第一部分　C 语言程序

A01　创建 Hello World 程序(简单的单文件程序)

本书中的程序并不是带有控件的 Windows 应用程序,而是所谓“工作台应用程序”(Console Application)。程序的创建过程包括程序文件(C / C++ 源文件、头文件)的编写、编译、连接和调试。整个过程在常用的 Visual C++ 6.0(简称 VC++)系统的 IDE(集成开发环境)中进行。

许多程序设计教科书都喜欢用 Hello World 程序与读者行见面礼。这里,我们照例首先创建这样一个非常简单的单文件程序,它只在屏幕上显示字符串“Hello World”。为此,要首先建立一个项目,然后在其中建立 C / C++ 源文件。

A01.1　创建项目

安装了 Visual C++ 6.0 之后,双击桌面上的 VC 快捷图标,便可进入 VC++ 集成开发环境,显示 VC++ 的 IDE 主界面,如图 A01-1 所示。

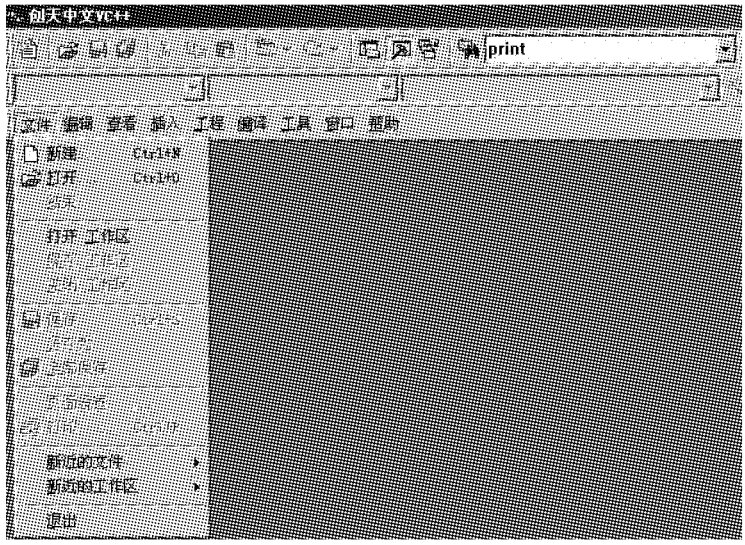


图 A01-1　VC++ 的 IDE 主界面

在菜单栏的“文件”菜单(如图 A01-1 所示)下,单击“新建”命令。在出现的对话框中,选中“工程”选项卡,如图 A01-2 所示。在该对话框的左边,选中 Win32 Console Application(32 位控制台应用程序)选项,而在右边填入路径(例如,C:\myC_CPP\A01)和项目名(例如,A0101)后,单击“确定”按钮。出现如图 A01-3 所示的对话框,询问要建立哪种控制台应用程序。这时,选中 An empty project 单选钮,单击“完成”按钮。其后,出现另一个信息框(如图 A01-4 所示)。确认后,就以 C:\myC_CPP\A01 为路径,建立了一个名为 A0101 的空项目(空目录)。以后有关该项目的所有文件,例如,项目文件、源文件、头文件等均被保存于此。

