &ref << endl;
  return ;
}
int main()
{ int val = 10 ;
  cout << "The address of val is : " << &val << endl;
  anonym( val ) ;
  anonym( val + 5 ) ;
}
```



3.2.3 引用参数

```
// 例3-10 常引用参数的匿名对象测试
#include<iostream>
using namespace std ;
void anonym ( const int & ref )
{ cout << "The address of ref is : " << &ref << endl;
  return ;
}
int main()
{ int val = 10 ;
  cout << "The address of val is : " << &val << endl;
  anonym( val ) ;
  anonym( val + 5 ) ;
}
```

它们是
同一对象的地址

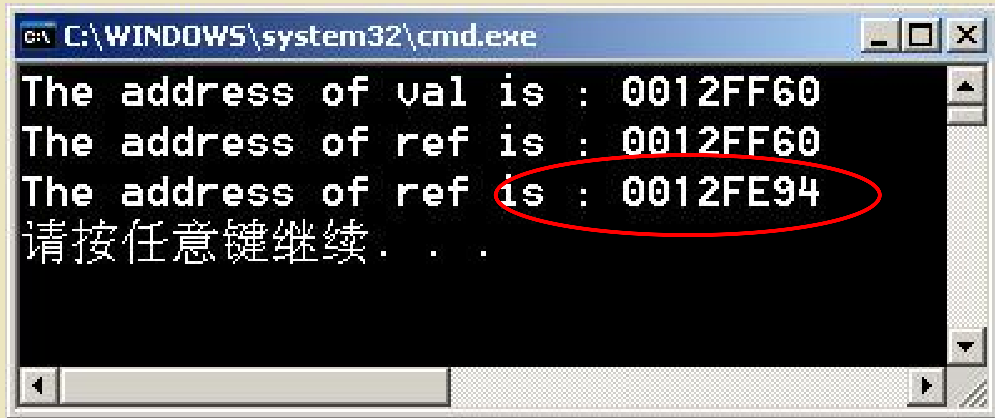


3.2.3 引用参数

// 例3-10 常引用参数的匿名对象测试

```
#include<iostream>
using namespace std ;
void anonym ( const int & ref )
{ cout << "The address of ref is : " << &ref << endl;
  return ;
}
int main()
{ int val = 10 ;
  cout << "The address of val is : " << &val << endl;
  anonym( val ) ;
  anonym( val + 5 ) ;
}
```

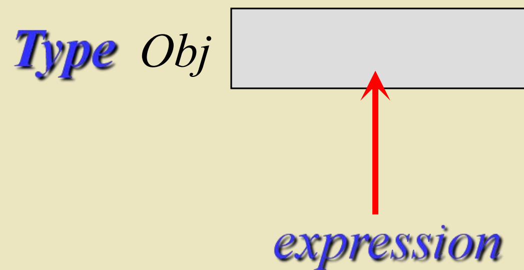
匿名对象的地址



3.2.4 函数的返回类型

- 函数通过匿名对象返回结果值
- 函数值的类型是匿名对象的类型
- **return** 语句把表达式的值赋给匿名对象

```
Type FunctionName ()
{ // statements
  return expression ;
}
```



- ***Type*** 可以为各种C++基本数据类型、类类型，以及这些类型的指针或引用

1. 返回基本类型

```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main()
{  double  vol,  r,  h ;
    cin  >>  r >> h ;
    vol = volume ( r,  h ) ;
    cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回过程：

1．返回基本类型

```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main()
{  double  vol,  r,  h ;
   cin  >>  r >> h ;
   vol = volume ( r,  h ) ;
   cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回过程：

- 对表达式求值

3.14 * radius * radius * height

1. 返回基本类型

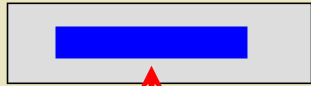
```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main ()
               r,  h ;
    cin  >>  r >> h ;
    vol = volume ( r,  h ) ;
    cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回类型

返回过程:

- 对表达式求值
- 向匿名对象赋值

double *Obj*



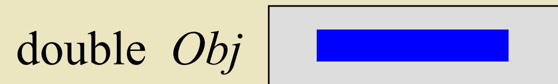
3.14 * radius * radius * height

1．返回基本类型

```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main()
{  double  vol,  r,  h ;
    cin  >>  r >> h ;
    vol = volume ( r,  h ) ;
    cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回过程：

- 对表达式求值
- 向匿名对象赋值
- 返回调用点，执行赋值操作

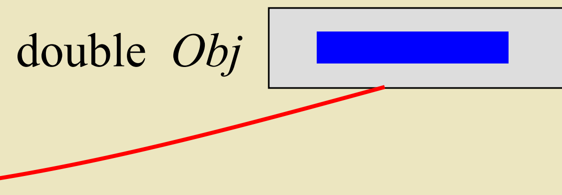


1．返回基本类型

```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main()
{  double  vol,  r,  h ;
   cin  >>  r >> h ;
   vol = volume ( r,  h ) ;
   cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回过程：

- 对表达式求值
- 向匿名对象赋值
- 返回调用点，执行赋值操作



1. 返回基本类型

```
// 例3-11 求圆柱体体积
#include<iostream>
using namespace std ;
double volume ( double ,  double ) ;
int main()
{  double  vol,  r,  h ;
   cin  >>  r >> h ;
   vol = volume ( r,  h ) ;
   cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{    return   3.14 * radius * radius * height ;  }
```

返回过程:

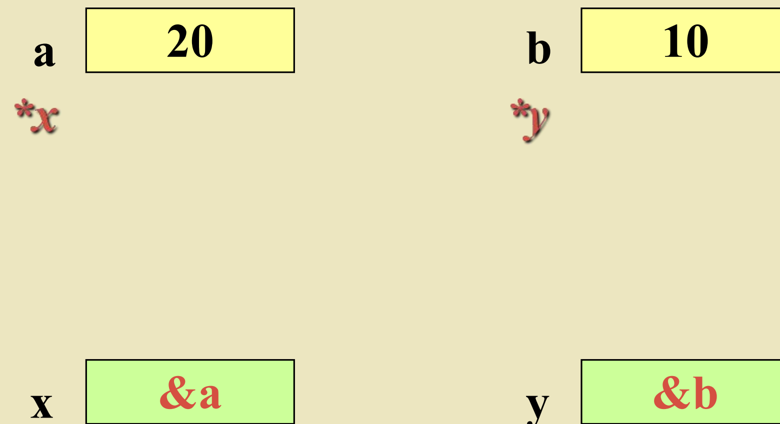
- 对表达式求值
- 向匿名对象赋值
- 返回调用点，执行赋值操作
- 撤销匿名对象

2. 返回指针

```
// 例3-12 返回较大值变量的指针
#include<iostream>
using namespace std ;
int * maxPoint(int * x, int * y ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << * maxPoint( &a, &b ) <<endl ;
}
int * maxPoint(int * x, int * y )
{ if ( *x > *y )  return x ;
  return y ;
}
```

返回过程:

- 对*地址*表达式求值

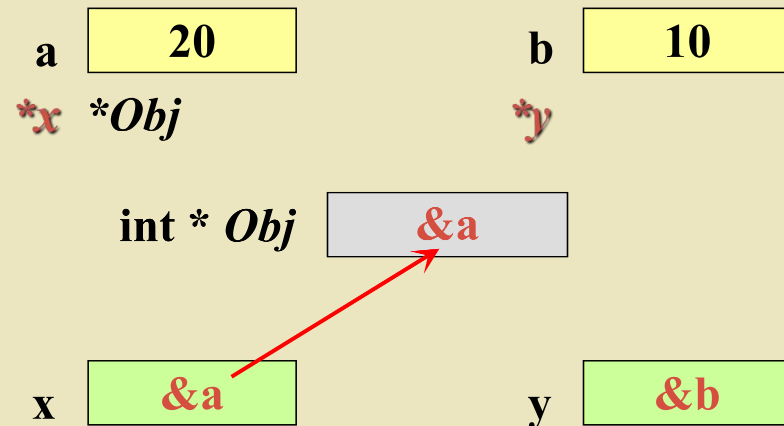


2. 返回指针

```
// 例3-12 返回较大值变量的指针
#include<iostream>
using namespace std ;
int * maxPoint(int * x, int * y ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << * maxPoint( &a, &b ) <<endl ;
}
int * maxPoint(int * x, int * y )
{ if ( *x > *y )  return x ;
  return y ;
}
```

返回过程：

- 对*地址*表达式求值
- 向匿名对象赋值

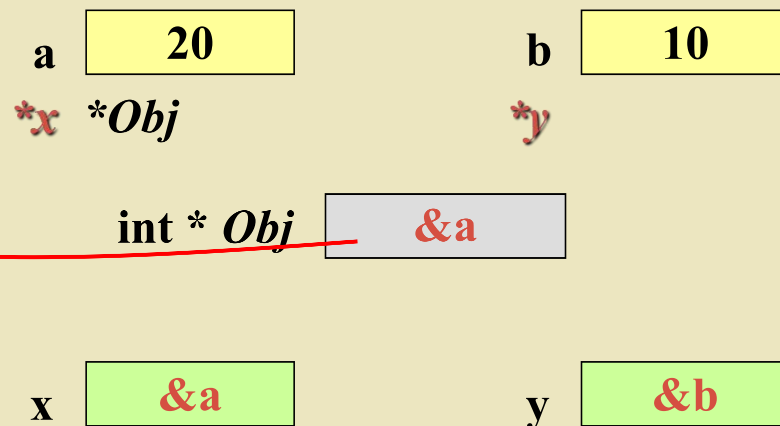


2. 返回指针

```
// 例3-12 返回较大值变量的指针
#include<iostream>
using namespace std ;
int * maxPoint(int * x, int * y ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << * maxPoint( &a, &b ) <<endl ;
}
int * maxPoint(int * x, int * y )
{ if ( *x > *y )  return x ;
  return y ;
}
```

返回过程：

- 对*地址*表达式求值
- 向匿名对象赋值
- 返回调用点，执行输出操作



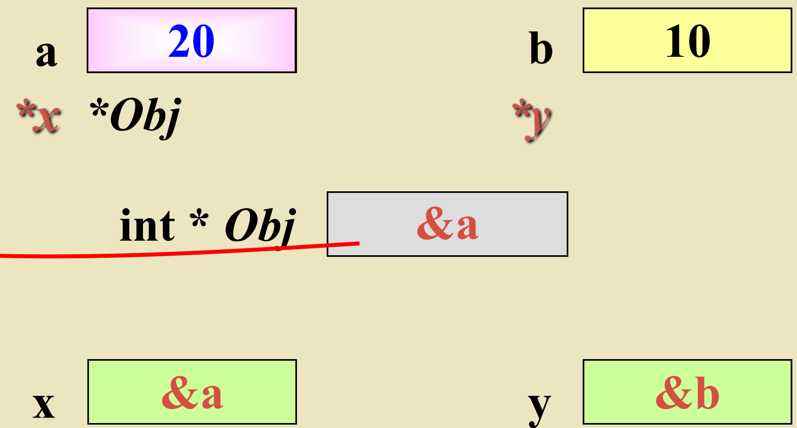
2. 返回指针

```
// 例3-12 返回较大值变量的指针
#include<iostream>
using namespace std ;
int * maxPoint(int * x, int * y ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << * maxPoint( &a, &b ) <<endl ;
}
int * maxPoint(int * x, int * y )
{ if ( *x > *y )  return x ;
  return y ;
}
```

20

返回过程:

- 对*地址*表达式求值
- 向匿名对象赋值
- 返回调用点，执行输出操作



2. 返回指针

```
// 例3-12 返回较大值变量的指针
#include<iostream>
using namespace std ;
int * maxPoint(int * x, int * y ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << * maxPoint( &a, &b ) <<endl ;
}
int * maxPoint(int * x, int * y )
{ if ( *x > *y )  return x ;
  return y ;
}
```

```
20
```

返回过程：

- 对*地址*表达式求值
- 向匿名对象赋值
- 返回调用点，执行输出操作
- 撤销匿名对象和形参对象

a	20	b	10

2. 返回指针

```
// 不应该返回局部量的指针
#include<iostream>
using namespace std ;
int * f1Warning()
{ int  temp = 100 ;
  return  & temp ;
}
int main()
{ cout << "temp=" << *f1Warning() << endl;
}
```

2. 返回指针

```
// 不应该返回局部量的指针
#include<iostream>
using namespace std ;
int * f1Warning()
{ int temp = 100 ;
  return & temp ;
}
int main()
{ cout << "temp=" << *f1Warning() << endl;
}
```

该对象只在函数体内有效

warning C4172: 返回局部变量或临时变量的地址

3．返回引用

- **C++函数返回对象引用时，不产生返回实际对象时的副本，返回时的匿名对象是实际返回对象的引用**

3. 返回引用

返回过程:

- 决定引用对象

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int & maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int & maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

a 20 b 10

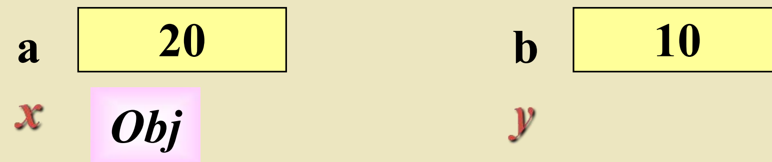
x *y*

3. 返回引用

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int & maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int & maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

返回过程:

- 决定引用对象
- 匿名对象绑定

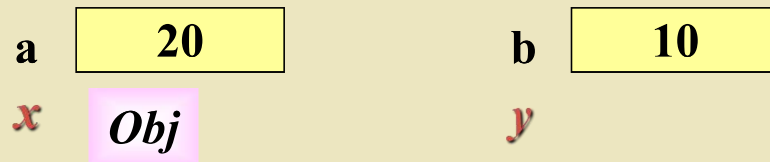


3．返回引用

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int & maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int & maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

返回过程：

- 决定引用对象
- 匿名对象绑定
- 返回调用点，执行输出操作



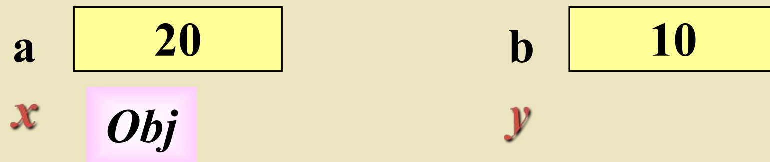
3．返回引用

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int & maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int & maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

函数返回
对象 a 的引用

返回过程:

- 决定引用对象
- 匿名对象绑定
- 返回调用点，执行输出操作



3．返回引用

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int & maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int & maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

返回过程：

- 决定引用对象
- 匿名对象绑定
- 返回调用点，执行输出操作
- 撤销匿名对象和形参对象的引用

a	20	b	10

3. 返回引用

```
// 例3-13  返回较大值变量的引用
#include<iostream>
using namespace std ;
int      maxRef( int & , int & ) ;
int main()
{ int a, b ;
  cout << "Input a, b : " ;
  cin >> a >> b ;
  cout << maxRef( a, b ) <<endl ;
}
int      maxRef( int & x, int & y )
{ if ( x > y ) return x ;
  return y ;
}
```

函数返回什么类型？

可以得到正确的结果吗？

运行机制有什么区别？

3．返回引用

```
// 不应该返回局部量的引用
#include<iostream>
using namespace std ;
int & f2Warning()
{int temp=100;
  return temp;
}
int main()
{ cout << "temp=" << f2Warning() << endl;
}
```

3. 返回引用

```
// 不应该返回局部量的引用
#include<iostream>
using namespace std ;
int & f2Warning()
{int temp = 100;
  return temp;
}
int main()
{ cout << "temp=" << f2Warning() << endl;
}
```

该对象只在函数体内有效

warning C4172: 返回局部变量或临时变量的地址

3.3 函数调用机制

函数之间的关系

- C++程序从主函数开始执行
- 主函数由操作系统调用
- 函数可以互相调用，自身调用
- 所有函数定义是平行的，不能嵌套定义

3.3 函数调用机制

函数之间的关系

主函数调用其它函数

其它函数互相调用

函数递归调用

main

fun1 fun2 fun3

fun4 fun5 fun6 fun7

3.3　函数调用机制

函数调用用堆栈管理

——堆栈是先进后出的数据结构

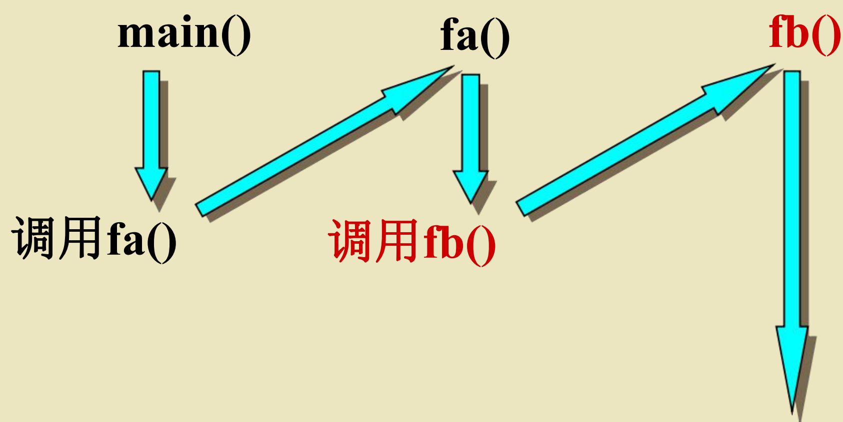
函数调用时入栈操作：

- 建立被调用函数的栈空间
- 保护调用函数运行状态和返回地址
- 传递参数
- 控制权交给被调用函数

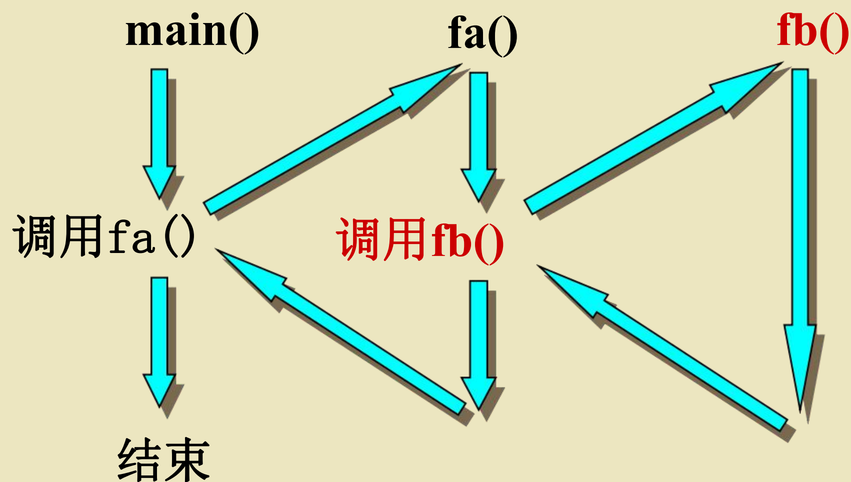
函数返回时出栈操作：

- 返回值保存在临时空间
- 恢复调用函数运行状态
- 释放栈空间
- 根据地址返回调用函数

3.3.1 嵌套调用



3.3.1 嵌套调用



函数嵌套调用示例

定义一个求 bin (n, k) 的函数。 $bin(n,k) = \dfrac{n!}{k!(n-k)!}$

分析:

定义函数 fact (m) = m !

bin (n , k) = fact (n) / (fact (k) * fact (n - k))

由主函数输入数据 a 、 b ， 求 bin (a , b)

```
//例3-14  函数嵌套调用示例
#include<iostream>
using namespace std ;
long  fact ( int  m )
{ int  i ;  long  sum = 1 ;
   for  ( i = 1 ;  i <= m ;  sum *= i ,  i++ ) ;
   return  sum ;
}
long  bin ( int  n ,  int  k )
{   return  ( fact ( n ) / ( fact ( k ) * fact ( n-k ) ) )  ;  }
int main ()
{ int a ,  b;  long  f1 ,  f2 ;
  cout << "Please input data a and b:" << endl ;
  cin >> a >> b ;
  f1 = fact ( a ) / ( fact ( b ) * fact ( a-b ) ) ;
  cout << " first:   bin(" << a << ', ' << b << ")= " << f1 << endl ;
  f2 = bin ( a , b ) ;
  cout << " second:  bin(" << a << ', ' << b << ")= " << f2 << endl ;
}
```

```
//例3-14  函数嵌套调用示例
#include<iostream>
using namespace std ;
long  fact ( int  m )
{ int  i ;  long  sum = 1 ;
   for  ( i = 1 ;  i <= m ;  sum *= i ,  i++ ) ;
   return  sum ;
}
long  bin ( int  n ,  int  k )
{   return  ( fact ( n ) / ( fact ( k ) * fact ( n-k ) ) )  ;  }
int main ()
{ int a ,  b;  long  f1 ,  f2 ;
  cout << "Please input data a and b:" << endl ;
  cin >> a >> b ;
  f1 = fact ( a ) / ( fact ( b ) * fact ( a-b ) ) ;
  cout << " first:   bin(" << a << ',' << b << ")= " << f1 << endl ;
  f2 = bin ( a , b ) ;
  cout << " second:  bin(" << a << ',' << b << ")= " << f2 << endl ;
}
```

```
//例3-14  函数嵌套调用示例
#include<iostream>
using namespace std ;
long  fact ( int  m )
{ int  i ;  long  sum = 1 ;
   for  ( i = 1 ;  i <= m ;  sum *= i ,  i++ ) ;
   return  sum ;
}
long  bin ( int  n ,  int  k )
{   return  ( fact ( n ) / ( fact ( k ) * fact ( n-k ) ) )  ;  }
int main ()
{ int a ,  b;  long  f1 ,  f2 ;
  cout << "Please input data a and b:" << endl ;
  cin >> a >> b ;
  f1 = fact ( a ) / ( fact ( b ) * fact ( a-b ) ) ;
  cout << " first:   bin(" << a << ', ' << b << ")= " << f1 << endl ;
  f2 = bin ( a , b ) ;
  cout << " second:  bin(" << a << ', ' << b << ")= " << f2 << endl ;
}
```

```
//例3-14  函数嵌套调用示例
#include<iostream>
using namespace std ;
long  fact ( int  m )
{ int  i ;  long  sum = 1 ;
   for  ( i = 1 ;  i <= m ;  sum *= i ,  i++ ) ;
   return  sum ;
}
long  bin ( int  n ,  int  k )
{   return  ( fact ( n ) / ( fact ( k ) * fact ( n-k ) ) )  ;  }
int main ()
{ int a ,  b;  long  f1 ,  f2 ;
  cout << "Please input data a and b:" << endl ;
  cin >> a >> b ;
  f1 = fact ( a ) / ( fact ( b ) * fact ( a-b ) ) ;
  cout << " first:   bin(" << a << ',' << b << ")= " << f1 << endl ;
  f2 = bin ( a , b ) ;
  cout << " second:  bin(" << a << ',' << b << ")= " << f2 << endl ;
}
```

3.3.2 递归调用

递归定义

一个对象部分地由它自己定义，或者是按它自己定义

例 1：　自然数定义

（1）1是自然数　　　（2）自然数的后继是自然数

例 2：　阶乘定义

（1）1 的阶乘等于 1　　　（2）n 的阶乘等于n乘以 n-1 的阶乘

3.3.2 递归调用

递归函数 —— 函数直接或间接调用自己

递归函数要求

- 递归形式（算法）
- 递归条件（缩小问题规模）
- 递归终止条件（基本情况）

- 递归结构是更强的循环结构
- 所有的循环结构都可以写成递归结构，反之不一定行

例3-15 求阶乘

$$\text{Factorial}(n) = n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n \geq 1 \end{cases}$$

```
int  Factorial ( int  n )
{ if  ( n = = 0 )
        return  1 ;
  else
        return   n * Factorial ( n - 1 ) ;
}
```

例3-15 求阶乘

$$\text{Factorial}(n) = n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n \geq 1 \end{cases}$$

```
int  Factorial ( int  n )
{ if  ( n = = 0 )
        return  1 ;
  else
        return   n * Factorial ( n - 1 ) ;
}
```

递归形式

例3-15 求阶乘

$$\text{Factorial}(n) = n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n \geq 1 \end{cases}$$

```
int  Factorial ( int  n )
{ if  ( n = = 0 )
        return  1 ;
  else
        return   n * Factorial ( n - 1 ) ;
}
```

递归终止条件
基本情况

例3-15 求阶乘

$$\text{Factorial}(n) = n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n \geq 1 \end{cases}$$

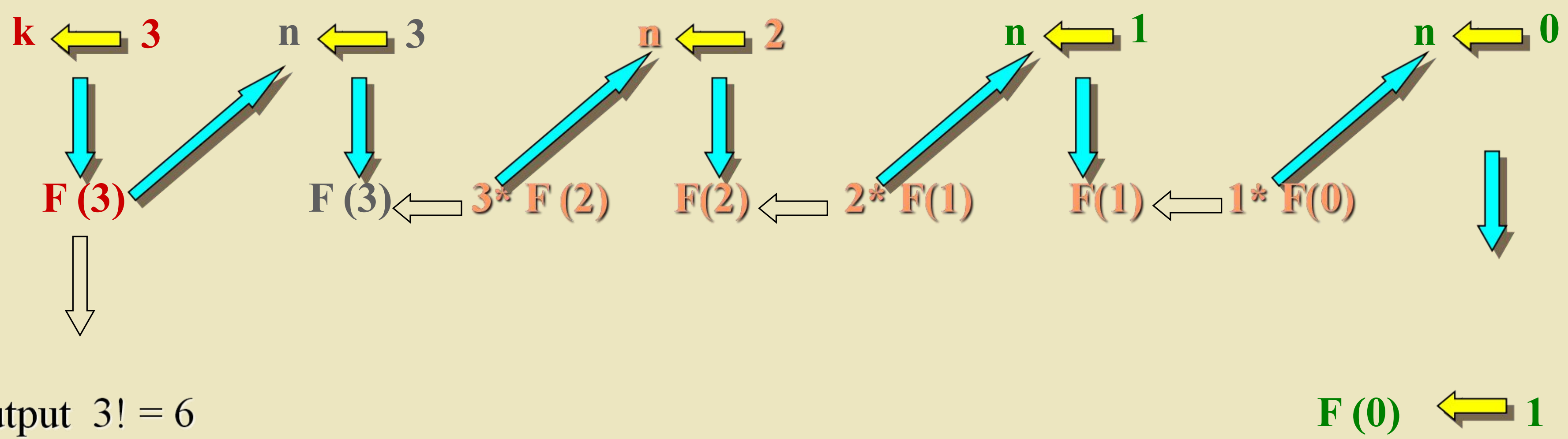
```
int  Factorial ( int  n )
{ if  ( n = = 0 )
        return  1 ;
  else
        return   n * Factorial ( n - 1 ) ;
}
```

