新标准C++程序设计

北京大学信息学院　郭 炜

GWPL@PKU.EDU.CN

类和对象(一)

面向对象语言的历史

Simula: 1967 年，Ole-Johan
发布Simula 67 提出了类（
概念,虽然实现并不是很完整
一个重要的里程碑。

Smalltalk: 1971 年， Xerox Pa
的Alan Kay 发明 Smalltalk。
对象的程序设计语言。

C++: 1979年，Bell实验室 Bjarn
刚开始的版本叫：C with Cla
时候正式命名为C++。

Java: 1995年，Sun 公司发布。支持跨平台可移植性、网络应用。

C#: 2002年1月，微软公司公布.NET Framework 1.0正式版。与此同时，Visual Studio.NET 2002也同步发行。

C++语言的历史

1983年8月， 第一个C++实现投入使用。
1983年12月，Rick Mascitti建议命名为CPlusPlus，即C++。
1985年10月，Bjarne博士完成了经典巨著The C++ Programming Language第一版。
1991年6月， The C++ Programming Language第二版完成。
1994年8月， ANSI/ISO委员会草案登记。
1997年7月， The C++ Programming Language第三版完成。
1997年10月，ISO标准通过表决被接受。
1998年11月，ISO标准被批准。

*历史上，**C++**编译器的“第一个”：*

1985年10月，Cfront Release 1.0发布。
1987年12月，GNU C++发布。
1988年1 月，第一个Oregon Software C++发布。
1998年6 月，第一个Zortech C++发布。
1990年5 月，第一个Borland C++发布。
1992年2 月，第一个Dec C++发布。
1992年3 月，第一个Microsoft C++发布。
1992年5 月，第一个IBM C++发布。

1994年，ANSI C++的标准发布。

1998年，ANSI和ISO标准委员会联合发布了至今最为广泛使用的C++标准，称为“C++98”。“C++98”的最重大改进就是加入了“标准模板库”（Standard Template Library,简称STL）,使得“泛型程序设计”成为C++除“面向对象”外的另一主要特点。

2003年，ISO的C++标准委员会又对C++略做了一些修订，发布了“C++03”标准。“C++03”和“C++98”的区别对大多数程序员来说可以不必关心。

2005年，一份名为“Library Technical Report 1”（简称TR1）的技术报告发布，加入了正则表达式、哈希表等重要类模板。虽然TR1当时没有正式成为C++标准，但如今的许多C++编译器都已经支持TR1中的特性。2011年9月，ISO标准委员会通过了新的C++标准，这就是“C++11”。“C++11”在酝酿的过程中，被称为“C++0x”，因为Bjarne Stroustrup原本预计它应该在2008年或2009年发布。

“C++11”刚通过不久，所以目前还没有编译器完全支持它。当前比较流行的C++编译器有GCC，微软的Visual C++ 10.0(包含在Visual Studio 2010中)、Dev C++、IBM的Eclipse、Borland C++Builder等。

