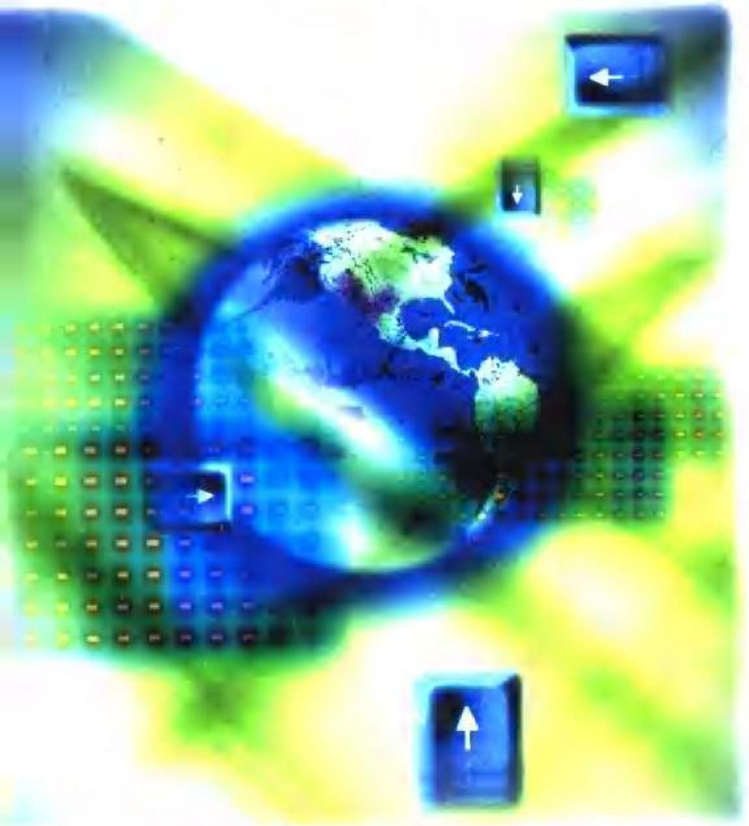
C++程序员首选参考手册

CD-ROM中包含所有范例程序的源代码

C++高级参考手册

C++ Master Reference

权威性 C++参考手册!



- 把核心语言和最常用的函数（类）库集于一体
- 所有条目按A～Z顺序编排并附范例代码，查找答案快捷方便
- 基于最新的技术规范，与ANSI C++全面兼容

[美]Clayton Walnum 著
苏帅华 陈维义 赵卫滨 等译
郝志恒 审校

IDG BOOKS WORLDWIDE

PHEI

電子工業出版社
Publishing House Of Electronics Industry
URL:http://www.phei.com.cn

C++高级参考手册

[美] Clayton Walnum 著

苏帅华 陈维义 赵卫滨 等译

郝志恒 审校

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是C++语言参考手册，按字母顺序介绍了C++语言的核心和最常用的类库及函数库，内容涵盖了C++的关键字、函数、运算符、类、概念和技术细节。本书解释细致易懂、范例程序简短明了。初学者和有经验的C++程序员都能够从本书中快速地找到所需问题的答案。

图书在版编目(CIP)数据

C++高级参考手册 /(美) 沃尔纳(Walnum,C.)著；
苏帅华译 .-北京：电子工业出版社，2000.1
书名原文：C++ Master Reference
ISBN 7-5053-5575-9
Ⅰ.C… Ⅱ.①沃…②苏… Ⅲ.C语言-程序设计-手册 Ⅳ.TP312.62
中国版本图书馆 CIP 数据核字(1999) 第 70010 号

书　　名：**C++高级参考手册**
著　　者：[美]Clayton Walunm
译　　者：苏帅华　陈维义　赵卫滨　等
审 校 者：郝志恒
责任编辑：嘉　益
印 刷 者：北京天竺颖华印刷厂
出版发行：电子工业出版社　　URL：http：//www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店经销
开　　本：787×1092　1 /16　　印张：53.75　字数：1548 千字
版　　次：2000 年 1 月第 1 版　　2000 年 1 月第 1 次印刷
书　　号：ISBN 7-5053-5575-9 / TP·2834　　著作权合同登记号：　图字：01-1999-2976
定　　价：108.00 元　(含光盘一张)

