Visual C++ 指导教程

目录

1.Visual Studio IDE 简介 (C++)

在这些主题中，您将创建一个新的标准 C++ 程序，并使用 Visual Studio 为 C++ 开发人员提供的功能测试该程序的功能。您创建的简单程序将跟踪有多少个玩家正在玩各种纸牌游戏。

本演练涵盖以下内容：
项目和解决方案 (C++)、生成项目 (C++)、测试项目 (C++)、调试项目 (C++)、部署程序 (C++)

1.1 项目和解决方案 (C++)

在 Visual Studio 中，可以将您的工作组织为项目和解决方案。一个解决方案可以包含多个项目，如一个 DLL 和一个引用该 DLL 的可执行文件。有关更多信息，请参见 介绍解决方案、项目和项。

先决条件
本主题假定您具备 C++ 语言的基础知识。

使用项目和解决方案
用 Visual Studio 编写 Visual C++ 程序的第一步是选择项目的类型。对于每种项目类型，Visual Studio 都为您设置编译器设置并生成起始代码。

创建新项目

1. 在**“文件”**菜单中，指向**“新建”**，然后单击**“项目...”**。
2. 在**“项目类型”**区域中，单击**“Win32”**。然后，在**“Visual Studio 已安装的模板”**窗格中，单击**“Win32 控制台应用程序”**。
3. 键入项目名称。在此示例中，我们将使用**“游戏”**。

   创建新项目时，Visual Studio 将该项目放入一个解决方案。请接受解决方案的默认名称，该名称与项目的名称相同。

   您可以接受默认位置、键入一个不同的位置或者浏览到要保存项目的目录。

   按**“确定”**启动**“Win32 应用程序向导”**。
4. 在**“Win32 应用程序向导”**对话框的**“概述”**页中，单击**“下一步”**。

5. 在**“应用程序类型”**下的**“应用程序设置”**页，选择**“控制台应用程序”**。选择**“其他选项”**下的**“空项目”**设置并单击**“完成”**。

   现在，您得到了一个没有源代码文件的项目。

使用解决方案资源管理器

通过解决方案资源管理器，您可以轻松使用解决方案中的文件和其他资源。

在本步骤中，您要向项目添加一个类，Visual Studio 会将 **.h** 和 **.cpp** 文件添加到项目中。然后，为测试类的主程序向项目添加一个新的源代码文件。

向项目添加类

1. 如果**“解决方案资源管理器”**窗口不可见，请单击**“视图”**菜单上的**“解决方案资源管理器”**。
2. 右击**“解决方案资源管理器”**中的**“头文件”**文件夹并指向**“添加”**。然后单击**“类”**。

   在**“Visual C++”**类别中，单击**“Visual Studio 已安装的模板”**区域中的**“C++”**，然后单击**“C++ 类”**。单击**“添加”**。
3. 在**“一般 C++ 类向导”**中，键入**“Cardgame”**作为**“类名”**，并接受默认的文件名，然后单击和设置。然后单击**“完成”**。
4. 对编辑区域中显示的 **Cardgame.h** 文件进行下列更改：

   - 在类定义的左大括号之后添加两个私有数据成员：

     复制代码

     ```
     int players;
     static int totalparticipants;
     ```

   - 添加一个采用一个 `int` 类型的参数的公共构造函数原型：

     复制代码

     ```
     Cardgame(int p);
     ```

   - 删除为您生成的默认构造函数。默认构造函数是没有参数的构造函数。默认构造函数类似于如下所示：

     复制代码

     ```
     Cardgame(void);
     ```

5. 进行上述更改后，**Cardgame.h** 文件应如下所示：

复制代码

```
#pragma once

class Cardgame

{

        int players;

        static int totalparticipants;

    public:

        Cardgame(int p);

        ~Cardgame(void);

};
```

#pragma once 行指明编译器只包含该文件一次。有关更多信息，请参见 once。

有关此头文件中包含的其他 C++ 关键字的信息，请参见 class (C++)、int、Static (C++) 和 public (C++)。

6. 双击**“源文件”**文件夹中的**“Cardgame.cpp”**，以将其打开进行编辑。

7. 为采用一个 int 参数的构造函数添加代码：

复制代码

```
Cardgame::Cardgame(int p)

{

    players = p;

    totalparticipants += p;

    cout << p << " players have started a new game.  There are now "

         << totalparticipants << " players in total." << endl;

}
```

开始键入 pl 或 to 时，可以按 Ctrl-空格键，自动完成功能将为您完成键入 players 或 totalparticipants。

8. 删除自动生成的默认构造函数：

复制代码

```
Cardgame::Cardgame(void);
```

9. 进行上述更改后，**Cardgame.cpp** 文件应如下所示：

复制代码

```
#include "Cardgame.h"
#include <iostream>
using namespace std;

Cardgame::Cardgame(int p)
{
    players = p;
    totalparticipants += p;
    cout << p << " players have started a new game.  There are now "
         << totalparticipants << " players in total." << endl;
}
Cardgame::~Cardgame(void)
{
}
```

有关 `#include` 的说明，请参见 The #include Directive。

添加源文件

在本步骤中，您将为测试类的主程序添加一个源代码文件。

添加新源文件

1. 在**“项目”**菜单上，单击**“添加新项”**。

   也可以使用“解决方案资源管理器”来向项目添加新文件，方法是右击“解决方案资源管理器”中的**“源文件”**文件夹，指向**“添加”**。然后单击**“新建项”**。

   在**“Visual C++”**区域中，选择**“代码”**。然后单击**“C++ 文件(.cpp)”**。

2. 键入**“testgames”**作为“名称”，然后单击**“添加”**。

3. 在 **testgames.cpp** 编辑窗口中，键入以下代码：

复制代码

```
#include "Cardgame.h"
int Cardgame::totalparticipants = 0;
int main()
{
    Cardgame *bridge = 0;
    Cardgame *blackjack = 0;
    Cardgame *solitaire = 0;
    Cardgame *poker = 0;

    bridge = new Cardgame(4);
    blackjack = new Cardgame(8);
    solitaire = new Cardgame(1);
    delete blackjack;
    delete bridge;
    poker = new Cardgame(5);
    delete solitaire;
    delete poker;

    return 0;
}
```

有关此源文件中包含的 C++ 关键字的信息，请参见 new Operator (C++) 和 delete Operator (C++)。

4. 在**“生成”**菜单上，单击**“生成解决方案”**。

在**“输出”**窗口中，您应当看到生成的输出，它指示已编译项目，并且未发生错误。如果未看到，请将您的代码与该主题较早显示的代码进行比较。

1.2 生成项目 (C++)

在本步骤中，您将故意在代码中引入一个 Visual C++ 语法错误，以了解什么是编译错误，以及如何修复它。 编译项目时，会显示错误消息以指示所发生的问题的性质和位置。

使用 IDE 修复编译错误

1. 在 **testgames.cpp** 中，删除最后一行中的分号，使代码如下所示：

   复制代码

   ```
   return 0
   ```

2. 在**“生成”**菜单上，单击**“生成解决方案”**。
3. **“输出”**窗口中显示一条消息，指示生成项目失败。

   单击**“输出”**窗口中的**“转到下一条消息”**按钮（指向右方的绿色箭头）。 **“输出”**窗口中的错误消息和状态栏区域指示右大括号前缺少一个分号。

   若要查看有关错误的更多帮助信息，请突出显示错误，并按 F1 键。
4. 将分号重新添加到导致语法错误的行的末尾：

   复制代码

   ```
   return 0;
   ```

5. 在**“生成”**菜单上，单击**“生成解决方案”**。

   “输出”窗口中显示一条消息，指示项目已正确编译。

1.3 测试项目 (C++)

以“调试”模式运行程序使您可以使用断点来暂停程序，以检查变量和对象的状态。
在本步骤中，您将在程序运行时观察变量的值，并推断为什么值与预期的不同。

以“调试”模式运行程序

1. 如果**“testgames.cpp”**文件不可见，请在编辑区域中单击该文件对应的选项卡。

2. 单击以下行，将其设置为编辑器中的当前行：

   复制代码

   ```
   solitaire = new Cardgame(1);
   ```

3. 若要在该行上设置断点，请单击**“调试”**菜单上的**“切换断点”**，或者按 F9。也可以单击代码行左侧的区域来设置或清除断点。

   设置了断点的代码行的左侧会显示一个红色圆圈。

4. 在**“调试”**菜单上，单击**“开始调试”**，或者按 F5。

   当程序运行到包含断点的行时，执行将暂时停止（因为程序处于“中断”模式）。代码行左侧的黄色箭头指示该行是要执行的下一个代码行。

5. 若要检查 `Cardgame::totalparticipants` 变量的值，请将鼠标指针悬停在该变量上方。该变量的名称及其值 12 即显示在工具提示窗口中。

   右击 `Cardgame::totalparticipants` 变量。选择**“表达式:‘totalparticipants’”**，并单击**“添加监视”**以在**“监视”**窗口中显示该变量。您也可以选择该变量并将其拖动到**“监视”**窗口。

6. 在**“调试”**菜单上，单击**“逐过程”**，或者按 F10 步进到下一行代码。

   `Cardgame::totalparticipants` 的值现在显示为 13。

7. 右击 `main` 方法的最后一行 (`return 0;`)，并单击**“运行到光标处”**。代码左侧的黄色箭头指向要执行的下一个语句。

8. 在 Cardgame 终止时，`Cardgame::totalparticipants` 数应当减小。此时，`Cardgame::totalparticipants` 应当等于 0，这是因为所有的 Cardgame 指针都已删除，但是**“监视 1”**窗口指示 `Cardgame::totalparticipants` 等于 18。

   代码中存在一个 bug，您将在下一节中检测并修复它。

9. 在**“调试”**菜单上，单击**“停止调试”**或者按 Shift-F5 停止程序。

1.4 调试项目 (C++)

更新：2007 年 11 月

在本步骤中，您将修改程序以修复在测试项目时发现的问题。

先决条件

本主题假定您具备 C++ 语言的基础知识。如果您是刚开始学习 C++，建议您参阅 Herb Schildt 编写的 C++ Beginner's Guide（《C++ 初学者指南》），该书可从 http://go.microsoft.com/fwlink/?LinkId=115303 在线获得。

修复包含 bug 的程序

1. 若要明白在 Cardgame 对象销毁时会发生什么，请查看 Cardgame 类的析构函数。

   在**“视图”**菜单上，单击**“类视图”**，或者单击**“解决方案资源管理器”**窗口中的**“类视图”**选项卡。

   展开**“game”**项目树并单击**“Cardgame”**类。

   下方的区域显示类的成员和方法。

   右击**“~Cardgame(void)”**析构函数并单击**“转到定义”**。

2. 要在 Cardgame 终止时减少 totalparticipants ，请在 Cardgame::~Cardgame 析构函数的左大括号和右大括号之间键入以下代码：

   复制代码

```
totalparticipants -= players;
cout << players << " players have finished their game.  There are now "
<< totalparticipants << " players in total." << endl;
}
```