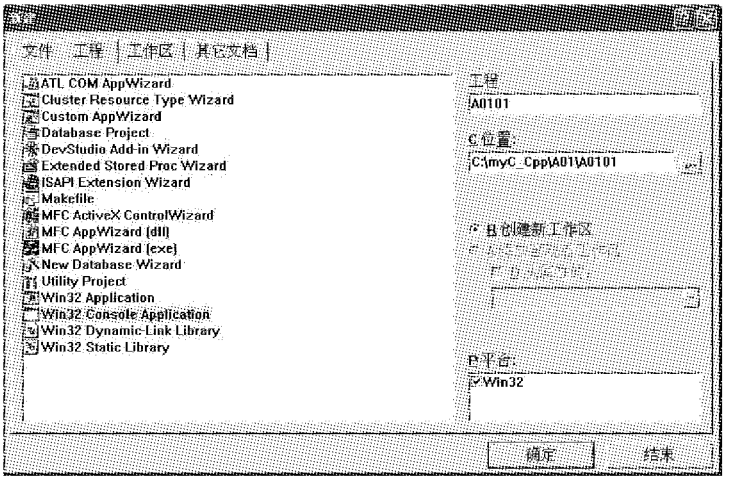


图 A01-2 建立项目

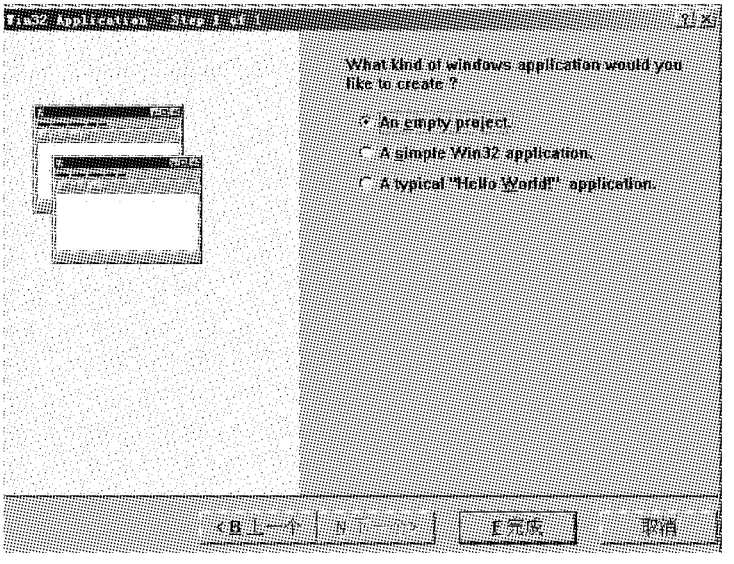


图 A01-3 询问框

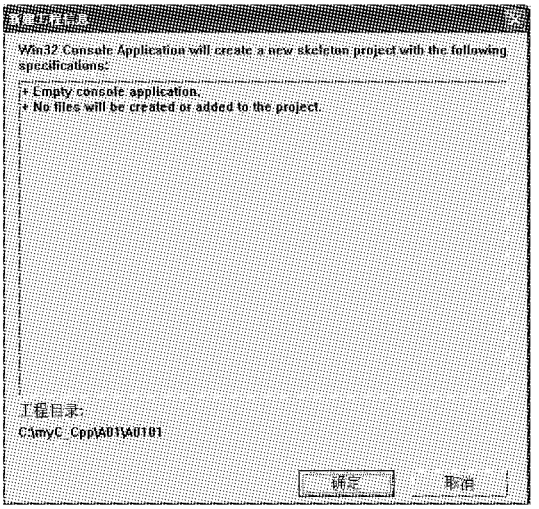


图 A01-4　信息框

A01.2　建立文件

按上述方法建立项目后，接着要在这个项目中建立 CPP 源文件。

在 IDE 中，单击“文件”|“新建”命令，便出现如图 A01-5 所示的对话框。如果要建立 CPP 源文件，应选中对话框中左边的 C ++ Source File 选项。对话框右边的第一栏是已建立的项目名称，第三栏是项目的路径。在第二栏输入文件名，例如，A0101（默认扩展名是.CPP）。确认后，便在项目中建立了空的源文件 A0101.CPP，并显示如图 A01-6 所示的界面。图 A01-6 中右边的区域出现闪烁的光标，这是程序编辑区。在该区域输入代码，得到源程序文件（A0101.CPP）。

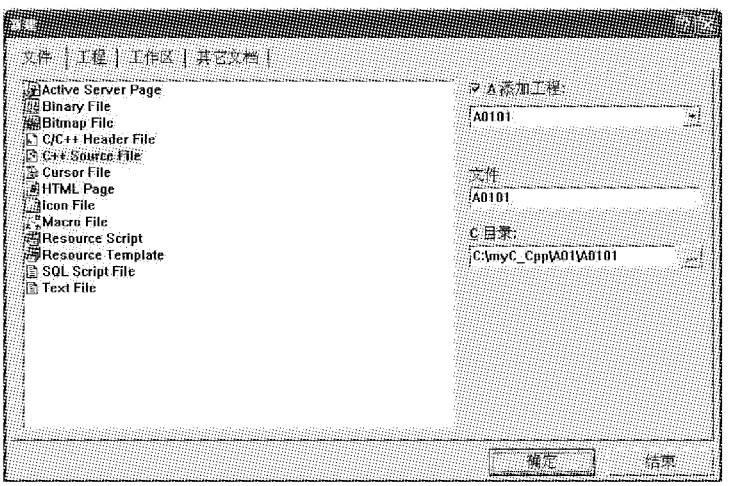


图 A01-5　“新建”对话框

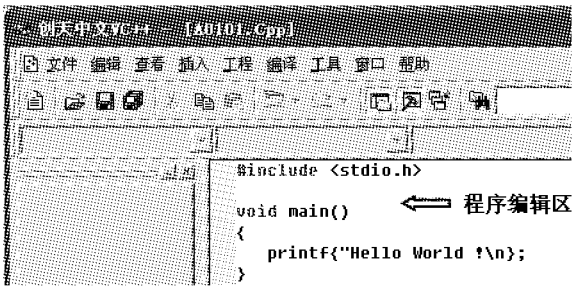


图 A01-6 在编辑区输入程序代码

重复上述操作,可建立其他 CPP 源文件。用上述方法也可以建立头文件,即在如图 A01-5 所示的对话框左边应选取 C / C++ Head File 项,其他步骤同上。头文件的默认扩展名是.H。

完成上述操作后,在指定的路径上生成一个与工程名字相同的目录,CPP 源文件位于其中。当然,CPP 源文件名字也可以有别于工程名字。

⚠ **注意**:所有 CPP 源文件中,只能有一个文件带有主函数 main()。

A01.3 编译

如果程序编辑区已出现待编译的主文件(包含函数 main() 的源文件),可用"编译"命令(如图 A01-7 所示)对其进行编译。否则,单击"文件"|"打开"命令,输入文件名,使待编译的主文件出现在编辑区中,然后进行编译。

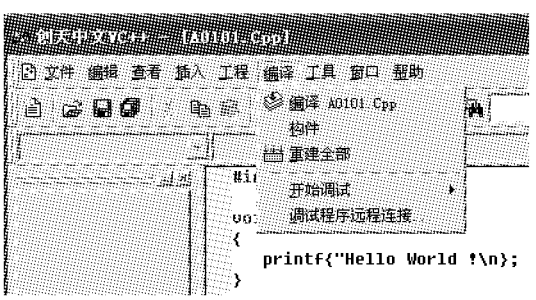


图 A01-7 用"编译"命令编译文件

A01.4 连接与运行

在单文件情况中,程序的主文件没有包含自定义的 CPP 文件或头文件。这时,如果主文件经过编译,正确无误,则可以单击"执行"命令(如图 A01-8 所示),执行该程序,不必显式地进行连接操作。事实上,源程序若包含了系统提供的头文件,则单击"执行"命令后,系统就自动将该头文件(它只是一张"菜单",其中罗列了可供用户使用的一些函数)的实现文件(CPP 源文件)与主文件进行连接,如果正确,就生成可供运行的执行代码。一个项目只能有一个主文件。

图 A01-8　用“执行”命令执行已编译好的源文件

A01.5　复制程序运行结果

运行上述 A0101.CPP 程序后，出现显示运行结果的窗体，其中有程序输出字符串“Hello World !”。右击该窗体，即出现一个对话框，如图 A01-9 所示。单击其中的“标记”选项，并用鼠标左键选中字符串“Hello World !”（在字母 H 左边按下鼠标左键，向右拖动），如图 A01-10 所示。至此，就可以利用“编辑”|“粘贴”命令把选中的程序运行结果粘贴到程序文本的末尾。

