前 言

Qt是Trolltech公司的产品，Trolltech是挪威的一家软件公司，主要开发两种产品：一种是跨平台应用程序界面框架；另外一种就是提供给做嵌入式Linux开发的应用程序平台，能够应用到PDA和各种移动设备，Trolltech公司网址： http://www.trolltech.com/。

Qt 是一个多平台的 C++图形用户界面应用程序框架，它提供给应用程序开发者建立艺术级的图形用户界面所需的所用功能。Qt 是完全面向对象，很容易进行扩展，并且允许真正的组件编程。1996 年开始，Qt 正式进入商业领域，它成为了全世界范围内数千种成功的应用程序的基础。Qt 同时也是流行的 Linux 桌面环境 KDE 的基础，注：KDE 是所有主要的 Linux 发行版的一个标准组件。Qt 目前支持以下平台：

- **MS/Windows** - 95、98、NT 4.0、ME、XP、2000、2003（新版 Qt 兼容.NET）
- **Unix/X11** - Linux、Sun Solaris、HP-UX、Compaq Tru64 UNIX、IBM AIX、SGI IRIX 和其它很多 X11 平台
- **Macintosh** - Mac OS X
- **Embedded** - 有帧缓冲(framebuffer)支持的 Linux 平台。

Qt 有以下几个版本，基中 Qt 专业版和企业版是 Qt 的商业版本，Qt 自由版是 Qt 的非商业版本，可以免费下载。

- Qt 企业版和 Qt 专业版 提供给商业软件开发。它们提供传统商业软件发行版并且提供免费升级和技术支持服务。
- Qt自由版是Qt仅仅为了开发*自由和开放源码软件* 提供的Unix/X11 版本。在Q公共许可证和GNU通用公共许可证 下，它是免费的。
- Qt/嵌入式自由版是Qt为了开发*自由软件*提供的嵌入式版本。在GNU通用公共许可证下，它是免费的。

只有你购买了专业版或企业版，你才能够编写商业的，私人的或收费的软件。如果你购买了这些商业版本，你也可以获得技术支持和升级服务。运行微软公司的 Windows 操作系统的 Qt 只提供了专业版和企业版。

专业版/企业版比较表	专业版	企业版
Qt 的基本模块（工具、核心、窗口部件、对话框） 与平台无关的 Qt 图形用户界面工具包和应用类	√	√
Qt 设计器 可视化的 Qt 图形用户界面的生成器	√	√
图标视图模块 几套图形用户交互操作的可视化效果。	√	√
工作区模块 多文档界面（MDI）支持	√	√
OpenGL 三维图形模块 在 Qt 中集成了 OpenGL		√

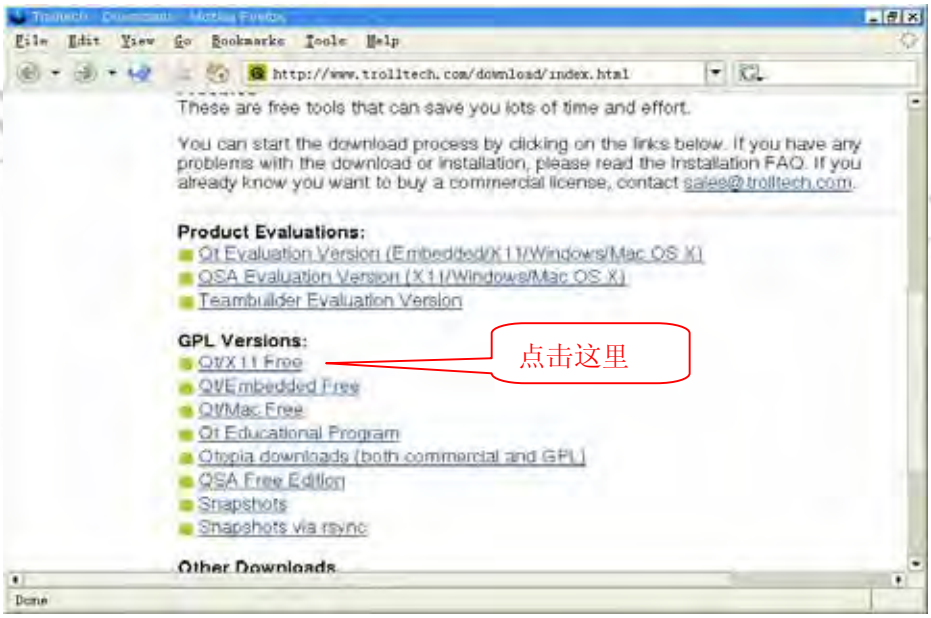
网络模块 一些套接字，TCP、FTP 和异步 DNS 查询并且与平台无关的类。		√
画布模块 为可视化效果，图表和其它而优化的二维图形领域。		√
表格模块 灵活的可编辑的表格/电子表格		√
XML 模块 通过 SAX 接口和 DOM Level 1 很好且已经成形的 XML 解析器。		√
SQL 模块 SQL 数据库访问类。		√