3. 进行上述更改后，**Cardgame.cpp** 文件应如下所示：

   复制代码

```
#include "Cardgame.h"
#include <iostream>
using namespace std;
Cardgame::Cardgame(int p)
{
    players = p;
    totalparticipants += p;
    cout << p << " players have started a new game.  There are now "
         << totalparticipants << " players in total." << endl;
```

```
}


Cardgame::~Cardgame(void)
{
    totalparticipants -= players;
    cout << players << " players have finished their game.  There are now "
         << totalparticipants << " players in total." << endl;
}
```

4. 在**"生成"**菜单上，单击**"生成解决方案"**。

5. 在**"调试"**菜单上，单击**"运行"**，或按 F5，以"调试"模式运行该程序。程序将在第一个断点处暂停。

6. 在**"调试"**菜单上单击**"逐过程"**或者按 F10 逐句通过程序。

   请注意，执行每个 Cardgame 构造函数后，`totalparticipants` 的值会增大。而在删除每个指针（并调用析构函数）后，`totalparticipants` 的值会减小。

7. 单步执行至程序的最后一行。恰好在执行 return 语句之前，`totalparticipants` 等于 0。继续逐句通过程序，直到程序退出；或者，在**"调试"**菜单上单击**"运行"**或按 F5，允许程序继续运行，直到退出。

1.5 部署程序 (C++)

现在，我们创建了应用程序，最后一步是创建可供其他用户在其计算机上安装该程序的安装程序。为此，我们需要将新项目添加到现有解决方案。此新项目的输出是 setup.exe 文件，该文件用于安装我们之前在另一台计算机上创建的应用程序。

说明

本主题中的信息不适用于 Visual C++ 学习版。

本演练将使用 Windows Installer 来部署应用程序。您还可以使用 ClickOnce 部署应用程序。有关更多信息，请参见 Visual C++ 应用程序的 ClickOnce 部署。有关常规部署的更多信息，请参见部署应用程序和组件。

创建安装项目和安装程序

1. 在**"文件"**菜单上，单击**"新建"**，再单击**"项目"**。

随即出现**“添加新项目”**对话框。

2. 在**“已安装的模板”**下，展开**“其他项目类型”**节点。接着，展开**“安装和部署”**节点并单击**“Visual Studio Installer”**。
3. 从**“模板”**窗格中，选择**“安装向导”**。键入安装项目的名称，例如 **gameInstaller**。在**“解决方案”**列表框中，选择**“添加到解决方案”**。单击**“确定”**按钮。
4. 将出现**“安装向导”**。单击**“下一步”**继续。
5. 从向导的**“选择一种项目类型”**窗格中，选择**“为 Windows 应用程序创建一个安装程序”**选项，并单击**“下一步”**继续。
6. 从向导的**“选择要包括的项目输出”**窗格，选择**“主输出 来自 游戏”**，并单击**“下一步”**继续。
7. 无需在安装程序中包括任何其他文件，因此，从安装程序的**“选择要包括的文件”**窗格中，单击**“下一步”**。
8. 检查向导的更改，并验证所有内容是否正确。单击**“完成”**创建项目。

   “解决方案资源管理器”中将列出新的 **gameInstaller** 项目。此项目将列出应用程序依靠的依赖项（如 C 运行库或 .NET Framework）以及安装程序中将包括的项目文件。

   创建安装项目后，有许多选项可以更改。有关更多信息，请参见 Visual Studio Installer 部署。
9. 通过在**“解决方案资源管理器”**中选择安装程序并从**“生成”**菜单单击**“生成 gameInstaller”**来生成安装程序。
10. 找到上一节创建的 setup.exe 和 gameInstaller.msi 程序。在计算机上双击任一文件安装应用程序。

2.创建命令行应用程序 (C++)

我们已经学习了 Visual Studio IDE，现在可以开始使用 Visual C++ 编写程序了。 我们将学习创建的第一个应用程序类型是命令行应用程序。 命令行应用程序不包含图形用户界面 (GUI)。 通常，命令行应用程序从控制台读取输入，并将输出写入该控制台，而不是写入图形窗口。
在本节中，您将学习如何创建 C 和 C++ 命令行应用程序。 还将学习如何创建不使用 Microsoft 扩展的标准 C 和 C++ 程序。 如果您希望使用 Visual C++ 创建在其他操作系统上使用的应用程序，这将很有用。

2.1 创建标准 C++ 程序 (C++)

可以使用 Visual C++ 2010 在 Visual Studio 集成开发环境 (IDE) 中创建标准 C++ 程序。通过采用此演练中的步骤，您可以创建一个项目，向该项目添加一个新文件，修改该文件以添加 C++ 代码，然后使用 Visual Studio 编译并运行程序。

您可以键入自己的 C++ 程序，或者使用示例程序之一。此演练中的示例程序是一个控制台应用程序。此应用程序使用标准模板库 (STL) 中的 **set** 容器。

Visual C++ 使用 2003 C++ 标准进行编译，但有以下几点主要例外之处：两阶段名称查找、异常规范和导出。此外，Visual C++ 支持若干 C++0x 功能，例如，lambda、自动、static_assert、rvalue 引用和 extern 模板。

说明

如果要求符合标准，请使用 **/Za** 编译器选项来禁用对该标准的 Microsoft 扩展。有关更多信息，请参见 /Za、/Ze（禁用语言扩展）。

创建项目并添加源文件

1. 通过以下方式创建一个项目：指向**“文件”**菜单上的**“新建”**，然后单击**“项目”**。
2. 在**“Visual C++”**项目类型窗格中，单击**“Win32”**，然后单击**“Win32 控制台应用程序”**。
3. 键入项目名称。

   默认情况下，包含项目的解决方案与项目同名，但您可以键入其他名称。您也可以为项目键入其他位置。

   单击**“确定”**创建项目。
4. 在**“Win32 应用程序向导”**中，单击**“下一步”**，选择**“空项目”**，然后单击**“完成”**。
5. 如果未显示**“解决方案资源管理器”**，请在**“视图”**菜单上，单击**“解决方案资源管理器”**。
6. 将一个新源文件添加到项目，如下所示。

   a. 在**“解决方案资源管理器”**中，右击**“源文件”**文件夹，指向**“添加”**，然后单击**“新建项”**。

   b. 在**“代码”**节点中单击**“C++ 文件(.cpp)”**，为文件键入名称，然后单击**“添加”**。

   该 .cpp 文件即显示在**“解决方案资源管理器”**中的“源文件”文件夹中，并且文件将在 Visual Studio 编辑器中打开。
7. 在编辑器内的文件中，键入使用标准 C++ 库的有效 C++ 程序，或者复制示例程序之一并将其粘贴在文件中。

   例如，您可以使用 set::find (STL Samples)示例程序，该程序是帮助中附带的标准模板库示例之一。

   如果使用该示例程序，请注意 `using namespace std;` 指令。此指令使程序能够使用 **cout** 和 **endl**，而无需完全限定名（**std::cout** 和 **std::endl**）。
8. 保存该文件。
9. 在**“生成”**菜单上，单击**“生成解决方案”**。

   “输出”窗口显示有关编译过程的信息，例如，生成日志的位置，以及指示生成状态的消息。

10. 在**“调试”**菜单上，单击**“开始执行(不调试)”**。

    如果使用了示例程序，将显示一个命令窗口，其中显示是否在集合中找到了特定的整数。

2.2 在命令行上编译本机 C++ 程序 (C++)

Visual C++ 包括一个 C++ 编译器，可用来创建从基本 Visual C++ 程序到 Windows 窗体应用程序和组件的各种程序。
通过按此演练的过程进行操作，您可以通过使用文本编辑器创建基本的 Visual C++ 程序，然后在命令行上对其进行编译。
也可以编译使用 Visual Studio 集成开发环境 (IDE) 创建的 Visual C++ 程序。有关更多信息，请参见演练：在 Visual Studio 中编译面向 CLR 的 C++ 程序 (C++)。
您可以使用自己的 Visual C++ 程序，而不是键入下面步骤中所示的程序。也可以使用其他帮助主题中的任何 Visual C++ 代码示例程序。

创建 Visual C++ 源文件并在命令行上对其进行编译

1. 打开**“Visual Studio 2010 命令提示”**窗口，方法是单击**“开始”**，指向**“所有程序”**、**“Microsoft Visual Studio 2010”**、**“Visual Studio 工具”**，然后单击**“Visual Studio 2010 命令提示”**。

   可能需要管理员凭据才能成功编译此演练中的代码，具体情况视计算机的操作系统和配置而定。

   若要以管理员身份运行**“Visual Studio 2010 命令提示”**窗口，请右击**“Visual Studio 2010 命令提示”**，然后单击**“以管理员身份运行”**。
2. 在命令提示符下，键入 **notepad basic.cpp**，并按 **Enter**。

   在系统提示是否创建文件时，单击**“是”**。
3. 在记事本中，键入下列各行。

   复制代码

```
#include <iostream>


int main()
{
    std::cout << "This is a native C++ program." << std::endl;
    return 0;
}
```

4. 在**“文件”**菜单上，单击**“保存”**。

   这样就创建了一个 Visual C++ 源文件。

5. 关闭记事本。
6. 在命令提示符下，键入 **cl /EHsc basic.cpp**，并按 **Enter**。**/EHsc** 命令行选项指示编译器启用 C++ 异常处理。有关更多信息，请参见 /EH（异常处理模型）。

   cl.exe 编译器将生成一个名为 **basic.exe** 的可执行程序。

   您可以在编译器显示的多行输出信息中看到可执行程序的名称。
7. 若要查看目录中具有名称 **basic** 以及任何文件扩展名的文件的列表，请键入 **dir basic.*** 并按 **Enter**。

   .obj 文件是一个中间格式文件，可以安全地忽略它。
8. 若要运行 basic.exe 程序，请键入 **basic** 并按 **Enter**。

   该程序显示以下文本并退出：

   This is a native C++ program.
9. 若要关闭**“Visual Studio 2010 命令提示”**窗口，请键入 **exit** 并按 **Enter**。

编译使用 .NET 类的 Visual C++ 程序

下面的步骤演示如何编译使用 .NET Framework 类的 Visual C++ 程序。

您必须使用 /clr（公共语言运行时编译）编译器选项，因为此程序使用 .NET 类并且必须包括必要的 .NET 库。Visual C++ 编译器生成的 .exe 文件包含 MSIL 代码，而不是可由计算机执行的指令。

按照本过程中的步骤编译帮助主题中的任何 Visual C++ 示例程序。

在命令行上编译 Visual C++ .NET 控制台应用程序

1. 打开**“Visual Studio 2010 命令提示”**窗口，方法是单击**“开始”**，指向**“所有程序”**、**“Microsoft Visual Studio 2010”**、**“Visual Studio 工具”**，然后单击**“Visual Studio 2010 命令提示”**。

   可能需要管理员凭据才能成功编译此演练中的代码，具体情况视计算机的操作系统和配置而定。

   若要以管理员身份运行**“Visual Studio 2010 命令提示”**窗口，请右击**“Visual Studio 2010 命令提示”**，然后单击**“以管理员身份运行”**。
2. 在命令提示符下，键入 **notepad basicclr.cpp**，并按 **Enter**。

   在系统提示是否创建文件时，单击**“是”**。
3. 在记事本中，键入下列各行。

   复制代码

```
int main()
{
    System::Console::WriteLine("This is a Visual C++ program.");
}
```

4. 在**“文件”**菜单上，单击**“保存”**。

   您已经创建了一个使用 .NET 类 (Console) 的 Visual C++ 源文件，该文件位于 System 命名空间。

5. 关闭记事本。

6. 在命令提示符下，键入 **cl /clr basicclr.cpp**，并按 **Enter**。cl.exe 编译器将生成一个名为 **basicclr.exe** 的可执行程序。

7. 若要查看目录中具有名称 **basicclr** 以及任何文件扩展名的文件的列表，请键入 **dir basicclr.*** 并按 Enter。

   .obj 文件是一个中间格式文件，可以安全地忽略它。

   .manifest 文件是包含有关程序集的信息的 XML 文件。（程序集是 .NET 部署单元，例如 .exe 程序或 .dll 组件或库。）

8. 若要运行 basicclr.exe 程序，请键入 **basicclr** 并按 **Enter**。

   该程序显示以下文本并退出：

   This is a Visual C++ program.