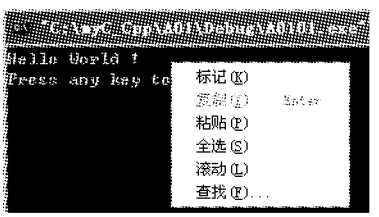


图 A01-9　右击运行结果窗体所出现的对话框

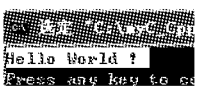


图 A01-10　选定运行结果

A01.6　Workspace 文件

建立项目后，就在 VC++ 中建立了一个“工作区”。一个工作区对应一个 Workspace 类型的文件（*.dsw 文件）。Workspace 类型文件是 VC++ 中特有的一种文件。这个文件用于存放一个或多个项目文件的有关信息。与开发工作有关的各种文件（例如，源文件、头文件、各种文档等）信息都存放在该项目文件中。

生成可执行文件后，如需编译和运行另一个程序，则先执行“文件”|“保存工作区”命令（如图 A01-11 所示），保存当前的 Workspace 文件，然后执行“文件”|“关闭工作区”命令，关闭当前的程序。然后执行“文件”|“打开工作区”命令，可以打开另一个 Workspace 文件。也可以在“Windows 资源管理器”中，直接单击某程序的.dsw 文件，把它打开，其后用“编译”|“执行”命令即可直接执行该程序。如果项目中某个源文件被修改，则下次

运行该文件时，需要执行“编译”|“重建全部”命令，重新编译该文件并进行连接，生成新的可执行文件；或者，也可以直接执行“编译”|“执行”命令，由系统自动重新编译、连接和执行。

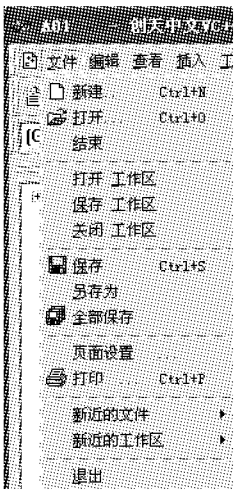


图 A01-11 对工作区的操作命令

A01.7 建立程序文件的简易方法

☞ 可以用简单方法建立 CPP 源文件或头文件，方法如下：

如果在计算机内已有某个 CPP 源文件或头文件，则可用“文件”|“打开”命令打开该文件，并用“文件”|“另存为”命令将该文件保存到指定的目录中。这样，原文件丝毫未变，而后来得到的文件可以被修改为所需的 CPP 源文件或头文件，然后用“文件”|“保存”命令保存下来。对 CPP 源文件编译后也可以得到项目文件和其他文件。这些文件都用 CPP 源文件名字作为默认名字。

A01.8 注意事项

⚠ 前面曾指出：在一个项目中，只能有一个 CPP 源文件带有主函数 main()。但在实际操作中，这个问题往往容易被忽视。

例如，按照前述方法，先建立项目 A0101，然后在该目录建立唯一的一个带主函数 main() 的 CPP 源文件。这一点比较容易做到，但是，如果运行了某个程序，例如运行了 A0101.CPP，就建立了一个工作区。这时，如果要运行另一个 CPP 文件，则必须先用“文件”|“关闭工作区”命令，将原来的工作区关闭，然后打开另一个 CPP 文件，进行编译、连接、运行等操作。否则，在原有工作区未被关闭场合，若试图运行另一个 CPP 文件，系统将给出如下两条错误信息：

_main already defined in A0101.obj (主函数已在 A0101.obj 中定义)
Debug/A0101.exe : fatal error LNK1169: one or more multiply defined symbols found
(对 A0101.obj 查错，发现重大连接错误 A0101.obj：发现一个或多个已定义符号)

实验 A01

1. 实验目的

(1) 学习建立一个简单的单文件程序。

(2) 学习对源文件进行编译、连接和运行等操作，并在程序末尾标出运行结果。

2. 实验内容

(1) 建立目录 C:\myC_CPP\A0101，在其中建立如下的 CPP 源文件 A0101.CPP，对这个文件进行编译、连接，并在程序末尾标出运行结果。

A0101.CPP

```
#include <stdio,h>
void main()
{
   printf("Hello World" \n");
}
```

▲

(2) 存储并关闭工作区。

(3) 用“文件”|“打开工作区”命令打开工作区文件C:\myC_CPP\A01\A0101.dsw。

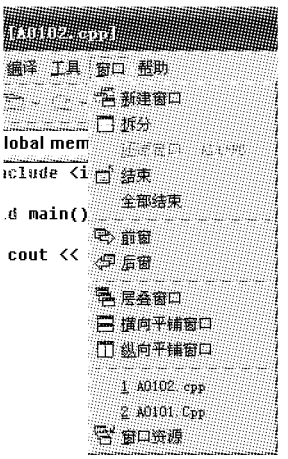
(4) 建立如下的CPP源文件A0102.CPP。

A0102.CPP

```
#include <iostream.h>
Void main()
{    cout << "Hello World !" << endl;   }
```

▲

这时,IDE的窗口菜单如图A01-12所示。



图A01-12　IDE的窗口菜单

(5) 激活主文件A0101.CPP或A0102.CPP,进行编译和连接。

为什么对这两个文件编译和连接,均出错?

A02　源代码隐蔽机制(多文件程序的编译与连接)

程序通常由多个文件组成。例如,下列A0201程序包括主文件A0201.CPP和实现文件A0201A.CPP。主文件是主函数main()所在的源文件。整个项目只能有一个主文件。这里,主函数需要调用加法函数add()和减法函数sub()。但这两个函数并不由主函数提供。它们在A0201A.CPP中列出。所以,主文件需要用预处理语句把A0201A.CPP包含进来。当用户修改主文件的数据时,需要重新编译主文件。这时,如果没有实现文件A0201A.CPP,系统将会因找不到add()和sub()的函数体而报错。因此,软件开发商向用户提供函数时,必须同时提供实现文件。换句话说,函数的源代码是不能隐藏的。

C:\myC_CPP\A0201\A0201.CPP　　// 设计者提供的主文件

```
#include <iostream.h>
#include "C:\myC_CPP\A0201\A0201A.CPP";
void main()
{
   float x = 20, y = 5, z;
   z = add(x, y) + sub(x, y);
   cout <<  "z = " << z << endl;
}
```