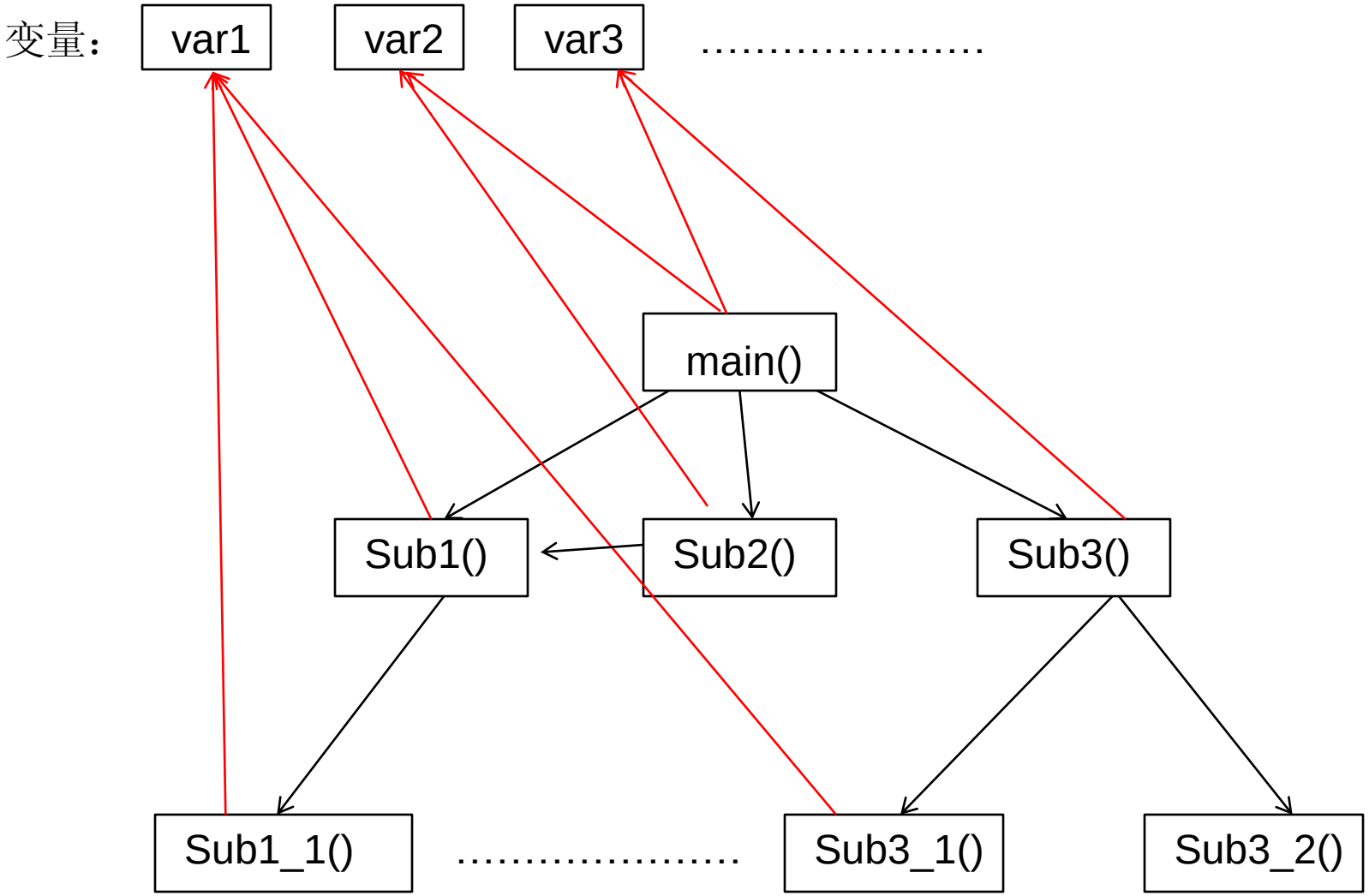
结构化程序设计

- C语言使用结构化程序设计：

  程序 = 数据结构 + 算法

  - 程序由全局变量以及众多相互调用的函数组成。
  - 算法以函数的形式实现，用于对数据结构进行操作。

结构化程序设计的程序模式：



结构化程序设计

- 结构化程序设计中，函数和其所操作的数据结构，没有直观的联系。
- 随着程序规模的增加，程序逐渐难以理解，很难一下子看出来：
  - 某个数据结构到底有哪些函数可以对它进行操作？
  - 某个函数到底是用来操作哪些数据结构的？
  - 任何两个函数之间存在怎样的调用关系？

结构化程序设计

- 结构化程序设计没有“封装”和“隐藏”的概念。要访问某个数据结构中的某个变量，就可以直接访问，那么当该变量的定义有改动的时候，就要把所有访问该变量的语句找出来修改，十分不利于程序的维护、扩充。
- 难以查错，当某个数据结构的值不正确时，难以找出到底是那个函数导致的。