9. 若要关闭**“Visual Studio 2010 命令提示”**窗口，请键入 **exit** 并按 **Enter**。

2.3 在 Visual Studio 中编译面向 CLR 的 C++ 程序 (C++)

通过使用 Visual Studio 开发环境，您可以创建使用 .NET 类的 Visual C++ 程序，并对它们进行编译。在本过程中，您可以键入自己的 Visual C++ 程序，也可以使用示例程序之一。本过程中使用的示例程序创建一个名为 textfile.txt 的文本文件，并将其保存到项目目录中。

在 Visual Studio 中创建新项目并添加新的源文件

1. 创建新项目。在**“文件”**菜单上，指向**“新建”**，然后单击**“项目”**。

2. 在“Visual C++ 项目类型”中单击**“CLR”**，然后单击**“CLR 空项目”**。

3. 键入项目名称。

   默认情况下，包含项目的解决方案与新项目同名，当然，您也可以键入其他名称。如果愿意，您可以为项目输入一个不同的位置。

   单击**“确定”**创建新项目。

4. 如果“解决方案资源管理器”不可见，请单击**“视图”**菜单上的**“解决方案资源管理器”**。

5. 向该项目添加新的源文件：

- 在解决方案资源管理器中右击**“源文件”**文件夹，指向**“添加”**并单击**“新建项...”**。
- 单击**“C++ 文件(.cpp)”**，键入一个文件名，然后单击**“添加”**。

该 **.cpp** 文件即显示在“解决方案资源管理器”中的**“源文件”**文件夹中，并且，在键入要包含在该文件中的代码的位置，出现一个选项卡式窗口。

6. 在 Visual Studio 中，在新创建的选项卡中单击并键入有效的 Visual C++ 程序，或者复制并粘贴示例程序之一。

   例如，您可以使用 如何：编写文本文件 示例程序（位于“编程指南”中的**“文件处理和 I/O”**节点）。

   如果要使用示例程序，请注意在创建 .NET 对象时，您可以使用 **gcnew** 关键字（而非 **new**），且 **gcnew** 返回一个句柄 (^) 而不是指针 (*)：

   ```
   StreamWriter^ sw = gcnew StreamWriter(fileName);
   ```

   有关新 Visual C++ 语法的更多信息，请参见 Language Features for Targeting the CLR。

7. 在**“生成”**菜单上，单击**“生成解决方案”**。

   “输出”窗口显示有关编译过程的信息，如生成日志的位置，以及指示生成状态的消息。

   如果进行了更改，并在未执行生成的情况下运行该程序，则对话框可能指示该项目已过期。如果要让 Visual Studio 始终使用文件的当前版本，并且在每次生成应用程序时不发出提示，请在单击**“确定”**之前选中此对话框上的复选框。

8. 在**“调试”**菜单上，单击**“开始执行(不调试)”**。

9. 如果您使用的是示例程序，则在运行程序时将显示一个命令窗口，指示已创建了该文本文件。按任意键，关闭该命令窗口。

   textfile.txt 文本文件现在位于您的项目目录中。您可以使用记事本打开此文件。

> **说明**
>
> 选择空 CLR 项目模板会自动设置 **/clr** 编译器选项。若要验证这一点，请在**“解决方案资源管理器”**中右击该项目，再单击**“属性”**，然后选中**“配置属性”**的**“常规”**节点中的**“公共语言运行时支持”**选项。

2.4 编译 C 程序

Visual C++ 2010 中包括一个 C 编译器，可用来创建从基本的 C 程序到 Windows API 应用程序的各种程序。

此演练演示如何使用文本编辑器创建一个基本的 C 程序，然后在命令行上对其进行编译。

您可以使用自己的 C 程序，而不是键入此演练中所示的示例程序。也可以使用帮助主题中包含的任何 C 代码示例程序。

默认情况下，Visual C++ 编译器将以 .c 结尾的所有文件视为 C 源代码，将以 .cpp 结尾的所有文件视为 C++ 源代码。若要强制编译器将所有文件视为 C（而不管文件扩展名如何），请使用 /Tc 编译器选

项。

系统必备

您必须了解 C++ 语言的基础知识。如果您刚刚开始学习 C++，建议阅读 Herb Schildt 编写的 C++ Beginner's Guide（《C++ 初学者指南》），MSDN 网站上提供了该指南。

创建 C 源文件并在命令行上对其进行编译

1. 单击**“开始”**，指向**“所有程序”**、**“Microsoft Visual Studio 2010”**和**“Visual Studio 工具”**，然后单击**“Visual Studio 2010 命令提示”**。

   根据计算机上的 Windows 版本和系统安全配置，可能必须右击**“Visual Studio 2008 命令提示”**，然后单击**“以管理员身份运行”**，才能成功运行按下列步骤创建的应用程序。

   > **说明**
   >
   > **“Visual Studio 2010 命令提示”**会自动设置 C 编译器和所需的任何库的正确路径。应使用它而不是使用普通的“命令提示符”窗口。有关更多信息，请参见为命令行生成设置路径和环境变量。

2. 在命令提示符下，键入 **notepad simple.c**，并按 Enter。

   在系统提示是否创建文件时，单击**“是”**。

3. 在记事本中，键入下列各行。

   复制代码

   ```
   #include <stdio.h>


   int main()
   {
       printf("This is a native C program.\n");
       return 0;
   }
   ```

4. 在**“文件”**菜单上，单击**“保存”**，以创建 C 源文件。

5. 关闭记事本。

6. 在命令提示符下，键入 **cl simple.c**，并按 Enter。

   cl.exe 编译器将生成一个可执行程序 Simple.exe。

   您可以在编译器显示的多行输出信息中看到可执行程序的名称。

   复制代码

   ```
   Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00 for 80x86
   ```

```
Copyright (C) Microsoft Corporation.All rights reserved.simple.c
Microsoft (R) Incremental Linker Version 10.00
Copyright (C) Microsoft Corporation.All rights reserved./out:simple.exe
simple.obj
```

7. 若要查看 \simple\ 目录中的所有文件的列表，请键入 **dir simple.*** 并按 Enter。

   .obj 文件是一个中间格式文件，可以安全地忽略它。

8. 若要运行 Simple.exe，请键入 **simple** 并按 Enter。

   该程序显示以下文本并退出：

   ```
   This is a native C program.
   ```

9. 若要关闭命令提示符窗口，请键入 **exit** 并按 Enter。

3.创建 Windows 应用程序 (C++)

现在我们学习了 Visual Studio IDE 和命令行应用程序，下面将学习如何创建 Windows 应用程序。使用 Visual C++，可以通过使用多种不同的技术来创建 Windows 应用程序，如 Windows API（也称为 Win32 API）和 .NET Framework。
在本节中，我们将通过使用 Win32 API 和 .NET Framework 创建两个简单的 Windows 应用程序。我们还将通过使用 .NET Framework 创建 Windows 窗体控件，最后将通过使用 DirectX 创建一个简单的游戏。

3.1 创建 Win32 应用程序 (C++)

Win32 API（也称为 Windows API）是用于创建 Windows 应用程序的基于 C 的框架，自 Windows 1.0 以来就已存在。在 Windows API 中可以找到有关此 API 的大量文档。
在本过程中，我们将创建向窗口显示"Hello, World!"的简单 Win32 应用程序。过程中的步骤对于所有 Win32 应用程序都是相同的。完成此过程后，您可以将这里创建的代码用作创建任何其他 Win32 应用程序的主干。

创建新的 Win32 项目

1. 在**"文件"**菜单上，单击**"新建"**，然后单击**"项目..."**。
2. 在**"项目类型"**窗格中，选择**"Visual C++"**节点中的**"Win32"**，然后在**"模板"**窗格中选择**"Win32 项目"**。

   键入项目的名称，如 **win32app**。您可以接受默认位置、键入一个位置或者导航到要保存项目

的目录。

3. 在**“Win32 应用程序向导”**中，选择**“下一步”**。
4. 在**“Win32 应用程序向导”**中，在**“应用程序类型”**下选择**“Windows 应用程序”**。在**“附加选项”**下选择**“空项目”**。原样保留剩余的选项。单击**“完成”**创建项目。
5. 在**“项目”**菜单中选择**“添加新项...”**，将 C++ 文件添加到项目中。在**“添加新项”**对话框中选择**“C++ 文件(.cpp)”**。为文件键入一个名称，如 **GT_HelloWorldWin32.cpp**，并单击**“添加”**。

启动 Win32 应用程序

1. 正如您所了解的，每个 C 和 C++ 应用程序必须具有一个 **main** 函数。此函数是应用程序的起始点。类似地，在 Win32 应用程序中，每个应用程序必须具有一个 **WinMain** 函数。**WinMain** 的语法如下所示：

复制代码

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow);
```

有关此函数的参数和返回值的解释，请参见 WinMain 函数。

2. 因为应用程序代码必须使用现有的定义，所以应将 include 语句添加到文件中以使用它们。例如：

复制代码

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>
```

3. 除 **WinMain** 外，每个 Win32 应用程序还必须具有第二个函数（通常称为 **WndProc**），它代表窗口过程。**WndProc** 的语法如下所示：

复制代码

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

此函数的用途是处理应用程序从操作系统接收的任何消息。应用程序何时从操作系统接收消息？始终接收！例如，假设我们创建了包含**“确定”**按钮的对话框。当用户单击该按钮时，操作系统向应用程序发送消息，使我们知道某位用户按下了此按钮。**WndProc** 函数负责响应该事件。在本示例中，适当的响应可能是关闭对话框。

有关更多信息，请参见窗口过程。

向 WinMain 添加功能

1. 首先，在 **WinMain** 函数内部创建 WNDCLASSEX 类型的窗口类结构。此结构包含有关窗口的信息，如应用程序图标、窗口的背景色、在标题栏中显示的名称、窗口过程函数的名称等等。典型的 **WNDCLASSEX** 结构如下：