▲

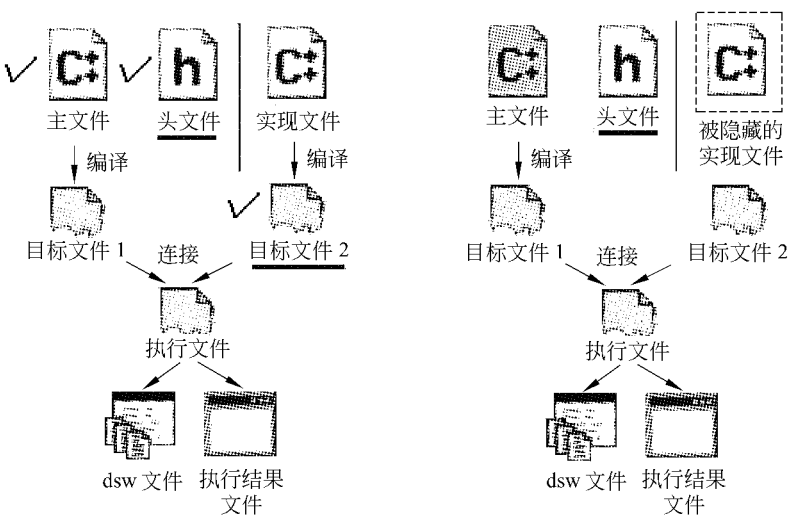
C:\myC_CPP\A0201\A0201A.CPP // 设计者必须提供的实现文件

```
float add(float a, float b)
{  return a + b;   }
float sub(float a, float b)
{  return a - b;   }
```

▲

下列程序具有隐藏源代码的功能。它由三个文件组成，即主文件 A0201.CPP、头文件 A0201A.h 和实现文件 A0201A.CPP。头文件其实是一张菜单，它向用户表明有哪些函数和常数可供用户选用。这个文件隐藏了函数的细节。函数细节由实现文件提供。

源代码隐藏又称信息隐蔽，其原理可用图 A02-1 说明。图 A02-1(a)示出程序设计者的编译、连接过程。



(a) 软件设计者的编译与连接过程　　(b) 用户的编译与连接过程

图 A02-1　信息隐藏的原理

C:\myC_CPP\A02\ A0202.CPP // 设计者提供的主文件

```
#include <iostream.h>
#include "C:\myC_CPP\A02\A0201A.h";
void main()
{
   float x = 30, y = 5, z;
   z = add(x, y) + sub(x, y);
   cout <<  "z = " << z << endl;
}
```

▲

C:\myC_CPP\A02\A0202A.h // 设计者提供的头文件

```
float add(float, float);
float sub(float, float);
```

▲

C:\myC_CPP\A02\A0202A.CPP　　// 设计者必须提供的实现文件

```
float add(float a, float b)
{   return a + b;    }
float sub(float a, float b)
{   return a - b;    }
```

▲

这里，主文件和实现文件被分别连接，生成目标文件 1 和目标文件 2。其后，这两个文件被连接成执行文件。在 C++ 6.0 中，多文件的编译、连接过程如下：

(1) 先编译主文件 A0202.CPP （见图 A02-2）。

(2) 单击“工程”|“添加工程”|“文件”命令(见图 A02-3)，出现如图 A02-4 所示的对话框。

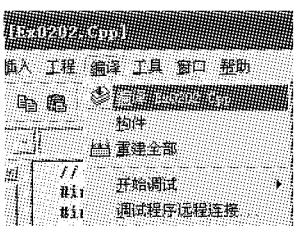


图 A02-2　编译主文件

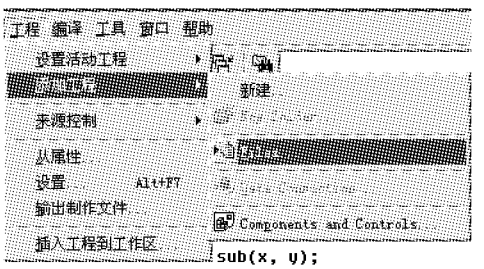


图 A02-3　提示添加实现文件

(3) 在对话框中输入实现文件名 A0202A.CPP。

(4) 单击“编译”|“重建全部”命令(见图 A02-5)。

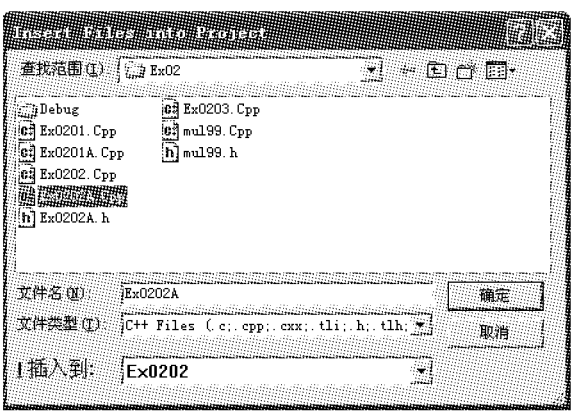


图 A02-4　选出实现文件

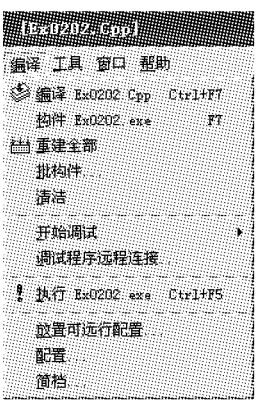


图 A02-5　生成可执行文件

这样，就完成了编译、连接操作，生成目标文件 1、目标文件 2 和执行文件。执行后，产生 dsw 文件和执行结果文件。

请注意：被连接的是 CPP 源文件；在步骤(3)中，不能输入头文件名。

在步骤(4)中，也可以直接单击“编译”|“执行”命令。这时，系统连续地进行编译、连接和执行等操作。

至此，程序设计者可以只将图 A02-1(a)中用“√”符号标出的主文件、头文件和目标文件 2 交付用户使用，而把实现文件隐藏起来。这时，可按图 A02-1(b)进行操作：若主文件已被修改，就先对它进行编译，得到目标文件 1。然后单击图 A02-5 的“编译”|“重建全部”或“编译”|“执行”命令，即可得到新的可执行文件。这时，无须实现文件参与，因为目标文件 2 是现成的，可参与连接。当然，如果设计者需要修改实现文件，则在修改后，仍需要按照上述 4 个步骤重新操作。