译 者 序

C++是C语言的超集,它不仅保持了C语言的表达能力强、目标代码效率高、可移植性好的特点,而且还提供了面向对象的程序设计能力。在硬件技术飞速发展的今天,人们对软件功能的要求也越来越高,利用面向对象的程序设计技术实现软件重用,是加速软件开发进程的根本途径。C++已经成为当今的主流程序设计语言。

总的来说,对C++语言的学习可分为两部分。一是学习C++语言本身,二是学习如何使用其丰富的类库和标准库。对C++语言学习的最困难之处是利用其丰富的功能和复杂的特点,灵活高效地开发实际软件系统,这需要大量的实践和学习。但是,即使对经验非常丰富的程序员来说,要记住C++中的全部内容(特别是类库和标准库)几乎是不可能的,也是没有必要的。虽然我们可以从C++编程环境中获得所需要的文档,但是这些海量文档查阅不便,而且许多内容几乎是永远也不会使用。本书正是根据实际程序设计需求而编写的,它按照字母顺序介绍了C++中最常用的函数、类和语法,为使读者快速领会所介绍的条目,每一个条目都用一个或多个简短且针对性强的范例来说明其用法。本书不是一本C++教程,但是可用作实际编程时的案头参考资料。

参与本书翻译的人员有苏帅华、陈维义、赵卫滨、左静、纪宁、谢一平、陈文、覃天、杨开开、严敏、文达、魏玉明、汪华、蒋晓军、单东淇、杨阳、马平初、半途、吴言、张燕燕、甘雨、李华、陈晓文,全书由郝志恒统稿。

由于时间关系和水平所限,疏漏不当之处在所难免,敬请广大读者批评指正。

译 者

1999年12月

关于作者

1982年，Clayton Walnum通过担当IBM Selectric公司的打字员挣得了一台只有16K内存的Atari 400计算机，从那时起他就开始从事计算机程序设计。Clay很快就学会把他的写作兴趣与其最新获得的程序设计技巧结合起来，并开始向计算机杂志投稿。1985年，全国性的计算机杂志《ANALOG Computing》聘请他担任技术编辑；在1989年离开该杂志前，他已经是一名自由撰稿人且升任到了执行主编。此后，他获得了计算机学学位，编写了40余本计算机图书，这些图书已经被翻译成多国语言，内容涉及计算机游戏编程到3D图形程序设计；他还在杂志上发表了数百篇文章和软件评述以及大量的程序。最近编写的书籍包括《Visual Basic 6 Master Reference》(IDG世界图书公司出版)和《Windows 98 Programming Secrets》(IDG世界图书公司出版)，其它图书包括《AFC Black Book，Special Edition Using MFC and ATL》、《Java By Example》以及获奖图书《Building Windows 95 Applications with Visual Basic》。Clay在生活中的最大遗憾是没有能够成为甲壳虫合唱队(Beatles)的一员；作为补偿，他在其家庭制作室内谱写并录制摇滚乐。访问他的主页www.claytonwalnum.com可与他取得联系。

前　言

C++ 是全球使用最广泛的程序设计语言之一。其成功的原因是显而易见的，即 C++ 是一门几乎具有汇编语言功能的高级语言（当然，具备这种功能也是有代价的）。尽管 C++ 语言的基本内容实际上并不多，但是掌握其复杂性要花上好几年的时间。而且，C++ 在其基本语言之外还定义了庞大的函数和类库集合，掌握这些函数和类库需要大量的实践。

一个程序员，无论他多么富有经验，都不可能靠记住全部所需的信息来编写 C++ 程序（就象我们不能把整个大西洋装入一个茶杯内），这就是我编写《C++ 编程大全》的原因。有了这本书，无需在浩如烟海的文档中苦苦搜寻，你就能够很快地找到大多数问题的答案。本书按字母顺序列出了关键字、函数、运算符、类、概念、技术以及其它方面的内容，所有这些内容都是按照易于阅读、查阅快速的原则来编排的。