本书主要讲述 Qt 在 Linux 下的编程基础知识。
系统环境：操作系统红旗 LINUX 4.1 桌面版，qt-x11-free-3.3.2
下载地址：
红旗 LINUX 4.1 桌面版：http://www.redflag-linux.com/xiazai/xiazai.php?id=1361
qt-x11-free-3.3.2：http://www.trolltech.com/download/qt/x11.html

第一章 Qt 的安装

因为本书主要介绍的是 Linux 下 Qt 的编程，这里只介绍 Qt 在 Linux 下的安装。
关于 Linux 的安装介绍，不是本书的讲述的内容，如果您还没有接触过 Linux，先从学习 Linux 基础开始吧。

首先下载 Qt，进入 http://www.trolltech.com/download/index.html 下载页面，选择 GPL Versions，点击 Qt/X11 Free 链接地址，



点击后，有相应的下载链接，下载 qt-x11-free-3.3.2.tar.gz 文件。
安装前，你可能需要 root 权限，这取决于你要安装 Qt 的路径的权限，首先，解压开压缩文件。

```
cd /usr/local
gunzip qt-x11-free-3.3.2.tar.gz     # 对这个包进行解压缩
tar xf qt-x11- free-3.3.2.tar       # 对这个包进行解包
```

或者直接

```
cd /usr/local
tar –zxvf qt-x11-free-3.3.2.tar.gz
```

执行完后会生成一个包含主要的包中文件的/usr/local/ qt-x11- free-3.3.2 目录，把 qt-x11- free-3.3.2 重新命名为 qt（或者建立一个链接）：

```
mv qt-x11-free-3.3.2 qt
```

这里假设 Qt 要被安装到/usr/local/qt 路径下
你的主目录下的.profile 文件（或者.login 文件，取决于你的 shell）中设置一些环境变量

- QTDIR -- 你安装 Qt 的路径
- PATH -- 用来定位 moc 程序和其它 Qt 工具
- MANPATH -- 访问 Qt man 格式帮助文档的路径
- LD_LIBRARY_PATH -- 共享 Qt 库的路径

示例：

在.profile 文件（如果你的 shell 是 bash、ksh、zsh 或者 sh）中，添加下面这些行：

```
QTDIR=/usr/local/qt
PATH=$QTDIR/bin:$PATH
MANPATH=$QTDIR/man:$MANPATH
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH

export QTDIR PATH MANPATH LD_LIBRARY_PATH
```

在.login 文件（如果你的 shell 是 csh 或者 tcsh 的情况下），添加下面这些行：

```
setenv QTDIR /usr/local/qt
setenv PATH $QTDIR/bin:$PATH
setenv MANPATH $QTDIR/man:$MANPATH
setenv LD_LIBRARY_PATH $QTDIR/lib:$LD_LIBRARY_PATH
```

做完这些之后，你需要重新登录，或者在继续工作之前重新指定你的配置文件，这样至少$QTDIR 被设置了。否则的话安装程序就会给出一个错误信息并且不再进行下去了

安装你的许可证文件。对于自由版本，你不需要一个许可证文件。对于专业版和企业版，你就需要安装一个和你的发行版一致的许可证文件。

编译 Qt 库，并且连编实例程序、教程和工具（比如 Qt 设计器），就像下面这样。

输入：

```
./configure
```

这样的话就为你的机器配置 Qt 库。注意在默认条件下 GIF 文件支持选项是关闭的。运行./configure -help 就会得到配置选项的一个列表。阅读 PLATFORMS 文件能够得到被支持的平台的列表。

生成库和编译所有的例程和教程：

```
make
```

根据您机器的配置，编译速度会有不多，需要等待一段时间。

如果你有问题，请看http://www.trolltech.com/platforms/。

在很少的情况下，如果你使用了共享库，在这个地方你也许需要运行/sbin/ldconfig 或者其它相似的东西。

如果你在运行实例程序的时候遇到问题，比如消息如下

can't load library 'libqt.so.2'