实验 A02

1. 实验目的

(1) 掌握多文件程序的编译、连接方法。

(2) 熟悉源代码隐藏机制。

2. 实验内容

(1) 在实现文件 A0201A.CPP 存在和被删除情况下分别运行程序 A0201。

(2) 在实现文件 A0202A.CPP 存在和被删除情况下分别运行程序 A0202。

(3) mul99.h 是头文件，其实现文件 mul99.CPP 给出 3 个函数，即 mul99_1()、mul99_2() 和 mul99_3()，分别以矩形、下三角形和上三角形等形式画出九九表。

程序 A0203 的主文件 A0203.CPP 如下，它调用函数 mul99_1()，画出矩形九九表。

A0203.CPP

```
#include "C:\myC_CPP\A02\mul99.h"
void main()
{   mul99_1();   }                                          ▲
```

- 请在实现文件 mul99.CPP 存在和不存在情况下分别运行这个程序。
- 修改主文件，调用函数 mul99_2() 或 mul99_3()，运行这个程序。

(1) 为什么只是 CPP 源文件参与连接，而头文件不能参与？

(2) 说明源代码隐藏机制。

A03 运算符 sizeof 的用法和几种类型的数据所占的字节数

设计程序前，要先确定算法，而算法的描述要涉及数据，即常量和变量。对常量和变量，都要说明其数据类型。不同数据类型所占用的存储单元数(字节数)是不同的，相应的表示范围和精度也各异。

应该指出：在不同的系统中，各种类型数据在内存中所占的字节数略有不同。教材中，表 2-2 列出了适用于 Borland C++ 3.0 的 15 种数据类型。

其实，除了无值类型 void 外，基本类型是 char、int、float 和 double。添上修饰字

signed、unsigned、short 和 long 后，便有了这 15 种类型。

运算符 sizeof 可以用来确定作为它的操作数的不同类型数据(变量或常量)在内存中所占的字节数。A0301.CPP 演示了运算符 sizeof 的用法，结果将显示几种类型数据在 Visual C++ 6.0 中所占的字节数。

A0301.CPP

```
#include <stdio.h>
void main()
{
  int byteNum;
  byteNum = sizeof( char );
  printf(" char 型数据所占的字节数          : %d \n", byteNum);
  byteNum = sizeof( int );
  printf(" int 数据所占的字节数             : %d \n", byteNum);
  byteNum = sizeof( long int );
  printf(" long int 数据所占的字节数        : %d \n", byteNum);
  byteNum = sizeof( float );
  printf(" float 数据所占的字节数           : %d \n", byteNum);
  byteNum = sizeof( double );
  printf(" double 数据所占的字节数          : %d \n", byteNum);
  byteNum = sizeof( long double );
  printf(" long double 数据所占的字节数     : %d \n", byteNum);
  int n, num = 100;
  printf("假设内存中可以提供 100 个字节存放 float 变量。能存放的变量个数为 : \n");
  printf("     n =    %d \n", num / sizeof(float));
}
```

▲

实验 A03

1. 实验目的

(1) 掌握运算符 sizeof 的用法。

(2) 熟悉几种基本类型数据在 Visual C++ 6.0 中所占的字节数。

2. 实验内容

运行程序 A0301.CPP，了解运算符 sizeof 的用法，记下运行结果。

3. 参考资料

教材 2.5 节。

A04　printf 函数和 scanf 函数中的基本格式字符的用法

C 语言使用格式化输出函数 printf()和格式化输入函数 scanf()进行 I/O 操作。在使用这两个函数时，要添加预处理指令 #include <stdio.h>。本课题介绍 printf() 函数

和 scanf() 函数的基本格式字符的用法。

A04.1 printf()函数中的基本格式字符

printf() 函数的作用是按照用户指定的格式向终端(或系统隐含指定的输出设备)输出内部类型的数据。

☞ printf() 函数的格式如下：

printf(格式化字符串,输出数据序列);

下面以程序 A0401.CPP 为例来说明。在主函数中,已定义 a 和 b 是浮点变量,故 a + b 也是浮点变量。

A0401.CPP ---- 格式化字符 f 的应用

```
#include <stdio.h>
void main()
{
   float a, b;
       a = 10.254;   b = 15.368;
       printf(" a + b = %2.3f + %2.3f = %2.2f \n", a, b, a + b);
   printf("      %2.3f    \n", a);
   printf("    +%2.3f    \n", b);
   printf("    ----------   \n");
   printf("      %2.2f    \n", a + b);
}
/*
运行结果：
a + b = 10.254 +   15.368 = 25.62
   10.254
+  15.368
---------
   25.62
*/
```

▲

在第一个 printf() 语句

```
printf(" a + b = %2.3f + %2.3f = %2.2f \n", a, b, a + b);
```

中,a、b 和 a+b 是输出数据序列。"a+b=%2.3f+%2.3f=%2.2f\n"是格式化字符串。f 是格式化字符,以字符%为前缀,其中的 2.3 用来修饰格式化字符 f,表示浮点数的整数部分占 2 位,小数部分占 3 位。这样的格式化字符有 2 个,依次对应变量 a 和 b。第 3 个格式化字符为%2.2f,表示在显示变量 a + b 时,整数部分和小数部分都取 2 位。

☞ 格式字符数目应等于被显示的变量、常量或表达式数目,而且格式字符在类型上应与被显示对象相匹配。格式化字符串中,前缀 % 和表 A04-1 所示的格式字符以及其中