复制代码

```
    WNDCLASSEX wcex;


    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = NULL;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
```

有关此结构的字段解释，请参见 WNDCLASSEX。

2. 现在已经创建了窗口类，接下来您必须注册它。使用 RegisterClassEx 函数，并将窗口类结构作为参数传递：

复制代码

```
    if (!RegisterClassEx(&wcex))
    {
        MessageBox(NULL,
            _T("Call to RegisterClassEx failed!"),
            _T("Win32 Guided Tour"),
            NULL);

        return 1;
    }
```

3. 现在已经注册了您自己的类，接下来创建窗口。使用 CreateWindow 函数，如下所示：

复制代码

```
static TCHAR szWindowClass[] = _T("win32app");
static TCHAR szTitle[] = _T("Win32 Guided Tour Application");
// The parameters to CreateWindow explained:
// szWindowClass: the name of the application
// szTitle: the text that appears in the title bar
// WS_OVERLAPPEDWINDOW: the type of window to create
// CW_USEDEFAULT, CW_USEDEFAULT: initial position (x, y)
// 500, 100: initial size (width, length)
// NULL: the parent of this window
// NULL: this application dows not have a menu bar
// hInstance: the first parameter from WinMain
// NULL: not used in this application
HWND hWnd = CreateWindow(
```

```
    szWindowClass,
    szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    500, 100,
    NULL,
    NULL,
    hInstance,
    NULL
);
if (!hWnd)
{
    MessageBox(NULL,
        _T("Call to CreateWindow failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}
```

此函数返回 HWND，它是某个窗口的句柄。有关更多信息，请参见 Windows 数据类型。

4. 创建了窗口后，我们可以使用以下代码将其显示在屏幕上：

复制代码

```
// The parameters to ShowWindow explained:
// hWnd: the value returned from CreateWindow
// nCmdShow: the fourth parameter from WinMain
ShowWindow(hWnd,
    nCmdShow);
UpdateWindow(hWnd);
```

到目前为止，此窗口还不会显示，因为我们尚未实现 **WndProc** 函数。

5. **WinMain** 的最后一步是消息循环。此循环的用途是侦听操作系统发送的消息。应用程序收到消息后，将该消息调度到 **WndProc** 函数，以便进行处理。消息循环类似于：

复制代码

```
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
```

有关消息循环中使用的结构和函数的更多信息，请参见 MSG、GetMessage、TranslateMessage 和 DispatchMessage。

您刚才完成的步骤为大多数 Win32 应用程序所共用。有关此应用程序所需要的 include 指令和全局变量声明，请参见本主题末尾的完整代码示例。

此时，**WinMain** 函数应该与下面的内容类似：

复制代码

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
```

```
wcex.cbClsExtra     = 0;
wcex.cbWndExtra     = 0;
wcex.hInstance      = hInstance;
wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName   = NULL;
wcex.lpszClassName  = szWindowClass;
wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));

if (!RegisterClassEx(&wcex))
{
    MessageBox(NULL,
        _T("Call to RegisterClassEx failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}

hInst = hInstance; // Store instance handle in our global variable

// The parameters to CreateWindow explained:
// szWindowClass: the name of the application
// szTitle: the text that appears in the title bar
// WS_OVERLAPPEDWINDOW: the type of window to create
// CW_USEDEFAULT, CW_USEDEFAULT: initial position (x, y)
// 500, 100: initial size (width, length)
// NULL: the parent of this window
// NULL: this application dows not have a menu bar
// hInstance: the first parameter from WinMain
// NULL: not used in this application
```

```
HWND hWnd = CreateWindow(
    szWindowClass,
    szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    500, 100,
    NULL,
    NULL,
    hInstance,
    NULL
);

if (!hWnd)
{
    MessageBox(NULL,
        _T("Call to CreateWindow failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}

// The parameters to ShowWindow explained:
// hWnd: the value returned from CreateWindow
// nCmdShow: the fourth parameter from WinMain
ShowWindow(hWnd,
    nCmdShow);
UpdateWindow(hWnd);

// Main message loop:
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
```

```
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
}
```

向 WndProc 添加功能

1. **WndProc** 函数的用途是处理应用程序接收的消息。通常使用 Switch 函数实现此操作。

   我们将处理的第一个消息是 WM_PAINT 消息。当必须更新应用程序窗口的一部分时，应用程序会收到此消息。首次创建窗口时，必须更新整个窗口，并传递此消息以指示此操作。

   当处理 **WM_PAINT** 消息时，首先应做的是调用 BeginPaint，最后应做的是调用 EndPaint。在这两个函数调用之间，您可以处理所有的逻辑，以在窗口中排列文本、按钮和其他控件。对于此应用程序，我们在窗口中显示字符串“Hello, World!”。若要显示文本，请使用 TextOut 函数，如下所示：

   复制代码

```
PAINTSTRUCT ps;
HDC hdc;
TCHAR greeting[] = _T("Hello, World!");

switch (message)
{
case WM_PAINT:
    hdc = BeginPaint(hWnd, &ps);

    // Here your application is laid out.// For this introduction, we just print out "Hello,
World!"
    // in the top left corner.TextOut(hdc,
```

```
        5, 5,
        greeting, _tcslen(greeting));
    // End application-specific layout section.EndPaint(hWnd, &ps);
    break;
}
```

2. 应用程序通常会处理许多其他消息，如 WM_CREATE 和 WM_DESTROY。一个简单而完整的 **WndProc** 函数如下：

复制代码

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR greeting[] = _T("Hello, World!");

    switch (message)
    {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);

        // Here your application is laid out.// For this introduction, we just print out
"Hello, World!"
        // in the top left corner.TextOut(hdc,
            5, 5,
            greeting, _tcslen(greeting));
        // End application specific layout section.EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
```

```
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        break;
    }

    return 0;
}
```

示例

说明

完成所有步骤之后，代码应该与下面的内容类似：若要生成应用程序，请从**“生成”**菜单选择**“生成解决方案”**。如果应用程序编译时没有任何错误，您可以通过按 F5 来运行该应用程序。在屏幕的左上角附近将显示带有文本“Hello, World!”的简单窗口。

代码

复制代码

```
// GT_HelloWorldWin32.cpp
// compile with: /D_UNICODE /DUNICODE /DWIN32 /D_WINDOWS /c

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>

// Global variables

// The main window class name.static TCHAR szWindowClass[] = _T("win32app");

// The string that appears in the application's title bar.static TCHAR szTitle[] = _T("Win32 Guided Tour Application");

HINSTANCE hInst;

// Forward declarations of functions included in this code module:
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
```

```
WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);
wcex.style          = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc    = WndProc;
wcex.cbClsExtra     = 0;
wcex.cbWndExtra     = 0;
wcex.hInstance      = hInstance;
wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName   = NULL;
wcex.lpszClassName  = szWindowClass;
wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));

if (!RegisterClassEx(&wcex))
{
    MessageBox(NULL,
        _T("Call to RegisterClassEx failed!"),
        _T("Win32 Guided Tour"),
        NULL);

    return 1;
}

hInst = hInstance; // Store instance handle in our global variable

// The parameters to CreateWindow explained:
// szWindowClass: the name of the application
// szTitle: the text that appears in the title bar
// WS_OVERLAPPEDWINDOW: the type of window to create
// CW_USEDEFAULT, CW_USEDEFAULT: initial position (x, y)
// 500, 100: initial size (width, length)
// NULL: the parent of this window
// NULL: this application does not have a menu bar
// hInstance: the first parameter from WinMain
// NULL: not used in this application
HWND hWnd = CreateWindow(
    szWindowClass,
    szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    500, 100,
    NULL,
```

```
        NULL,
        hInstance,
        NULL
    );

    if (!hWnd)
    {
        MessageBox(NULL,
            _T("Call to CreateWindow failed!"),
            _T("Win32 Guided Tour"),
            NULL);

        return 1;
    }

    // The parameters to ShowWindow explained:
    // hWnd: the value returned from CreateWindow
    // nCmdShow: the fourth parameter from WinMain
    ShowWindow(hWnd,
        nCmdShow);
    UpdateWindow(hWnd);

    // Main message loop:
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE:  Processes messages for the main window.//
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
```

```
    HDC hdc;
    TCHAR greeting[] = _T("Hello, World!");

    switch (message)
    {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);

        // Here your application is laid out.// For this introduction, we just print out "Hello,
World!"
        // in the top left corner.TextOut(hdc,
            5, 5,
            greeting, _tcslen(greeting));
        // End application-specific layout section.EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        break;
    }

    return 0;
}
```

3.2 通过使用.NETFramework 创建 Windows 窗体应用程序 (C++)

使用 Visual C++ 开发 Windows 窗体项目，通常与使用任何其他 .NET 语言（如 Visual Basic 或 Visual C#）进行开发并无不同。

使用 Visual C++ 编写的 Windows 窗体应用程序通过新的 Visual C++ 语法使用 .NET Framework 类和其他 .NET 功能。有关更多信息，请参见 Language Features for Targeting the CLR。

在本过程中，您将使用工具箱中的几种标准控件创建 Windows 窗体应用程序。用户可以在完成后的应用程序中选择一个日期，此时将出现一个文本标签，显示用户选择的日期。

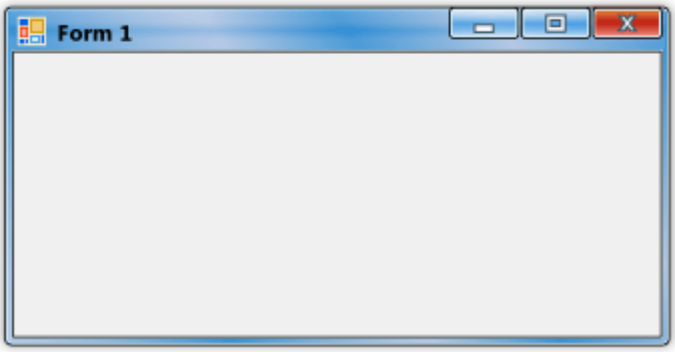
说明

对于在以下说明中使用的某些 Visual Studio 用户界面元素，您的计算机可能会显示不同的名称或位置。这些元素取决于您所使用的 Visual Studio 版本和您所使用的设置。有关更多信息，请参见 使用设置。

创建新的 Windows 窗体项目

1. 在**“文件”**菜单上，单击**“新建”**，然后单击**“项目”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**节点中的**“CLR”**，然后在**“模板”**窗格中选择**“Windows 窗体应用程序”**。