你也许需要在配置文件中给定一个 qt 库的定位，并且以 root 的身份在你的系统中运行 /sbin/ldconfig 。并且你不要忘记了在上面的第二步中提到的设置一个 LD_LIBRARY_PATH 环境变量。

在线的 HTML 文档被安装到了/usr/local/qt/doc/html/，主页面是 /usr/local/qt/doc/html/index.html。man 帮助文档被安装到了/usr/local/qt/doc/man/。

你已经做完了。Qt 已经安装完毕。 安装完毕后不会像 WINDOWS 安装程序一样，会在开始菜单上添加菜单。
在Linux命令模式下，键入以下命令

```
designer
```

就可以看到 Qt Designer 界面了。为了方便使用，您可以在桌面上添加一个快捷方式。

Qt 提供了几种命令行和图形工具来减轻和加速开发过程。

- **Qt 设计器** 可视化地设计视窗
- **Qt 语言学家** 翻译应用程序使之能够进入国际市场
- **Qt 助手** 快速地发现你所需要的帮助
- **Qmake** 由简单的宇平台无关的项目文件生成 Makefile
- **qembed** 转换数据，比如把图片转还为 C++代码
- **qvfb** 在桌面上运行和测试嵌入式应用程序
- **makeqpf** 为嵌入式设备提供预先做好的字体
- **moc** 元对象编译器
- **uic** 用户界面编译器
- **qtconfig** 一个基于 Unix 的 Qt 配置工具，这里是在线帮助

小节

本章主要介绍了 Qt 在红旗 Linux Desktop 4.1 下的安装，至少 Qt 在其它系统下的安装方法，可详细查看安装说明，在 Linux 下可以在解压缩后，参考/usr/local/qt 目录下的 install 文档。

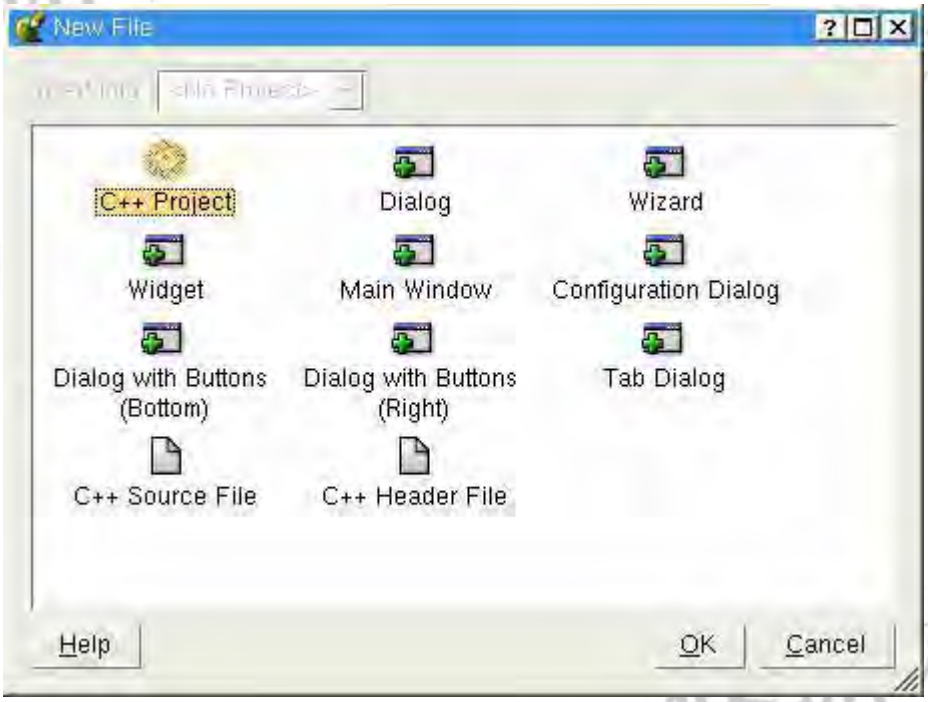
第二章 Hello World

作为一个初学者，安装完 Qt 后第一件事，当然 Hello World 一下，通常介绍编程的教科书都是从 Hello World 开始的，我不知道如果打破这个传统会带来什么后果，我现在还没有勇气去做第一个吃螃蟹的人。如果你不是第一次接触 Qt，可以跳过本章节。下面用两个经典的示例来写讲述 Hello World。