读者对象

本书不是一本 C++ 教程（尽管某些 C++ 话题是以"微型"教程的形式来介绍的）。要想理解本书所介绍的内容，至少要对 C++ 的基本知识有所了解。如果对 C++ 还比较陌生，先应该挑选一本优秀的 C++ 编程书籍加以研读，如 Teach Yourself...C++（第 5 版，Al Stevens 编著，IDG 世界图书公司出版）；一旦掌握了该书中的内容（或类似教材中的内容），就可把本书作为主要参考手册。

本书内容

在开始计划本书内容的时候，我们原想使本书覆盖 C++ 的全部知识。但是，我们很快就发现，按照这种方式编写本书，其篇幅将加长一倍，而且还将包括数百页读者可能永远也用不到的深奥内容。

因此，在编写本书时，我们从核心语言入手来取舍全书的内容，既使本书保持适度的篇幅，又使其包含 C++ 语言中最重要的信息。在写完全书时，我们又添加了 C++ 语言的基本知识以及最常用的库和类。本书的条目类型有：

- 类 —— 包括对类的简要描述、类的声明和一张描述类成员的表。例如，exception、fstream 和 stdiostream 等等都属于这种类型的条目。
- 概念 —— 描述了 C++ 编程的概念，如抽象类、头文件和虚函数等等。
- 指令 —— 描述了编译指令并举例说明其用法，如 #include、#define 和 #if 等都属于这种类型的条目。
- 函数 —— 包括函数描述和函数声明，还包括演示函数用法的简短的范例程序。_wcsinc()、printf() 和 sin() 等都属于这种类型的条目。
- 关键字 —— 描述了 C 或 C++ 中的关键字，包括演示其用法的范例。
- 操纵算子 —— 包括对流操纵算子（如 dec、endl 和 hex）的描述和其用法举例。
- 运算符 —— 包含对运算符（如 &、-> 和 sizeof）的描述和其用法举例。
- C 标准库头文件 —— 包括文件描述和在该文件中定义的函数、常量、结构等等的符号。这种类型的条目有 float.h、stdlib.h、string.h 等等。
- C++ 标准库头文件 —— 包括对文件的简要描述以及在该文件中定义的符号（函数、常量、结构等等）。cfloat、cstdlib 和 cstring 等等都属于这类条目。
- 流 —— stdout、stdin 和 stderr 等等的流对象条目包括对流的描述和用法举例
- 流对象 —— cerr、cin 和 cout 等等的流对象条目中包括对该对象的描述和用法举例
- 技术 —— 该种类型的条目都是"微型"教程，描述了重要的 C++ 程序设计技术，如怎样创建类、多态性的用法和异常处理等等。
- 模板类 —— 包括对模板类的简要描述和一张描述类成员的表，例如 bitset、list 和 map 等都属于模板类。

除了上述的条目类型外，本书还包括注释符类型和宏类型，本书都对它作了解释并举例说明了其用法。

组织方式

所有的条目都按照字母顺序编排,因此可在本书中快速地找到所需的内容,例如要了解头文件stdlib只要在S部分查找就可以了,而在C部分可找到对函数cprintf()的介绍。

注意,本书第一章的条目都是非字母字符,如&、!和/*等等。其余的章节则按字母顺序编排;在这些章节中,以一个或多个下划线开始的条目(如_close()、_exit、__int16等等)和以英镑符开始的条目(如#define、#if和#include等等)总是放在相应章节的开始;也就是说,#define放在D章的开始,而_close()放在C章的开始位置。

理解程序清单

本书的大多数条目都配有演示该特定语言要素用法的简短程序。其中的程序清单是在Microsoft Visual C++上开发的。附带的光盘上包含了本书中所有程序的源代码,这些源代码有Visual C++和Borland C++ Builder两种版本。

注意,所有的VC++和BC++项目都是用开发环境的代码产生器(code-generator)来启动的。详情参见本书附录"CD-ROM中的内容"。

系统要求

本书中的程序对系统和软件的要求与所运行的系统以及所用的编译器的要求相同。

结束语

为使本书的内容尽可能地准确,我们已经做了大量的努力,但是如此巨大的工作量难免会有一些错误。因此,请读者对其中的错误予以谅解并通知我,非常感谢你的帮助。读者可以通过我的主页www.claytonwalnum.com与我联系。