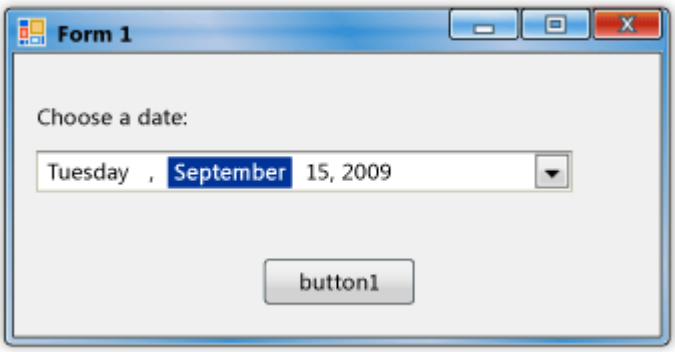
   键入项目的名称，如“winformsapp”。您可以接受默认位置、键入一个位置或者导航到要保存项目的目录。
3. 随即打开 Windows 窗体设计器，显示所创建项目的**“Form1”**，如下所示：



向窗体添加控件

1. 如果看不到**“工具箱”**窗口，请在**“视图”**菜单上单击**“工具箱”**。
2. 将**“工具箱”**中的三个控件放到**“Form1”**设计图面上：
   a. 将一个 Label 控件拖动到靠近**“Form1”**左上角的位置。
   b. 将一个 DateTimePicker 控件拖动到 Label 控件正下方。
   c. 将一个 Button 控件拖动到窗体底部靠近中心点的位置。

   窗体应该与下面的内容类似：



设置窗体和控件的属性

1. 单击窗体图面上的空白区域以选择窗体。
2. 如果没有显示**“属性窗口”**，请单击**“视图”**菜单上的**“属性窗口”**（或按 F4）。

   您可能需要关闭**“工具箱”**以获得更多空间。
3. 设置窗体的**“Text”**属性（显示在窗体标题栏中），方法是在**“属性窗口”**中**“Text”**属性的右侧单击，

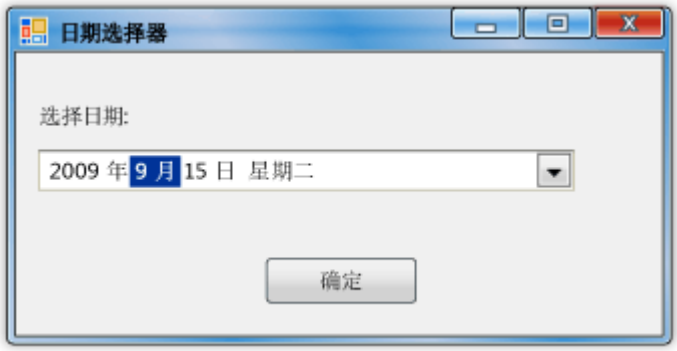
   并键入：

   日期选择器
4. 单击以选择标签，将其**“Text”**属性设置为：

   “选择日期：”。
5. 单击以选择按钮，将其**“Text”**属性设置为：

   “确定”。

   窗体应该与下面的内容类似：



编写事件处理程序代码

在本节中，您将编写在发生以下事件时运行的代码：

- Button 控件上的 Click 事件。
- DateTimePicker 控件上的 ValueChanged 事件。

编写代码以处理事件

1. 双击按钮以添加按钮 Click 事件处理程序（按钮的默认事件为 Click 事件）。

   在选项卡页的编辑区域中显示的窗体的“代码”视图中，此操作创建了一个空事件处理程序方法。

   说明

   还将一行代码添加到 **InitializeComponent** 函数中，此函数创建事件处理程序，并将其分配给与

控件相关联的“单击”字段。如果您双击“设计”视图中的控件以添加相关代码，然后决定稍后移除它，则删除两个添加项（不仅仅是空的事件处理程序）。

2. 将光标移动到 **button1_Click** 方法的左大括号之后，并键入在发生该事件时运行的以下代码：

```
Application::Exit();
```

在键入范围解析运算符 (::) 之后，IntelliSense 将显示可能的有效选项的列表。您可以从该列表中选择一个选项并按 **Tab**，双击它，或者继续键入。

3. 返回“设计”视图，方法是单击编辑区域中的**“Form1.h [设计]”**选项卡，或者单击**“视图”**菜单上的**“设计器”**。
4. 单击 DateTimePicker 控件。
5. 若要向 DateTimePicker 控件添加 ValueChanged 事件处理程序，请单击**“属性”**窗口中的闪电形图标，显示该控件的事件。
6. 双击**“ValueChanged”**事件，在“代码”视图中生成一个空事件处理程序。

说明

ValueChanged 是 DateTimePicker 控件的默认事件。因此，您还可以双击 DateTimePicker 控件，以生成空事件处理程序。

7. 将光标移动到 **dateTimePicker1_ValueChanged** 方法的左大括号之后，按 Enter 键，并键入在发生该事件时运行的以下代码：

```
label1->Text=String::Format("New date: {0}", dateTimePicker1->Text);
```

当应用程序的用户选择了新的日期时，标签的 Text 属性将设置为后跟 DateTimePicker 的 Text 属性的字符串 `"New date:"`。

Visual Studio 提供了几个可以简化代码键入的功能：

- 当键入箭头运算符 (->) 时，IntelliSense 将显示可从中选择的有效选项列表。
- 当键入方法的左括号时，将出现一个工具提示窗口，其中显示该方法的各个重载的有效参数。要查看不同的重载，请使用向上键或向下键。
- 自动完成可以根据您已键入的部分完成变量名或成员的键入。例如，如果键入了 `String::Fo` 并按 Ctrl-空格键或 Tab，Visual Studio 将自动完成键入 `String::Format`。

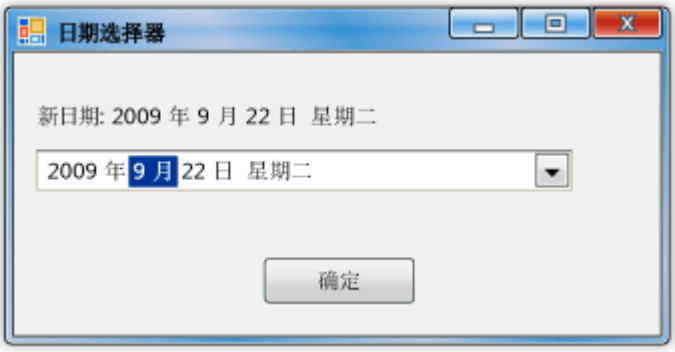
生成并运行程序

1. 在**“生成”**菜单中，单击**“生成解决方案”**。

如果存在错误，请单击**“输出”**窗口中的**“转到下一条消息”**按钮。错误消息文本显示在状态栏中。

您可以双击任何错误，转到源代码中包含该错误的行。

2. 在**“调试”**菜单中，单击**“不进行调试直接运行”**。将显示您生成的应用程序。
3. 测试该应用程序，方法是单击 DateTimePicker 上的向下箭头，选择一个日期。标签文本更改为显示所选的日期，如下所示：



4. 您可以向此应用程序添加更多功能，如菜单、其他窗体和帮助文件。不要畏惧实验。

3.3 创建 Windows 窗体控件 (C++)

Windows 窗体控件是可以添加到 Windows 窗体应用程序（面向公共语言运行时的 GUI 应用程序）的组件。使用 Visual C++ 编写的 Windows 窗体应用程序通过新的 Visual C++ 语法使用 .NET Framework 类和其他 .NET 功能。

在本过程中，您将创建显示数字的 Windows 窗体控件。用户每次单击应用程序中的标签时将递增此数字。您还将创建一个 Windows 窗体应用程序项目来测试该控件。

本演练涵盖以下内容：

- 创建新项目。
- 设计控件。
- 向控件添加自定义属性。
- 添加用于测试控件的项目。
- 将控件放在应用程序中。
- 运行应用程序。

创建新项目

在本节中，您将使用“Windows 窗体控件”项目模板创建一个用户控件，该控件是一个包含其他控件的复合控件。

您也可以通过直接从 Control 类（代码负责绘制控件）或者 Component 类（无 UI 的控件）派生一个类来创建 Windows 窗体控件。

创建新的 Windows 窗体控件项目

1. 在**“文件”**菜单上，单击**“新建”**，再单击**“项目...”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**节点中的**“CLR”**，然后在**“Visual Studio 已安装的模板”**窗格中选择**“Windows 窗体控件库”**。

   键入项目的名称，如“clickcounter”。

   为解决方案键入一个不同的名称，如“controlandtestapp”。

   您可以接受默认位置、键入所需的位置或者导航到要保存项目的目录。
3. Windows 窗体设计器将打开并显示一个区域，您可以将要放置到控件设计图面上的控件添加到该区域中。

设计控件

在本步骤中，您要将一个 Label 控件添加到控件设计图面中。然后，设置控件本身及其包含的 Label 控件的一些属性。

设置用户控件的属性

1. 如果没有显示**“属性”**窗口，请单击**“视图”**菜单上的**“属性窗口”**。

   在 Windows 窗体设计器中，单击控件将其选中并按如下方式设置它的属性：

   - 将**“Size”**属性设置为“100, 100”。
   - 将**“BorderStyle”**设置为**“Fixed3D”**。

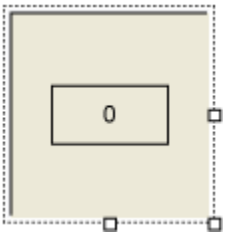
     将控件放置到应用程序中后，将显示标签的边框。
2. 如果**“工具箱”**窗口不可见，请从**“视图”**菜单中选择**“工具箱”**。

   将一个 Label 控件从**“工具箱”**拖动到设计图面上，将其放置在靠近控件中心的位置。

   设置标签的下列属性：

   - 将**“BorderStyle”**设置为**“FixedSingle”**。
   - 将**“Text”**设置为数字**“0”**（零）。
   - 将**“Autosize”**设置为**“False”**。
   - 将**“Size”**设置为**“30, 20”**。
   - 将**“TextAlign”**设置为**“MiddleCenter”**。

   保留**“Name”**属性（在代码中将使用它来引用该控件）为**“label1”**。该控件应如下所示：



3. 通过双击标签，为标签的 Click 事件（标签的默认事件）添加事件处理程序。
4. **clickcounter.h** 文件将显示在编辑区域中，并附带一个空事件处理程序方法。

> **说明**
>
> 如果需要更多空间，可以关闭“**工具箱**”或“**属性**”窗口，方法是单击相应的“关闭”框，或者解除窗口锁定使其自动隐藏。

5. 将光标移动到 **label1_Click** 方法的左大括号之后，按 Enter 并键入以下内容：

复制代码

```
int temp = System::Int32::Parse(label1->Text);

temp++;

label1->Text = temp.ToString();
```

在键入范围解析运算符 (::)、点运算符 (.)或箭头运算符 (->) 之后，IntelliSense 将显示有效选项的列表。您可以通过突出显示某个项并按 Tab 或 Enter，或者通过双击某个项，将该项插入代码中。

此外，当键入方法的左括号时，Visual Studio 将显示该方法的每个重载的有效参数类型。

向控件添加自定义属性

在本步骤中，您将定义一个自定义属性，它确定控件上显示的数字是在用户单击标签时递增还是在用户单击控件上的任何位置时递增。

向控件添加自定义属性

1. 将光标放置在 **clickcounterControl.h** 文件顶部的第一个 **public** 范围指示符 `public:` 的冒号之后，按 Enter，然后键入以下内容：