的修饰符(如果有)作为一个整体,不予显示。此外,在格式化字符串中,作为转义字符的\n是换行符,也不予显示。其余部分(包括空格)将被原样显示。

printf 函数中的基本格式字符是:

c, s, d, i, o, x(X), u, f, g(G), e(E), p

表 A04-1 列出了 printf 函数所用的格式字符。除了 X、E、G 外,其他格式字符都要用小写。

表 A04-1　printf 函数所用的格式字符

格式字符	说　明
d、i	以带符号的十进制形式输出整数(正数不带符号)
o	以八进制无符号形式输出整数(不输出前导 0)
x、X	以十六进制无符号形式输出整数(不输出前导符 0x)。若用小写 x,则输出十六进制数的 a～f 时,以小写字母输出。若用大写 X,则以大写字母输出
u	以无符号十进制形式输出整数
c	以字符形式输出一个字符
s	输出字符串
f	以小数形式输出实数,隐含输出 6 位小数
e、E	以指数(科学记数)形式输出实数。若用 e,则在输出中以小写字母 e 表示指数。若用 E,则以大写字母 E 表示指数
g、G	用以输出实数。根据数值的大小,在 f 格式和 e 格式二者中选取较短的一种。用格式字符 G 时,若以指数格式输出,则指数以大写字母 E 表示
p	以十六进制数输出变量的地址

程序 A0402.CPP 演示这些格式字符的应用。

A0402.CPP

```
#include <stdio.h>
void main()
{
  //   用格式字符 'c' 显示 char 型数据
  printf("--------- 1 ---------\n");
  char c1 = 'A';     char c2 = 66;
  printf("  c1 = %c,   c2 = %c \n", c1, c2);
  //   用格式字符 's' 显示字符串.
  printf("--------- 2 ---------\n");
  printf("  s1 = %s,   s2 = %s \n", "ABCDEFG", "abcdefg");
  //   用格式字符 'd' 或 'i' 显示有符号的 int 型数据.?
  printf("--------- 3 ---------\n");
  int i = -1;     int j = 32768;
```

```
    printf("  i = %d,  j = %d \n", i, j);     // j 的值超过最大表示范围,出现溢出
    printf("  i = %i,  j = %i \n", i, j);     // j 的值超过最大表示范围,出现溢出
    //   用格式字符'o'显示有符号的 int 型八进制数据
    // -1 的八进制显示不出现负号,它显示为-1 的补码 177776
    printf("--------- 4 --------\n");
    i = 33;   j = -1;
    printf("  i = %o,   j = %o \n", i, j );
    //   用格式字符 'x' ( 或 'X' ) 显示有符号的 int 型十六进制数据
    // -1 的十六进制显示不出现负号,它显示为-1 的补码 FFFF
    printf("--------- 5 --------\n");
    i = 33;    j = -1;
    printf("  i = %X,   j = %X \n", i, j );
    //   用格式字符 'u' 显示无符号的 unsigned int 型数据
    printf("--------- 6 --------\n");
    unsigned int u1 = 30000;    unsigned int u2 = 65537;
    printf("  u1 = %u,  u2 = %u \n", u1, u2);       // u2 的值超过最大表示范围,出现溢出
    //   用格式字符 'f' 显示 float 型数据
    printf("--------- 7 --------\n");
    float f1 =112233.44;  f2 =12345.67;
    printf("  f1 = %f,  f2 =  %f \n", f1, f2);
    printf("--------- 8 --------\n");
    printf("  f1 = %e,  f2 = %E \n", f1, f2);
    //   用格式字符 'p' 显示变量地址(地址操作符'&'放在变量之前,进行取地址操作)
    printf("--------- 9 --------\n");
    printf("  &i = %p, &j = %p \n", &i, &j);
    printf("  &u1 = %p, &u2 = %p \n", &u1, &u2);
    printf("  &f1 = %p, &f2 = %p \n", &f1, &f2);
}                                                                          ▲
```

请运行本程序,分析运行结果。

教材 2.8.1 节介绍了一种特殊形式的字符常量,即以“\”开头的字符序列。这些添加反斜杠的字符另有含义,称为转义字符。格式化字符串中可以使用这些转义字符。例如,如果要原样显示字符“\”,则在格式化字符串中要使用转义字符“\\”。

A04.2 scanf 函数中的基本格式字符

scanf 函数包含格式控制字符串和地址表列:

scanf(格式控制字符串, 地址表列);

格式控制的含义与 printf 函数相同。请注意,这里的地址表列是变量的地址或字符串的首地址(字符串的名字就代表该字符串的首地址)。格式化字符串包含格式字符和它的前缀字符“%”,以及分隔符。格式字符个数应等于地址表列中的项数。

⚠ 这里需要特别强调:scanf 语句中,表示取地址的运算符“&”不可漏写,否则会造成不

可预知的错误。例如，

```
int a;
scanf("%d", a);     // 从计算机的默认输入设备输入一个数到地址为 a 的存储单元
```

执行这两条语句后，输入的整数(格式字符 d 确定输入的数是整数)并不是存放到变量 a 所在的存储单元，而是存放到地址为 a 的存储单元。如果该单元已被系统占用，就会造成不可预知的错误。正确的写法是：

```
int a;
scanf("%d", &a);     // 从计算机的默认输入设备输入一个数，作为变量 a
```

程序 A0403.CPP 演示 scanf 函数中的基本格式字符的用法，这些格式字符是：

```
c、s、d、i、o、x(X)、u、f、e(E)
```

A0403.CPP

```
#include <stdio.h>
void main()
{
  //   用格式字符 'c' 输入 char 型数据
  printf(" 请输入一个字符!\n");
  char c1;
  scanf("%c", &c1);
  printf("  c1 =   '% c' \n", c1);
  //   用格式字符 's' 输入字符串
  //   关于字符串的定义方法，请参考论述字符串的章节．此处，字符串 s1 的总长度定义为 20
  printf(" 请输入一个字符串 (不超过 19 个字符)\n");
  char s1[20];
  scanf("% s", s1);
  printf("  s1 = \"%s\" \n", s1);
  //   用格式字符 'd' 或 'i' 输入有符号的 int 型数据
  printf(" 请输入一个 int 型十进制整数 \n");
  int i1;
  scanf("% i", &i1);
  printf("  i1 = %i \n", i1);
  printf(" 请输入一个 int 型十进制整数 \n");
  scanf("%d", &i1);
  printf("  i1 = %i \n", i1);
  //   用格式字符 'o' 输入有符号的 int 型八进制数据
  printf(" 请输入一个 int 型八进制整数 \n");
  scanf("%o", &i1);
  printf("  转换为十进制，i1 = %d \n", i1);
  printf(" 请输入一个 int 型十六进制整数 \n");
  scanf("%x", &i1);
  printf("  转换为十进制，i1 = %d \n", i1);
}
```