修改递归条件

例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )        return  1 ;
  else          return   n * Factorial ( n - 1 ) ;
}
```

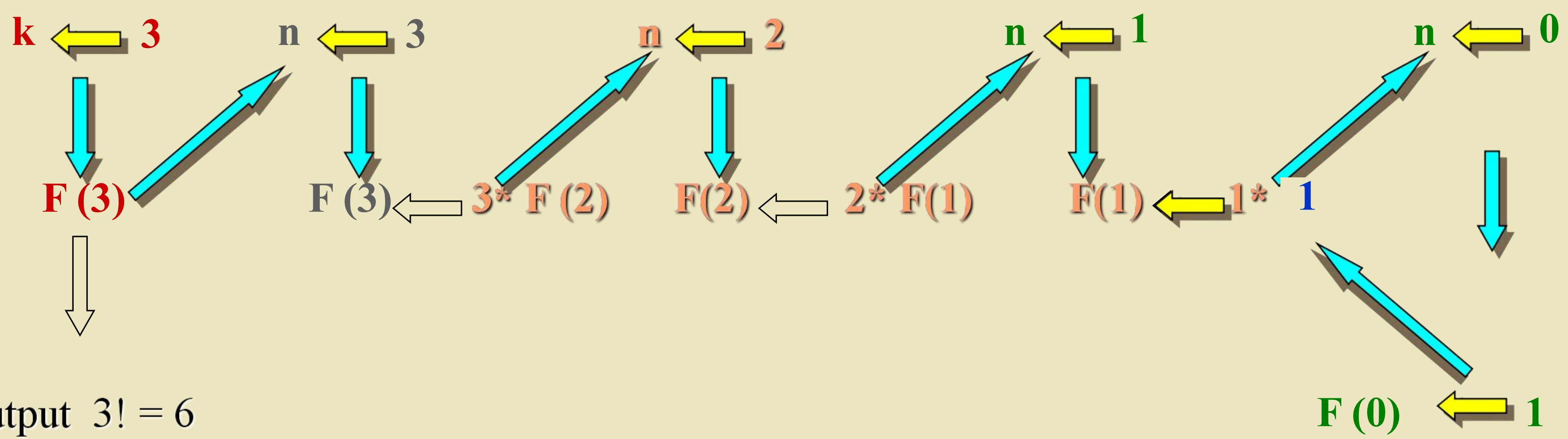
计算 Factorial (3) = 3!



例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )        return  1 ;
  else          return   n * Factorial ( n - 1 ) ;
}
```

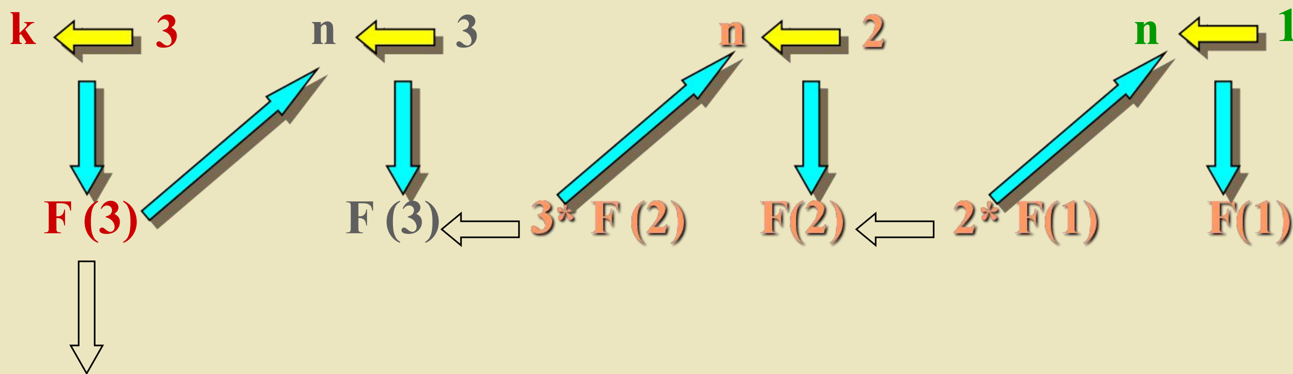
计算 Factorial (3) = 3!



例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )        return  1 ;
  else         return   n * Factorial ( n - 1 ) ;
}
```

计算 Factorial (3) = 3!

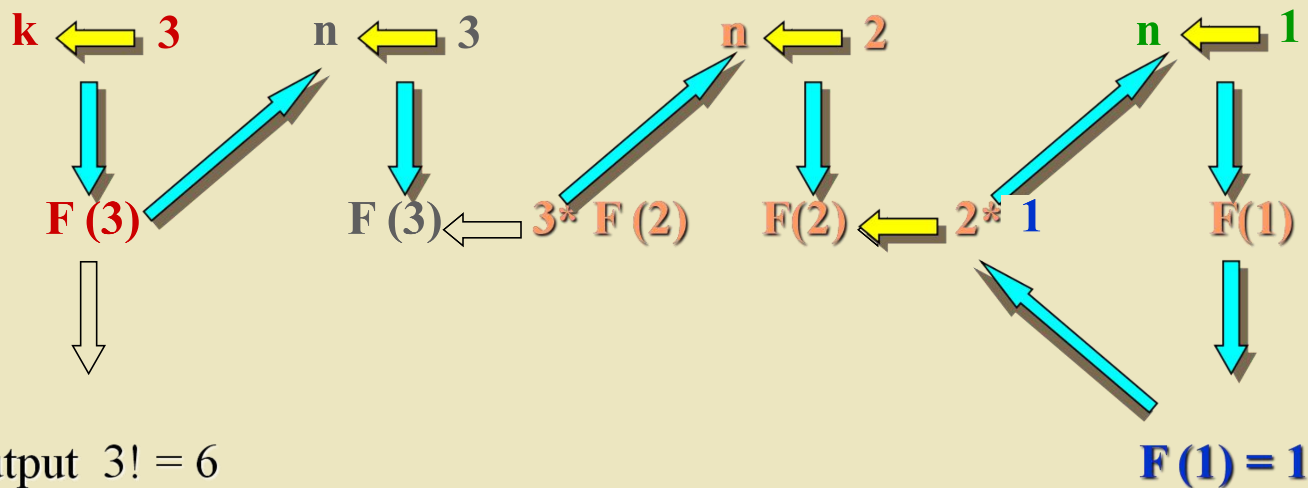


Output 3! = 6

例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else        return   n * Factorial ( n - 1 ) ;
}
```

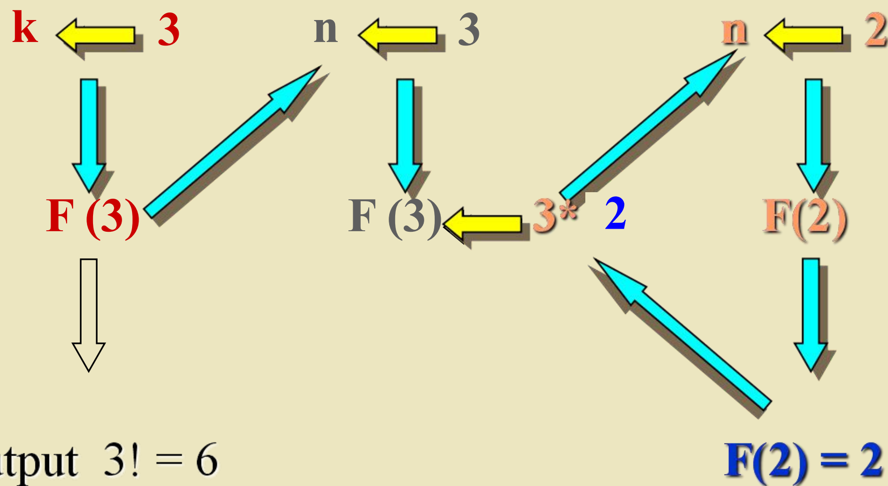
计算 Factorial (3) = 3!



例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else          return   n * Factorial ( n - 1 ) ;
}
```

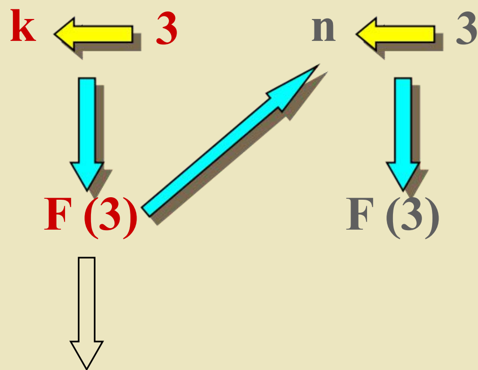
计算 Factorial (3) = 3!



例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else        return   n * Factorial ( n - 1 ) ;
}
```

计算 Factorial (3) = 3!

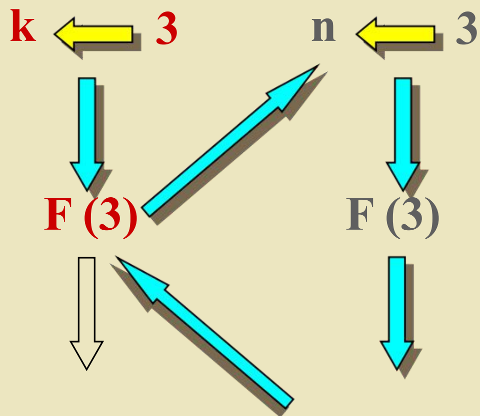


Output 3! = 6

例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )        return  1 ;
  else         return   n * Factorial ( n - 1 ) ;
}
```

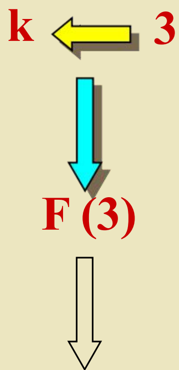
计算 Factorial (3) = 3!



例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else         return   n * Factorial ( n - 1 ) ;
}
```

计算 Factorial (3) = 3!

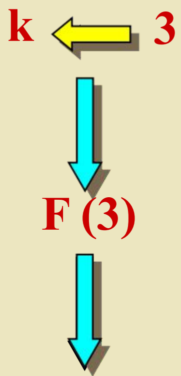


Output 3! = 6

例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else        return   n * Factorial ( n - 1 ) ;
}
```

计算 Factorial (3) = 3!

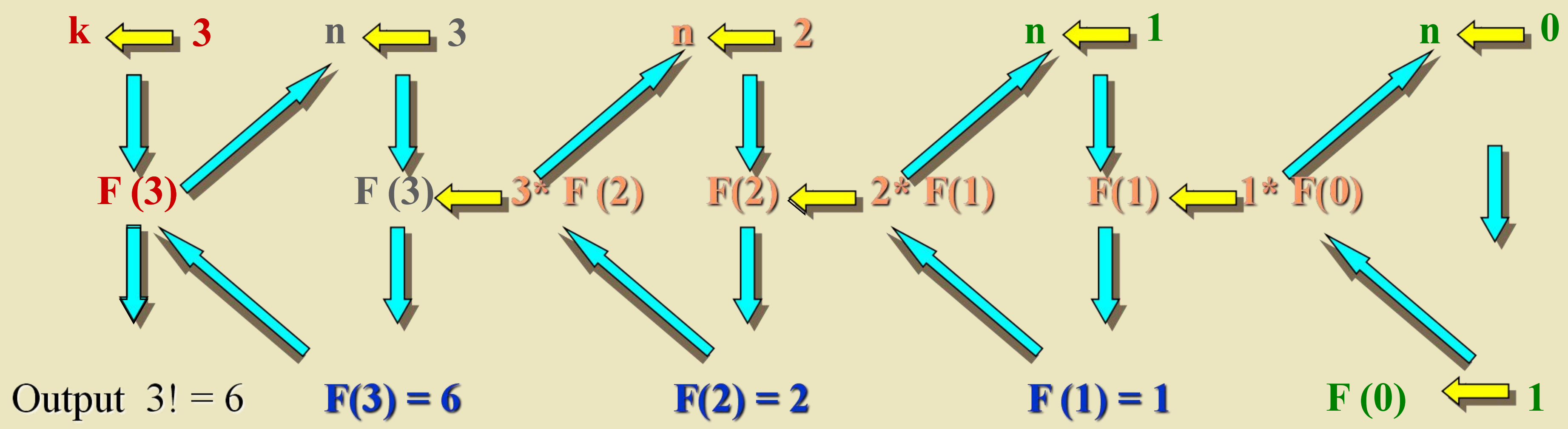


Output **3! = 6**

例3-15 求阶乘

```
int  Factorial ( int  n )
{ if  ( n = = 0 )       return  1 ;
  else        return   n * Factorial ( n - 1 ) ;
}
```

计算 Factorial (3) = 3!



例3-15 求阶乘

```
#include<iostream>
using namespace std ;
int  Factorial ( int ) ;
int main ()
{ int  k ;
  cout << "Compute Factorial(k) ,  Please input k: " ;
  cin >> k ;
  cout << k << "! = " << Factorial(k) << endl ;
}
int  Factorial ( int  n )
{ if ( n == 0 )
        return  1 ;
   else
        return   n * Factorial ( n - 1 ) ;
}
```

非递归调用

例3-15 求阶乘

```
#include<iostream>
using namespace std ;
int  Factorial ( int ) ;
int main ()
{ int  k ;
  cout << "Compute Factorial(k) ,  Please input k: " ;
  cin >> k ;
  cout << k << "! = " << Factorial(k) << endl ;
}
int  Factorial ( int  n )
{ if ( n == 0 )
        return  1 ;
   else
        return   n * Factorial ( n - 1 ) ;
}
```

递归调用

例3-15 求阶乘

```
#include<iostream>
using namespace std ;
int  Factorial ( int ) ;
int main ()
{ int  k ;
  cout << "Compute Factorial(k) ,  Please input k: " ;
  cin >> k ;
  cout << k << "! = " << Factorial(k) << endl ;
}
int  Factorial ( int  n )
{ if ( n == 0 )
       return  1 ;
   else
       return   n * Factorial ( n - 1 ) ;
}
```

注意传值参数

一般递归函数的形式

$$
\begin{aligned}
F(x_1,\ x_2,\ \ldots,\ x_n) \equiv\ & \text{if}\ (P_1)\ E_1 \\
& \text{else if}\ (P_2)\ E_2 \\
& \ldots\ldots \\
& \text{else if}\ (P_m)\ E_m \\
& \text{else}\ E_{m+1}
\end{aligned}
$$

P_i（$i = 1, 2, \ldots\ldots, m$）是测试表达式

E_i（$i = 1, 2, \ldots\ldots, m+1$）是表达式

F() 可以出现在 P_i 和 E_i 中

例3-16

递归定义斐波那契数列

$$F_n = \begin{cases} F_n = 1 & if \quad n = 1 \\ F_n = 1 & if \quad n = 2 \\ F_n = F_{n-1} + F_{n-2} & if \quad n > 2 \end{cases}$$

```
int  Fibonacci ( int  n )
  { if ( n <= 2 )
         return 1 ;
     else
        return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

Fibonacci (5)

Fibonacci (4)   *Fibonacci (3)*

Fibonacci (3)   *Fibonacci (2)*

Fibonacci (2)   *Fibonacci (1)*

n *stack*

2
3
4
5

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

Fibonacci (5)

Fibonacci (4)   *Fibonacci (3)*

Fibonacci (3)   *Fibonacci (2)*

1

Fibonacci (1)

n *stack*

1
3
4
5

```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

Fibonacci (5)

Fibonacci (4)

Fibonacci (3)

Fibonacci (3)

Fibonacci (2)

1

1

Fibonacci (1)

n *stack*

3
4
5

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else         return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

Fibonacci (5)

Fibonacci (4)   *Fibonacci (3)*

2

Fibonacci (3)   *Fibonacci (2)*

1   +   1

n *stack*

4
5

```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else         return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

Fibonacci (5)

Fibonacci (4)

Fibonacci (3)

2

Fibonacci (2)

n *stack*

2
4
5

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

Fibonacci (5)

Fibonacci (4)

Fibonacci (3)

2

1

Fibonacci (2)

n *stack*

4
5

```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

Fibonacci (5)

3

Fibonacci (4)

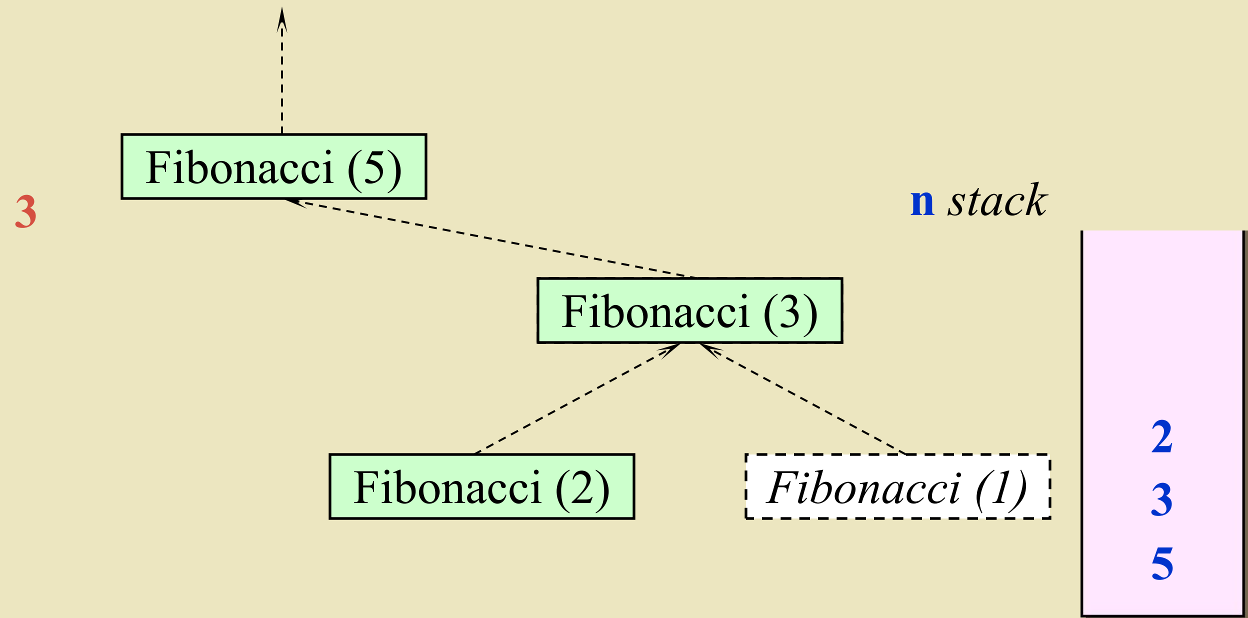
Fibonacci (3)

2 + 1

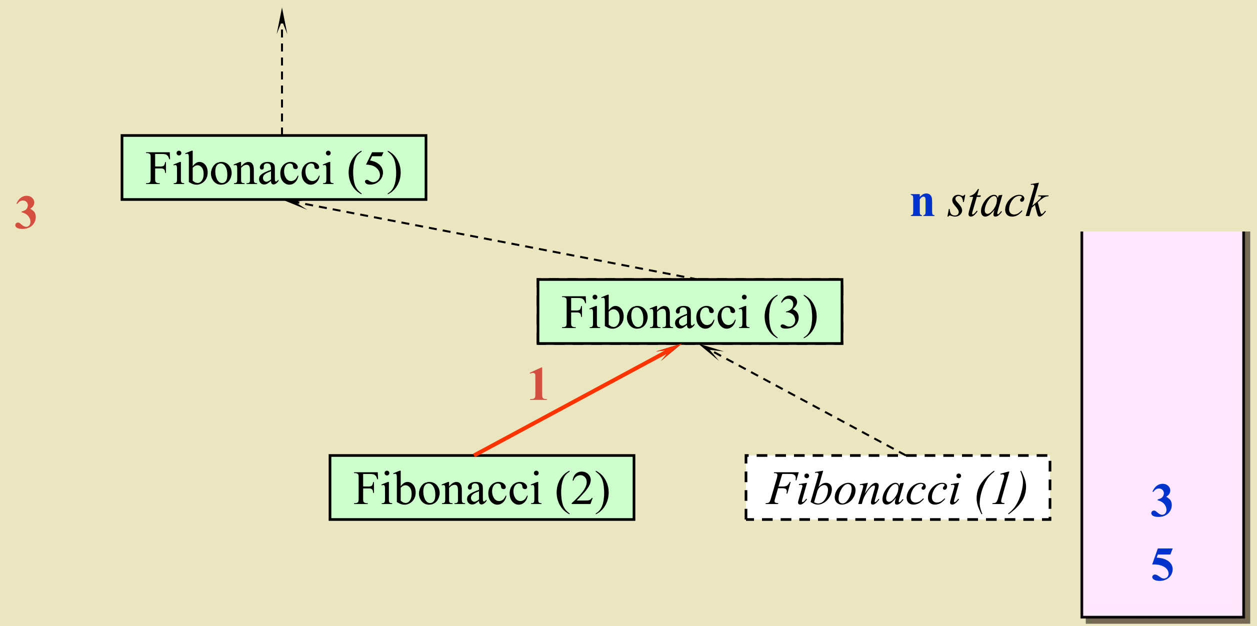
n *stack*

5

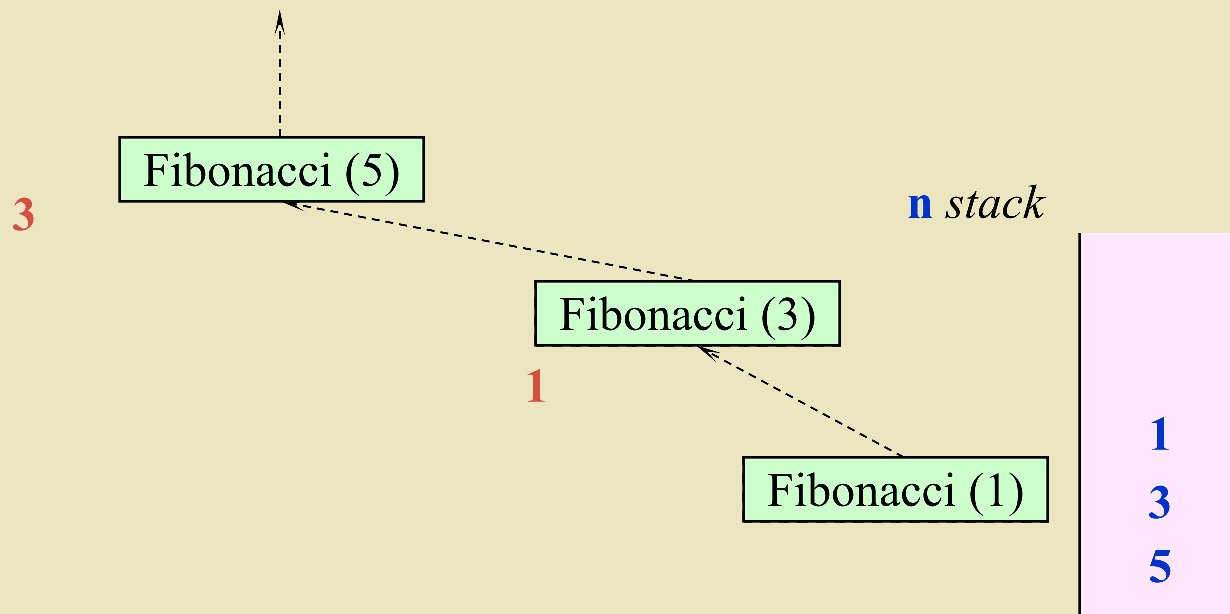
```
int  Fibonacci ( int  n )
  { if ( n <= 2 )           return 1 ;
    else           return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```