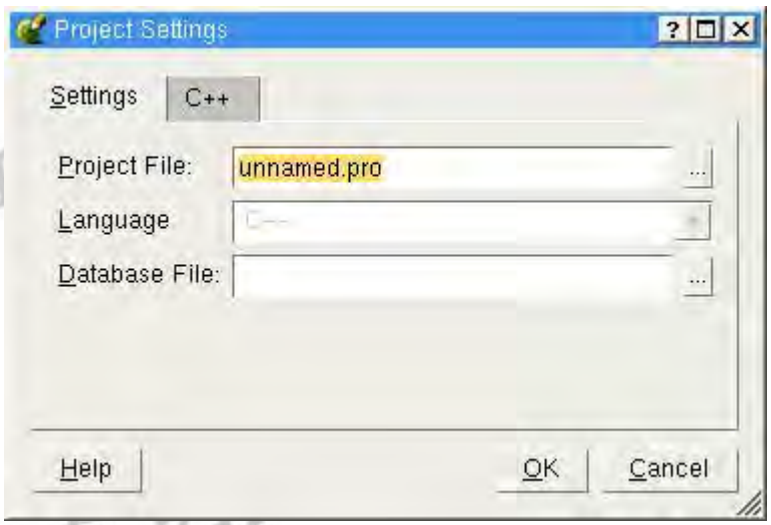
2.1 从两个例子开始

示例一：

运行 Qt Designer,点击菜单 File->new，新建一个项目，



这里选择 C++ Project，确定。接下来会提示项目保存位置，



选择保存路径和文件名，确定，然后，点击菜单 File->New,选择 C++ Source File,确定，录入以下内容：

```
#include <qapplication.h>
#include <qpushbutton.h>

int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    QPushButton hellobtn( "Hello world!", 0 );
    hellobtn.resize(100, 20 );
    a.setMainWidget( &hellobtn );
    hellobtn.show();
    return a.exec();
}
```

代码说明：

#include <qapplication.h>

这一行包含了 QApplication 类的定义。在每一个使用 Qt 的应用程序中都必须使用一个 QApplication 对象。QApplication 管理了各种各样的应用程序的广泛资源，比如默认的字体和光标。

#include <qpushbutton.h>

这一行包含了 QPushButton 类的定义。参考文档的文件的最上部分提到了使用哪个类就必须包含哪个头文件的说明。

QPushButton 是一个经典的图形用户界面按钮，用户可以按下去，也可以放开。它管理自己的观感，就像其它每一个 QWidget。一个窗口部件就是一个可以处理用户输入和绘制图形的用户界面对象。程序员可以改变它的全部观感和它的许多主要的属性（比如颜

色），还有这个窗口部件的内容。一个 QPushButton 可以显示一段文本或者一个 QPixmap。

```
int main( int argc, char **argv )
{
```

main()函数是程序的入口。几乎在使用 Qt 的所有情况下，main()只需要在把控制转交给 Qt 库之前执行一些初始化，然后 Qt 库通过事件来向程序告知用户的行为。

argc 是命令行变量的数量，argv 是命令行变量的数组。这是一个 C/C++特征。它不是 Qt 专有的，无论如何 Qt 需要处理这些变量（请看下面）。

```
    QApplication a( argc, argv );
```

a 是这个程序的 QApplication。它在这里被创建并且处理这些命令行变量(比如在 X 窗口下的-display)。请注意，所有被 Qt 识别的命令行参数都会从 argv 中被移除(并且 argc 也因此而减少）。关于细节请看 QApplication::argv()文档。

注意：在任何 Qt 的窗口系统部件被使用之前创建 QApplication 对象是必须的。

```
    QPushButton hellobtn( "Hello world!", 0 );
```

这里，在 QApplication 之后，接着的是第一个窗口系统代码：一个按钮被创建了。

这个按钮被设置成显示“Hello world!”并且它自己构成了一个窗口（因为在构造函数指定 0 为它的父窗口，在这个父窗口中按钮被定位）。

```
    hellobtn.resize( 100, 20 );
```

这个按酒被设置成 100 像素宽，20 像素高（加上窗口系统边框）。在这种情况下，我们不用考虑按钮的位置，并且我们接受默认值。

```
    a.setMainWidget( &hellobtn );
```