'(直接字符常量标志)

运算符

单引号用于表示直接字符常量。

语法

```
'char'
```

■ char —— 代表单个字符的值。

举例

下例显示一行信息"C++",其后跟有几个新行符。程序的输出结果为:

```
C++
Press any key to continue
```

尽管新行符在程序中是用反斜杠和"n"这两个字符表示的,但是仍应把它们看作是单个字符。反斜杠仅充当转义序列。下面是产生该输出结果的程序源代码:

```
#include"stdafx.h"

int main(int argc, char * argv[])
{
    putchar('C');
    putchar('+');
    putchar(43);
    putchar('\n');
    putchar('\n');

    return 0;
}
```

相关内容

另见"(直接字符串常量标志)。

-(减法)

运算符

减法运算符用运算符左边的值减去运算符右边的值。

语法

```
result = value1 - value2;
```

■ result —— 运算结果
■ value1 —— 被减数
■ value2 —— 减数

举例

下面的程序演示了减法运算符的使用。程序的输出结果为:

```
result: 13
result: 15
Press any key to continue
```

程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 20;
    int val2 = 7;
```

```
    int result = val1 - val2;
    cout << "result: " << result << endl;
    result = val1 - 5;
    cout << "result: " << result << endl;

    return 0;
}
```

相关内容

另见 —(自减)、-=(减法赋值)、+(加法)、+=(加法赋值)、*(乘法)、*=(乘法赋值)、/(除法)和 /=(除法赋值)。

—(自减)

运算符

自减运算符把其操作数的值减 1。如果自减运算符放置在其操作数之前,那么运算符在程序语句访问该值之前执行减法运算,这时该运算符称为"前置自减运算符";如果自减运算符放置在其操作数之后,那么运算符在程序语句访问该值之后执行减法运算,这时该运算符称为"后置自减运算符"。

语法

```
—value

或
value—
```

■ value —— 被自减的值

举例

下面的程序用到了前置和后置自减运算符,并显示了这两种减法的运算结果。程序的输出结果为:

```
9
9
20
19
Press any key to continue
```

对于变量 val1 来说,程序在调用 cout 显示其值之前先把其值减 1,这时 val1 的值变为 9;第二次调用 cout 显示了自减操作后的 val1 的值。对于变量 val2 来说,第三次调用 cout 显示了 val2 在执行减法之前的值,即显示结果为 20;第四次调用 cout 再次显示了 val2 的值,这时候的 val2 的值为 19,证明它是执行第一次显示后自减后的值。程序的源代码如下:

```
    #include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int val1 = 10;
    int val2 = 20;

    cout << —val1 << endl;
    cout << val1 << endl;
    cout << val2— << endl;
    cout << val2 << endl;

    return 0;
}
```

相关内容

另见 ++(自增)。

-=(自减赋值)

运算符

自减赋值运算符用运算符左边的值减去运算符右边的值,然后把运算结果赋给左边的值。例如,语句 val1 -= 3 等价于 val1 = val1 - 3。

语法

```
var1 -= exp1;
```

■ exp1 —— 合法的表达式

■ var1 —— 在执行运算之前,变量 var1 包含

了减去 exp1 之前的值;执行运算之后,var1 包含了运算的结果

举例

下例显示了变量 val1 在自减 - 赋值运算前后的值。程序的输出结果为:

```
val1 = 12
val1 = 9
Press any key to continue
```

执行运算之前,val1 的值为 12。执行运算之后,val1 的值为 9。程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int val1 = 12;
    cout << "val1 = " << val1 << endl;
    val1 -= 3;
    cout << "val1 = " << val1 << endl;

    return 0;
}
```

相关内容

另见 %=(求余赋值)、&=(按位"与"赋值)、*=(乘法赋值)、/=(除法赋值)、^=(按位"异或"赋值)、|=(按位"或"赋值)、+=(加法赋值)、<<=(左移位赋值)、=(简单的赋值)和 >>=(右移位赋值)。

->(对象指针复引用)