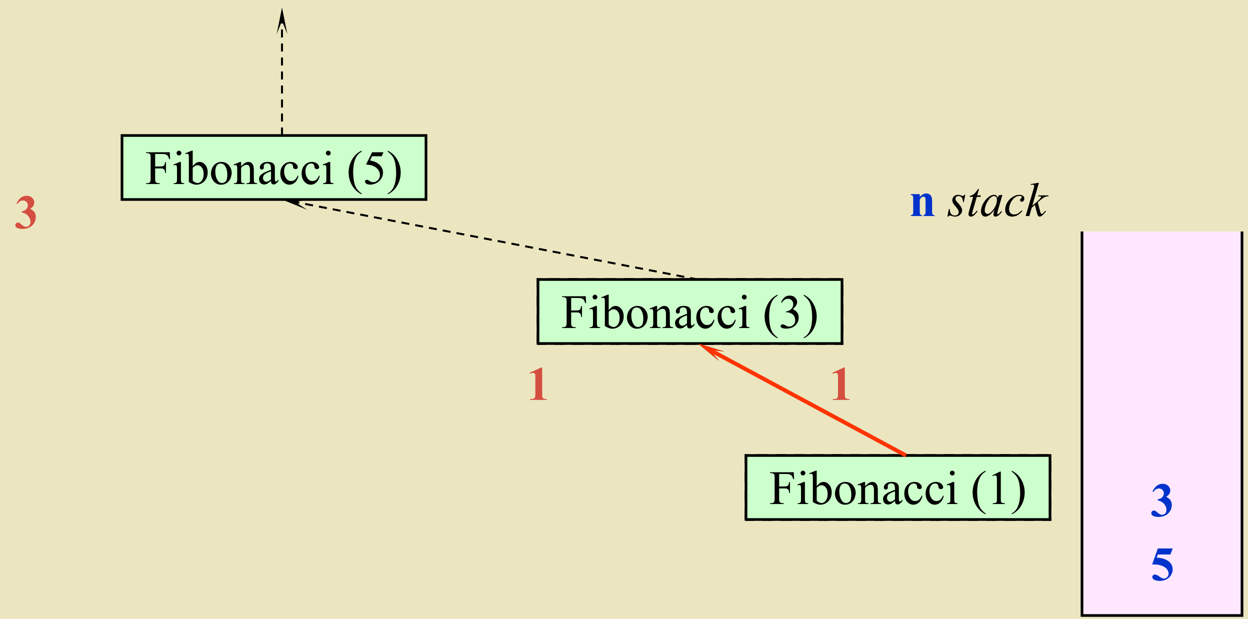
```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else         return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```



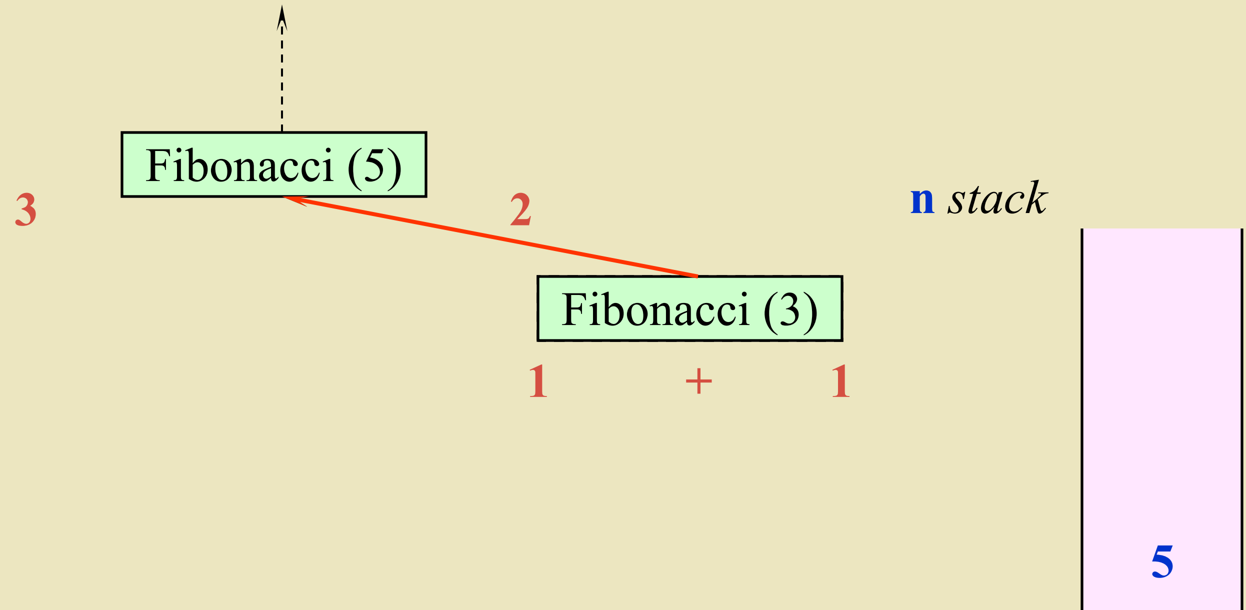
```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```



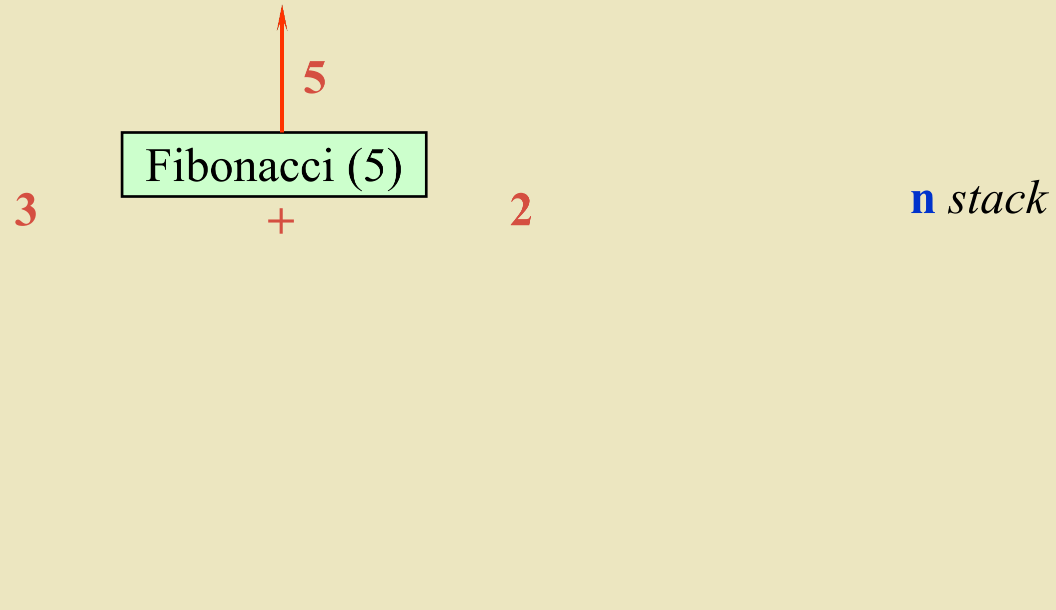
```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```



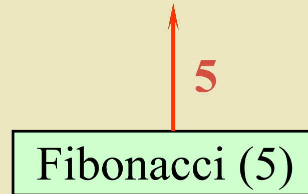
```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else         return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

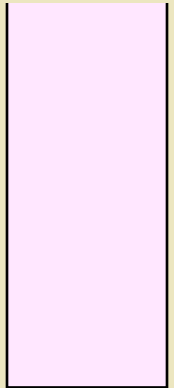


```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```



```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```





```
int  Fibonacci ( int  n )
  { if ( n <= 2 )          return 1 ;
    else          return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
  }
```

5

Fibonacci (5)

3 2

Fibonacci (4) Fibonacci (3)

2 1 1 1

Fibonacci (3) Fibonacci (2) Fibonacci (2) Fibonacci (1)

1 1

Fibonacci (2) Fibonacci (1)

```
int  Fibonacci ( int  n )
 { if ( n <= 2 )          return 1 ;
   else         return Fibonacci ( n-1 ) + Fibonacci ( n-2 ) ;
 }
```

5

Fibonacci (5)

3　　2

Fibonacci (4)　　***Fibonacci (3)***

2　　1　　1　　1

Fibonacci (3)　　***Fibonacci (2)***　　***Fibonacci (2)***　　***Fibonacci (1)***

1　　1

Fibonacci (2)　　***Fibonacci (1)***

```
int Fibonacci ( int n )
 { int i , f0=1, f1=1;
   for ( i = 2; i <= n/2 ; i ++ )
   { f0 = f0 + f1 ;
     f1 = f1 + f0 ;
   }
  if ( n%2 && n>2)
     f1=f0+f1 ;
  return f1;
 }
```

Fibonacci (5) 5

Fibonacci (4) 3

Fibonacci (3) 2

Fibonacci (2) 1

Fibonacci (1) 1

例3-17

```
// 输入一串字符，然后反序输出
#include<iostream>
using namespace std ;
void  reverse ()
{  char  ch ;          // 局部量
     cin >> ch;
     if ( ch != '.' )
          reverse() ;
     cout << ch ;
}
int main ()
 { cout << " Input a string : " << endl ;
    reverse() ;
    cout << endl ;
 }
```

输入数据进栈

例3-17

```
// 输入一串字符，然后反序输出
#include<iostream>
using namespace std ;
void  reverse ()
{  char  ch ;           // 局部量
    cin >> ch;
    if ( ch != '.' )
        reverse() ;
    cout << ch ;
}
int main ()
 { cout << " Input a string : " << endl ;
    reverse() ;
    cout << endl ;
 }
```

弹出堆栈数据输出

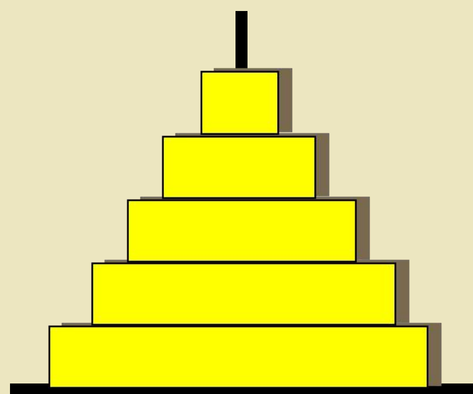
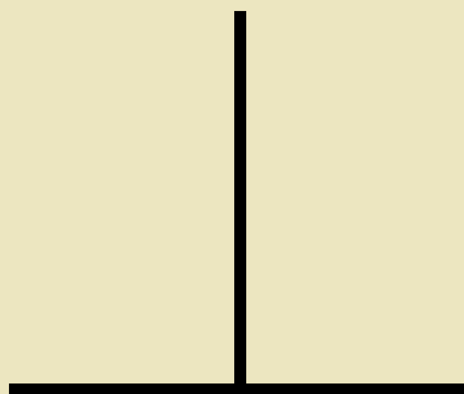
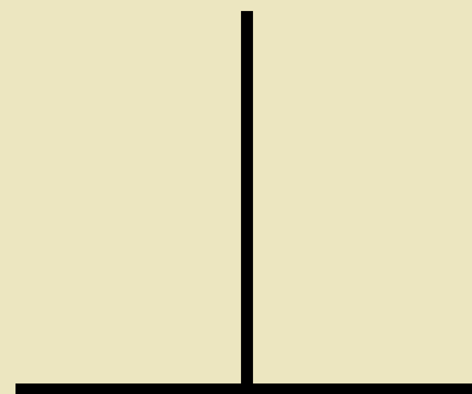
例3-18

```
// 反序输出正整数数字串
#include<iostream>
using namespace std ;
void  reverse ( int n )
  {  cout << n % 10 ;          // 输出最右一位数字
     if  ( n/10 != 0 )
          reverse ( n/10 );    // 求商，递归
  }
int main ()
  { int  k ;
    cout << "Input a integer n
    cin >> k ;
    reverse ( k ) ;
    cout << endl ;
}
```

可以直接取消递归
用循环代替吗？
请试一试

例3-19

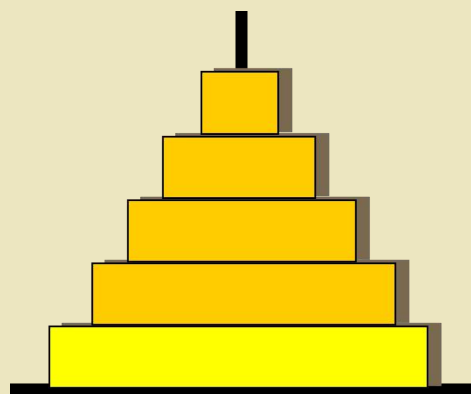
// 汉诺塔

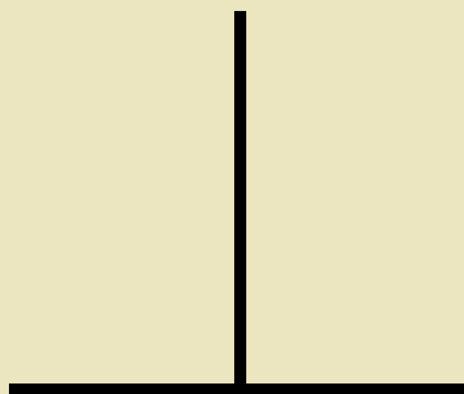




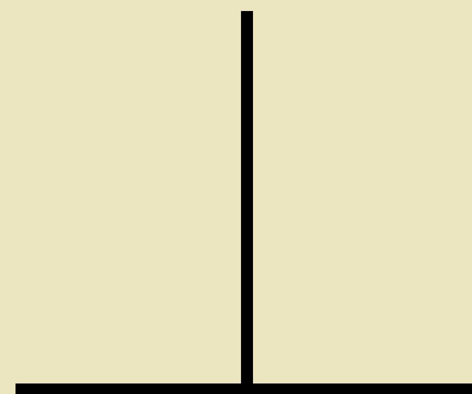
A　　　　B　　　　C

例3-19

// 汉诺塔

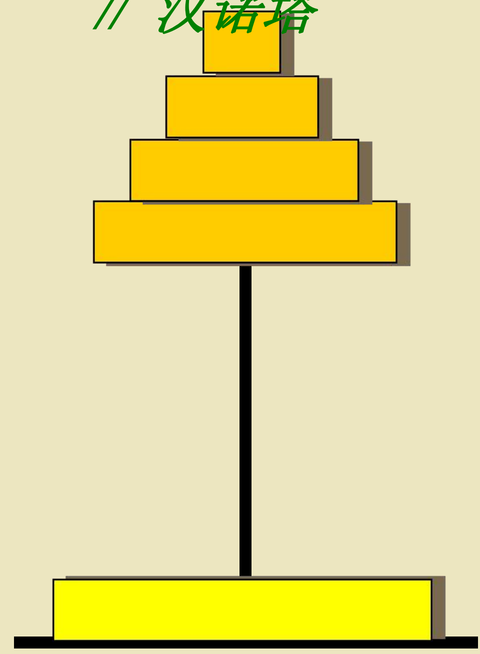


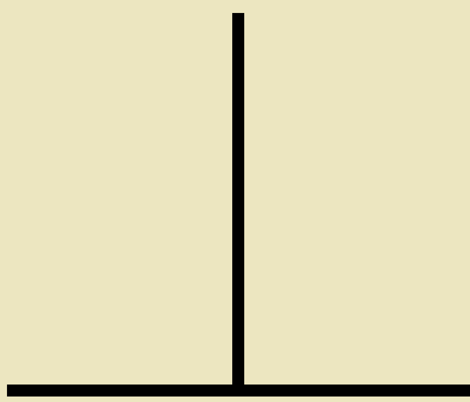


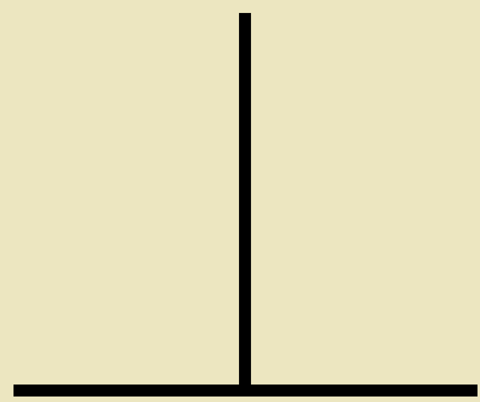


例3-19

// 汉诺塔







例3-19

// 汉诺塔

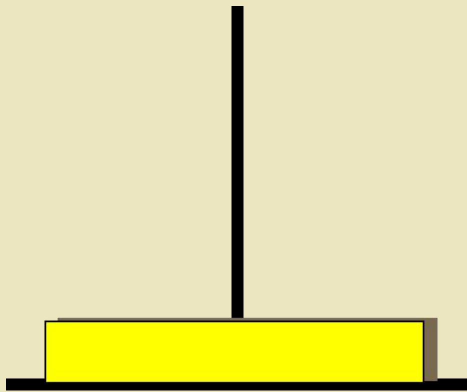
A

B

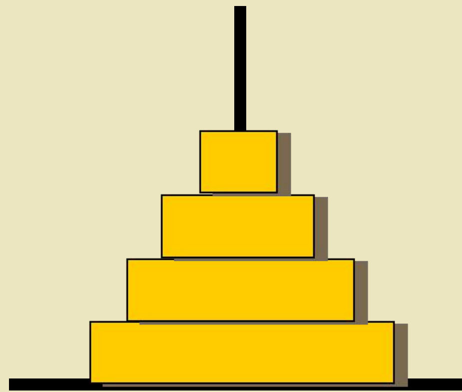
C

例3-19

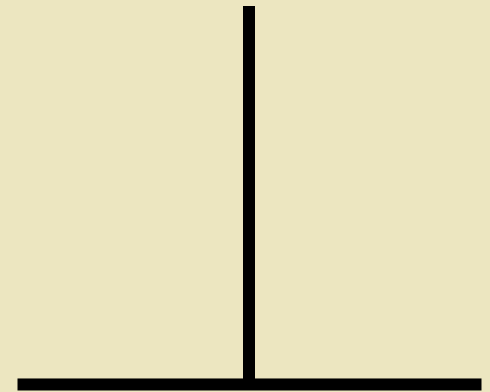
// 汉诺塔



A



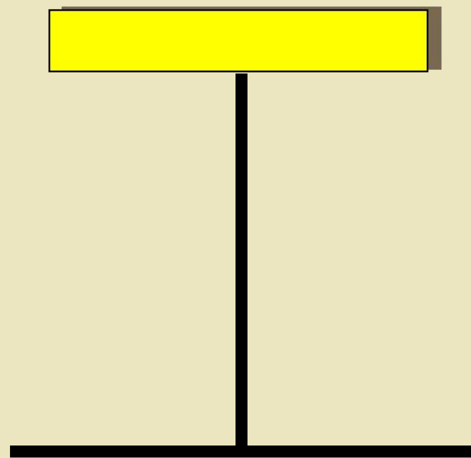
B

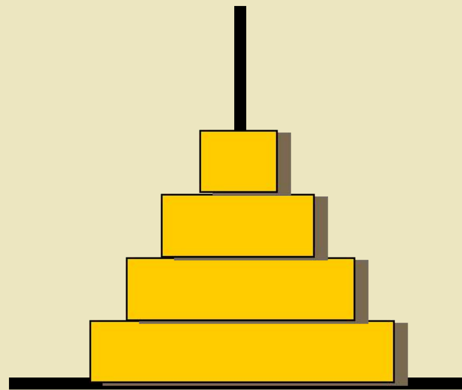


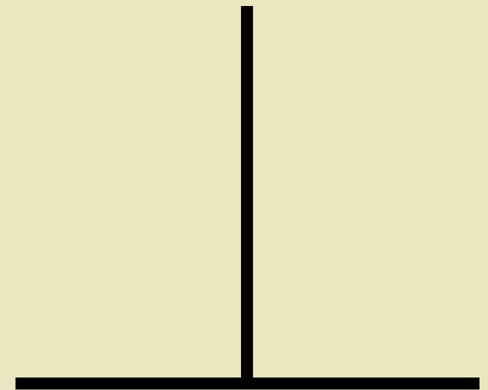
C

例3-19

// 汉诺塔

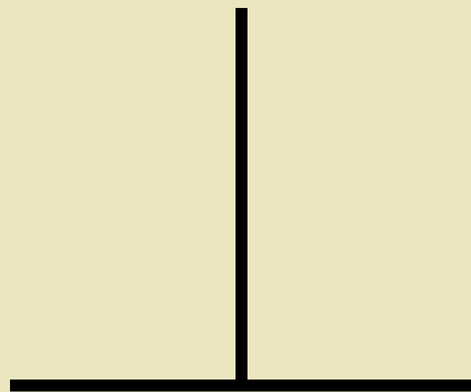


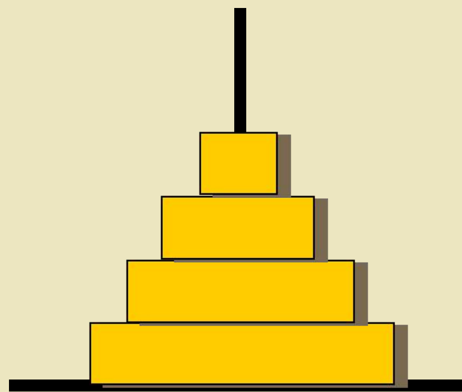


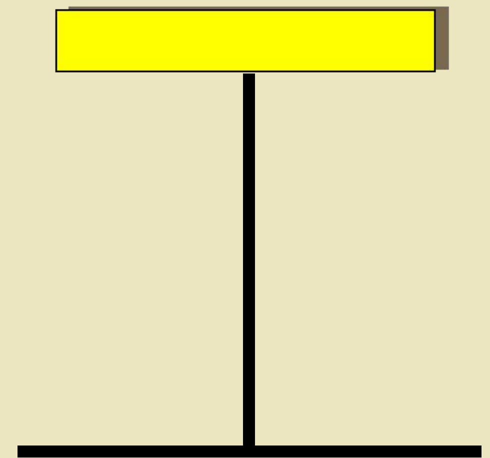


例3-19

// 汉诺塔



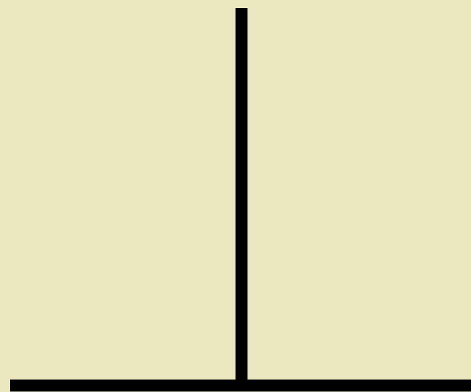


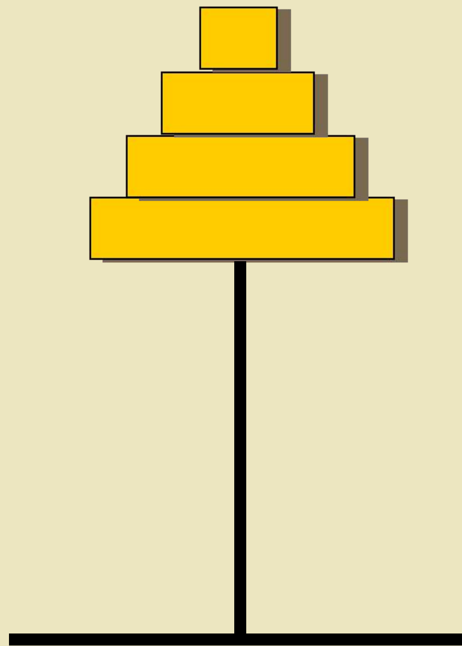


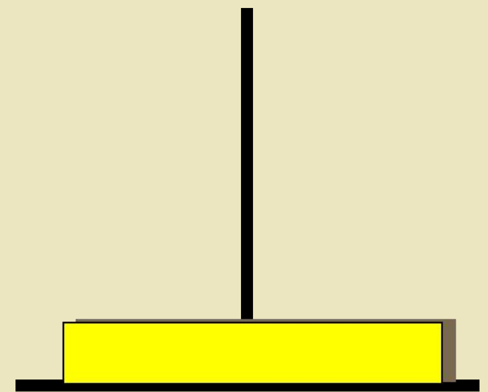
A　　B　　C

例3-19

// 汉诺塔



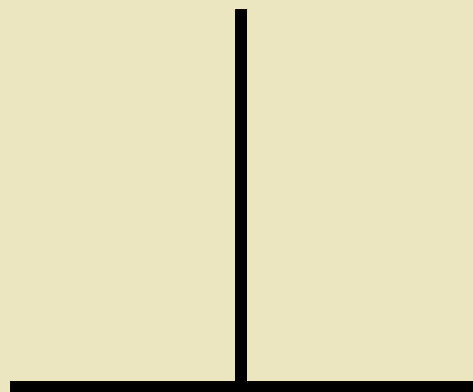


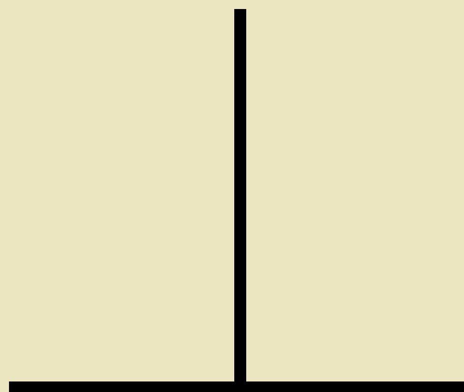


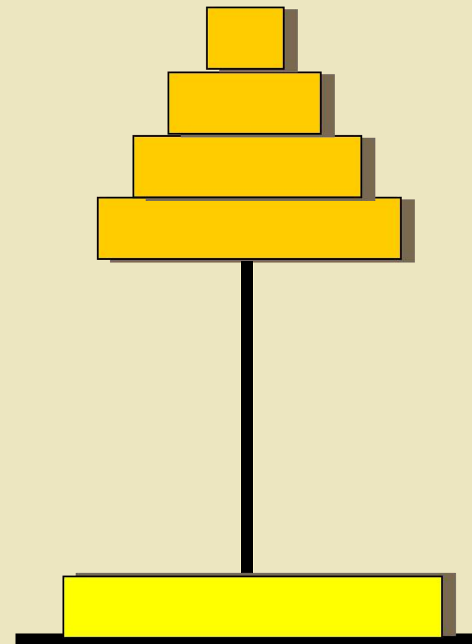
A　　B　　C

例3-19

// 汉诺塔



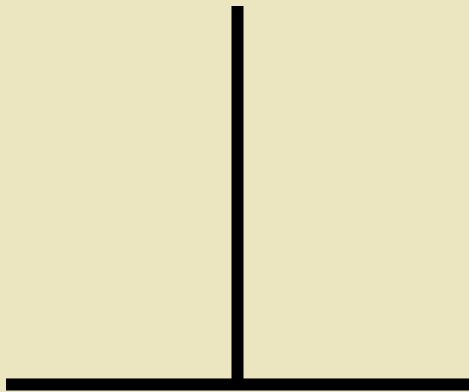


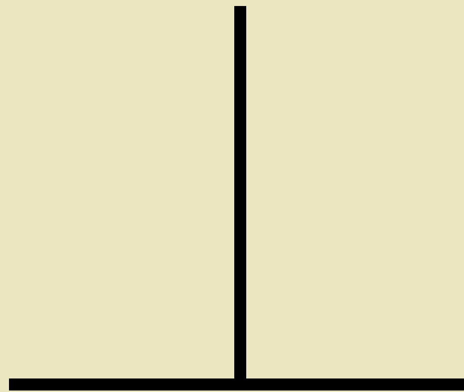


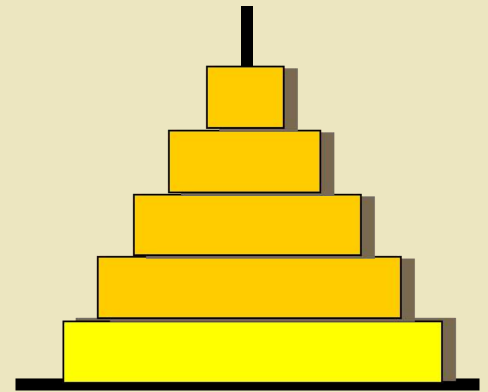
A　　B　　C

例3-19

// 汉诺塔

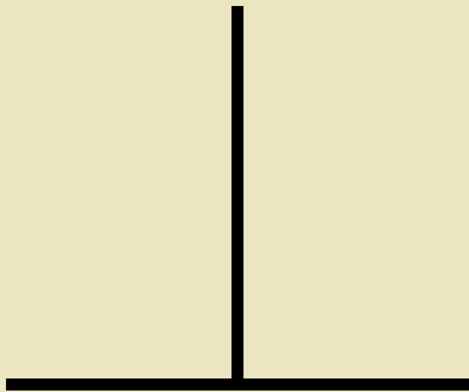
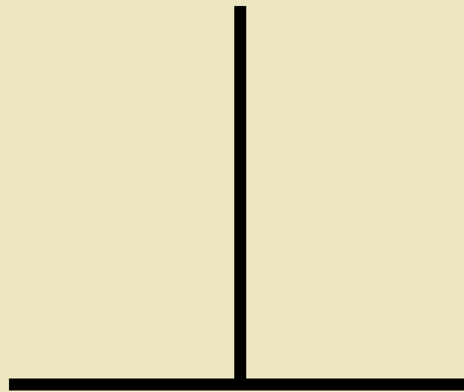
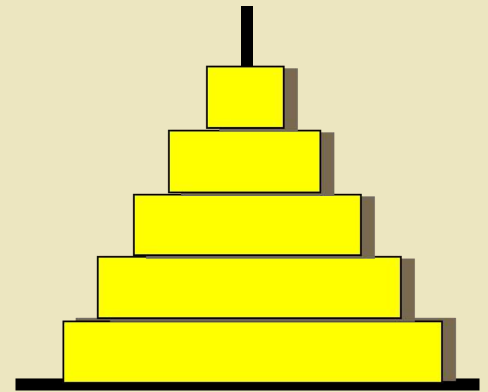