结构化程序设计

- 重用：在编写某个程序时，发现其需要的某项功能，在现有的某个程序里已经有了相同或类似的实现，那么自然希望能够将那部分代码抽取出来，在新程序中使用。
- 在结构化程序设计中，随着程序规模的增大，由于程序大量函数、变量之间的关系错综复杂，要抽取这部分代码，会变得十分困难。

结构化程序设计

- 总之，结构化的程序，在规模庞大时，会变得难以理解，难以扩充（增加新功能），难以查错，难以重用。

结构化程序设计

- 软件业的目标是更快、更正确、更经济地建立软件。
  - 如何更高效地实现函数的复用？
  - 如何更清晰的实现变量和函数的关系？使得程序更清晰更易于修改和维护。

面向对象的程序设计

- 面向对象的程序设计方法，能够较好解决上述问题。

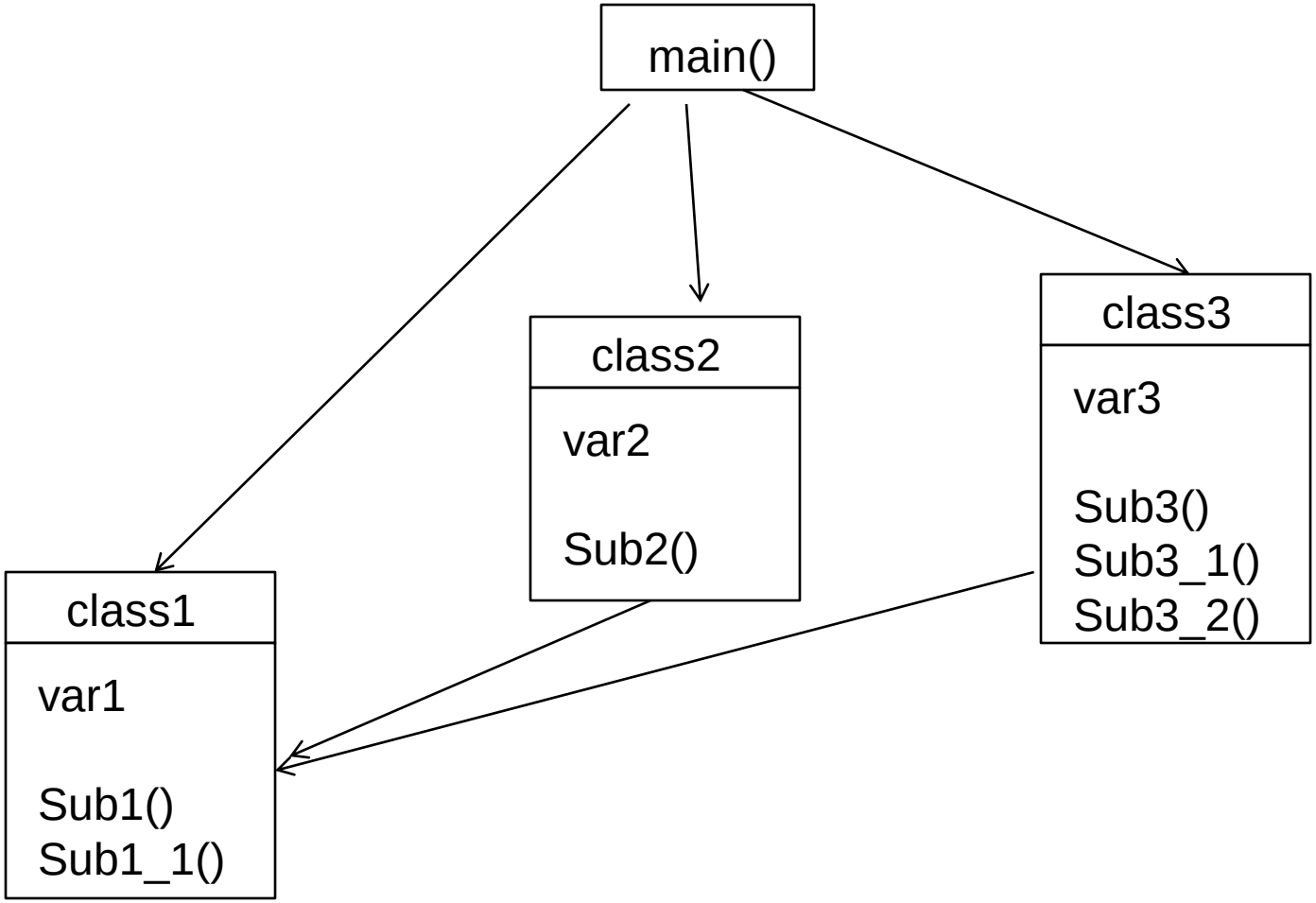
  面向对象的程序 = 类 + 类 + …+ 类

- 设计程序的过程，就是设计类的过程。

面向对象的程序设计

- 面向对象的程序设计方法：
  - 将某类客观事物共同特点（属性）归纳出来，形成一个数据结构（可以用多个变量描述事物的属性）；
  - 将这类事物所能进行的行为也归纳出来，形成一个个函数，这些函数可以用来操作数据结构(这一步叫“抽象”）。
- 然后，通过某种语法形式，将数据结构和操作该数据结构的函数“捆绑”在一起，形成一个“类”，从而使得数据结构和操作该数据结构的算法呈现出显而易见的紧密关系，这就是“封装”。

面向对象的程序模式：


main()
class2
var2
Sub2()
class3
var3
Sub3()
Sub3_1()
Sub3_2()
class1
var1
Sub1()
Sub1_1()

从客观事物抽象出类的例子

- 写一个程序，输入矩形的长和宽，输出面积和周长。
- 比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢？
- 矩形的属性就是长和宽。因此需要两个变量，分别代表长和宽。
- 一个矩形，可以有哪些行为呢（或可以对矩形进行哪些操作）？
  - 矩形可以有设置长和宽，算面积，和算周长这三种行为（当然也可以有其他行为）。
  - 这三种行为，可以各用一个函数来实现，他们都需要用到长和宽这两个变量。

从客观事物抽象出类的例子

- 将长、宽变量和设置长，宽，求面积，以及求周长的三个函数“封装”在一起，就能形成一个“矩形类”。
- 长、宽变量成为该“矩形类”的“成员变量”，三个函数成为该类的“成员函数”。成员变量和成员函数统称为类的成员。