运算符

对象指针复引用运算符复引用指向类实例的指针。

语法

```
objectPtr->member
```

- objectPtr —— 指向类实例的指针
- member —— 实例化类的成员

举例

下例演示了如何使用对象指针复引用运算符。程序的输出结果为:

```
Class value: 10
Press any key to continue
```

程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>
#include "MyClass.h"

using namespace std;

int main(int argc, char* argv[])
{
    MyClass* pMyClass = new MyClass(10);
    int value = pMyClass->GetValue();
    cout << "Class value: " << value <<
    endl;
    delete pMyClass;

return 0;
}

// Class header file.
class MyClass
{
protected:
    int value;

public:
    MyClass(int val);
    ~MyClass();
```

```
    int GetValue();
};

// Class implementation file.
#include "stdafx.h"
#include "MyClass.h"

MyClass::MyClass(int val)
{
    value = val;
}

MyClass::~MyClass()
{
}

int MyClass::GetValue()
{
    return value;
}
```

相关内容

另见 &(取地址)、&(引用)和 *(指针)。

! (逻辑"非")

运算符

逻辑"非"运算符反转布尔型表达式的值,也就是说,"真"值表达式变为"假","假"值表达式变为"真"。

语法

```
! exp
```

■ exp —— 其值将要被反转的布尔型表达式

举例

下例演示了逻辑"非"运算符的用法。程序的输出结果为:

```
result: 1
! result: 0
Press any key to continue
```

程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int x = 10;
    bool result = (x == 10);
    cout << "result: " << result << endl;
    cout << "! result: " << ! result <<
    endl;

    return 0;
}
```

相关内容

另见 bool(关键字)。

! =(不等于)

运算符

不等于运算符对两个值予以比较,如果这两个值相等,返回 1(真),否则返回 0(假)。

语法

```
result = exp1 ! = exp2;
```

■ exp1 —— 合法的表达式
■ exp2 —— 合法的表达式
■ result —— 值 1 或 0

举例

下面的程序用不相等运算符对两个值进行比较。程序的输出结果为:

```
The values are not equal
Press any key to continue
```

程序的源代码如下：

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 10;
    int val2 = 15;

    if (val1 ! = val2)
        cout << "The values are not equal" <<
        endl;
    else
        cout << "The values are equal" <<
        endl;

    return 0;
}
```

相关内容

另见 <(小于)、<=(小于或等于)、==(等于)、>(大于)和 >=(大于或等于)。

"(直接字符串常量标志)

运算符

双引号用于表示一个直接字符串常量。

语法

```
"str"
```

■ str —— 字符串

举例

如下的程序显示了一个跟有新行符的字符串。程序的输出结果为：

```
C++
Press any key to continue
```

程序的源代码如下：

```
#include "stdafx.h"
#include <stdio.h>

int main(int argc, char * argv[])
{
    printf("C++ \n\n");
    return 0;
}
```

相关内容

另见 '(直接字符常量标志)。

%(求余)

运算符

求余运算符用运算符左边的值除以运算符右边的值，返回除法运算的余数。

语法

```
result = exp1 % exp2;
```

■ exp1 —— 合法的表达式
■ exp2 —— 合法的表达式
■ result —— 求余运算的结果

举例

如下的程序显示了变量 result 在求余运算前后的值。程序的输出结果为：

```
result = 6
result = 0
Press any key to continue
```

在执行运算之前，val1 的值为 20。在 val2 执行了对 val2 的求余运算之后，result 的值为 6(20 /7 的余数)。第二次执行求余运算后，result 为 0(20 / 5 的余数)。程序的源代码如下：

```
#include "stdafx.h"
```

```
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 20;
    int val2 = 7;

    int result = val1 % val2;
    cout << "result: " << result << endl;
    result = val1 % 5;
    cout << "result: " << result << endl;

    return 0;
}
```

相关内容

另见 %=(求余赋值)、/(除法)和 /=(除法赋值)。

%=(求余赋值)

运算符

求余赋值运算符用运算符左边的值除以运算符右边的值,然后把除法运算的余数赋给左边的值。例如,语句 val1 %= 4 等价于 val1 = val1 % 4 。

语法

```
var1 %= exp2;
```

- exp2 —— 合法的表达式
- var1 —— 在执行除法运算之前,该变量包含了被除数的值;在执行除法运算之后,该变量包含了运算结果

举例

如下的程序显示了变量 val1 在求余赋值运算前后的值。程序的输出结果为:

```
val1 = 25
val1 = 1
Press any key to continue
```

在执行求余运算之前,val1 的值为 25。执行求余运算之后,val1 的值为 1(25 /4 的余数)。程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 25;
    cout << "val1 = " << val1 << endl;
    val1 %= 4;
    cout << "val1 = " << val1 << endl;

    return 0;
}
```

相关内容

另见 -=(减法赋值)、&=(按位"与"赋值)、*=(乘法赋值)、/=(除法赋值)^=(按位"异或"赋值)、|=(按位"或"赋值)、+=(加法赋值)、<<=(左移位赋值)、=(简单的赋值)和 >>=(右移位赋值)。

&(取地址)

运算符

取地址运算符返回对象(包括变量、函数和结构)在内存中的地址。

语法

```
&obj
```

- obj —— 要获取在内存中地址的对象

举例

如下的程序显示了变量的地址并用该地址修

改变量的值。程序的输出结果类似于：

```
val1 = 25
&val1 = 006AFDF4
val1 = 50
&val1 = 006AFDF4
Press any key to continue
```

在执行运算之前，val1 的值为 25。前两个 cout 调用显示了 val1 的当前值和其地址。程序然后把 val1 的值赋给一个指针变量，并引用该指针所指向的值来修改存储在 val1 中的值。第三和第四次调用 cout 显示了变量 val1 的新值和其地址(地址没有变化)。程序的源代码如下：

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 25;
    cout << "val1 = " << val1 << endl;
    cout << "&val1 = " << &val1 << endl;
    int * adr = &val1;
    * adr = 50;
    cout << "val1 = " << val1 << endl;
    cout << "&val1 = " << adr << endl;
    return 0;
}
```

相关内容

另见 ->(对象指针复引用)、&(引用)和 *(指针)。

&(按位“与”)

运算符

按位“与”运算符把运算符左边的值和右边的值进行按位“与”运算并返回运算结果。

语法

```
result = exp1 & exp2;
```

- exp1 —— 合法的表达式
- exp2 —— 合法的表达式
- result —— 按位“与”运算的结果

举例

下面的程序执行按位“与”运算。程序的输出结果为：

```
result: 0
result: 4
Press any key to continue
```

程序的源代码如下：

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    int val1 = 20;
    int val2 = 2;

    int result = val1 & val2;
    cout << "result: " << result << endl;
    result = val1 & 5;
    cout << "result: " << result << endl;
    return 0;
}
```

相关内容

另见 &=(按位“与”赋值)、^(按位“异或”)、^=(按位“异或”赋值)、|(按位“或”)和 |=(按位“或”赋值)。

&(引用)

运算符

引用运算符允许函数按引用方式而不是传值方式接受参数。随后对该引用参数的使用就好象该参数是调用函数和被调用函数之间的"全局"变量一样。也就是说,对该参数值的修改也会修改其原始数据项。参数的这种传递方法类似与向函数传递变量的地址(指向变量的指针),只不过函数中不能对引用参数使用复引用运算符 * (用指针方法传递的变量必须用运算符 * 来访问变量的内容)。

语法

```
&val1
```

■ val1 —— 按引用方式接受的参数

举例

如下的程序演示了变量的引用传递方式。程序的输出结果为:

```
var1 in main: 25
arg1 in MyFunc: 25
arg1 in MyFunc: 50
var1 in main: 50
Press any key to continue
```

程序在把 var1 传递给函数 MyFunc()之前首先显示了 var1 的值。函数 MyFunc()接受对 var1 的引用,该引用命名为 arg1。也就是说,var1 和 arg1 代表了内存中的同一个存储单元。函数 MyFunc()显示了 arg1 的值,显示结果表明 arg1 确实等于 var1。然后,该函数把 arg1 的值修改为 50,并在返回到 main()函数之前显示了修改后的值。最后,main()函数显示了 var1 的值,该值也被修改为 50。程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

void MyFunc(int &arg1)
{
    cout << "arg1 in MyFunc: " << arg1
    << endl;
    arg1 = 50;
    cout << "arg1 in MyFunc: " << arg1
    << endl;
}