例3-19

// 汉诺塔

A B C

例3-19

```
// 汉诺塔
#include<iostream>
using namespace std ;
void hanoi ( int n, char a,  char b,  char c )
  {  if  ( n >= 1 )
       {  hanoi ( n-1,  a,  c,  b ) ;
          cout << a << " --> " << c << endl ;
          hanoi ( n-1,  b,  a,  c ) ;
       }
  }
int main ()
  {  int  m ;
     cout << " Input the number of diskes: " << endl ;
     cin >> m ;
     hanoi ( m,  'A' ,  'B' , 'C' ) ;
  }
```

```
// 汉诺塔
int main ()
  {  int  m ;
     cout << " Input the number of diskes: " << endl ;
     cin >> m ;
     hanoi ( m,  'A' ,  'B' , 'C' ) ;
  }
```

Stack

n a b c

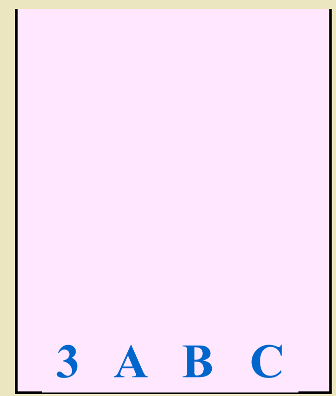
Output

// 汉诺塔

```
int main ()
  { int  m ;
    cout << " Input the number of diskes: " << endl ;
    cin >> m ;
    hanoi ( m,  'A' , 'B' , 'C' ) ;
  }
```

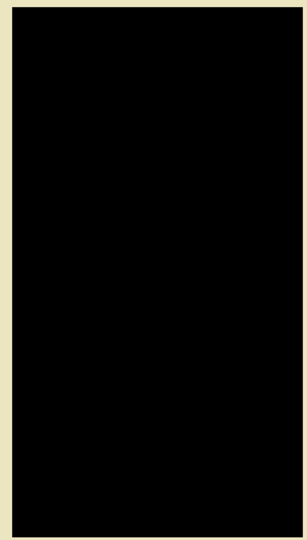
H(n,A,B,C) **H (3, A, B, C)**

Stack



n a b c

Output

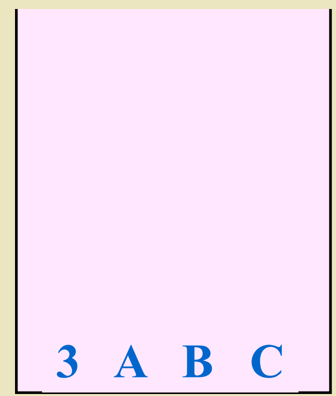


// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```

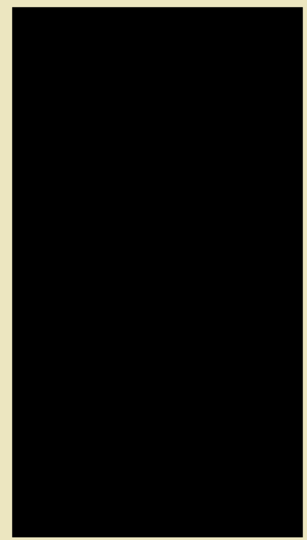
H(n,A,B,C) **H (3, A, B, C)**

Stack



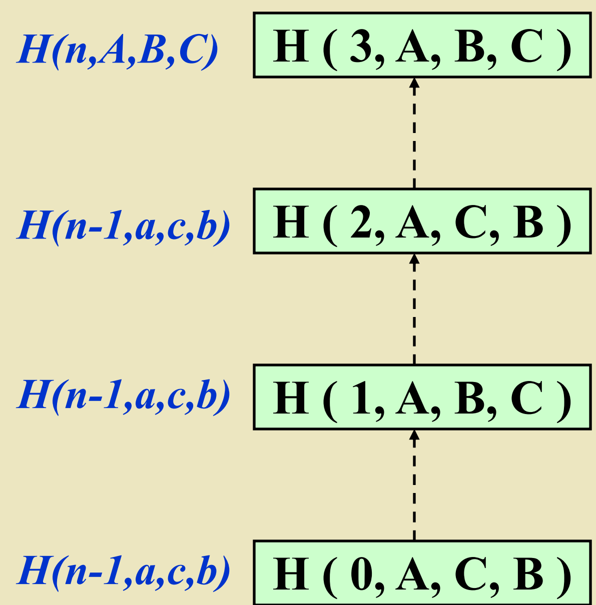
n a b c

Output



// 汉诺塔

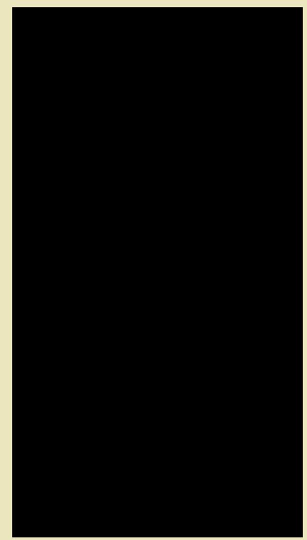
```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
        cout<<a<<" --> "<<c<<endl;
        hanoi (n-1, b, a, c) ;
     }
 }
```



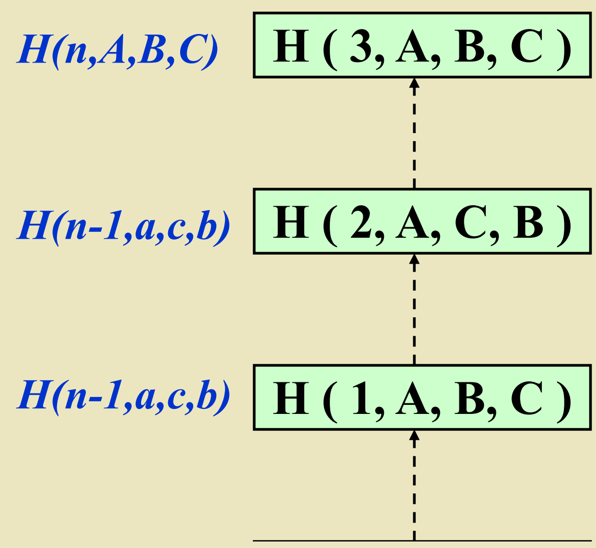
Stack

0	A	C	B
1	A	B	C
2	A	C	B
3	A	B	C
n	a	b	c

Output



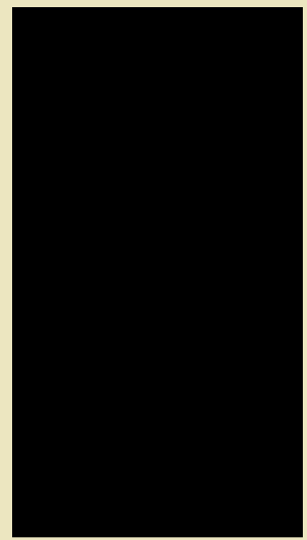
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
 }
```



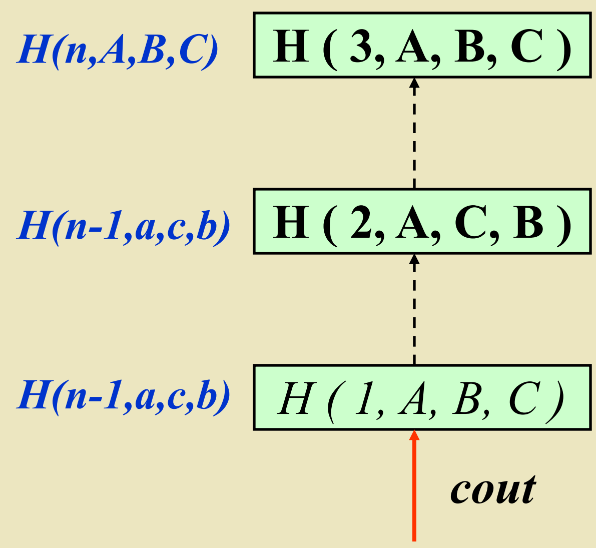
Stack

n	a	b	c
1	A	B	C
2	A	C	B
3	A	B	C

Output



```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
 }
```

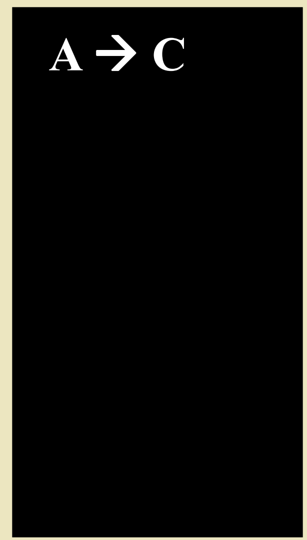


Stack

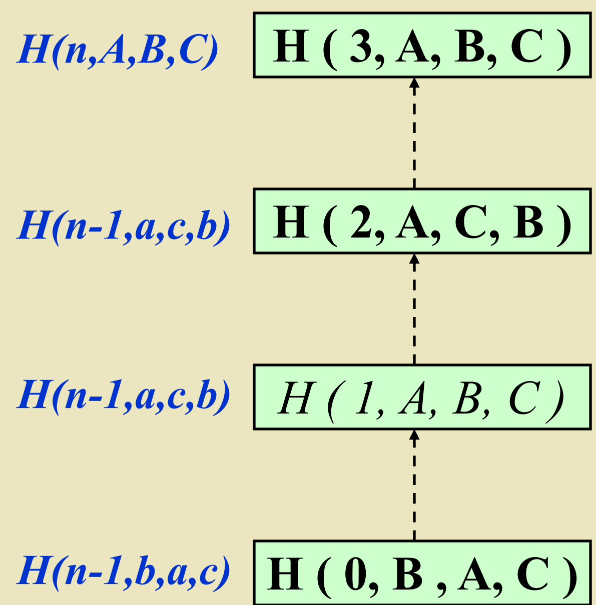
1 A B C
2 A C B
3 A B C

n a b c

Output



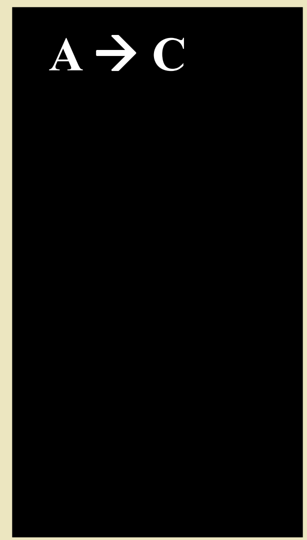
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



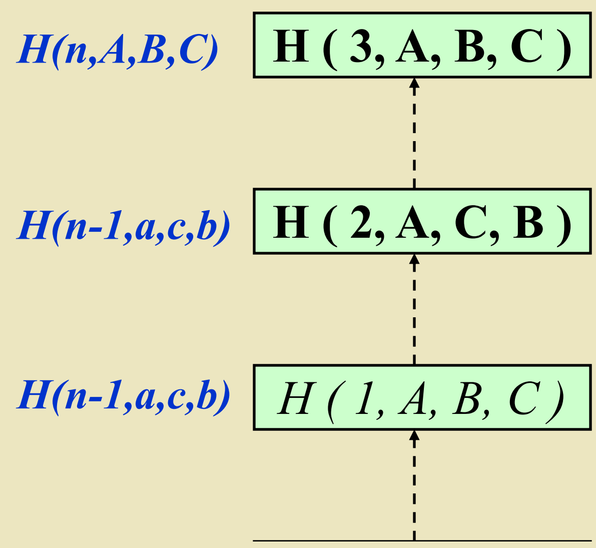
Stack

0	B	A	C
1	A	B	C
2	A	C	B
3	A	B	C
n	a	b	c

Output



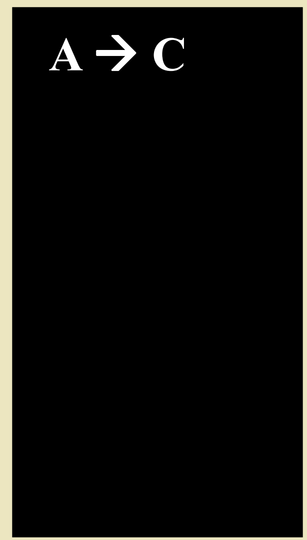
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



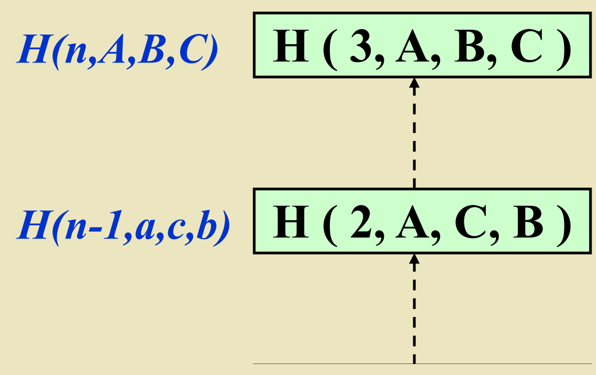
Stack

n	a	b	c
1	A	B	C
2	A	C	B
3	A	B	C

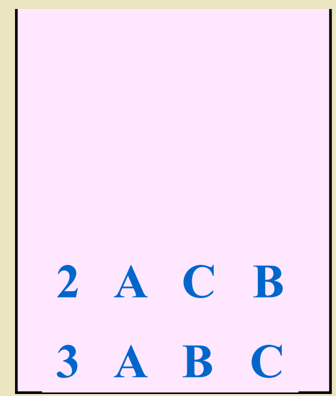
Output



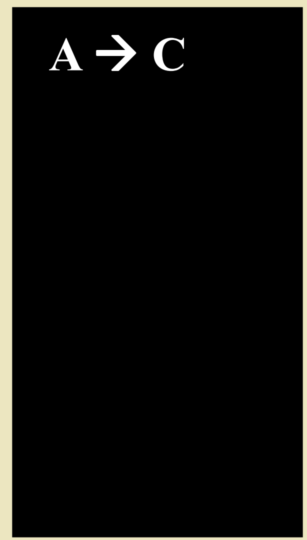
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
}
```



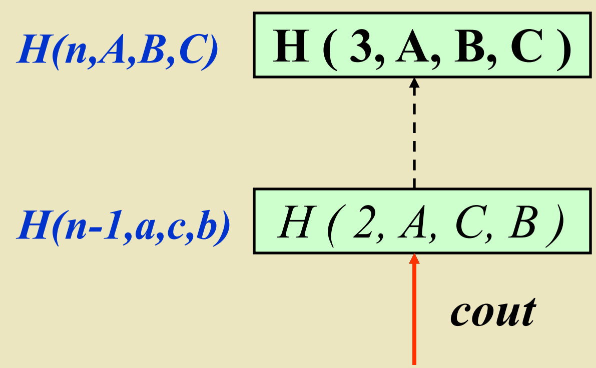
Stack



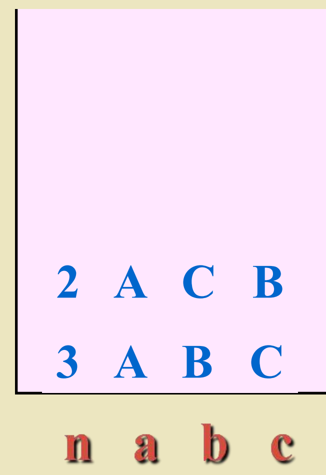
Output



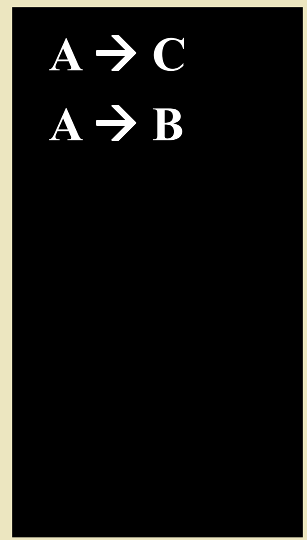
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



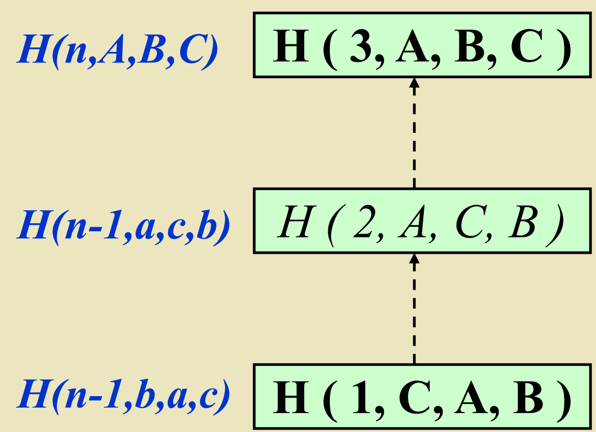
Stack



Output



```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



Stack

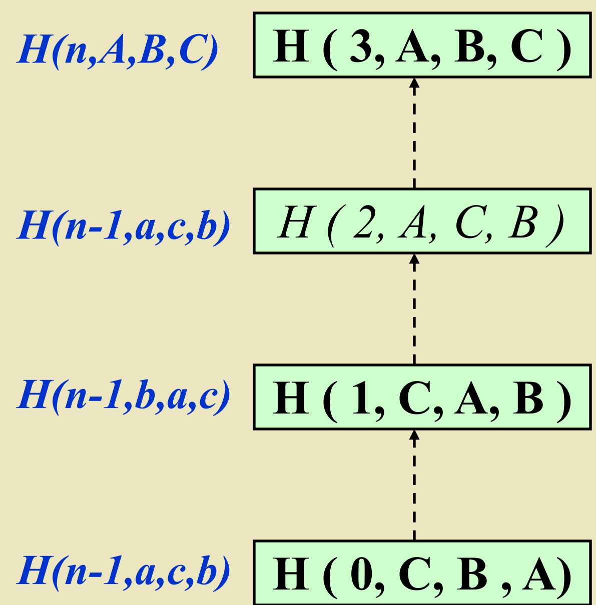
n	a	b	c
1	C	A	B
2	A	C	B
3	A	B	C

Output

A → C
A → B

// 汉诺塔

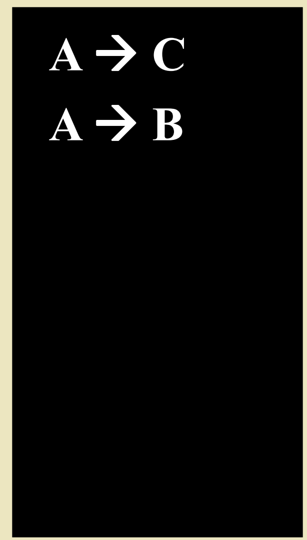
```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
        cout<<a<<" --> "<<c<<endl;
        hanoi (n-1, b, a, c) ;
     }
 }
```



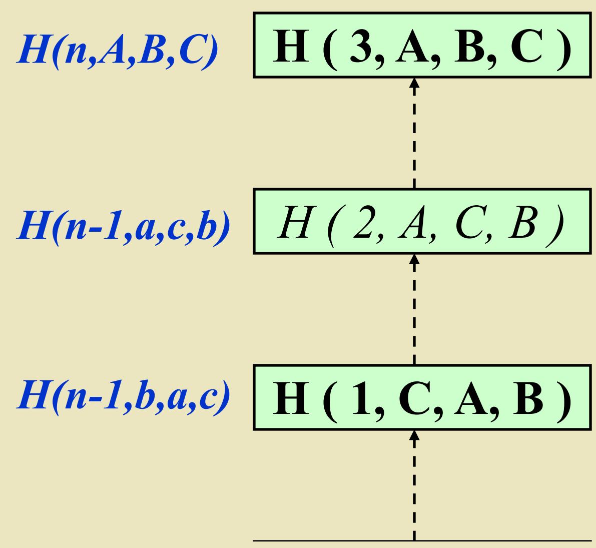
Stack

n	a	b	c
0	C	B	A
1	C	A	B
2	A	C	B
3	A	B	C

Output



```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



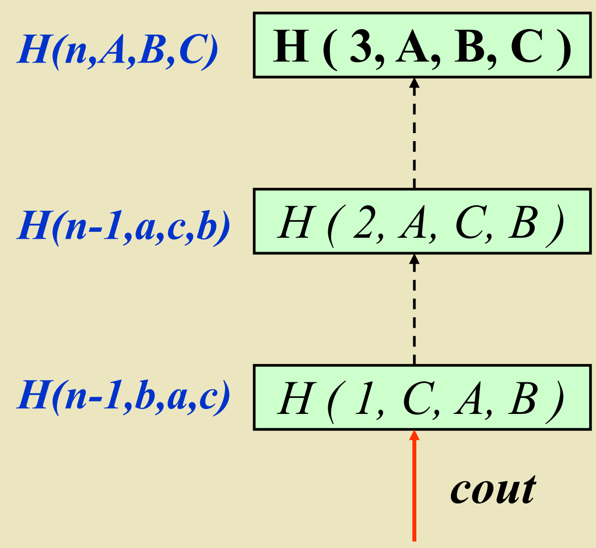
Stack

n	a	b	c
1	C	A	B
2	A	C	B
3	A	B	C

Output

A → C
A → B

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



Stack

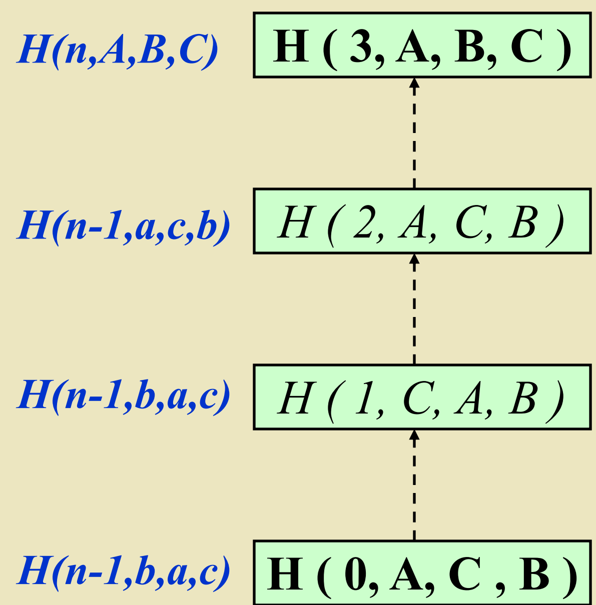
n	a	b	c
1	C	A	B
2	A	C	B
3	A	B	C

Output

A → C
A → B
C → B

// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



Stack

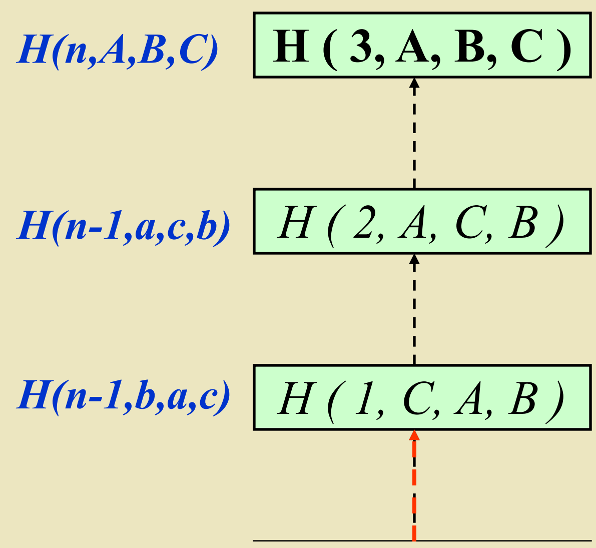
n	a	b	c
0	A	C	B
1	C	A	B
2	A	C	B
3	A	B	C

Output

A → C
A → B
C → B

// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



Stack

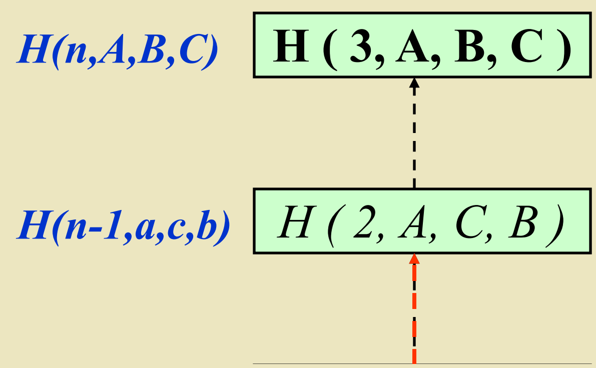
1 C A B
2 A C B
3 A B C

n a b c

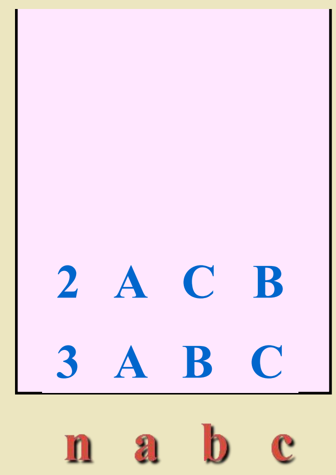
Output

A → C
A → B
C → B

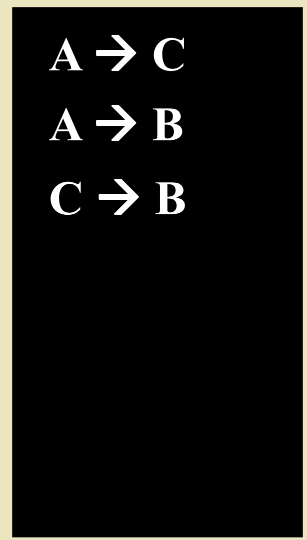
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