程序 A0404.CPP 说明对 scanf 函数的格式字符进行修饰，以便按照一定的格式输入数据。请注意所用的修饰符及其在格式化字符串中的位置。

A0404.CPP

```
#include <stdio.h>
void main()
{
  printf("请输入一个 long int 型十进制整数 \n");
  long int Li1;
  scanf("%ld", &Li1);                              //  'l'是修饰符
  printf("  十进制数 i1 = %ld \n", Li1);
  printf("请输入一个 long int 型八进制整数 \n");
  scanf("%lo", &Li1);                              //  'l'是修饰符
  printf("  转换为十进制,i1 = %ld \n", Li1);
  printf("请输入一个 long int 型十六进制整数 \n");
  scanf("%lx", &Li1);                              //  'l'是修饰符
  printf("  转换为十进制,i1 = %ld \n", Li1);
  printf("请用非指数格式输入一个 float 型实数 \n");
  float f1;
  scanf("%f",&f1);
  printf("  f1 = %f \n", f1);
  printf("请用非指数格式输入一个 float 型实数,小数位数为 8.  \n");
  scanf("%f10.8",&f1);                             //  数字 10.8 是修饰符
  printf("  f1 = %10.8f \n", f1);
  printf("请用非指数格式输入一个 double 型实数 \n");
  double d1;
  scanf("%lf",&d1);                                //  'l'是修饰符
  printf("  d1 = %lf \n", d1);
  printf("请用非指数格式输入一个 double 型实数,小数位数为 8.  \n");
  scanf("%lf15.8",&d1);
  printf("  d1 = %10.8lf \n", d1);                 //  'l'和数字 15.8 是修饰符
  printf("请用非指数格式输入一个 long double 型实数 \n");
  long double Ld1;
  scanf("%Lf",&Ld1);                               //  'L'是修饰符
  printf("  Ld1 = %Lf \n", Ld1);
  printf("请用非指数格式输入一个 long double 型实数,小数位数为 12  \n");
  scanf("%Lf15.12",&Ld1);                          //  'L'和数字 15.12 是修饰符
  printf("  Ld1 = %10.12Lf \n", Ld1);
  printf("请用指数格式输入一个 long double 型实数 \n");
  scanf("%LE",&Ld1);                               //  'L'是修饰符
  printf("  Ld1 = %Le \n", Ld1);
  printf("请用指数格式输入一个 long double 型实数,尾数部分的小数位数为 10  \n");
  scanf("%LE15.10",&Ld1);                          //  'L'和数字 15.10 是修饰符
  printf("  Ld1 = %15.10Le \n", Ld1);
}
```

▲

请运行本程序，分析运行结果。

⚠ 程序 A0405.CPP 演示：在 scanf 函数的格式化字符串中，可以选用空格(“ ”)、逗号(“,”)或分号(“;”)作为分隔符。但必须注意：在输入数据时，也要用相应的分隔符，否则出错。

A0405.CPP

```
#include <stdio.h>

void main()
{
  int a, b;     float x, y;     char c1, c2;
  printf("请输入数据 :\n");
  printf("用空格作为分隔符，输入整型数 a 和 b，然后按下回车键 :\n");
  scanf("  %d %d", &a, &b);
  printf("用空格作为分隔符，输入实型数 x 和 y,然后按下回车键 :\n");
  scanf("  %f %e", &x, &y);
  printf("用空格作为分隔符，输入字符 c1 和 c2,然后按下回车键 :\n");
  scanf("  %c %c", &c1, &c2);
  printf("读出输入的数据 :\n");
  printf("  a = %d   b = %d \n", a, b);
  printf("  x = %f   y = %e \n", x, y);
  printf("  c1 = '%c'   c2 = '%c' \n\n", c1, c2);
  printf("请输入数据 :\n");
  printf("用逗号作为分隔符，依次输入整型数 a 和 b，实型数 x 和 y，以及字符 c1 和 c2,然
          后按下回车键 :\n");
  scanf("  %d, %d, %f, %e, %c, %c", &a, &b, &x, &y, &c1, &c2);
  printf("读出输入的数据 :\n");
  printf("  a = %d   b = %d \n", a, b);
  printf("  x = %f   y = %e \n", x, y);
  printf("  c1 = %c   c2 = %c \n", c1, c2);
}
```

▲

实验 A04

1. 实验目的

(1) 掌握函数 printf() 和 scanf() 的用法。

(2) 熟悉函数 printf() 和 scanf() 的几种基本格式字符。

2. 实验内容

(1) 运行程序 A0401.CPP ～ A0404.CPP，掌握函数 printf() 和 scanf() 的用法，了解它们的各种格式字符。

(2) 运行程序 A0405.CPP，牢记在使用 scanf() 输入多个变量时的注意事项。

A05 初识 C++ 的 I/O 流

上面介绍的格式化输出函数 printf() 和输入函数 scanf() 均适用于 C 和 C++。下面讲述的输入输出操作只适用于 C++。

C++ 中的输入输出(I/O)操作是由一个 I/O 流库提供的。教材第 16 章详细介绍了这个流库所提供的 I/O 操作。这里只介绍在以后的实验中经常采用的几种输入输出操作。

一般来说,计算机常以键盘作为标准输入设备,以显示器作为标准输出设备。输入操作是从键盘接收一个字符序列,由计算机的处理器处理后,以一定的格式送入存储单元。这里键盘是数据源。数据从源流向存储单元,这种数据流称为输入流。相反,存储单元的数据经过处理器处理,以一定的格式送往显示器。显示器作为接收数据的设备,称为目标。从计算机存储单元到显示器的数据流称为输出流。“流”(stream)这个词形象地描述了数据的流动。从输入设备输入数据可以理解为从一个输入流提取信息;反之,向输出设备送出信息可以理解为向输出流插入信息。应该注意:这里所谓的信息实际上包含数据以及控制其格式的控制符,如图 A05-1 所示。