int main(int argc, char * argv[])
{
    int var1 = 25;
    cout << "var1 in main: " << var1 << endl;
    MyFunc(var1);
    cout << "var1 in main: " << var1 << endl;

    return 0;
}
```

如下是等价的源代码,该源代码用指向 var1 的指针代替了引用。

```
#include "stdafx.h"
#include <iostream>

using namespace std;

void MyFunc(int * arg1)
{
    out << "arg1 in MyFunc: " << * arg1 <<
    endl;
    * arg1 = 50;
    cout << "arg1 in MyFunc: " << * arg1 <<
    endl;
}

int main(int argc, char * argv[])
{
    int var1 = 25;
    cout << "var1 in main: " << var1 << endl;
    MyFunc(&var1);
    cout << "var1 in main: " << var1 << endl;

    return 0;
}
```

相关内容

另见->(对象指针复引用)、&(引用)和*(指针)。

&&(逻辑“与”)

运算符

逻辑“与”运算符在运算符两边的表达式都为真时返回真,否则返回假。

语法

```
result = expr1 && expr2;
```

- expr1 —— 任何合法的表达式
- expr2 —— 任何合法的表达式
- result —— 布尔型值真(非0)或假(0)。

举例

下面的程序演示了逻辑“与”运算符的使用。程序的输出结果为:

```
TRUE
FALSE
TRUE
```

程序的源代码如下:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int var1 = 25;
    int var2 = 35;

    if ((var1 == 25) && (var2 == 35))
        cout << "TRUE" << endl;
    else
        cout << "FALSE" << endl;

    if ((var1 == 20) && (var2 == 35))
        cout << "TRUE" << endl;
    else
        cout << "FALSE" << endl;

    if ((var1) && (var2))
        cout << "TRUE" << endl;
    else
        cout << "FALSE" << endl;

    return 0;
}
```

相关内容

另见 ||(逻辑“或”)。

&=(按位“与”赋值)

运算符

按位“与”赋值运算符对运算符左右两边的值进行按位“与”运算并把运算结果赋给左边的值。例如,语句 val1 &= 3 等价于 val1 = val1 & 3 。

语法

```
var1 &= exp2;
```

- exp2 —— 合法的表达式
- var1 —— 在执行按位“与”运算之前,该变量包含了与 exp2 进行按位“与”运算前的值。在执行了按位“与”运算之后,var1 包含了运算的结果

举例

下面的程序显示了 val1 在进行按位“与”赋值运算前后的值。程序的输出结果为:

```
val1 = 27
val1 = 3
Press any key to continue
```

在执行运算之前,val1 的值为 27。执行运算

之后，val1 的值为 3。程序的源代码如下：

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int val1 = 27;
    cout << "val1 = " << val1 << endl;
    val1 &= 3;
    cout << "val1 = " << val1 << endl;

    return 0;
}
```

相关内容

另见 -=(减法赋值)、%=(求余赋值)、*=(乘法赋值)、/=(除法赋值)、^=(按位"异或"赋值)、|=(按位"或"赋值)、+=(加法赋值)、<<=(左移位赋值)、=(简单赋值)和 >>=(右移位赋值)。

*(乘法)

运算符

乘法运算符用运算符左边的值乘以运算符右边的值。

语法

```
result = value1 * value2;
```

- result —— 运算结果
- value1 —— 被乘数
- value2 —— 乘数

举例

下面的程序演示了乘法运算符的使用。程序的输出结果为：

```
result: 140
result: 100
Press any key to continue
```

程序的源代码如下：

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int val1 = 20;
    int val2 = 7;

    int result = val1 * val2;
    cout << "result: " << result << endl;
    result = val1 * 5;
    cout << "result: " << result << endl;

    return 0;
}
```

相关内容

另见 -(减法)、--(自减)、-=(减法赋值)、+(加法)、+=(加法赋值)、*=(乘法赋值)、/(除法)和 /=(除法赋值)。

*(指针)

运算符

指针运算符用来声明或复引用一个指针。

语法

```
type* var1

或
*var1
```

- type —— 指针所指向数据的数据类型
- var1 —— 代表指针的符号