Stack

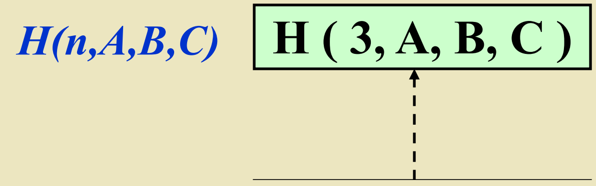


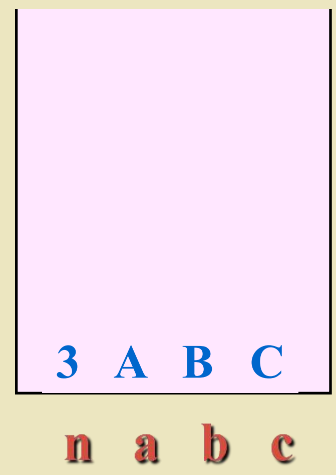
Output

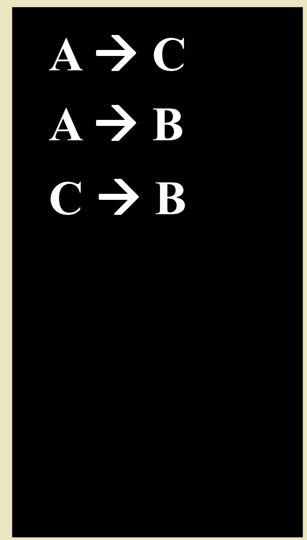


// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
}
```

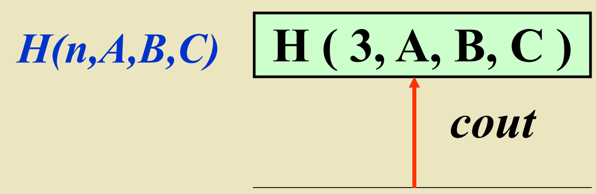




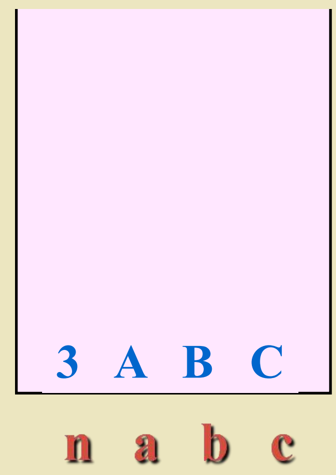


```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```

H(n,A,B,C)

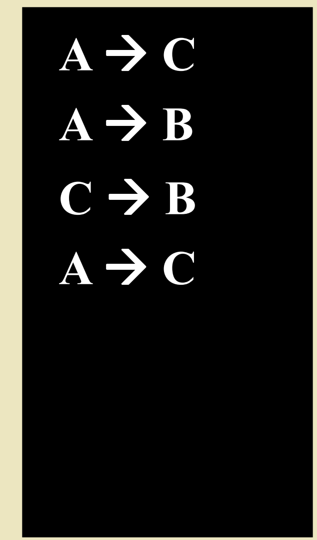


Stack

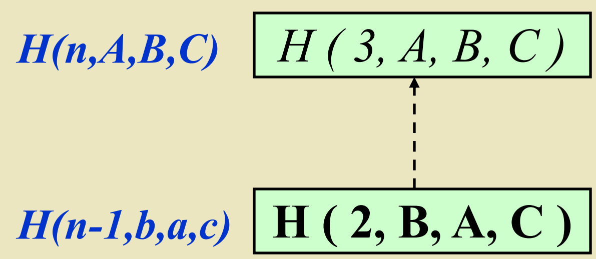


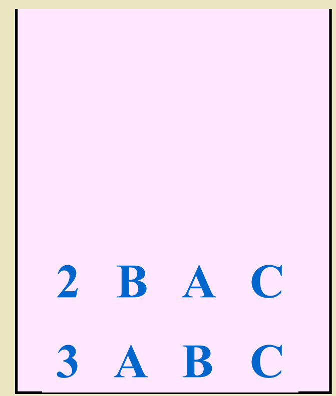
n a b c

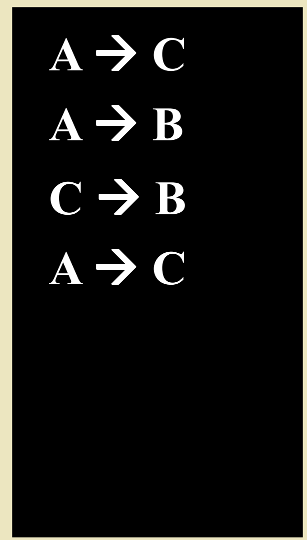
Output



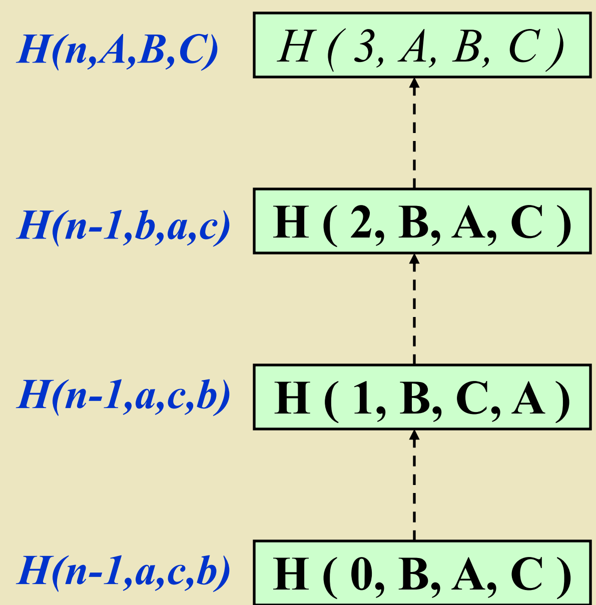
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```







```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
        cout<<a<<" --> "<<c<<endl;
        hanoi (n-1, b, a, c) ;
     }
 }
```



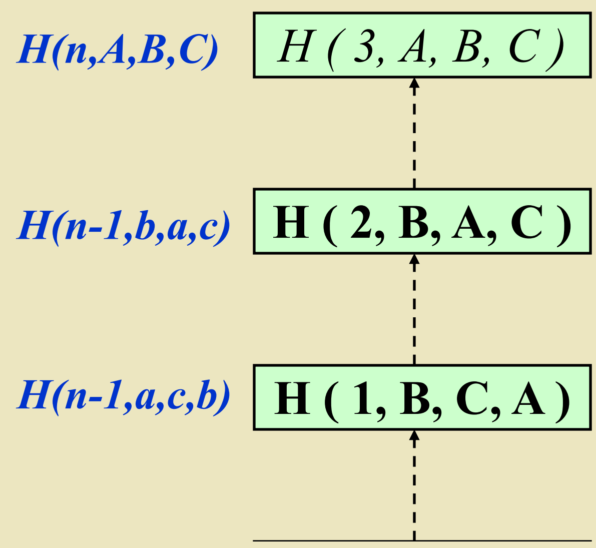
Stack

n	a	b	c
0	B	A	C
1	B	C	A
2	B	A	C
3	A	B	C

Output

A → C
A → B
C → B
A → C

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
}
```



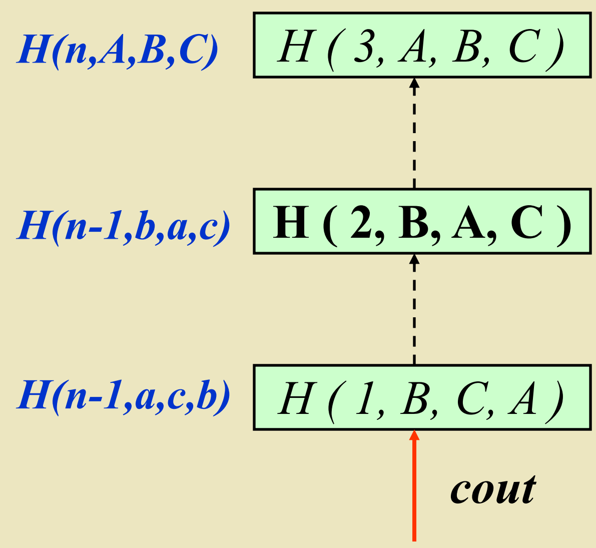
Stack

n	a	b	c
1	B	C	A
2	B	A	C
3	A	B	C

Output

A → C
A → B
C → B
A → C

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



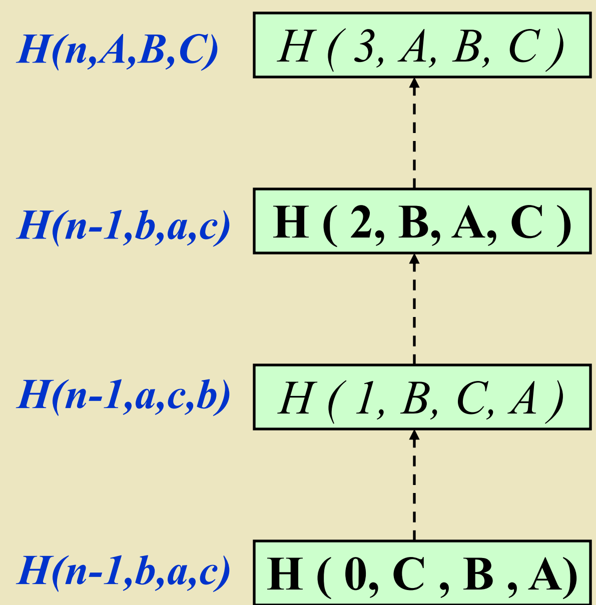
Stack

1	B	C	A
2	B	A	C
3	A	B	C
n	a	b	c

Output

A → C
A → B
C → B
A → C
B → A

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



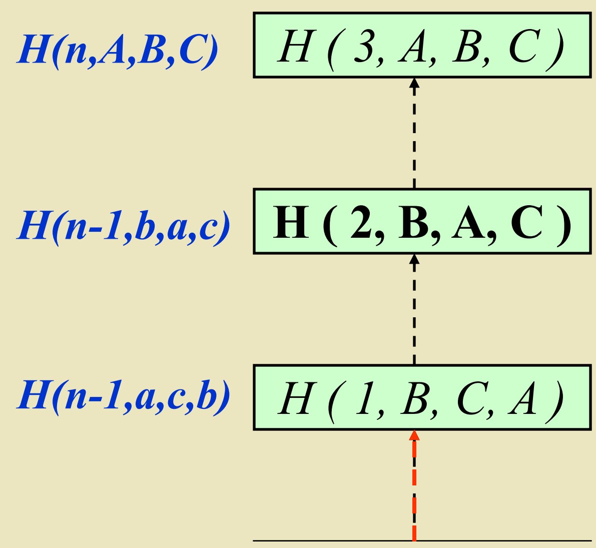
Stack

0	C	B	A
1	B	C	A
2	B	A	C
3	A	B	C
n	a	b	c

Output

A → C
A → B
C → B
A → C
B → A

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
 }
```



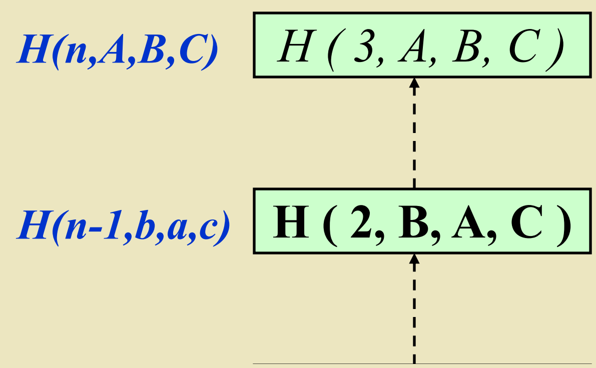
Stack

n	a	b	c
1	B	C	A
2	B	A	C
3	A	B	C

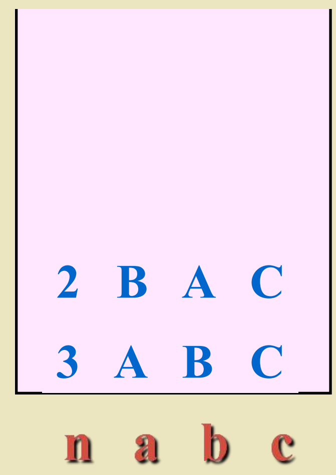
Output

A → C
A → B
C → B
A → C
B → A

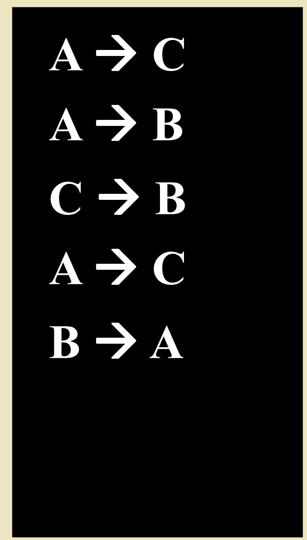
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



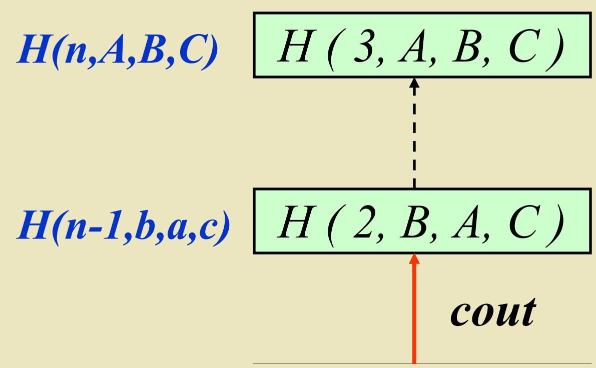
Stack



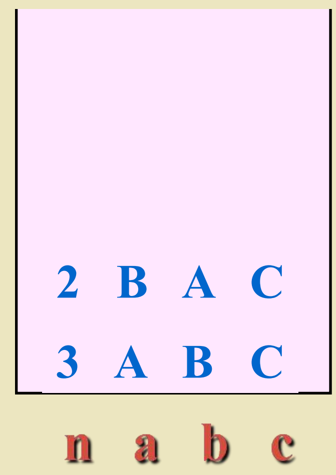
Output



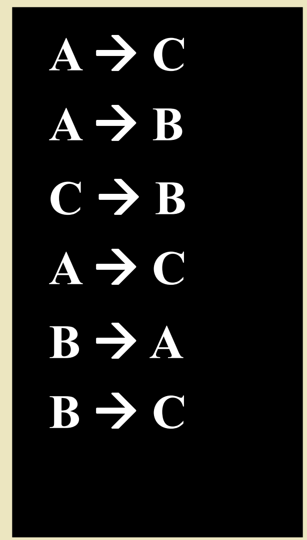
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



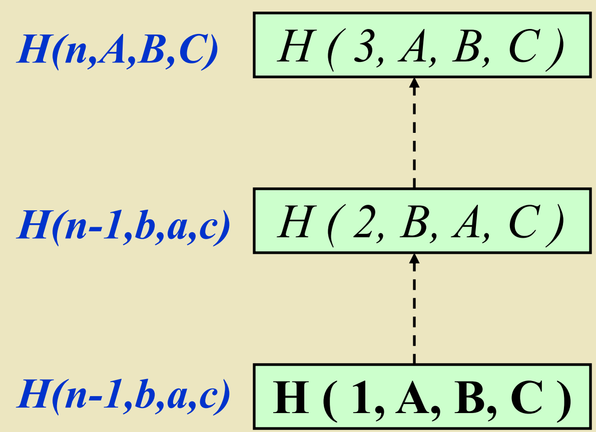
Stack



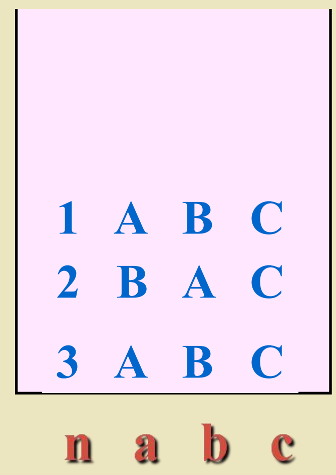
Output



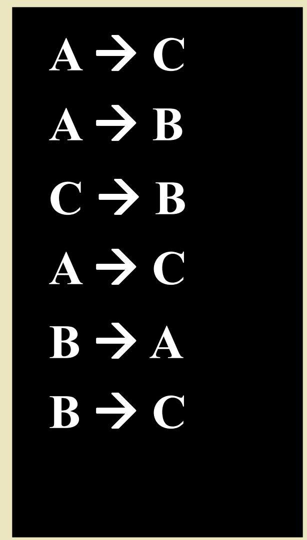
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



Stack

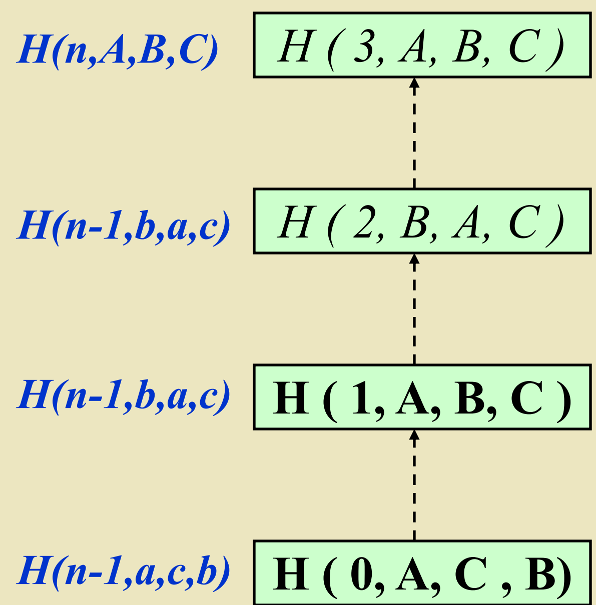


Output



// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
      { hanoi ( n-1, a, c, b) ;
         cout<<a<<" --> "<<c<<endl;
         hanoi (n-1, b, a, c) ;
      }
 }
```



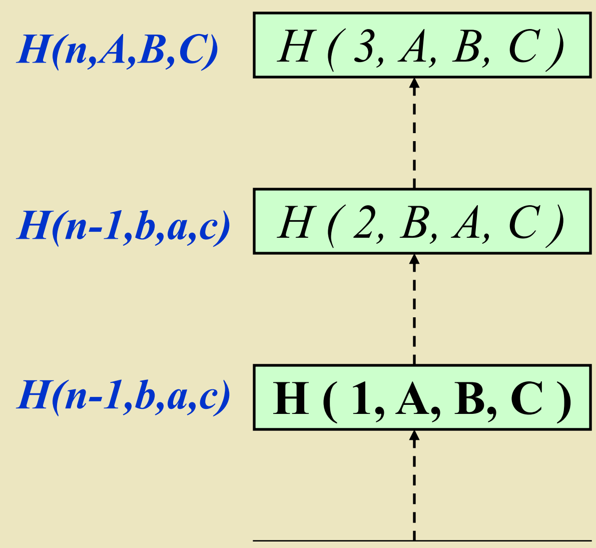
Stack

0	A	C	B
1	A	B	C
2	B	A	C
3	A	B	C
n	a	b	c

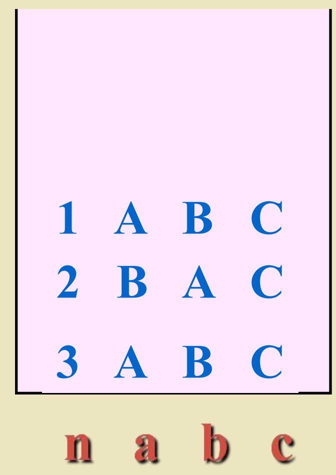
Output

A → C
A → B
C → B
A → C
B → A
B → C

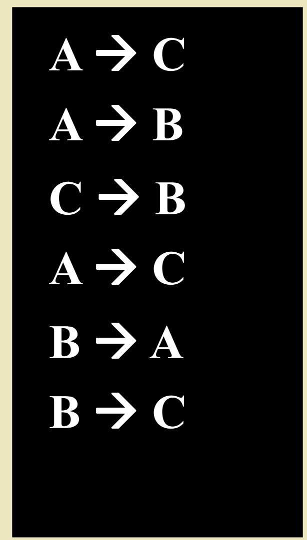
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
        cout<<a<<" --> "<<c<<endl;
        hanoi (n-1, b, a, c) ;
     }
}
```



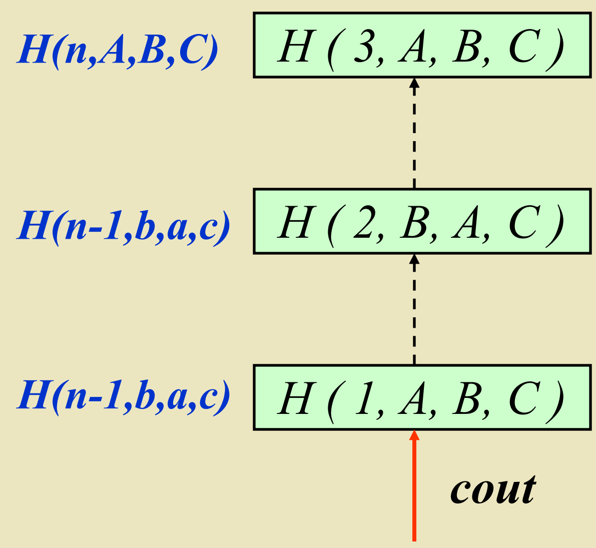
Stack



Output



```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



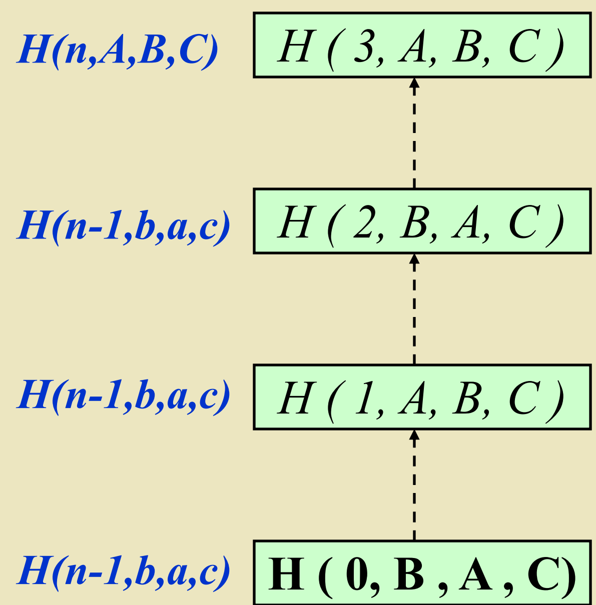
Stack

n	a	b	c
1	A	B	C
2	B	A	C
3	A	B	C

Output

A → C
A → B
C → B
A → C
B → A
B → C
A → C

```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
 }
```



Stack

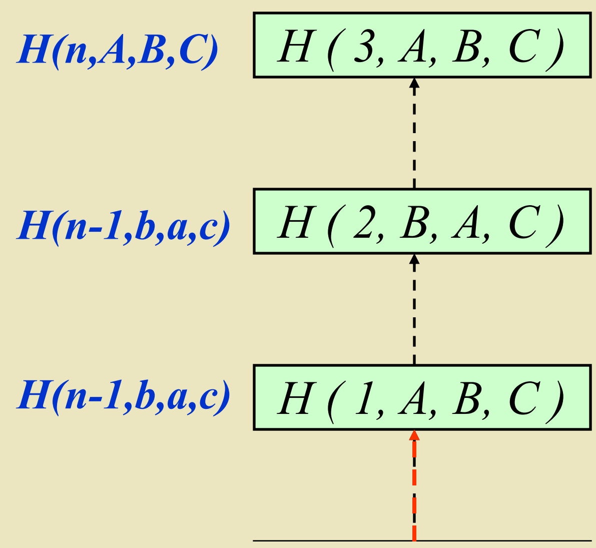
0	B	A	C
1	A	B	C
2	B	A	C
3	A	B	C
n	a	b	c

Output

A → C
A → B
C → B
A → C
B → A
B → C
A → C

// 汉诺塔