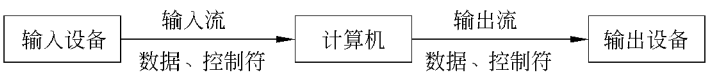


图 A05-1 输入流、输出流示意图

在 C++ 中,标准输入流 cin 是从键盘到计算机的信息流(包括控制符),标准输出流 cout 是从计算机到显示器的信息流(包括控制符)。

在使用输入、输出流之前,要包含系统头文件 iostream.h。而流的控制符则在头文件 iomanip.h 中定义。

在 C++ 中,输入运算符是“>>”。输入操作的格式如下:

```
cin >> 控制符 1 >> 变量 1 >> 变量 2 >> …… >> 控制符 2 >> ……;
```

由此看出:可以利用一个语句依次输入多个基本类型的变量和各种控制符。变量和控制符的输出次序根据需要而定。

输出(提取)运算符是“≪”。输出操作的格式如下:

```
cout << 控制符 1 << 变量 1 << 变量 2 << …… << 控制符 2 << ……;
```

同样地,可以利用一个语句依次输出多个基本类型的变量和各种控制符。变量和控制符的输出次序也根据需要而定。

下面介绍几个常用的控制符。

(1) endl:换行符,与转义字符“\n”等效。

(2) setw(n):设定输出变量所占的宽度(列数)为 n(整数)。每输出一个变量,都要重新设定宽度。

(3) setprecision(n)：参数 n 是一个整数。控制符 setprecision(n) 在 VC++ 和其他编译系统中，有不同的含义。VC++ 的 float、double 和 long double 类型数据的最大有效位数 k 分别为 6 或 7、15 或 16、18 或 19。在 VC++ 中，setprecision(n) 用于设定有效位数 n。可以任意选择 n 值，但显示的位数却不会超过 k。

(4) dec：用十进制表示法显示数值。

(5) hex：用十六进制表示法显示数值。

(6) oct：用八进制表示法显示数值。

程序 A0501.CPP 和 A0502.CPP 演示 C++ 的 I/O 操作。

A0501.CPP

```
#include <iostream.h>        // 使用输入流，要包含头文件 iostream.h
#include <iomanip.h>         // 使用控制符，要包含头文件 iomanip.h
void main()
{
  int i1, i2;
  float f1, f2;
  cout << " 请以空格为分隔符输入两个十六进制整数，然后按下回车键 \n";
  cin >> i1 >> i2;
  cout << " 请以空格为分隔符输入两个实数，然后按下回车键 " << endl;
  cin >> f1 >> f2;
  cout << " 您输入两个十六进制的整数是 ： \n";
  cout << " i1 = " <<   setw(5) << hex << i1 << " ,   i2 = " <<  i2 << setw(10) << endl;
  cout << " 相应的两个十进制的整数是 ： \n";
  cout << " i1 = " <<   setw(5) << dec << i1 << " ,   i2 = " << i2 << setw(10) << endl;
  cout << " 您输入的实数是 ： \n";
  cout << " f1 = " << setw(10) << setprecision(5) << f1 << ",   f2 = " << setw(10) <<
       setprecision(7) << f2 << endl;
}                                                                    ▲
```

请读者运行本程序，分析运行结果。

程序 A0501.CPP 表明 I/O 操作可以是连续的。输入、输出量可以是基本数据类型(包括字符串)和控制符。

A0502. CPP

```
#include <iostream.h>        // 使用输入流，要包含头文件 iostream.h
#include <iomanip.h>         // 使用控制符，要包含头文件 iomanip.h
void main()
{
  float f1, f2;
  f1 = f2 = 12.14159265;
  cout << " f1 = " << setw(16) << setprecision(3) << f1 << ",   f2 = " << setw(16) <<
       setprecision(4) << f2 << endl;
  cout << " f1 = " << setw(16) << setprecision(5) << f1 << ",   f2 = " << setw(16) <<
       setprecision(6) << f2 << endl;
```

```
    cout << " f1 = " << setw(16) << setprecision(7) << f1 << ",   f2 = " << setw(16) <<
        setprecision(8) << f2 << endl;
    cout << " f1 = " << setw(16) << setprecision(9) << f1 << ",   f2 = " << setw(16) <<
        setprecision(10) << f2 << endl;
    f1 = f2 = 12345.14159265;
    cout << " f1 = " << setw(16) << setprecision(3) << f1 << ",   f2 = " << setw(16) <<
        setprecision(4) << f2 << endl;
    cout << " f1 = " << setw(16) << setprecision(5) << f1 << ",   f2 = " << setw(16) <<
        setprecision(6) << f2 << endl;
    cout << " f1 = " << setw(16) << setprecision(7) << f1 << ",   f2 = " << setw(16) <<
        setprecision(8) << f2 << endl;
    cout << " f1 = " << setw(16) << setprecision(9) << f1 << ",   f2 = " << setw(16) <<
        setprecision(10) << f2 << endl;
}                                                                          ▲
```

请读者运行本程序,分析运行结果。

实验 A05

1. 实验目的

(1) 熟悉 C++ 的输入、输出运算符的用法。

(2) 熟悉输入、输出操作的几个常用控制符。

2. 实验内容

(1) 编制程序 A0503.CPP。已知圆周率 Pi=3.14159265。要求用一个循环语句和一个输出运算符,使输出显示为:

```
小数点后取 2 位,Pi = 3.14
小数点后取 3 位,Pi = 3.142
小数点后取 4 位,Pi = 3.1416
小数点后取 5 位,Pi = 3.14158
```

(2) 编制程序 A0504.CPP。要求输入 3 个十六进制整数,并将它们显示为八进制和十六进制数。

A06 调试程序

A06.1 VC++ 的程序调试功能

即使程序员编程时自认为非常"谨慎",但十之八九仍然会出现各种各样的错误:语法上的、算法上的……其中语法错误和连接错误由编译系统报出。这类错误要由程序员自己亲自纠正。事实上,若程序员稍有经验,这类错误是不难消除的。而消除算法上的错误以及其他错误(例如数据类型定义错误、逻辑错误、循环体定义错误等),则往往很费精