- 实际上，“类”看上去就像“带函数的结构”。

从客观事物抽象出类的例子

```
class CRectangle
{
        public:
                int w, h;
                int Area() {
                        return w * h;
                }
                int Perimeter(){
                        return 2 * ( w + h);
                }
                void Init( int w_,int h_ ) {
                        w = w_;  h = h_;
                }
};//必须有分号
```

从客观事物抽象出类的例子

```
int main( )
{
        int w,h;
        CRectangle  r;  //r是一个对象
        cin >> w >> h;
        r.Init( w,h);
        cout << r.Area() << endl << r. Perimeter();
        return 0;
}
```

- 通过类，可以定义变量。类定义出来的变量，也称为类的实例，就是我们所说的“对象”。
- C++中，类的名字就是用户自定义的类型的名字。可以象使用基本类型那样来使用它。CRectangle 就是一种用户自定义的类型。

对象的内存分配

- 和结构变量一样，对象所占用的内存空间的大小，等于所有成员变量的大小之和。

- 每个对象各有自己的存储空间。一个对象的某个成员变量被改变了，不会影响到另一个对象。

如何使用类的成员变量和成员函数

用法1：对象名.成员名

```
CRectangle r1,r2;
r1.w = 5;
r2.Init(5,4);
```

- Init函数作用在 r2 上，即Init函数执行期间访问的 w 和 h是属于 r2 这个对象的, 执行r2.Init 不会影响到 r1。

用法2. 指针->成员名

```
CRectangle  r1,r2;
CRectangle * p1 = & r1;
CRectangle * p2 = & r2;
p1->w = 5;
p2->Init(5,4);   //Init作用在p2指向的对象上
```