```
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
}
```



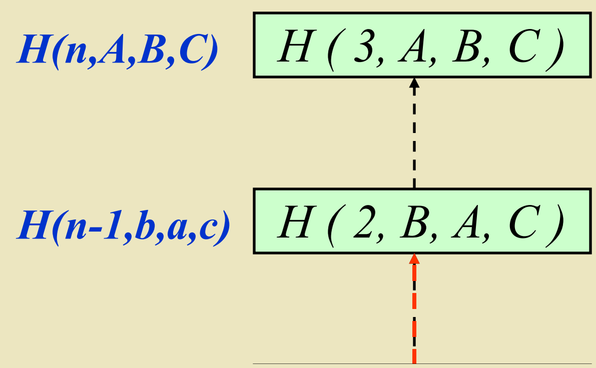
Stack

1	A	B	C
2	B	A	C
3	A	B	C
n	a	b	c

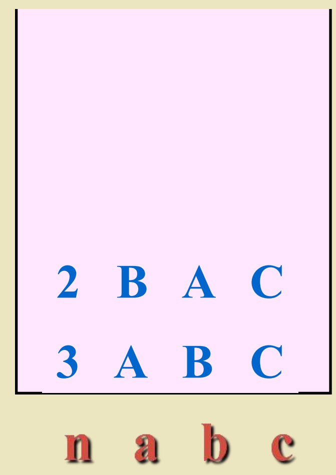
Output

A → C
A → B
C → B
A → C
B → A
B → C
A → C

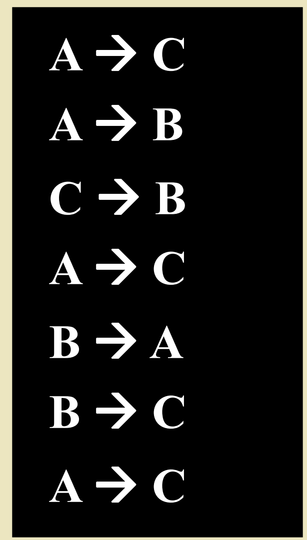
```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```



Stack

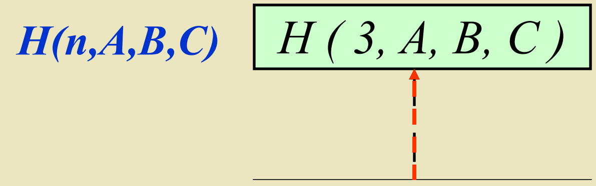


Output

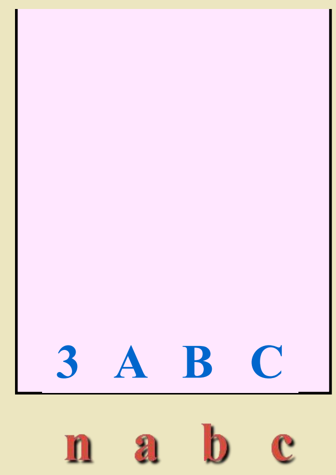


```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
     { hanoi ( n-1, a, c, b) ;
       cout<<a<<" --> "<<c<<endl;
       hanoi (n-1, b, a, c) ;
     }
}
```

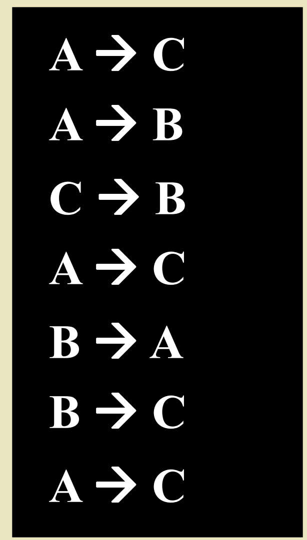
H(n,A,B,C)



Stack



Output



```
// 汉诺塔
void hanoi ( int n, char a,  char b,  char c )
 {  if  ( n >= 1 )
    { hanoi ( n-1, a, c, b) ;
      cout<<a<<" --> "<<c<<endl;
      hanoi (n-1, b, a, c) ;
    }
}
```

Over

Stack

n a b c

Output

A → C
A → B
C → B
A → C
B → A
B → C
A → C

```
// 汉诺塔
int main ()
 {  int  m ;
    cout << " Input the number of diskes: " << endl ;
    cin >> m ;
    hanoi ( m,  'A' ,  'B' , 'C' ) ;
 }
```

Output

```
A → C
A → B
C → B
A → C
B → A
B → C
A → C
```

3.4　函数指针

- 函数、应用程序是编译器处理的对象
- 每一个函数模块都有一个首地址，称为函数的入口地址，
  （函数指针）
- 函数调用：找到函数入口地址；传递参数
- 不带括号的函数名就是函数入口地址

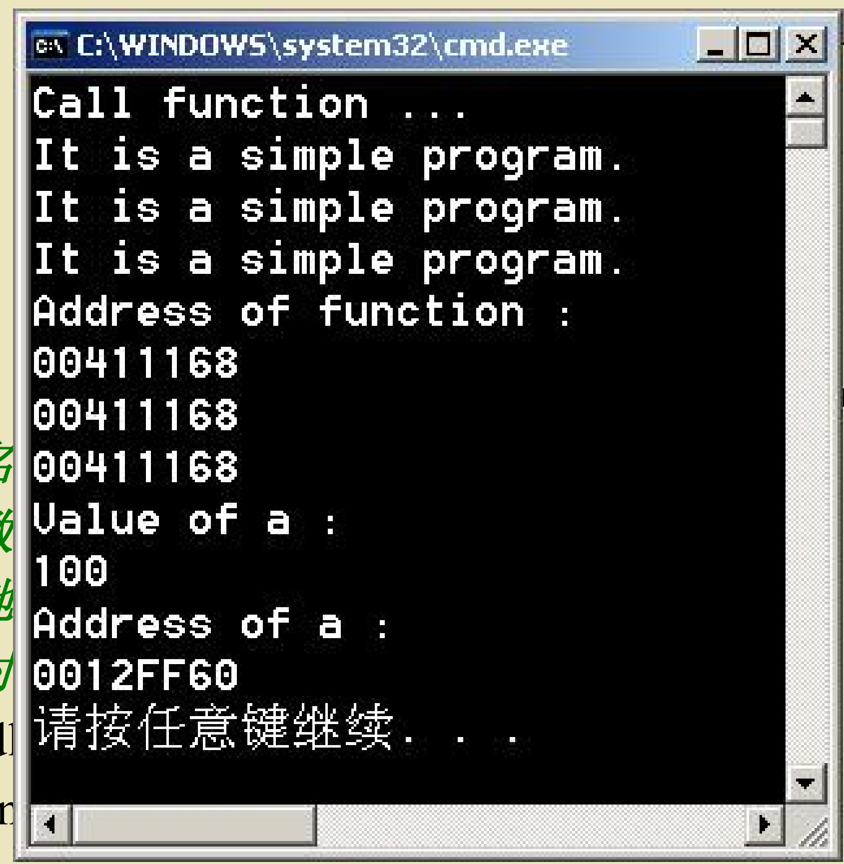
3.4.1 函数的地址

```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                          // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
   simple() ;                          // 名方式调用
   ( & simple )() ;                    // 地址方式调用
   ( * & simple )() ;                  // 间址调用
   cout << "Address of function :\n" ;
   cout << simple << endl ;                      // 函数名是地址
   cout << & simple << endl ;                    // 取函数地址
   cout << *&simple << endl ;                    // 函数地址所指对象
   int a = 100 ;                       // 声明一个整数对象
   cout << "Value of a :\n" ;      cout << a << endl;
   cout << "Address of a :\n" ;    cout << &a << endl;
}
```

```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
   simple() ;                    // 名方式调用
   ( & simple )() ;              // 地址方式调用
   ( * & simple )() ;            // 间址调用
   cout << "Address of function :\n" ;
   cout << simple << endl ;              // 函数名是地址
   cout << & simple << endl ;            // 取函数地址
   cout << *&simple << endl ;            // 函数地址所指对象
   int a = 100 ;                 // 声明一个整数对象
   cout << "Value of a :\n" ;      cout << a << endl;
   cout << "Address of a :\n" ;    cout << &a << endl;
}
```

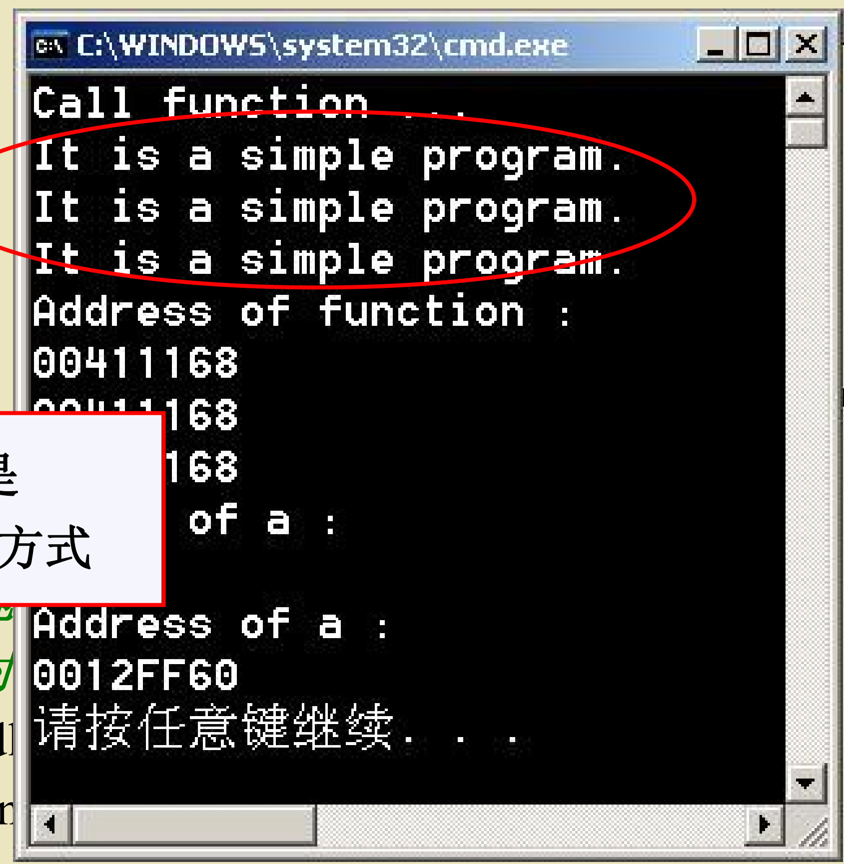
一个无参数的简单函数

```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
   simple() ;                    // 名方式调用
   ( & simple )() ;              // 地址方式调用
   ( * & simple )() ;            // 间址调用
   cout << "Address of function :\n" ;
   cout << simple << endl ;                 // 函数名
   cout << & simple << endl ;               // 取函数
   cout << *&simple << endl ;               // 函数地
   int a = 100 ;                 // 声明一个整数对
   cout << "Value of a :\n" ;       cout << a << endl
   cout << "Address of a :\n" ;     cout << &a << en
}
```



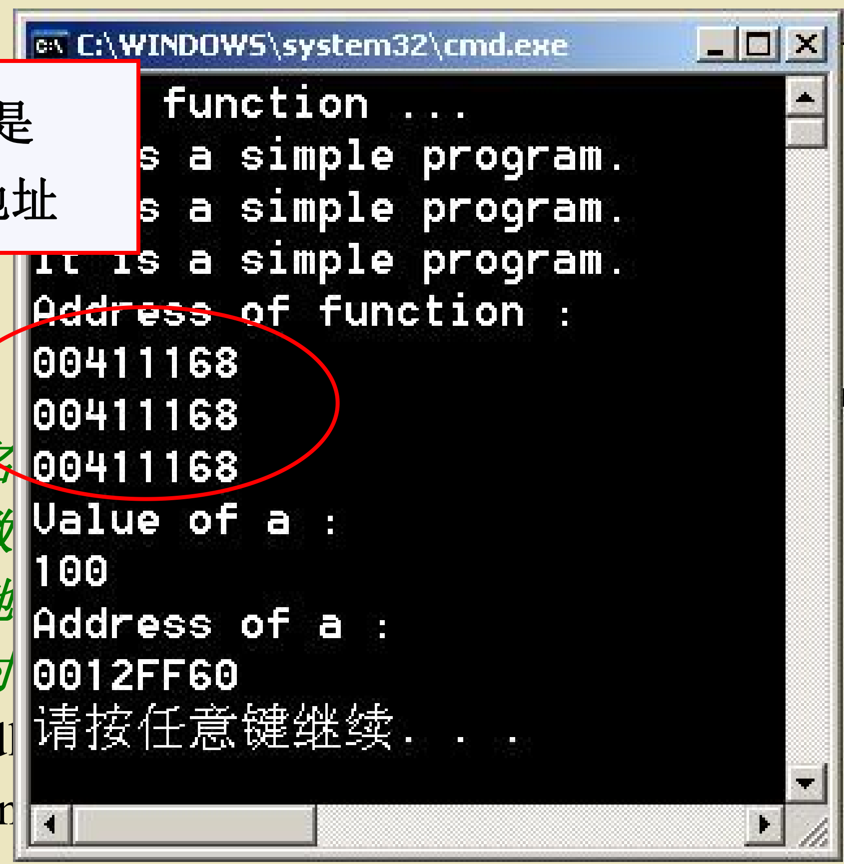
```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
  simple() ;                     // 名方式调用
  ( & simple )() ;               // 地址方式调用
  ( * & simple )() ;             // 间址调用
  cout << "Address of function :\n" ;
  cout << simple << endl ;
  cout << & simple << endl ;
  cout << *&simple << endl ;           // 函数地
  int a = 100 ;                  // 声明一个整数对
  cout << "Value of a :\n" ;     cout << a << endl
  cout << "Address of a :\n" ;   cout << &a << en
}
```

它们都是
正确的调用方式



```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
   simple() ;                    // 名
   ( & simple )() ;              // 地址方式调用
   ( * & simple )() ;            // 间址调用
   cout << "Address of function :\n" ;
   cout << simple << endl ;            // 函数名
   cout << & simple << endl ;          // 取函数
   cout << *&simple << endl ;          // 函数地
   int a = 100 ;                 // 声明一个整数对
   cout << "Value of a :\n" ;       cout << a << endl
   cout << "Address of a :\n" ;     cout << &a << en
}
```

它们都是
函数的地址

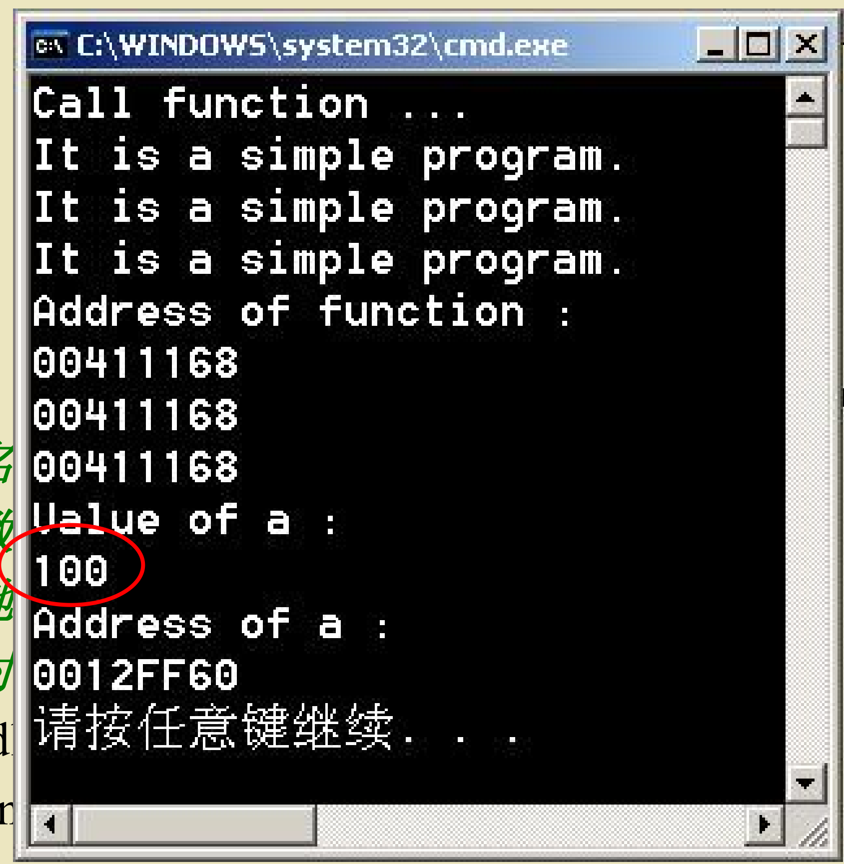


```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
   simple() ;                    // 名方式调用
   ( & simple )() ;              // 地址方式调用
                                 // 间址调用
                    ction :\n" ;
                                       // 函数名
                    ndl ;              // 取函数
   cout << *&simple << endl ;          // 函数地
   int a = 100 ;                 // 声明一个整数对
   cout << "Value of a :\n" ;      cout << a << endl
   cout << "Address of a :\n" ;    cout << &a << en
}
```

数据对象的值

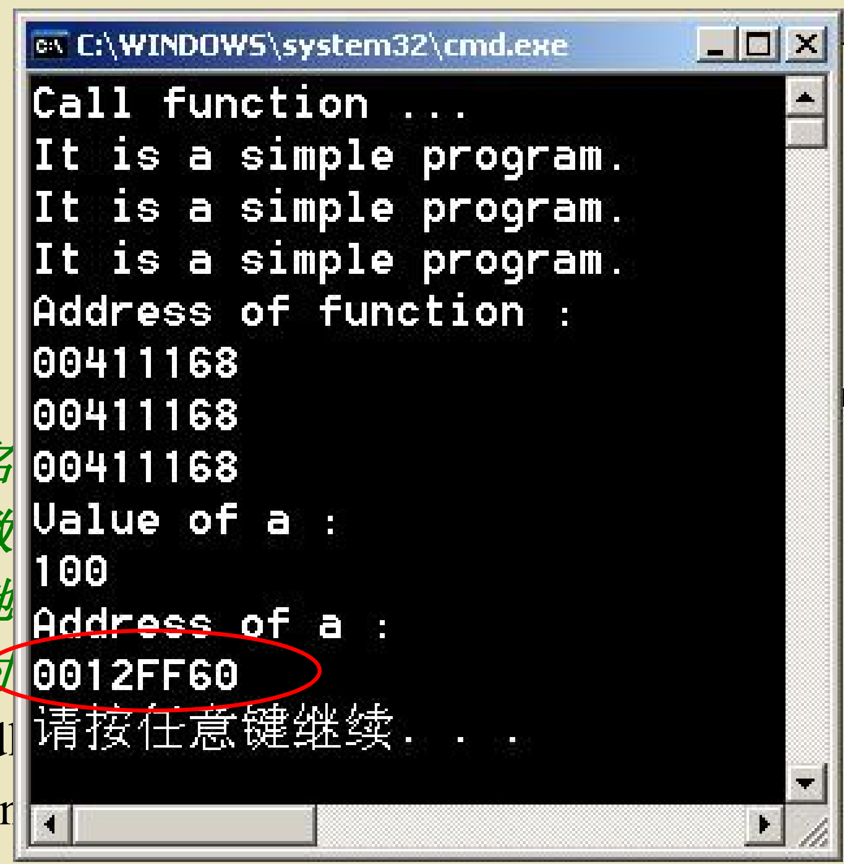
等价于 *&a

由类型 int 解释内存



```
// 例3-20  函数和数据的测试
#include<iostream>
using namespace std ;
void simple()                    // 定义一个简单函数
{ cout << "It is a simple program.\n" ; }
int main()
{ cout << "Call function ...\n" ;
  simple() ;                     // 名方式调用
  ( & simple )() ;               // 地址方式调用
  ( * & simple )() ;             // 间址调用
  cout <<                     " ;
  cout <<                                // 函数名
  cout << & simple << endl ;             // 取函数
  cout << *&simple << endl ;             // 函数地
  int a = 100 ;                  // 声明一个整数对
  cout << "Value of a :\n" ;       cout << a << endl
  cout << "Address of a :\n" ;     cout << &a << en
}
```

数据对象的地址



3.4.1 函数的地址

C++的函数与数据对象

	函数 FunctionObj		数据 DataObj
地址	FunctionObj *FunctionObj	&FunctionObj *&FunctionObj	&DataObj
对象	FunctionObj() *FunctionObj()	&FunctionObj() *&FunctionObj()	DataObj *&DataObj
访问	调用函数 *地址表达式* (*参数表*) 执行代码		读：右值表达式 写：左值表达式 由 *类型* 解释数据

3.4.2　函数指针

- 指向函数的指针变量简称为函数指针
- 函数的类型是函数的接口
- 可以通过指针变量的间址方式调用函数

3.4.2　函数指针

1. 函数的类型

以下是类型相同的函数：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
```

定义函数类型：　**typedef** *类型 函数类型*（ *形式参数表* ）；

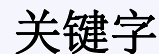
3.4.2　函数指针

1. 函数的类型

以下是类型相同的函数：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
```

定义函数类型：　**typedef** *类型 函数类型*（ *形式参数表* ）；



3.4.2 函数指针

1. 函数的类型

以下是类型相同的函数：

double max (double, double) ;

double min (double, double) ;

double average (double, double) ;

定义函数类型： **typedef** *类型* *函数类型* （ *形式参数表* ） ；

3.4.2　函数指针

1. 函数的类型

以下是类型相同的函数：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
```

定义函数类型：　typedef *类型* *函数类型* （ *形式参数表* ）；

用户定义标识符

3.4.2 函数指针

1. 函数的类型

以下是类型相同的函数：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
```

定义函数类型： typedef *类型 函数类型 （形式参数表）* ；

3.4.2 函数指针

1. 函数的类型

以下是类型相同的函数：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double
```

它们的类型为：

```
double  ( double, double )
```

或定义为：

```
typedef double  functionType  ( double, double) ;
```

```
//integer 是 int 的同义词:
typedef  int integer;

//real 是 double 的同义词:
typedef  double real;
```

3.4.2 函数指针

2. 函数指针

若有函数类型为：

```
double  ( double, double ) ;
```

或：

```
typedef double  functionType  ( double, double ) ;
```

定义指向这类函数的指针变量：

```
double  ( * fp ) ( double, double ) ;
```

或：

```
functionType  * fp1 ,  * fp2 ;
```

函数指针

3.4.2 函数指针

2．函数指针

例如:

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp1 ,  * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

3.4.2 函数指针

2. 函数指针

例如：

double* max *(double, double) ;

double* min *(double, double) ;

double* average *(double, double) ;

函数原型

```
typedef double  functionType  ( double, double ) ;
functionType  * fp1 ,  * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

3.4.2 函数指针

2．函数指针

例如：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
typedef double functionType ( double, double ) ;
functionType * fp1 , * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

定义函数类型

3.4.2 函数指针

2. 函数指针

例如：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp1 ,  * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

声明函数指针

3.4.2 函数指针

2. 函数指针

例如:

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp1 ,  * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

获取函数地址

3.4.2 函数指针

2. 函数指针

例如：

```
double max ( double, double ) ;
double min ( double, double ) ;
double average ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp1 ,  * fp2 ;
fp1 = max ;
fp2 = min ;
fp1 = average ;
fp1 = fp2 ;
```

改变指针指向

3.4.2 函数指针

2. 函数指针

例如：

```
typedef double  functionType  ( double, double ) ;

functionType  max, min, average ;

functionType  * fp1 ,  * fp2 ;

fp1 = max ;

fp2 = min ;

fp1 = average ;

fp1 = fp2 ;
```

说明了3个函数原型

3.4.2　函数指针

3．用函数指针调用函数

例如：

```
double max ( double, double ) ;
double min ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp ;
fp = max ;
double x = fp ( 3.14, 0.516 ) ;
fp = min ;
cout << fp ( 0.75, x ) << endl ;
```

3.4.2 函数指针

3．用函数指针调用函数

例如：

```
double max ( double, double ) ;
double min ( double, double ) ;
typedef double  functionType  ( double, double ) ;
functionType  * fp ;
fp = max ;
double x = fp ( 3.14, 0.516 ) ;
fp = min ;
cout << fp ( 0.75, x ) << endl ;
```

指向函数max

3.4.2 函数指针

```
// 例3-21 用函数指针调用函数
#include<iostream>
using namespace std ;
int  sum ( int  x ,  int  y )  {  return  x + y ;  }
int  product ( int  x ,  int  y )  {  return  x * y ;  }
int main()
{  int  ( * pf )  ( int ,   int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ; cin >> a ;
   cout << "b = " ; cin >> b ;
   pf = sum ;      result = pf ( a ,  b )  ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;     result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

3.4.2 函数指针

```
// 例3-21 用函数指针调用函数
#include<iostream>
using namespace std ;
int  sum ( int  x ,  int  y ) {  return  x + y ;  }
int  product ( int  x ,  int  y ) {  return  x * y ;  }
int main()
{  int  ( * pf )  ( int ,  int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ;   cin >> a ;
   cout << "b = " ;   cin >> b ;
   pf = sum ;      result = pf ( a ,  b )  ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;     result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

函数指针

3.4.2 函数指针

```
// 例3-21  用函数指针调用函数
#include<iostream>
using namespace std ;
int  sum ( int  x ,  int  y )  {  return  x + y ;  }
int  product ( int  x ,  int  y )  {  return  x * y ;  }
int main()
{  int  ( * pf )  ( int ,   int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ;   cin >> a ;
   cout << "b = " ;   cin >> b ;
   pf = sum ;      result = pf ( a ,  b ) ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;     result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

调用不同函数

3.4.2 函数指针

// 例3-21 用函数指针调用函数

