函数的调用与返回 (1)


```
class A {
    char *p;
public:
    A( ) {  p = new char [10]; }
    ~A( ) { if(p) delete p; p = 0; }
} a;
int f(A a)  { return 0; }
int main( ) {  f(a); }
```

程序崩溃！

程序有问题吗?

函数的调用与返回 (2)

调用子程序时, 编译器在堆栈中构建所需要的参数。

- 如果参数是指针 (引用)类型，则将实参的地址拷贝到堆栈。
- 如果参数是简单类型的变量，则将实参的值拷贝到堆栈。
- 如果参数是对象 (例如 A a)，则调用 A(const A &) 在堆栈中构建对象，该临时对象在失去作用范围时将被析构。

函数的调用与返回 (3)


```
struct A {
    A() { cout << "A() "; }
    A(const A &a) { cout << "A(a) "; }
    ~A() { cout << "~A() "; }
};
int f(A *a) { return 0; }
int g(A &a) { return 0; }
int h(A a)  { return 0; }
int main()
{
    A  a;
    A b = a;
    f(&a);
    g(a);
    h(a);
}
```

```
int main()
{
    A  a;        //A()
    A b = a;     //A(a)
    f(&a);       //
    g(a);        //
    h(a);        //A(a), ~A()
}                //~A(), ~A()
```

如果去除 A(const A &)，将怎样？

```
struct A {
    A() { cout << "A() "; }
   ~A() { cout << "~A() "; }
};
int main()
{
    A  a;        //A()
    A b = a;     //
    f(&a);       //
    g(a);        //
    h(a);        //~A()
}                //~A(), ~A()
```

函数的调用与返回 (4)

子程序返回机制：

- 如果返回类型是void，则不做任何事情。
- 如果返回类型是指针（引用）、简单的整型类型(char、short、unsigned int等)，则地址、简单的整型类型的值保存到EAX(AX、AL)寄存器中。主程序从EAX（AX、AL)获取返回值。
- 如果返回类型是简单的非整型类型（float、double等)，则在堆栈中构建一个临时存贮单元以存贮返回值，主程序从该存贮单元读取返回值然后释放该存贮单元。
- 如果返回1个对象 (例如 A a)，则调用 A(const A &) 在堆栈中构建1个临时对象。主程序调用 operator=(const A &)将该临时对象赋值给变量，然后析构该临时对象。

函数的调用与返回 (5)


```
struct A {
    A() { cout << "A() "; }
    A(const A &a) { cout << "A(a) "; }
    ~A() { cout << "~A() "; }
    A &operator=(const A &a) {
        cout << "=() "; return *this; }
};
int f(A *a) { return 0; }
int g(A &a) { return 0; }
A h(A a)  { return a; }
```

```
int main()
{
    A  a;
    A  b = a;
    f(&a);
    g(a);
    b = h(a);
    A  c = h(a);
}
```

```
int main()
{
    A  a;          //A()
    A  b = a;      //A(a)
    f(&a);         //
    g(a);          //
    b = h(a);      //A(a), A(a), ~A(), =(), ~A()
    A c = h(a);    //A(a), A(a), ~A()
}                  //~A(), ~A(), ~A()
```

函数的调用与返回 (6)


```
class A {
    int  i, j, k;
public:
    A(int x) {  i = x;
                cout << "Ad" << i;
             }
    A(const A &a) {  i = a.i;
                     cout << "Aa" << i;
                  }
    ~A() { cout << "~A" << i; }
    friend A abs(A a);
};

A  abs(A a)
{
   A  b( a.i < 0? -a.i : a.i );
   return b;
}
```

```
int main( )
{
    A a(-1), b(2);
    b = abs(a);
    return 0;
}
```

```
A a(-1), b(2);  //Ad-1 Ad2
b = abs(a);     //Aa-1 Ad1 Aa1 ~A1 ~A-1 ~A1
return 0;       //~A2 ~A-1
```

函数的调用与返回 (7)


```
class A {
    int  i, j, k;
public:
    A(int x) {  i = x;
                cout << "Ad" << i;
             }
    A(const A &a) {  i = a.i;
                     cout << "Aa" << i;
                  }
    ~A() { cout << "~A" << i; }
    friend A abs(A &a);
};

A  abs(A &a)
{
   A  b( a.i < 0? -a.i : a.i );
   return b;
}
```

```
int main( )
{
    A a(-1), b(2);
    b = abs(a);
    return 0;
}
```

```
A a(-1), b(2);  //Ad-1Ad2
b = abs(a);     //Ad1Aa1~A1~A1
return 0;       //~A2~A-1
```

函数的调用与返回 (8)

结论：

当一个函数的输入参数是对象 或者 返回一个对象时，若对象类中的实例成员函数中有申请内存的操作，则一定要定义 深构造 和 深拷贝（深赋值）的成员函数。