如何使用类的成员变量和成员函数

用法3：引用名.成员名

```
CRectangle r2;
CRectangle & rr = r2;
rr.w = 5;
rr.Init(5,4);     //rr的值变了，r2的值也变

void PrintRectangle(CRectangle & r)
{
    cout << r.Area() << ","<<  r.Perimeter();
}
CRectangle r3;
r3.Init(5,4);
PrintRectangle(r3);
```

引用的概念

类型名 & 引用名 = 某变量名;

➢此种写法就定义了一个引用，并将其初始化为引用某个变量。

➢某个变量的引用，和这个变量是一回事，相当于该变量的一个别名。

```
int n = 4;
int & r = n;
r = 4;
cout << r;  //输出 4
cout << n; //输出 4
n = 5;
cout << r;  //输出5
```

➢定义引用时一定要将其初始化成引用某个变量，不初始化编译不过。

➢初始化后，它就一直引用该变量，不会再引用别的变量了

➢引用只能引用变量，不能引用常量和表达式。

引用的作用

➢C语言中，要写交换两个整型变量值的函数，只能通过指针。

```
void swap( int * a, int * b)
{
      int tmp;
      tmp = * a; * a = * b; * b = tmp;
}
int n1, n2;
swap(& n1,& n2) ; // n1,n2的值被交换
```

引用的作用

➢有了引用后：

```
void swap( int & a, int & b)
{
      int tmp;
      tmp = a; a = b; b = tmp;
}
int n1, n2;
swap(n1,n2) ; // n1,n2的值被交换
```

引用作为函数的返回值

➢函数的返回值可以是引用,如:

```
#include <iostream>
using namespace std;
int n = 4;
int & SetValue()  { return n;  }
int main()
{
     SetValue() = 40;
     cout << n;
     return 0;
}
```

该程序输出结果是 40

类的成员函数的另一种写法

➢成员函数体和类的定义分开写。

```
class CRectangle
{
        public:
                int w,h;
                int Area();     //成员函数仅在此处声明
                int Perimeter() ;
                void Init( int w_,int h_ );
};
```

类的成员函数的另一种写法


```
int CRectangle::Area()  {
        return w * h;
}
int CRectangle::Perimeter()  {
        return 2 * ( w + h);
}
void CRectangle::Init( int w_,int h_ ) {
        w = w_;  h = h_;
}
```

- CRectangle::说明后面的函数是CRectangle类的成员函数，而非普通函数。那么，一定要通过对象或对象的指针或对象的引用才能调用。

对象成员的访问权限

- 在类的定义中，用下列权限关键字来说明对象成员的访问权限：
  - private: 私有成员，只能在成员函数内访问
  - public : 公有成员，可以在任何地方访问
  - protected: 保护成员，以后再说
- 以上三种关键字出现的次数和先后次序都没有限制。

对象成员的访问权限

- 定义一个类

```
class className  {
   private:
     私有属性和函数
   public:
     公有属性和函数
   protected:
     保护属性和函数
};
```

说明类成员的可见性

- 如过某个成员前面没有上述关键字，则缺省地被认为是私有成员。

```
class  Man {
   int nAge; //私有成员
   char szName[20]; // 私有成员
public:
   void SetName(char * szName){
     strcpy( Man::szName,szName);
   }
};
```

对象成员的访问权限

- 在类的成员函数内部，能够访问：
  - 当前对象的全部属性、函数；
  - 同类其它对象的全部属性、函数。
- 在类的成员函数以外的地方，只能够访问该类对象的公有成员。

```
#include <iostream>
#include <cstring>
using namespace std;
class CEmployee  {
    private:
            char szName[30]; //名字
    public :
            int salary; //工资
            void setName(char * name);
            void getName(char * name);
          void averageSalary(CEmployee e1,CEmployee e2);
};
```

```
void CEmployee::setName( char * name)  {
    strcpy( szName, name); //ok
}
void CEmployee::getName( char * name)  {
        strcpy( name,szName); //ok
}
void CEmployee::averageSalary(CEmployee
e1,CEmployee e2)
{
        salary = (e1.salary + e2.salary )/2;
}
```

```
int main()
{
    CEmployee e;
    strcpy(e.szName,"Tom1234567889"); //编译错，不能访问私有成员
    e.setName( "Tom"); // ok
    e.salary = 5000;   //ok
    return 0;
}
```