```
#include<iostream>
using namespace std ;
int sum ( int  x ,  int  y )  {  return  x + y ;  }
int product ( int  x ,  int  y )  {  return  x * y ;  }
int main()
{  int  ( * pf )  ( int ,   int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ;   cin >> a ;
   cout << "b = " ;   cin >> b ;
   pf = sum ;      result = pf ( a ,  b )  ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;      result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

(*pf) (a, b)
等价吗？

3.4.2 函数指针

// 例3-21 用函数指针调用函数

```
#include<iostream>
using namespace std ;
int  sum ( int  x ,  int  y )  {  return  x + y ;  }
int  product ( int  x ,  int  y )  {  return  x * y ;  }
int main()
{  int  ( * pf )  ( int ,   int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ;   cin >> a ;
   cout << "b = " ;   cin >> b ;
   pf = sum ;      result = pf ( a ,  b )  ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;     result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

(&pf) (a, b)

等价吗？

3.4.2 函数指针

```
// 例3-21 用函数指针调用函数
#include<iostream>
using namespace std ;
int sum ( int x , int y ) { return x + y ; }
int product ( int x , int y ) { r
int main()
{ int ( * pf ) ( int , int ) ;
   int a , b , result ;
   cout << "a = " ;  cin >> a ;
   cout << "b = " ;  cin >> b ;
   pf = sum ;    result = pf ( a , b ) ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;    result = pf ( a , b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

(&pf) (a, b) 与

(&sum) (a, b) 有何区别？

3.4.2 函数指针

```
// 例3-21  用函数指针调用函数
#include<iostream>
using namespace std ;
int  sum ( int  x ,  int  y )  {  return  x + y ;  }
int  product ( int  x ,  int  y )  {  retur
int main()
{  int  ( * pf )  ( int ,   int ) ;
   int  a ,  b ,  result ;
   cout << "a = " ;   cin >> a ;
   cout << "b = " ;   cin >> b ;
   pf = sum ;      result = pf ( a ,  b )  ;
   cout << a << " + " << b << " = " << result << endl ;
   pf = product ;     result = pf ( a ,  b ) ;
   cout << a << " * " << b << " = " << result << endl ;
}
```

pf 是指针变量，存放函数的地址

sum 是函数的直接地址

pf 的值等于 *sum* （ *&sum* ）

&pf 不等于 *sum* （ *&sum* ）

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                    // 定义函数类型
funType circlePerimeter ;                            // 用函数类型名定义函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;               // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                     // 定义函数类型
funType circlePerimeter ;                            // 用函数类型名定义函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;               // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

函数类型

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                    // 定义函数类型
funType circlePerimeter ;                                        函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;               // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

函数原型

```
double circlePerimeter ( double );
double circleArea ( double );
double ballArea ( double );
double ballVolume ( double );
```

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                    // 定义函数类型
funType circlePerimeter ;                            // 用函数类型名定义函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;               // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

```
funType circlePerimeter, circleArea, ballArea, ballVolume;
```

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                     定义函数类型
funType circlePerimeter ;                          // 用函数类型名定义函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;             // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

函数原型

// 例3-22 使用函数指针参数调用函数

```
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );                    // 定义函数类型
funType circlePerimeter ;                                              数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;               // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

函数指针参数

double (*) (double)

```
// 例3-22  使用函数指针参数调用函数
#include <iostream>
using namespace std ;
#include <cmath>
typedef double funType( double );              // 定义函数类型
funType circlePerimeter ;                      // 用函数类型名定义函数原型
funType circleArea ;
funType ballArea ;
funType ballVolume ;
double callFun( funType * , double ) ;         // 第一个参数是函数指针参数
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
```

实际参数是函数地址

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolu
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

circlePerimeter(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolur
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

circlePerimeter(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;     cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume,
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

circleArea(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume,
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

circleArea(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume,
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

ballArea(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume,
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

ballArea(r)

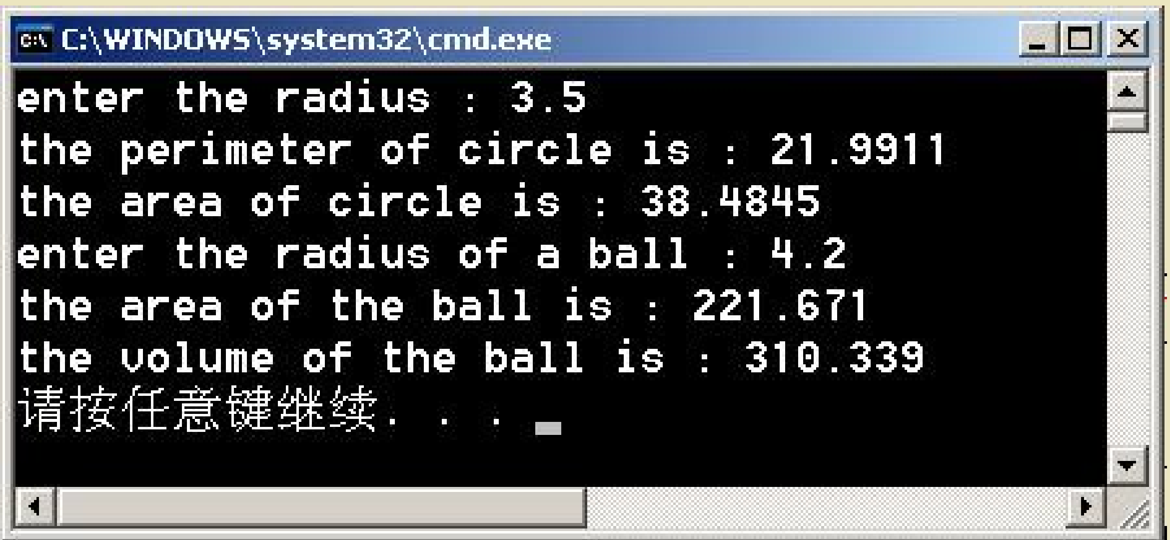
```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;      cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

ballVolume(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
const double pi = 3.14159 ;
double callFun(funType * qf, double r)  { return qf( r ) ; }
double circlePerimeter( double radius)  { return 2 * pi * radius ; }
double circleArea( double radius )  { return pi * radius * radius ; }
double ballArea( double radius )  { return 4 * pi * radius * radius ; }
double ballVolume( double radius)  { return 4.0 / 3 * pi * pow( radius, 3 ) ; }
```

ballVolume(r)

```
int main()
{ double r ;
  cout << "enter the radius : " ;    cin >> r ;
  cout << "the perimeter of circle is : "<< callFun(circlePerimeter, r) << endl ;
  cout << "the area of circle is : "<< callFun(circleArea, r) << endl ;
  cout << "enter the radius of a ball : " ;     cin >> r ;
  cout << "the area of the ball is : "<< callFun(ballArea, r) << endl ;
  cout << "the volume of the ball is : "<< callFun(ballVolume, r) << endl ;
}
const double pi = 3.14159
double callFun(funType *
double circlePerimeter( do
double circleArea( double
double ballArea( double r
double ballVolume( doubl
```



3.5 内联函数和重载函数

- 内联函数是C++为降低小程序调用开销的一种机制
- 函数重载是以同一个名字命名多个函数实现版本

3.5.1 内联函数

内联函数作用

减少频繁调用小子程序的运行的时间开销

内联函数机制

编译器在编译时，将内联函数的调用以相应代码代替

内联函数声明

inline *函数原型*

注： 内联函数仅在函数原型作一次声明。

适用于只有**1～5**行的小函数

不能含有复杂结构控制语句，不能递归调用

3.5.1 内联函数

例:

```
inline  int  smallf ( ) ;
int main ( )
{ ……
    a = smallf();
    ……
}
int  smallf ( )
{
    ……
}
```

```
inline  int  smallf ( )
{
    ……
}
int main ( )
{ ……
    a = smallf ( )
    ……
}
```

3. 5. 1 内联函数

错误说明:

```
inline  int  smallf ( ) ;
int main ( )
{ ……
    a = smallf();
    ……
}
inline  int  smallf ( )
{
    ……
}
```

重复说明，语法错误

3.5.1 内联函数

错误说明:

```
inline  int  smallf ( ) ;
int main ( )
{ ……
    a = smallf();
    ……
}
inline  int  smallf ( )
{
    ……
}
```

```
int  smallf ( ) ;
int main ( )
{ ……
    a = smallf();
    ……
}
inline  int  smallf ( )
{
    ……
}
```

作普通函数处理

3. 5. 1 内联函数

```
// 例3-23  内联函数示例
#include<iostream>
using namespace std ;
inline double volume ( double ,   double  ) ;        // 函数原型
int main ( )
{
    double  vol,  r,  h ;
    cin  >>  r >> h ;
    vol = volume ( r,  h ) ;
    cout << "Volume = " <<  vol  << endl ;
}
double volume ( double radius,  double  height )
{
    return   3.14 * radius * radius * height;
}
```

编译器变换为：

vol = 3.14 * r * r * h ;

3.5.2 函数重载

- 多个同名函数有不同的参数集
- 编译器根据不同参数的类型和个数产生调用匹配
- 函数重载用于处理不同数据类型的类似任务

3.5.2 函数重载

重载示例:

参数个数相同

参数类型不同

```
#include<iostream>
using namespace std ;
int  abs ( int  a ) ;
double  abs ( double  f ) ;
int main ()
  {  cout << abs ( -5 )  << endl ;
     cout << abs ( -7.8 ) << endl ;
  }
int  abs ( int  a )
  {  return  a <  0  ?  -a  :  a ;  }
double  abs ( double  f )
  {  return  f < 0  ?  -f  :  f ;  }
```

3.5.2 函数重载

重载示例:

参数个数不同

```
#include<iostream>
using namespace std ;
int  max ( int a ,  int b ) ;
int  max ( int a ,  int b,  int c ) ;
int main ()
 { cout << max ( 5, 3 ) << endl ;
    cout << max (4, 8, 2 ) << endl ;
 }
int  max ( int a ,  int  b )
 { return a > b ? a : b ;  }
int  max ( int a ,  int  b,  int  c )
 { int  t ;
    t = max ( a ,  b ) ;
    return  max ( t ,  c ) ;
 }
```

3.5.2 函数重载

默认参数

重载示例:

```
void  ferror ( int x , int y = 0 ) ;
void  ferror ( int x ) ;
```

有调用:

ferror (3) ;

错误!

编译器无法唯一确定调用函数

3.5.2 函数重载

重载示例:

void ferror (int x , int y = 0) ;

void ferror (int x) ;

有调用:

ferror (3) ;

编译器无法唯一确定调用函数

错误!

int average(*int, int*) ;

double average(*int, int*) ;

仅返回类型不同

函数重定义

3.6 变量存储特性与标识符作用域

- 标识符存储特性确定内存的生存时间和连接特性
- 标识符作用域是在程序正文中能够被引用的那部分区域
- 标识符的连接特性决定能否被工程中的其他文件引用

3.6.1 存储特性

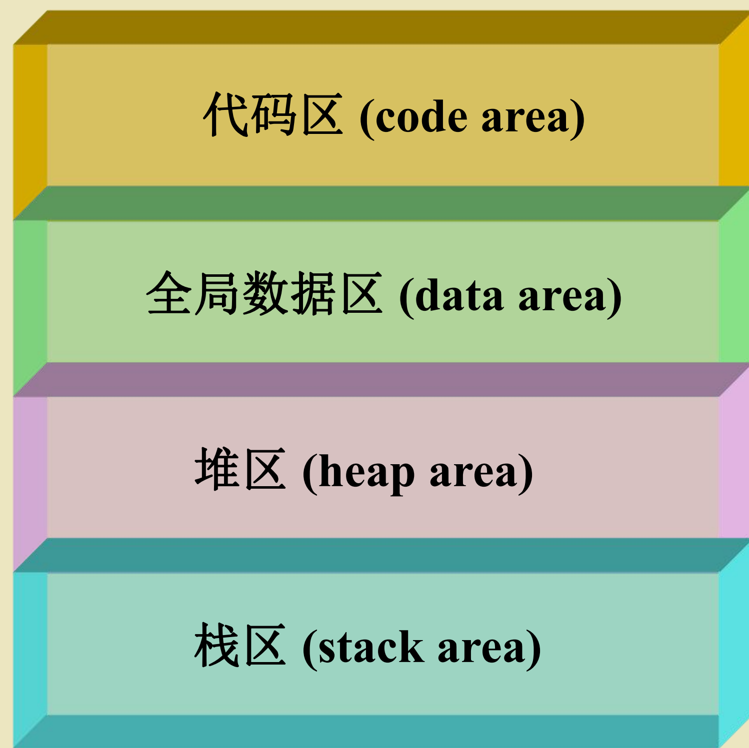
程序的内存区域

存放程序代码

存放程序的全局数据和静态数据

存放程序的动态数据

存放程序的局部数据



3.6.1 存储特性

1． 自动存储类

程序的内存区域

自动存储变量

- 自动存储变量存放在栈区
- 进入声明的块时生成，在结束块时删除
- 函数的参数和局部变量都是自动存储类
- 自动存储是变量的默认状态

```
int  max ( int a , int b, int c )
 { int  t ;
   t = max ( a ,  b ) ;
   return  max ( t ,  c ) ;
 }
```

3.6.1 存储特性

2. 静态存储类

- 关键字extern和static声明静态存储类变量和函数标识符
- extern声明全局量（全局量默认为extern），static声明局部量
- extern和static说明变量时，程序开始执行时分配和初始化存储空间
- extern和static说明函数，表示从程序执行开始就存在这个函数名

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
   cout << func() << endl ;
}
int func()
{ int a = 0 ;            // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
   cout << func() << endl ;
}
int func()
{ int a = 0 ;            // 自动变量
  static int b = 1 ;     // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

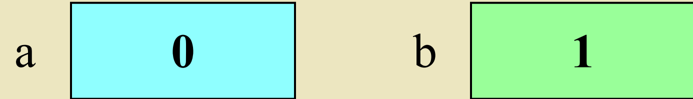
```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
   cout << func() << endl ;
}
int func()
{ int a = 0 ;             // 自动变量
  static int b = 1 ;      // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

// 例3-24 静态变量与自动变量的测试

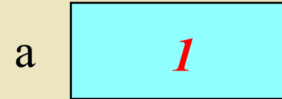
```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

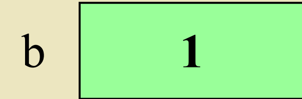
a | 0

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

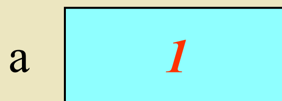


```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

a


b


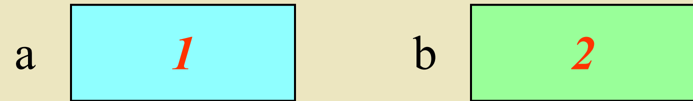
```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

a 

b

// 例3-24 静态变量与自动变量的测试

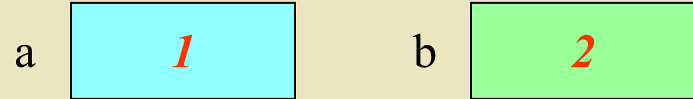
```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



auto a = 1

// 例3-24 静态变量与自动变量的测试

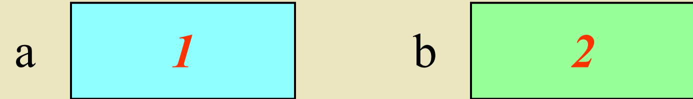
```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
```

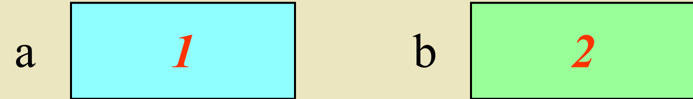
// 例3-24 静态变量与自动变量的测试

```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
```

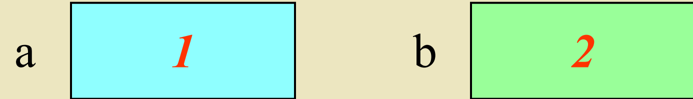
```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



auto a = 1

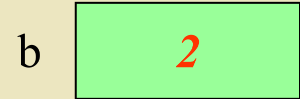
static b = 2

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

b 2

```
auto a = 1
static b = 2
3
```

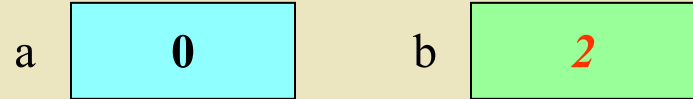
// 例3-24 静态变量与自动变量的测试

```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

b 2

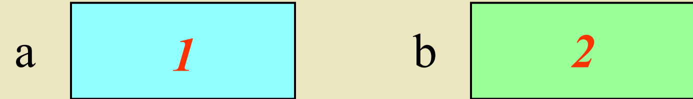
```
auto a = 1
static b = 2
3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



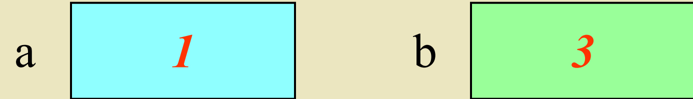

```
auto a = 1
static b = 2
3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

```
auto a = 1
static b = 2
3
```

// 例3-24 静态变量与自动变量的测试

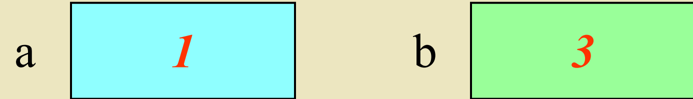
```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;            // 自动变量
  static int b = 1 ;     // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
```

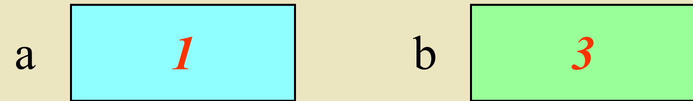
// 例3-24 静态变量与自动变量的测试

```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



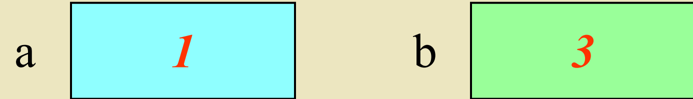

```
auto a = 1
static b = 2
3
auto a = 1
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;            // 自动变量
  static int b = 1 ;     // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
auto a = 1
```

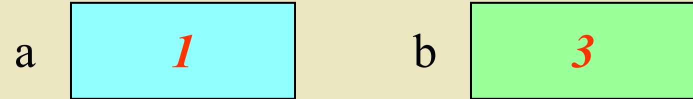
```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



auto a = 1
static b = 2
3
auto a = 1
static b = 3

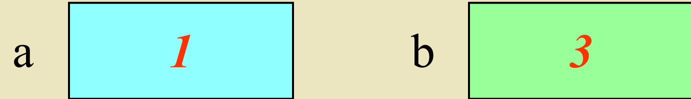
// 例3-24 静态变量与自动变量的测试

```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```



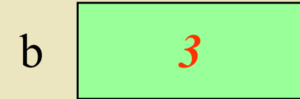

```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;            // 自动变量
  static int b = 1 ;     // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```




```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
4
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
4
```

```
// 例3-24  静态变量与自动变量的测试
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;           // 自动变量
  static int b = 1 ;    // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

C++不对自动变量初始化

```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
4
```

// 例3-24 静态变量与自动变量的测试

```
#include <iostream>
using namespace std ;
int func();
int main()
{ cout << func() << endl ;
  cout << func() << endl ;
}
int func()
{ int a = 0 ;          // 自动变量
  static int b = 1 ;   // 静态变量
  a ++ ;
  b ++ ;
  cout << "auto a = " << a << endl ;
  cout << "static b = " << b << endl ;
  return a + b ;
}
```

静态变量默认初始化值为 0

```
auto a = 1
static b = 2
3
auto a = 1
static b = 3
4
```

```
// 例3-25  用静态变量测试密码输入次数
#include<iostream>
using namespace std;
int password( const int & key );
int main()
{  if( password(123456) )                    //调用函数，测试用户输入密码
      cout << "Welcome!" << endl;
   else
      cout << "Sorry,you are wrong!" << endl;
}
```

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{  if( p
      co
   else
      co
}
```

```
int password( const int & key )
{  static int n = 0;                          //静态变量
    int k;
    cout<< "Please input your password: ";
    cin >> k;                                 //输入密码
    n ++;                                     用次数
    if( n<3 )                                 //输入次数  法
      {  if( k==key )  return 1;              //密码正确
          else  password(key);                //递归，重新输入
      }
   else                                       //连续输入3次错误
      {  if( k!=key )    return 0;
      }
}
```

静态变量，用于记录输入次数

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{  if( p
      co
   else
      co
}
```

```
int password( const int & key )
{  static int n = 0;                              //静态变量
   int k;
   cout<< "Please input your passw
   cin >> k;                                      //输入密码
   n ++;                                          //记录输入次数，即函数调用次数
   if( n<3 )                                      //输入次数合法
    {  if( k==key )  return 1;                    //密码正确
       else  password(key);                       //递归，重新输入
    }
   else                                           //连续输入3次错误
    {  if( k!=key )    return 0;
    }
}
```

函数被调用一次自增1

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{ if( p
     co
  else
     co
}
```

```
int password( const int & key )
{ static int n = 0;                          //静态变量
   int k;
   cout<< "Please input your password: ";
   cin >> k;                                 //输入密码
   n ++;                                     //记录输 … 次数
   if( n<3 )                                 //输入次
    { if( k==key )  return 1;                //密码正
       else  password(key);                  //递归，重新输入
    }
  else                                       //连续输入3次错误
    { if( k!=key )    return 0;
    }
}
```

递归函数的参数
为常引用参数

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{  if( p
      co
   else
      co
}
```

```
int password( const int & key )
{  static int n = 0;                              //静态变量
   int k;
   cout<< "Please input your password: ";
   cin >> k;                                      //输入密码
   n ++;                                          //i            用次数
   if( n<3 )                                      //
   {  if( k==key )  return 1;                     //
      else  password(key);                        //递归，重新输入
   }
  else                                            //连续输入3次错误
   {  if( k!=key )    return 0;
   }
}
```

递归条件

n<3 && k!=key

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{  if( p
      co
   else
      co
}
```

```
int password( const int & key )
{  static int n = 0;                              //静态变量
   int k;
   cout<< "Please input your password: ";
   cin >> k;                                      //输入密码
   n ++;                                          //记录输入次数，即函数调用次数
   if( n<3 )                                      //输入次
     { if( k==key )  return 1;                    //密码正
       else  password(key);                       //递归，重新输入
     }
   else                                           //连续输入3次错误
     { if( k!=key )     return 0;
     }
}
```

结束函数的

两种情况

// 例3-25 用静态变量测试密码输入次数

```
#include<iostream>
using namespace std;
int password( const int & key );
int mai
{  if( p
      co
   else
      co
}
```

```
int password( const int & key )
{  static int n = 0;                          //静态变量
   int k;
   cout<< "Please input your password: ";
   cin >> k;                                  //输入密码
   n ++;                                      //记录输入次数，即函数调用次数
   if( n<3 )                                  //输入次数合法
     {  if( k==key )  return 1;               //密码正确
          else  password(key);                //递归，重新输入
     }
   else                                       //连续输入3次错误
     {  if( k!=key )     return 0;
     }
}
```

3.6.2 标识符作用域

1. 函数原型作用域

函数原型形式参数表中的标识符具有函数原型作用域

例如，以下函数原型编译器认为是相同的：

```
double funPrototype ( double , double ) ;

double funPrototype ( double a , double b ) ;

double funPrototype ( double x , double y ) ;
```

3.6.2 标识符作用域

2．块作用域

在语句块中声明的标识符具有块作用域

```
// 例3-25  不同作用域的同名变量
#include<iostream>
using namespace std ;
int main()
{ int a = 1;                          // 外层的a
 { int a = 1 ;                        // 内层的a
     a ++ ;
    cout << "inside a = " << a << endl ;
 }                                    // 内层的a作用域结束
cout << "outside a = " << a << endl ;
}                                     // 外层的a作用域结束
```

3.6.2 标识符作用域

2．块作用域

在语句块中声明的标识符具有块作用域

```
// 例3-25  不同作用域
#include<iostream>
using namespace std ;
int main()
{ int a = 1;                        // 外层的a
 { int a = 1 ;                      // 内层的a
    a ++ ;
    cout << "inside a = " << a << endl ;
 }                                  // 内层的a作用域结束
cout << "outside a = " << a << endl ;
}                                   // 外层的a作用域结束
```

内层的 a 覆盖了外层的 a

```
inside a = 2
outside a = 1
```

3.6.2 标识符作用域

3．函数作用域

- 语句标号（后面带冒号的标识符）是惟一具有函数作用域的标识符
- 语句标号用于switch结构中的case标号，goto语句转向入口的语句标号
- 标号可以在函数体中任何地方使用，但不能在函数体外引用

3.6.2 标识符作用域

4．文件作用域

- 任何在函数之外声明的标识符具有文件作用域
- 这种标识符从声明处起至文件尾的任何函数都可见

```
// 例3-26  使用文件作用域变量
#include <iostream>
using namespace std ;
int a = 1, b = 1 ;
void f1( int x )
{ a = x * x ;
   b = a * x ;
}
int c ;
void f2( int x, int y )
{ a = x > y ? x : y ;
   b = x < y ? x : y ;
   c = x + y ;
}
int main()
{ f1( 4 ) ;
  cout << "call function f1 :\n" ;
  cout << "a = " << a << " , b = " << b << endl ;
  f2 (10, 23 ) ;
  cout << "call function f2 :\n" ;
  cout << "a = " << a << " , b = " << b << " , c = " << c << endl ;
}
```

```
// 例3-26  使用文件作用域变量
#include <iostream>
using namespace std ;
int a = 1, b = 1 ;              // a, b的作用域从这里开始
void f1( int x )                // f1函数可以访问a,b
{ a = x * x ;
  b = a * x ;
}
int c ;
void f2( int x, int y )         // f2函数可以访问a, b
{ a = x > y ? x : y ;
  b = x < y ? x : y ;
  c = x + y ;
}
int main()                      // main 函数可以访问a, b
{ f1( 4 ) ;
  cout << "call function f1 :\n" ;
  cout << "a = " << a << " , b = " << b << endl ;
  f2 (10, 23 ) ;
  cout << "call function f2 :\n" ;
  cout << "a = " << a << " , b = " << b << " , c = " << c << endl ;
}
```

```
// 例3-26  使用文件作用域变量
#include <iostream>
using namespace std ;
int a = 1, b = 1 ;            // a, b的作用域从这里开始
void f1( int x )              // f1函数可以访问a,b
{ a = x * x ;
  b = a * x ;
}
int c ;                       // c 的作用域从这里开始，默认初始值为0
void f2( int x, int y )       // f2函数可以访问a, b, c
{ a = x > y ? x : y ;
  b = x < y ? x : y ;
  c = x + y ;
}
int main()                    // main 函数可以访问a,
{ f1( 4 ) ;
  cout << "call function f1 :\n" ;
  cout << "a = " << a << " , b = " << b << endl ;
  f2 (10, 23 ) ;
  cout << "call function f2 :\n" ;
  cout << "a = " << a << " , b = " << b << " , c = " << c << endl ;
}
```