复制代码

```
property bool ClickAnywhere {
```

```
    bool get() {
        return (label1->Dock == DockStyle::Fill);
    }
    void set(bool val) {
        if (val)
            label1->Dock = DockStyle::Fill;
        else
            label1->Dock = DockStyle::None;
    }
}
```

当控件的 **ClickAnywhere** 属性设置为 **true** 时，标签的 Dock 属性将设置为 **DockStyle::Fill**，该标签将占据整个控件图面。单击控件图面上的任何位置将引发标签的 Click 事件，使标签上的数字递增。

当 **ClickAnywhere** 属性为 **false**（默认值）时，标签的 Dock 属性将设置为 **DockStyle::None**。标签不填充整个控件，并且单击控件时，必须单击标签边框内部才会引发标签的 Click 事件，使数字递增。

2. 生成用户控件。在**“生成”**菜单上，选择**“生成解决方案”**。

   如果没有错误，将生成文件名为 **clickcounter.dll** 的 Windows 窗体控件。您可以在项目目录结构中找到此文件。

添加用于测试控件的项目

在本步骤中，要创建一个 Windows 窗体应用程序项目，您将在其中的一个窗体上放置“clickcounter”控件的实例。

说明

您所创建的用于测试控件的 Windows 窗体应用程序可以使用 Visual C++ 或其他 .NET 语言（如 C# 或 Visual Basic）来编写。

创建 Windows 窗体应用程序项目

- 在**“文件”**菜单上选择**“新建”**，然后单击**“项目...”**。

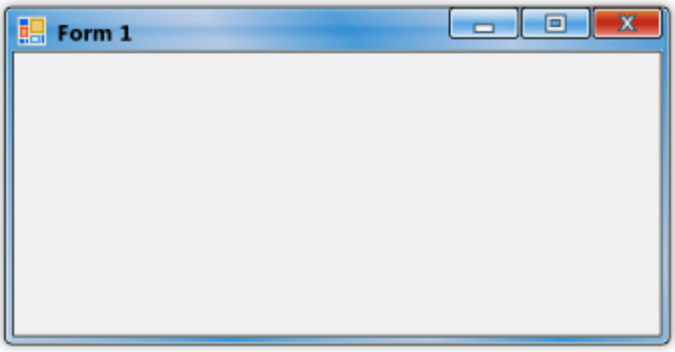
也可以通过以下方法将项目添加到解决方案中：右击**“解决方案资源管理器”**中的 controlandtestapp 解决方案，指向**“添加”**，然后单击**“新建项目...”**。

1. 在**“项目类型”**窗格中，选择**“Visual C++”**节点中的**“CLR”**，然后在**“Visual Studio 已安装的模板”**窗格中选择**“Windows 窗体应用程序”**。

键入项目的名称，如“testapp”。

确保选择**“添入解决方案”**，而不是接受**“解决方案”**下拉列表中默认的**“创建新解决方案”**设置，然后单击**“确定”**。

2. 将为新项目打开 Windows 窗体设计器，其中显示一个名为**“Form1”**的新窗体，如下图所示：



将控件添加到工具箱

1. 添加对控件的引用。右击**“解决方案资源管理器”**中的 testapp 项目，然后单击**“引用”**。

   单击**“添加新引用”**按钮，单击**“项目”**选项卡（这是在此解决方案中添加对另一个项目的引用），然后选择 clickcounter 项目。单击**“确定”**两次。

2. 如果看不到**“工具箱”**窗口，请从**“视图”**菜单中选择**“工具箱”**。

3. 如果在工具箱中找不到带有“齿轮”图标的 clickCounter 控件，则右击**“工具箱”**，然后单击**“选择项”**。

   单击**“浏览”**按钮，定位到解决方案目录结构中的 **clickcounter.dll 文件。**选择该文件并单击**“打开”**。

   clickcounter 控件即出现在**“.NET Framework 组件”**列表中，并带有一个选中标记。单击**“确定”**。

   控件即显示在**“工具箱”**中，带有默认的“齿轮”图标。

将控件放在应用程序中

在本步骤中，您要将控件的两个实例放到应用程序窗体上并设置其属性。

将控件的实例放置到窗体上

1. 从**“工具箱”**拖出 clickcounte 控件的两个实例。将它们放在窗体上，避免使其重叠。

   如果需要加宽窗体，请单击以选择窗体，向外拖动一个选择手柄。

2. 如果看不到**“属性”**窗口，请从**“视图”**菜单选择**“属性”**。

如果属性是按类别组织的，**“ClickAnywhere”**属性将位于**“属性窗口”**的**“杂项”**部分。

3. 单击以选择窗体上的一个 clickcounter 控件实例，然后将其**“ClickAnywhere”**属性设置为 **true**。
4. 将 clickcounter 控件的另一个实例的**“ClickAnywhere”**属性设置为 **false**（默认值）。
5. 在解决方案资源管理器中，右击 testapp 项目，并选择**“设为启动项目”**。
6. 从**“生成”**菜单中选择**“重新生成解决方案”**。

   您应当看到生成了两个项目，并且没有出现错误。

运行应用程序

在本步骤中，您将运行应用程序，并单击控件测试它们。

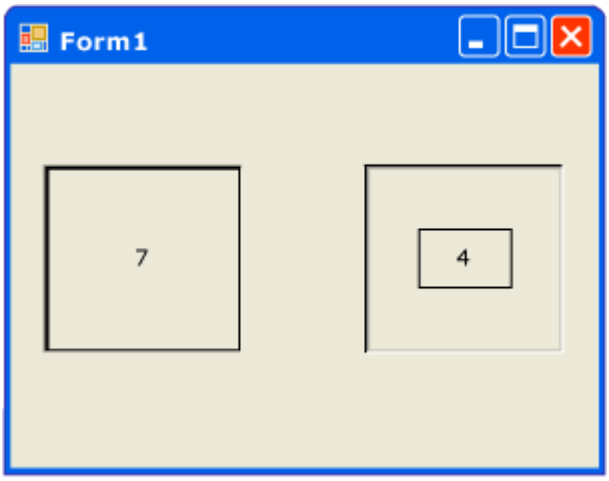
测试应用程序

1. 从**“调试”**菜单中选择**“启动调试”**。

   将显示窗体，其中控件的两个实例都可见。
2. 运行应用程序并单击两个“clickcounter”控件：
   - 单击**“ClickAnywhere”**设置为 **true** 的控件。

     单击控件上的任何位置，标签上的数字都会递增。
   - 单击**“ClickAnywhere”**设置为 **false** 的控件。

     仅当在标签的可见边框内单击时，标签上的数字才会递增。下面的屏幕快照演示了应用程序在单击数次后的外观情况：



1. 单击“Form1”窗口右上角的“关闭”框，关闭测试应用程序。

3.4 使用 DirectX 创建游戏 (C++)

由于 C++ 非常强大和灵活，所以它是创建游戏的优秀语言。 通过使用 Visual C++ 和 DirectX，您可以用本机代码或托管代码编写游戏。 此灵活性允许您在最熟悉的平台上创建游戏。
创建一个好游戏不是一件轻而易举的事，这不在本指导教程的范围内。 如果要创建游戏，请查看下面的链接，其中的信息可以帮助您创建第一个游戏。

图形编程入门

1. 若要使用 DirectX 创建游戏，您必须从以下位置安装 DirectX SDK：DirectX Developer Center（DirectX 开发人员中心）。 安装了 SDK 后，您会发现几个示例，这些示例将帮助您了解 DirectX 编程的入门知识。
2. 在 MSDN 上查看 Visual C++ Express Edition 网页，获得可以下载、学习和随意修改的现有游戏。在该页面上，您可以从 Microsoft Research 下载游戏，甚至可以下载流行游戏 Quake II .NET 的完整源代码。

4.创建可重用代码（C++）

现在我们学习了如何使用 Visual Studio IDE 以及如何创建命令行应用程序和 Windows 应用程序，下面我们将学习如何编写代码，以便可以让多个应用程序使用该代码。 执行此操作的一种方法是创建包含相关类和算法的库。 例如，Visual C++ 附带了许多任何 C 或 C++ 应用程序都可以使用的库，如 C 运行库和标准 C++ 库。 如果没有这些库，则 C 或 C++ 应用程序没有写入控制台或确定当前日期和时间的标准方法。
每个 C 或 C++ 应用程序都可能用到前面提到的库之一。您还可以创建任何应用程序都可以使用的自己的类和算法库。 使用 Visual C++，您可以创建三种类型的库：

- 动态链接库 (DLL)。
- 静态库。
- 托管程序集。

通常，如果创建可供本机 C++ 代码使用的库，则可以创建动态链接库或静态库。 有关如何确定应创建何种类型库的更多信息，请参见 DLL。 如果要创建可供 C++/CLI 或任何其他 .NET 语言（如 C# 或 Visual Basic）使用的库，则应创建托管程序集。
在本部分中，我们将创建简单的标准数学运算（如加法和乘法）库，并将演示应用程序如何使用此库。

4.1 创建和使用动态链接库 (C++)

我们将创建的第一种类型的库是动态链接库 (DLL)。使用 DLL 是一种重用代码的绝佳方式。您不必在自己创建的每个程序中重新实现同一例程，而只需对这些例程编写一次，然后从需要该功能的应用程序引用它们即可。

本演练涵盖以下内容：

- 创建新的动态链接库 (DLL) 项目。
- 向动态链接库添加类。
- 创建引用动态链接库的应用程序。
- 在控制台应用程序中使用类库的功能。
- 运行应用程序。

创建新的动态链接库（DLL）项目

1. 从**“文件”**菜单中，选择**“新建”**，然后选择**“项目...”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“Win32”**。
3. 在**“模板”**窗格中，选择**“Win32 控制台应用程序”**。
4. 为项目选择一个名称，如 MathFuncsDll，并将其键入**“名称”**字段。为解决方案选择一个名称，如 DynamicLibrary，并将其键入**“解决方案名称”**字段。
5. 单击**“确定”**启动 Win32 应用程序向导。在**“Win32 应用程序向导”**对话框的**“概述”**页中，单击**“下一步”**。
6. 在**“Win32 应用程序向导”**中的**“应用程序设置”**页中，选择**“应用程序类型”**下的**“DLL”**（如果可用），或者选择**“控制台应用程序”**（如果**“DLL”**不可用）。某些版本的 Visual Studio 不支持通过使用向导创建 DLL 项目。您可以稍后对此进行更改，以将项目编译为 DLL。
7. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，选择**“附加选项”**下的**“空项目”**。
8. 单击**“完成”**创建项目。

向动态链接库添加类

1. 若要为新类创建头文件，请从**“项目”**菜单中选择**“添加新项...”**。将显示**“添加新项”**对话框。在**“类别”**窗格中，选择**“Visual C++”**下的**“代码”**。在**“模板”**窗格中选择**“头文件(.h)”**。为头文件选择一个名称，如 MathFuncsDll.h，并单击**“添加”**。将显示一个空白文件。
2. 添加一个名为“MyMathFuncs”的简单类，以执行常见的算术运算，如加、减、乘和除。代码应与以下内容类似：