设置私有成员的目的是强制对成员变量的访问一定要通过成员函数进行，那么以后成员变量的类型等属性修改后，只需要更改成员函数即可。否则，所有直接访问成员变量的语句都需要修改。

➢设置私有成员的机制，叫“隐藏”

- 如果将上面的程序移植到内存空间紧张的手持设备上，希望将 szName 改为 char szName[5]，若szName不是私有，那么就要找出所有类似

```
strcpy(e.szName,"Tom1234567889");
```

这样的语句进行修改，以防止数组越界。这样做很麻烦。

- 如果将szName变为私有，那么程序中就不可能出现（除非在类的内部）

```
strcpy(e.szName,"Tom1234567889");
```

这样的语句，所有对 szName的访问都是通过成员函数来进行，比如：

```
e.setName( “Tom12345678909887”);
```

- 那么，就算szName改短了，上面的语句也不需要找出来修改，只要改 setName成员函数，在里面确保不越界就可以了。

函数重载

- 一个或多个函数，名字相同，然而参数个数或参数类型互不相同，这叫做函数的重载。
- 如：

```
int Max(double f1,double f2) {
}
int Max(int n1,int n2) {
}
int Max(int n1,int n2,int n3) {
}
```

- 函数重载使得函数命名变得简单。

函数的缺省参数：

- C++中，写函数的时候可以让参数有缺省值，那么调用函数的时候，若不写参数，参数就是缺省值。

```
void func( int x1 = 2, int x2 = 3) { }

func( ) ; //等效于 func(2,3)
func(8) ; //等效于 func(8,3)
func(, 8) ; //不行
```

- 函数参数可缺省的目的在于提高程序的可扩充性。
- 即如果某个写好的函数要添加新的参数，而原先那些调用该函数的语句，未必需要使用新增的参数，那么为了避免对原先那些函数调用语句的修改，就可以使用缺省参数。

```
void func( int x1 = 2, int x2 = 3){
   cout<<x1+x2<<endl;
}
void testFunc(){
   int a=1; int b=1;
   func(a,b);
}

void func( int x1 = 2, int x2 = 3, int x3 = 3){
   cout<<x1+x2<<endl;
   int k=x3;
}
```

成员函数的重载及参数缺省

- 成员函数也可以重载（普通函数也可以）；
- 成员函数和构造函数可以带缺省参数（普通函数也可以）。

```
#include <iostream>
using namespace std;
class Location {
   private :
       int x, y;
  public:
      void init( int x=0 , int y = 0 );
      void valueX( int val ) { x = val ;}
      int valueX() { return x; }
};
void Location::init( int X, int Y)
{
  x = X;
  y = Y;
}
```

```
int main() {
    Location A,B;
    A.init(5);
    A.valueX(5);
    cout << A.valueX();
    return 0;
}
```

➢ 使用缺省参数要注意避免有函数重载时的二义性。

```
class Location {
     private :
            int x, y;
    public:
       void init( int x =0, int y = 0 );
       void valueX( int val = 0) { x = val; }
       int valueX() { return x; }
};
```

```
Location A;
A.valueX();          //错误，编译器无法判断调用哪个valueX
```

内联成员函数

在成员函数前面加上inline关键字后，成员函数就成为内联的成员函数。如果不写inline关键字，而是将整个成员函数的函数体写在类定义里面，那么该成员函数也会自动成为内联成员函数。下面的class B中的func1和func2都是内联成员函数：

```
class B
{
  inline void func1();
  void func2()
  {
  };
};
void B::func1 () {   }
```

内联成员函数

全局函数前面也可以加inline关键字，成为内联函数

内联函数在被编译时，不会生成函数调用的指令，而是将函数体的指令拷贝到调用语句处

好处是加快了函数执行的速度，坏处是增大了可执行程序的体积