3.6.2 标识符作用域

5．全局变量和局部变量

- 具有文件作用域的变量称为全局变量；具有函数作用域或块作用域的变量称为局部变量
- 全局变量声明时默认初始值为0
- 当局部量与全局量同名，在块内**屏蔽**全局量
- 为了在块内访问全局量，可以用域运算符〞::〞

3.6.2 标识符作用域

5．全局变量和局部变量

```
// 在函数体内访问全局变量
#include<iostream>
using namespace std ;
int x ;
int main()
{ int x = 256 ;
   cout << "global variable x = " << ::x <<endl ;
   cout << "local variable x = " << x << endl ;
}
```

3.6.2 标识符作用域

5. 全局变量和局部变量

```
// 在函数体内访问全局变量
#include<iostream>
using namespace std ;
int x ;
int main()
{ int x = 256 ;
   cout << "global variable x = " << ::x <<endl ;
   cout << "local variable x = " << x << endl ;
}
```

访问全局量

3.6.2 标识符作用域

5. 全局变量和局部变量

```
// 在函数体内访问全局变量
#include<iostream>
using namespace std ;
int x ;
int main()
{ int x = 256 ;
  cout << "global variable x = " << ::x <<endl ;
  cout << "local variable x = " << x << endl ;
}
```

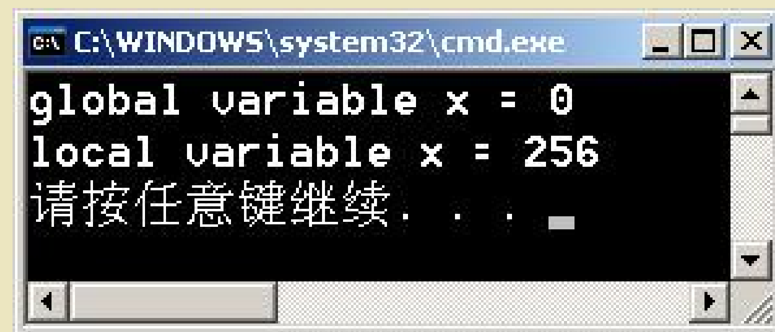
访问局部量

3.6.2 标识符作用域

5．全局变量和局部变量

// 在函数体内访问全局变量

```
#include<iostream>
using namespace std ;
int x ;
int main()
{ int x = 256 ;
   cout << "global variable x = " << ::x <<endl ;
   cout << "local variable x = " << x << endl ;
}
```

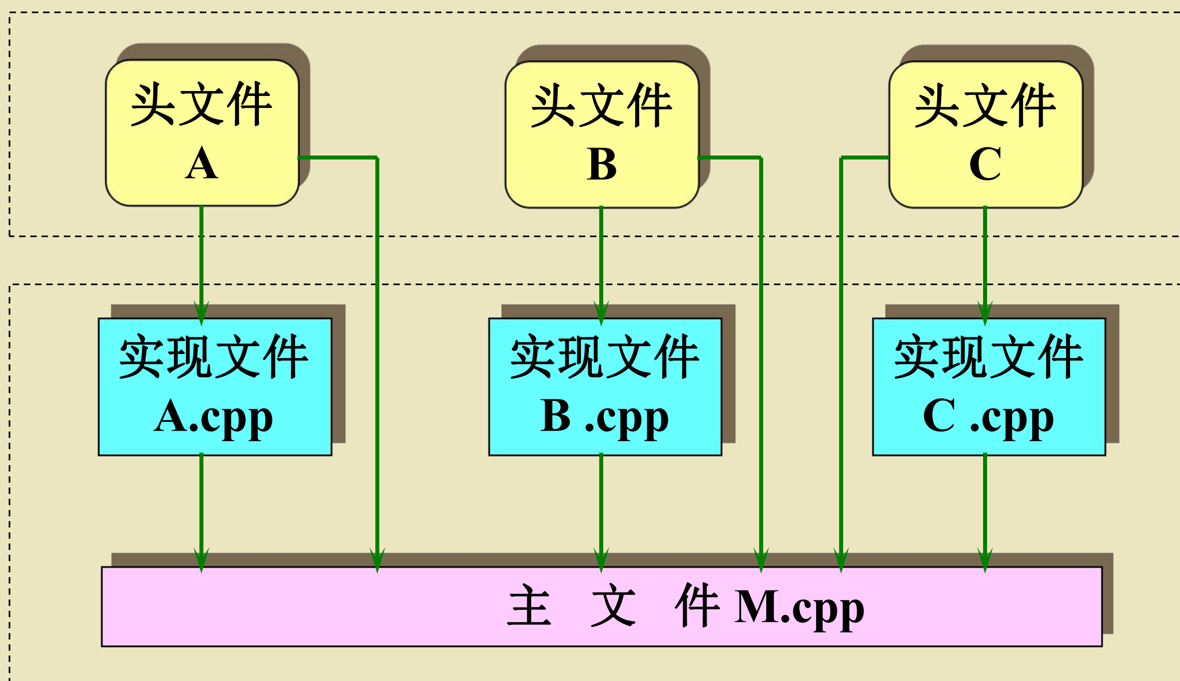


3.7 多文件程序结构

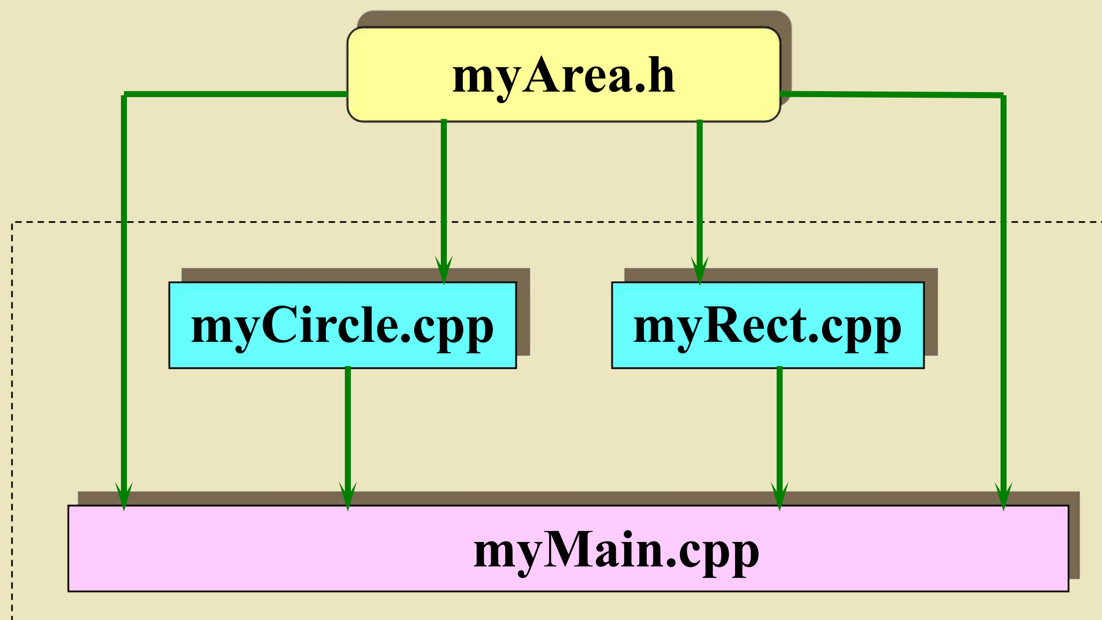
- 一个C++程序称为一个工程（.dsp）
- 一个工程由一个或多个文件组成
- 一个文件可以包含多个函数定义，但一个函数的定义必须完整地存在于一个文件中
- 一个文件可以被多个应用程序共享

3.7.1 多文件结构

一个好的软件系统应当分解为各种同构逻辑文件



例3-27 计算圆面积和矩形面积



例3-27 计算圆面积和矩形面积

myArea.h

```
double circle( double radius ) ;
double rect( double width, double length ) ;
```

myCircle.cpp

```
const double pi = 3.14 ;
double circle ( double radius )
{ return pi * radius * radius ; }
```

myRect.cpp

```
double rect ( double with, double length )
{ return with * length ; }
```

例3-27 计算圆面积和矩形面积

myMain.cpp

```
#include<iostream>
using namespace std ;
#include "myArea.h"
int main()
{ double width, length ;
    cout << "Please enter two numbers:\n" ;
    cin >> width >> length ;
    cout << "Area of recttangle is: " << rect( width, length ) << endl ;
    double radius ;
    cout << "Please enter a radius:\n" ;
    cin >> radius ;
    cout << "Area of circle is: " << circle( radius ) << endl ;
}
```

例3-27 计算圆面积和矩形面积

example - Microsoft Visual Studio

文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 数据(A) 工具(T) 体系结构(C) 测试(S) 分析(N) 窗口(W) 帮助(H)

解决方案资源管理器

解决方案"example"(1 个项目)
example
外部依赖项
头文件
源文件
资源文件

添加新项 - example

已安装的模板

Visual C++
UI
代码
数据
资源
Web
实用工具
属性表

排序依据: 默认值

搜索 已安装的模板

模板	类型
Windows 窗体	Visual C++
C++ 文件(.cpp)	Visual C++
HTML 页(.htm)	Visual C++
静态发现文件(.disco)	Visual C++
头文件(.h)	Visual C++
Midl 文件(.idl)	Visual C++
资源文件(.rc)	Visual C++
服务器响应文件(.srf)	Visual C++

类型: Visual C++
创建 C++ 头文件

名称(N): myArea
位置(L): D:\example\example
浏览(B)...
添加(A) 取消

添加头文件

例3-27 计算圆面积和矩形面积

example - Microsoft Visual Studio

myArea.h*

```
double circle( double radius ) ;
double rect( double width, double length ) ;
```

输入代码

例3-27 计算圆面积和矩形面积

添加源文件

example - Microsoft Visual Studio

文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 数据(A) 工具(T) 体系结构(C) 测试(S) 分析(N) 窗口(W) 帮

解决方案资源管理器

解决方案"example"(1 个项目)
example
外部依赖项
头文件
myArea.cpp
源文件
资源文件

myArea.cpp*

(全局范围)

circle(double radius)

添加新项 - example

已安装的模板

Visual C++
UI
代码
数据
资源
Web
实用工具
属性表

排序依据: 默认值

搜索 已安装的模板

模板	类型
Windo 窗体	Visual C++
C++ 文件(.cpp)	Visual C++
HTML 页(.htm)	Visual C++
静态发现文件(.disco)	Visual C++
头文件(.h)	Visual C++
Midl 文件(.idl)	Visual C++
资源文件(.rc)	Visual C++
服务器响应文件(.srf)	Visual C++

类型: Visual C++
创建包含 C++ 源代码的文件

名称(N): myCircle
位置(L): D:\example\example
浏览(B)...
添加(A) 取消

解决方案... 工具箱

例3-27 计算圆面积和矩形面积

example - Microsoft Visual Studio

解决方案"example"(1 个项目)
- example
  - 外部依赖项
  - 头文件
    - myArea.h
  - 源文件
    - myCircle.cpp
  - 资源文件

myCircle.cpp* | myArea.h*

(全局范围)

```
const double pi = 3.14 ;
double circle ( double radius )
{ return pi * radius * radius ; }
```

输入代码

例3-27 计算圆面积和矩形面积

example - Microsoft Visual Studio

解决方案资源管理器

- 解决方案"example"(1 个项目)
  - example
    - 外部依赖项
    - 头文件
      - myArea.h
    - 源文件
      - myCircle.cpp
      - myMain.cpp
      - myRect.cpp
    - 资源文件

myMain.cpp　myRect.cpp　myCircle.cpp　myArea.h

```
#include<iostream>
using namespace std ;
#include "myArea.h"
int main()
{ double width, length ;
      cout << "Please enter two numbers:\n" ;
      cin >> width >> length ;
      cout << "Area of recttangle is: " << rect( width, length )
      double radius ;
      cout << "Please enter a radius:\n" ;
      cin >> radius ;
      cout << "Area of circle is: " << circle( radius ) << endl ;
}
```

完整项目文件

输出

```
========== 生成: 成功 0 个，失败 0 个，最新 1 个，跳过 0 个 ==========
```

生成成功

3.7.2　预处理指令

C++编译器工作过程

源程序 → 预处理器 → 词法分析 → 语法分析 → 代码生成 → 目标程序

符号表

错误处理程序 → 错误信息

预处理器　　改善程序的组织和管理

预处理指令　所有编译指令以 # 开头，每条指令单独占一行

3.7.2　预处理指令

1．*文件包含*

include指令在编译之前把指定文件包含到该命令所在位置

形式为：

系统头文件

 # include <*文件名*>

或　　# include "*文件名*"

3.7.2 预处理指令

1. 文件包含

include指令在编译之前把指定文件包含到该命令所在位置

形式为：

 # include <*文件名*>

或 # include "*文件名*"

自定义头文件

3.7.2 预处理指令

2. 条件编译

- 形式1

```
# if 常量表达式
程序段
# endif
```

- 形式2

```
# if 常量表达式
程序段1
# else
程序段2
# endif
```

- 形式3

```
# if 常量表达式1
程序段1
# elif 常量表达式2
程序段2
…
# elif 常量表达式n
程序段n
# else
程序段n+1
# endif
```

3.7.2　预处理指令

3．宏定义指令

用指定正文替换程序中出现的标识符

形式　　　　**#define** *标识符 文本*

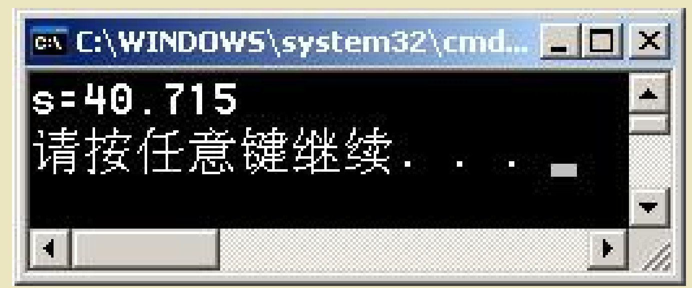
C语言的宏替换直接做文本替换，没有类型检查。C++也支持。

```
#include<iostream>
using namespace std ;
//不带参数宏替换。在程序正文中，用3.1415926代替PI
#define PI  3.1415926
//带参数宏替换。在程序正文中，PI*r*r代替area(x)，r是参数
#define area(r)  PI*r*r
int main()
{ double x, s;
  x=3.6;
  s=area(x);
  cout<<"s="<<s<<endl;
}
```

C语言的宏替换直接做文本替换，没有类型检查。C++也支持。

```
#include<iostream>
using namespace std ;
//不带参数的宏替换，在C++中使用常量定义：
const double PI=3.1415926;
//带参数宏替换，C++使用内联函数：
inline double area(double r) {return PI*r*r;}
int main()
{ double x, s;
  x=3.6;
  s=area(x);
  cout<<"s="<<s<<endl;
}
```

是参数



3.7.2　预处理指令

3．*宏定义指令*

用指定正文替换程序中出现的标识符

形式　　　　　**#define** *标识符 文本*

#define 的一个有效应用是在条件编译指令中，避免程序中多次用**include**指令包含这个头文件，出现重定义的错误

3.7.2 预处理指令

3．宏定义指令

```
// calculate
#ifndef CALCULATE_H    // 若CALCULATE_H未定义，执行下一宏指令
   #define CALCULATE_H    // 用后续3行正文代替CALCULATE_H
   double circle(double radius)
         {   const double pi = 3.14159 ;
             return pi * radius * radius;
         }
#endif
```

3.7.2 预处理指令

3. 宏定义指令

```
// calculate
#ifndef CALCULATE_H      // 若CALCULATE_H未定义，执行下一宏指令
   #define CALCULATE_H      // 用后续3行正文代替CALCULATE_H
   double circle(double radius)
        {   const double pi = 3.14159 ;
            return pi * radius * radius;
        }
#endif
```

3.8 命名空间

- 命名空间是类、函数、对象、类型和其他名字的集合。
- 命名空间令软件组件之间不会产生命名冲突。
- **std**是**C++**的标准名空间，包含了标准头文件中各种名字的声明。

3.8.1 标准名空间

- **C++标准头文件没有扩展名。**

  iostream iomanip limit fstream string typeinfo stdexcept

- **使用标准类库的组件时，需要指定名空间。**

- **C++标准名空间 std**

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>
using namespace std;
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>                    //包含头文件
using namespace std;
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>              //包含头文件
using namespace std;            //使用标准名空间std
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>                    //包含头文件
using namespace std;                  //使用标准名空间std
int main()
{ int a, b;
  cin>>a;                             //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>                //包含头文件
using namespace std;              //使用标准名空间std
int main()
{ int a, b;
  cin>>a;                         //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';        //使用std的元素cout
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法一：
#include<iostream>                    //包含头文件
using namespace std;                  //指定使用名空间std
int main()                            //使用std的元素cin
{ int a, b;
  cin>>a;                             //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';            //使用std的元素cout
}
```

3.8.1　标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;
using std::cout;
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;                    //指定使用std的元素cin
using std::cout;
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1　标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;                    //指定使用std的元素cin
using std::cout;                   //指定使用std的元素cout
int main()
{ int a, b;
  cin>>a;
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;                       //指定使用std的元素cin
using std::cout;                      //指定使用std的元素cout
int main()
{ int a, b;
  cin>>a;                             //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;                    //指定使用std的元素cin
using std::cout;                   //指定使用std的元素cout
int main()
{ int a, b;
  cin>>a;                          //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';         //使用std的元素cout
}
```

3.8.1　标准名空间

使用标准名空间

```
//方法二：
#include<iostream>
using std::cin;                    //指定使用std的元素cin
using std::cout;                   //指定使用std的元素cout
int main()
{ int a, b;
  cin>>a;                          //使用std的元素cin
  cin>>b;
  cout<<"a+b="<<a+b<<'\n';         //使用std的元素cout
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法三：
#include<iostream>
int main()
{ int a, b;
  std::cin>>a;
  std::cin>>b;
  std::cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法三：
#include<iostream>
int main()
{ int a, b;
  std::cin>>a;                    //指定使用std的元素cin
  std::cin>>b;
  std::cout<<"a+b="<<a+b<<'\n';
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法三：
#include<iostream>
int main()
{ int a, b;
  std::cin>>a;                           //指定使用std的元素cin
  std::cin>>b;
  std::cout<<"a+b="<<a+b<<'\n';          //指定使用std的元素cout
}
```

3.8.1 标准名空间

使用标准名空间

```
//方法三：
#include<iostream>
int main()
{ int a, b;
  std::cin>>a;                          //指定使用std的元素cin
  std::cin>>b;
  std::cout<<"a+b="<<a+b<<'\n';    //指定使用std的元素cout
}
```

3.8.2 定义名空间

定义命名空间语法：

namespace <标识符>
{ <语句序列> }

```
namespace A
 { void f();
   void g();
 }
namespace B
 { void h();
   namespace C          //嵌套命名空间
    { void i();
    }
 }
namespace A    //为namespace A追加说明
 { void j();
 }
```

3.8.3　使用名空间

使用命名空间语法：

using namespace 名空间 ;

或

using 名空间::元素 ;

//例3-28 演示命名空间的使用

```
#include<iostream>
using namespace std;
namespace A
{ void f()
  { cout << "f() : fron
 void g()
  { cout << "g() :
 namespace B
  { void f()
     { cout << "f() : from namespace B\n" ; }
       namespace C
        { void f()
           { cout << "f() : from namespace C\n" ;
           }
        }
   }
}
void g()
{ cout << "g() : from global namespace“
        << endl ;
}
```

```
int main()
{ g() ;
   using namespace A;
   f() ;
   B::f() ;
   B::C::f() ;
   A::g() ;
}
```

调用非命名空间

使用命名空间A

调用命名空间

调用函数A::B::f()

调用函数A::B::C::f()

调用函数A::g()

3.9　终止程序执行

1．abort函数

函数原型：**void abort (void) ;**

功能：中断程序的执行，返回C++系统

2．assert函数

函数原型：**void assert (int *expression*) ;**

功能：若*expression*的值为false，中断程序的执行，　　　　　　行所在文件和程序行，返回C++系统。在assert声明。

这些中断语句

只应该出现在程序调试过程

3．exit函数

函数原型：**void exit (int *status*) ;**

功能：中断程序的执行，返回退出代码，回到C++系统。在stdlib声明。其中退出代码*status*是整型常量，返回操作系统，C++看不到exit的返回值。

小结

- 函数的作用是程序的功能划分和代码重用。
- 函数的参数是函数与外部通信的接口。形式参数与实际参数有三种传递方式：传值参数、指针参数和引用参数。
- return语句可以通过匿名对象使函数返回一个表达式的值。
- 内联函数是为减少调用开销的小程序。重载函数是名字相同，实现版本不同的函数。
- 函数可以用语句或表达式调用。已经定义的函数可以互相调用，可以递归调用。main函数是程序的启动函数。
- 调用一个函数需要的信息包括：函数地址和对应的实际参数。
- C++程序可以由多个程序文件构成。
- 标识符有特定的存储特性和作用域。一个结构性好的程序，应该遵循最低权限访问的原则，尽量不要使用全局变量。

规则与建议

C++程序通常分为两类文件：

头文件（.h）——保存程序的声明（**declaretion**）；

定义文件（.cpp）——保存程序的实现。

规则与建议

1、头文件的结构

（1）版本说明

（2）预处理块

（3）函数和类结构说明

【规则1-1】用#include<filename>引用标准库头文件。编译器将从标准库目录开始搜索。

【规则1-2】用#include"filename.h"引用非标准的头文件。编译器将从用户的工作目录开始搜索。

【规则1-3】头文件中只存放“声明”，不存放“定义”

【规则1-4】为防止头文件的定义内容被重复引用，使用 ifndef/define/endif 结构产生预处理块。

【规则1-5】不提倡使用全局变量，尽量不要在头文件中出现如

```
extern int value;
```

这样的说明。

规则与建议

2、定义文件的结构

（1）版本说明、功能说明

（2）对头文件的引用

（3）程序的实现体，包括数据和代码

规则与建议

3、代码书写

代码书写的良好风格，是阅读、调试程序的基础

【规则2-1】用空行分隔逻辑块。如类、函数、语句功能块。

【规则2-2】一行代码只做一件事情。便于阅读和跟踪。

【规则2-3】一行代码不超过80个字符，便于打印。

【建议2-3】尽量在变量说明的同时初始化。

【规则2-4】程序分界符大括号“{”与匹配的反括号“}”独占一行，并且位于同一列对齐。括号对{ }中的语句按逻辑以缩进格式书写。

【规则2-5】注释准确简洁易懂。注释放在代码的上方或右方。

【规则2-6】修改代码的同时修改注释。

规则与建议

4、函数设计

【建议4-1】函数功能要单一，不要设计多用途的函数。

【建议4-2】函数体规模要小，尽量控制在50行代码之内。

【建议4-3】函数原型书写形式参数名可以增加可读性。

【规则4-4】对不需要修改的指针参数和引用参数用const约束。

【规则4-5】默认参数只能从后向前挨个设置。

【规则4-6】不要使用参数传递的隐式类型转换处理数据。

【规则4-7】函数类型和函数返回类型是两个不同的概念。

【建议4-8】求值算术函数使用传值参数，用return返回计算值。

【建议4-9】功能性函数用return返回错误信息。

【规则4-10】名字相同，参数不同（包括类型、顺序不同）的函数才是重载函数。仅仅返回值类型不同则错误。

【建议4-11】尽量用内联函数取代宏代码，提高程序执行效率。

规则与建议

5、程序效率

程序的时间效率是指运行速度；

空间效率是指程序占用内存或外存的状况；

全局效率是指站在整个系统的角度上考虑的效率；

局部效率是指站在模块或函数的角度上考虑的效率。

【规则5-1】在满足正确性、可靠性、健壮性及可读性等程序质量的因素下，设法提高程序的效率。

【规则5-2】以提高全局效率为主，提供局部效率为辅。

【规则5-3】找出限制效率的“瓶颈”。

【规则5-4】先优化数据结构和算法，再优化代码。

【规则5-5】在时间优化和空间优化上做出平衡。

规则与建议

6、一些有益的建议

【建议6-1】当心视觉上不易分辨的操作符书写错误。例如：

== 和 =　　||　　&&　　<=　　>=

【建议6-2】变量（指针、数组）被创建后及时初始化，防止把未初始化的变量作为右值使用。

【建议6-3】当心变量初值、默认值错误，或精度不够。

【建议6-4】当心数据类型转换发生错误，尽量使用显式类型转换。

【建议6-5】当心变量发生上溢、下溢，以及数组下标越界。

【建议6-6】当心忘记编写错误处理程序，或者错误处理程序本身有误。

【建议6-7】当心文件I/O有错误。

【建议6-8】避免编写技巧性很高的代码。

【建议6-9】不要设计面面俱到、非常灵活的数据结构。

【建议6-10】重用高质量的代码；重写质量差的代码（不要修补）。

【建议6-11】尽量使用标准库函数，不要“发明”已经存在的库函数。