复制代码

```
// MathFuncsDll.h

namespace MathFuncs
{
    class MyMathFuncs
    {
    public:
        // Returns a + b
        static __declspec(dllexport) double Add(double a, double b);

        // Returns a - b
        static __declspec(dllexport) double Subtract(double a, double b);

        // Returns a * b
        static __declspec(dllexport) double Multiply(double a, double b);

        // Returns a / b
        // Throws DivideByZeroException if b is 0
        static __declspec(dllexport) double Divide(double a, double b);
    };
}
```

3. 请注意此代码方法声明中的 **__declspec(dllexport)** 修饰符。这些修饰符使 DLL 能够导出该方法以供其他应用程序使用。有关更多信息，请参见 dllexport, dllimport。
4. 若要为新类创建源文件，请从**“项目”**菜单中选择**“添加新项...”**。将显示**“添加新项”**对话框。在**“类别”**窗格中，选择**“Visual C++”**下的**“代码”**。在**“模板”**窗格中，选择**“C++ 文件(.cpp)”**。为源文件选择一个名称，如 MathFuncsDll.cpp，并单击**“添加”**。将显示一个空白文件。
5. 在源文件中实现“MyMathFuncs”的功能。代码应与以下内容类似：

复制代码

```
// MathFuncsDll.cpp
// compile with: /EHsc /LD

#include "MathFuncsDll.h"

#include <stdexcept>

using namespace std;

namespace MathFuncs
{
    double MyMathFuncs::Add(double a, double b)
    {
        return a + b;
    }

    double MyMathFuncs::Subtract(double a, double b)
    {
        return a - b;
    }

    double MyMathFuncs::Multiply(double a, double b)
    {
        return a * b;
    }

    double MyMathFuncs::Divide(double a, double b)
    {
        if (b == 0)
        {
            throw new invalid_argument("b cannot be zero!");
```

```
        }


        return a / b;
    }
}
```

6. 若要将项目生成为 DLL，请从**“项目”**菜单中选择 MathFuncsDll**“属性...”**。在左窗格中，选择**“配置属性”**下的**“常规”**。在右窗格中，将**“配置类型”**更改为**“动态库(.dll)”**。单击**“确定”**保存更改。

   说明

   如果您从命令行生成项目，请使用 **/LD** 编译器选项指定输出文件应为 DLL。有关更多信息，请参见 /MD、/MT、/LD（使用运行库）。

7. 编译该动态链接库，方法是选择**“生成”**菜单中的**“生成解决方案”**。这样就创建了一个可供其他程序使用的 DLL。有关 DLL 的详细信息，请参见 DLL。

创建引用动态链接库的应用程序

1. 若要创建将引用并使用刚创建的动态链接库的应用程序，请从**“文件”**菜单中选择**“新建”**，然后选择**“项目...”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“Win32”**。
3. 在**“模板”**窗格中，选择**“Win32 控制台应用程序”**。
4. 为项目选择一个名称（如 MyExecRefsDll），并将其键入**“名称”**字段。从**“解决方案”**旁边的下拉列表中选择**“添入解决方案”**。这会将新项目添加到该动态链接库所属的同一个解决方案中。
5. 单击**“确定”**启动**“Win32 应用程序向导”**。在**“Win32 应用程序向导”**对话框的**“概述”**页中，单击**“下一步”**。
6. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，选择**“应用程序类型”**下的**“控制台应用程序”**。
7. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，清除**“附加选项”**下的**“预编译头”**复选框。
8. 按**“完成”**创建项目。

在控制台应用程序中使用类库的功能

1. 创建新的控制台应用程序后，将为您创建一个空程序。源文件的名称与您在前面为项目选择的名称相同。在本示例中，名为“MyExecRefsDll.cpp”。

2. 若要使用在动态链接库中创建的算术例程，则必须引用该库。若要执行此操作，请在解决方案资源管理器中选择 MyExecRefsDll 项目，然后从**“项目”**菜单中选择**“引用...”**。在**“属性页”**对话框中，展开**“通用属性”**节点，选择**“框架和引用”**，然后选择**“添加新引用...”**按钮。有关**“引用...”**对话框的更多信息，请参见“<Projectname> 属性页”对话框 ->“通用属性”->“框架和引用”。
3. 显示**“添加引用”**对话框。此对话框列出了所有可以引用的库。**“项目”**选项卡列出了当前解决方案中的所有项目，以及它们包含的所有库。在**“项目”**选项卡中，选择 MathFuncsDll。然后单击**“确定”**。
4. 若要引用动态链接库的头文件，必须修改包含目录路径。为此，请在**“属性页”**对话框中展开**“配置属性”**节点，然后展开**“C/C++”**节点，并选择**“常规”**。在**“附加包含目录”**旁边，键入 MathFuncsDll.h 头文件所在位置的路径。
5. 可执行文件仅在运行时加载动态链接库。必须告诉系统在哪里查找“MathFuncsDll.dll”。您可以通过使用 **PATH** 环境变量做到这一点。为此，请在**“属性页”**对话框中展开**“配置属性”**节点，并选择**“调试”**。在**“环境”**旁边键入以下内容：`PATH=<MathFuncsDll.dll 文件的路径>`，其中 *<MathFuncsDll.dll 文件的路径>* 应替换为 MathFuncsDll.dll 的实际位置。单击**“确定”**保存所有更改。

> **说明**
>
> 如果要从命令行而不是从 Visual Studio 运行可执行文件，则必须在命令提示符处手动更新 **PATH** 环境变量，如下所示：`set PATH=%PATH%;<MathFuncsDll.dll 文件的路径>`，其中 *<MathFuncsDll.dll 文件的路径>* 应替换为 MathFuncsDll.dll 的实际位置。

6. 现在，可以在应用程序中使用“MyMathFuncs”类了。使用以下代码替换“MyExecRefsDll.cpp”的内容：

复制代码

```
// MyExecRefsDll.cpp
// compile with: /EHsc /link MathFuncsDll.lib

#include <iostream>

#include "MathFuncsDll.h"
```

```
using namespace std;


int main()
{
    double a = 7.4;
    int b = 99;


    cout << "a + b = " <<
        MathFuncs::MyMathFuncs::Add(a, b) << endl;
    cout << "a - b = " <<
        MathFuncs::MyMathFuncs::Subtract(a, b) << endl;
    cout << "a * b = " <<
        MathFuncs::MyMathFuncs::Multiply(a, b) << endl;
    cout << "a / b = " <<
        MathFuncs::MyMathFuncs::Divide(a, b) << endl;


    return 0;
}
```

7. 通过从**"生成"**菜单中选择**"生成解决方案"**，生成可执行文件。

运行应用程序

1. 确保选择"MyExecRefsDll"作为默认项目。在**"解决方案资源管理器"**中，选择 MyExecRefsDll，然后选择**"项目"**菜单中的**"设为启动项目"**。
2. 若要运行项目，请选择**"调试"**菜单中的**"开始执行（不调试）"**。输出应该与下面的内容类似：

4.2 创建和使用静态库 (C++)

我们将创建的下一个库类型是静态库 (LIB)。使用静态库是重用代码的一种绝佳方式。您不必在自己创建的每个程序中重新实现同一例程，而只需对这些例程编写一次，然后从需要该功能的应用程序引用它们即可。

本演练涵盖以下内容：

- 创建新的静态库项目。

- 向静态库添加类。
- 创建引用静态库的应用程序。
- 在控制台应用程序中使用静态库的功能。
- 运行应用程序。

创建新的静态库项目

1. 从**“文件”**菜单中，选择**“新建”**，然后选择**“项目...”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“Win32”**。
3. 在**“模板”**窗格中，选择**“Win32 控制台应用程序”**。
4. 为项目选择一个名称（例如 MathFuncsLib），并将该名称输入**“名称”**字段。为解决方案选择一个名称（例如 StaticLibrary），并将该名称输入**“解决方案名称”**字段。
5. 按**“确定”**启动“Win32 应用程序向导”。在**“Win32 应用程序向导”**对话框的**“概述”**页中，按**“下一步”**。
6. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，选择**“应用程序类型”**下的**“静态库”**。
7. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，清除**“附加选项”**下的**“预编译头”**复选框。
8. 按**“完成”**创建项目。

向静态库添加类

1. 若要为新类创建头文件，请从**“项目”**菜单中选择**“添加新项...”**。将显示**“添加新项”**对话框。从**“类别”**窗格中，选择**“Visual C++”**下的**“代码”**。从**“模板”**窗格中选择**“头文件(.h)”**。为头文件选择一个名称（例如 MathFuncsLib.h），并按**“添加”**。将显示一个空白文件。
2. 添加一个名为“MyMathFuncs”的简单类，以执行常见的算术运算，如加、减、乘和除。代码应与以下内容类似：

复制代码

```
// MathFuncsLib.h


namespace MathFuncs
{
    class MyMathFuncs
```

```
    {
    public:
        // Returns a + b
        static double Add(double a, double b);

        // Returns a - b
        static double Subtract(double a, double b);

        // Returns a * b
        static double Multiply(double a, double b);

        // Returns a / b
        // Throws DivideByZeroException if b is 0
        static double Divide(double a, double b);
    };
}
```

3. 若要为新类创建源文件，请从**“项目”**菜单中选择**“添加新项...”**。将显示**“添加新项”**对话框。从**“类别”**窗格中，选择**“Visual C++”**下的**“代码”**。从**“模板”**窗格中，选择**“C++ 文件(.cpp)”**。为源文件选择一个名称（例如 MathFuncsLib.cpp），并按**“添加”**。将显示一个空白文件。
4. 在源文件中实现 MyMathFuncs 的功能。代码应与以下内容类似：

复制代码

```
// MathFuncsLib.cpp
// compile with: /c /EHsc
// post-build command: lib MathFuncsLib.obj

#include "MathFuncsLib.h"

#include <stdexcept>
```

```
using namespace std;


namespace MathFuncs
{
    double MyMathFuncs::Add(double a, double b)
    {
        return a + b;
    }

    double MyMathFuncs::Subtract(double a, double b)
    {
        return a - b;
    }

    double MyMathFuncs::Multiply(double a, double b)
    {
        return a * b;
    }

    double MyMathFuncs::Divide(double a, double b)
    {
        if (b == 0)
        {
            throw new invalid_argument("b cannot be zero!");
        }

        return a / b;
    }
}
```

5. 若要将项目生成为静态库，请从**“项目”**菜单中选择**“MathFuncsLib 属性...”**。在左窗格中，选择**“配置属性”**下的**“常规”**。在右窗格中，将**“配置类型”**更改为**“静态库(.lib)”**。按**“确定”**保存更改。

说明

如果是从命令行生成，必须分两个步骤来生成程序。首先，通过使用带编译器选项 **/c** 的 **Cl.exe** 编译代码 (`cl /c /EHsc MathFuncsLib.cpp`)。这将创建名为“MathFuncsLib.obj”的对象文件。有关更多信息，请参见 /c（在不链接的情况下进行编译）。接着，使用库管理器 **Lib.exe** 链接代码 (`lib MathFuncsLib.obj`)。这将创建静态库“MathFuncsLib.lib”。有关库管理器的更多信息，请参见 LIB 参考。

6. 编译该静态库，方法是选择**“生成”**菜单中的**“生成解决方案”**。这将创建一个可供其他程序使用的静态库。

创建引用静态库的应用程序

1. 若要创建引用并使用刚刚创建的静态库的应用程序，请从**“文件”**菜单中选择**“新建”**，然后选择**“项目...”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“Win32”**。
3. 在**“模板”**窗格中，选择**“Win32 控制台应用程序”**。
4. 为项目选择一个名称（如 MyExecRefsLib），并将其键入**“名称”**字段。从**“解决方案”**旁边的下拉列表中选择**“添加解决方案”**。这会将新项目添加到该静态库所属的同一个解决方案中。
5. 按**“确定”**启动**“Win32 应用程序向导”**。在**“Win32 应用程序向导”**对话框的**“概述”**页中，按**“下一步”**。
6. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，选择**“应用程序类型”**下的**“控制台应用程序”**。
7. 在**“Win32 应用程序向导”**的**“应用程序设置”**页中，清除**“附加选项”**下的**“预编译头”**。
8. 按**“完成”**创建项目。

在控制台应用程序中使用静态库的功能

1. 创建新的控制台应用程序后，该向导将为您创建一个空程序。源文件的名称与您在前面为项目选择的名称相同。在本示例中，名为“MyExecRefsLib.cpp”。
2. 若要使用在静态库中创建的算术例程，必须引用该静态库。为此，请选择**“项目”**菜单中的**“引用...”**。在**“属性页”**对话框中，展开**“通用属性”**节点，并选择**“引用”**。然后选择**“添加新引用...”**按钮。有关**“引用...”**对话框的更多信息，请参见“<Projectname> 属性页”对话框 ->“通用属性”->“框架和引用”。

3. 显示**“添加引用”**对话框。此对话框列出了所有可以引用的库。**“项目”**选项卡列出了当前解决方案中的所有项目，以及它们包含的所有库。在**“项目”**选项卡中，选择 MathFuncsLib。然后选择**“确定”**。
4. 若要引用静态库的头文件，必须修改包含目录路径。为此，请在**“属性页”**对话框中，展开**“配置属性”**节点，然后展开**“C/C++”**节点，并选择**“常规”**。在**“附加包含目录”**旁边，键入 MathFuncsLib.h 头文件所在位置的路径。
5. 现在即可在此应用程序中使用 MyMathFuncs 类。使用以下代码替换“MyExecRefsLib.cpp”的内容：

复制代码

```
// MyExecRefsLib.cpp
// compile with: /EHsc /link MathFuncsLib.lib

#include <iostream>

#include "MathFuncsLib.h"

using namespace std;

int main()
{
    double a = 7.4;
    int b = 99;

    cout << "a + b = " <<
        MathFuncs::MyMathFuncs::Add(a, b) << endl;
    cout << "a - b = " <<
        MathFuncs::MyMathFuncs::Subtract(a, b) << endl;
    cout << "a * b = " <<
        MathFuncs::MyMathFuncs::Multiply(a, b) << endl;
```

```
    cout << "a / b = " <<
        MathFuncs::MyMathFuncs::Divide(a, b) << endl;

    return 0;
}
```

6. 通过从**“生成”**菜单中选择**“生成解决方案”**，生成可执行文件。

运行应用程序

1. 确保选择“MyExecRefsLib”作为默认项目。在**“解决方案资源管理器”**中，选择 MyExecRefsLib，然后从**“项目”**菜单中选择**“设为启动项目”**。
2. 若要运行项目，请选择**“调试”**菜单中的**“开始执行（不调试）”**。输出应该与下面的内容类似：

4.3 创建和使用托管程序集 (C++)

托管程序集是一种库，您可以创建该库以便高效地重用代码。这样，就不必在多个程序中重新实现同样的例程，而只需编写这些例程一次，然后在需要该功能的应用程序中引用它们即可。
本演练涵盖以下任务：

- 创建一个类库项目。
- 向该类库添加类。
- 创建引用该类库的应用程序。
- 在应用程序中使用类库的功能。
- 运行应用程序。

系统必备
若要完成本演练，您必须了解 C++ 语言的基础知识。如果您刚刚开始学习 C++，建议阅读 Herb Schildt 编写的“C++ Beginner's Guide”（《C++ 初学者指南》），MSDN 网站上的 Beginner Developer Learning Center（入门开发人员学习中心）提供了该指南。

创建类库项目

1. 在**“文件”**菜单上指向**“新建”**，然后单击**“项目”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“CLR”**。

   此组中的每个项目类型都将创建一个面向公共语言运行时 (CLR) 的项目。
3. 在**“模板”**窗格中，选择**“类库”**。

4. 在**“名称”**框中键入项目的名称，例如，MathFuncsAssembly。在**“解决方案名称”**字段中键入解决方案的名称，例如，ManagedAssemblies。
5. 单击**“确定”**创建项目。
6. 默认情况下，在创建项目时，会将项目设置为使用预编译头。若要为 MathFuncsAssembly 项目禁用预编译头，请在**“解决方案资源管理器”**中选择项目，然后在**“项目”**菜单上，单击**“属性”**。依次展开**“配置属性”**节点和**“C/C++”**节点，然后选择**“预编译头”**。在**“创建/使用预编译头”**旁边的列表中，选择**“不使用预编译头”**。单击**“确定”**保存这些更改。有关更多信息，请参见创建预编译的头文件。

向类库添加类

1. 在您创建 CLR 类库后，向导将为您生成一个基本类。生成的头文件和源文件的名称均与您在创建项目时为项目指定的名称相同。在本示例中，它们的名称为“MathFuncsAssembly.h”和“MathFuncsAssembly.cpp”。
2. 通过使用一个名为 **MyMathFuncsAssembly** 的基本类，替换 MathFuncsAssembly.h 中的现有代码。此类执行一些常见的算术运算，例如加、减、乘和除。代码应与下面的示例类似。

复制代码

```
// MathFuncsAssembly.h

using namespace System;

namespace MathFuncs
{
    public ref class MyMathFuncs
    {
    public:
        // Returns a + b
        static double Add(double a, double b);

        // Returns a - b
```

```
        static double Subtract(double a, double b);

        // Returns a * b
        static double Multiply(double a, double b);

        // Returns a / b
        // Throws DivideByZeroException if b is 0
        static double Divide(double a, double b);
    };
}
```

3. 在源文件中实现 **MyMathFuncs** 的功能。代码应与下面的示例类似。

复制代码

```
// MathFuncsAssembly.cpp
// compile with: /clr /LD

#include "MathFuncsAssembly.h"

namespace MathFuncs
{
    double MyMathFuncs::Add(double a, double b)
    {
        return a + b;
    }

    double MyMathFuncs::Subtract(double a, double b)
    {
        return a - b;
    }
```

```
    double MyMathFuncs::Multiply(double a, double b)
    {
        return a * b;
    }

    double MyMathFuncs::Divide(double a, double b)
    {
        if (b == 0)
        {
            throw gcnew DivideByZeroException("b cannot be zero!");
        }

        return a / b;
    }
}
```

4. 编译该类库，方法是在**“生成”**菜单中单击**“生成解决方案”**。这将创建一个可供其他程序使用的动态链接库 (DLL)。有关 DLL 的更多信息，请参见 DLL。

创建引用类库的控制台应用程序

1. 在**“文件”**菜单上指向**“新建”**，然后单击**“项目”**。
2. 在**“项目类型”**窗格中，选择**“Visual C++”**下的**“CLR”**。
3. 在**“模板”**窗格中，选择**“CLR 控制台应用程序”**。
4. 在**“名称”**框中键入项目的名称，例如，MyExecRefsAssembly。在**“解决方案”**旁边的列表中，选择**“添入解决方案”**以将新项目添加到包含类库的解决方案中。
5. 单击**“确定”**创建项目。
6. 在**“解决方案资源管理器”**中选中 MyExecRefsAssembly 项目，然后在**“项目”**菜单上，单击**“属性”**，为该项目禁用预编译头。依次展开**“配置属性”**节点和**“C/C++”**节点，然后选择**“预编译头”**。在**“创建/使用预编译头”**旁边的列表中，选择**“不使用预编译头”**。单击**“确定”**保存这些更改。

在控制台应用程序中使用类库的功能

1. 在您创建 CLR 控制台应用程序后，向导将生成一个仅向控制台写入“Hello World”的程序。生成的源文件的名称与您在创建项目时为项目指定的名称相同。在本示例中，名称为“MyExecRefsAssembly.cpp”。
2. 若要使用在类库中创建的算术例程，必须引用类库。为此，请在**“解决方案资源管理器”**中选择 MyExecRefsAssembly 项目，然后在**“项目”**菜单上，单击**“属性”**。在**“属性页”**对话框中展开**“通用属性”**节点，选择**“框架和引用”**，然后单击**“添加新引用”**。有关更多信息，请参见“<Projectname> 属性页”对话框 ->“通用属性”->“框架和引用”。
3. **“添加引用”**对话框列出了所有可以引用的库。**“.NET”**选项卡列出了 .NET Framework 附带的库。**“COM”**选项卡列出了计算机上的所有 COM 组件。**“项目”**选项卡列出了当前解决方案中的所有项目，以及它们包含的所有库。在**“项目”**选项卡上，选择**“MathFuncsAssembly”**，然后单击**“确定”**。

   说明

   通过包括 **#using** 指令（例如 `#using <MathFuncsAssembly.dll>`），您可以直接从源文件引用程序集。有关更多信息，请参见 The #using Directive。
4. 现在即可在此应用程序中使用 **MyMathFuncs** 类。在 MyExecRefsAssembly.cpp 中，使用下面的代码替换文件函数的内容。

复制代码

```
// MyExecRefsAssembly.cpp
// compile with: /clr /FUMathFuncsAssembly.dll

using namespace System;

int main(array<System::String ^> ^args)
{
    double a = 7.4;
    int b = 99;
```

```
    Console::WriteLine("a + b = {0}",
        MathFuncs::MyMathFuncs::Add(a, b));
    Console::WriteLine("a - b = {0}",
        MathFuncs::MyMathFuncs::Subtract(a, b));
    Console::WriteLine("a * b = {0}",
        MathFuncs::MyMathFuncs::Multiply(a, b));
    Console::WriteLine("a / b = {0}",
        MathFuncs::MyMathFuncs::Divide(a, b));

    return 0;
}
```

5. 通过在**"生成"**菜单上，单击**"生成解决方案"**来生成可执行文件。

运行应用程序

1. 通过在**"解决方案资源管理器"**中选择 MyExecRefsAssembly，然后在**"项目"**菜单上，单击**"设为启动项目"**，从而确保选择 MyExecRefsAssembly 作为默认项目。
2. 若要运行项目，请在**"调试"**菜单上，单击**"开始执行(不调试)"**。输出应与下面的示例类似。