这个按钮被选为这个应用程序的主窗口部件。如果用户关闭了主窗口部件，应用程序就退出了。

你不用必须设置一个主窗口部件，但绝大多数程序都有一个。

```
    hellobtn.show();
```

当你创建一个窗口部件的时候，它是不可见的。你必须调用 show()来使它变为可见的。

```
    return a.exec();
```

这里就是 main()把控制转交给 Qt，并且当应用程序退出的时候 exec()就会返回。

在 exec()中，Qt 接受并处理用户和系统的事件并且把它们传递给适当的窗口部件。

```
}
```

你现在可以试着编译和运行这个程序了。

编译

编译一个 C++应用程序，你需要创建一个 makefile。创建一个 Qt 的 makefile 的最容易的方法是使用 Qt 提供的连编工具 qmake。如果你已经把 main.cpp 保存到它自己的目录了，你所要做的就是这些：

```
qmake hello.pro
```

第一个命令调用 qmake 来生成一个.pro（项目）文件，运行后会生成一个 makefile。你现在可以输入 make（或者 nmake，如果你使用 Visual Studio），然后运行你的第一个 Qt 应用程序！

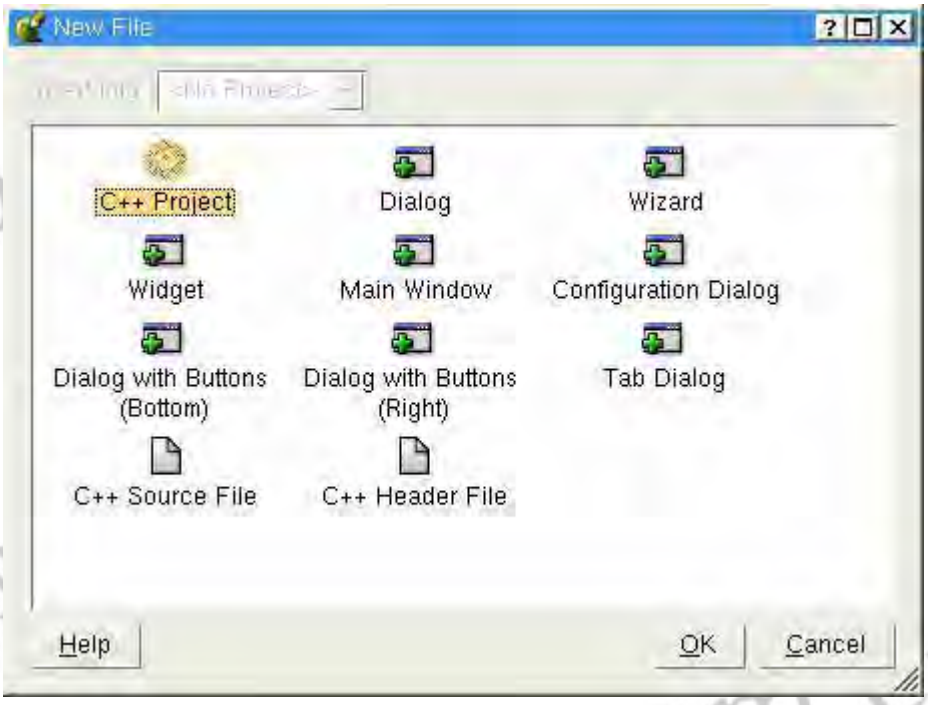
```
qmake hello.pro
make
./hello
```

运行后显示窗口

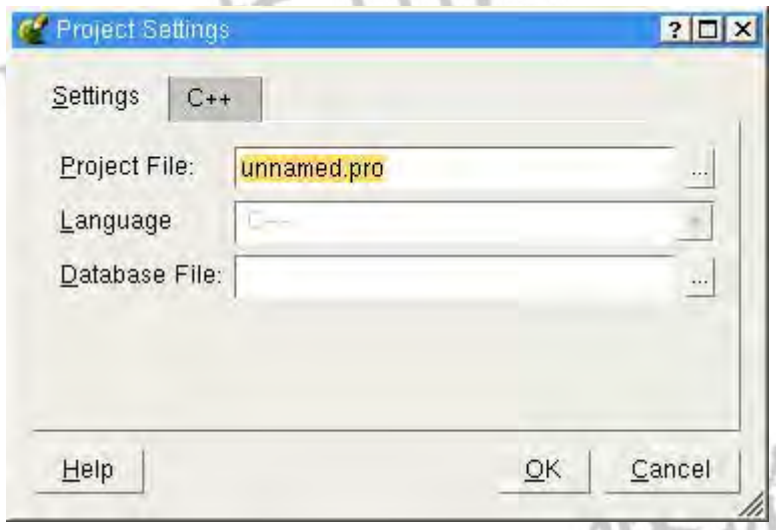
其实使用 Qt Designer 完全可以生成上面的代码，下面我们再看一个例子，

示例二：

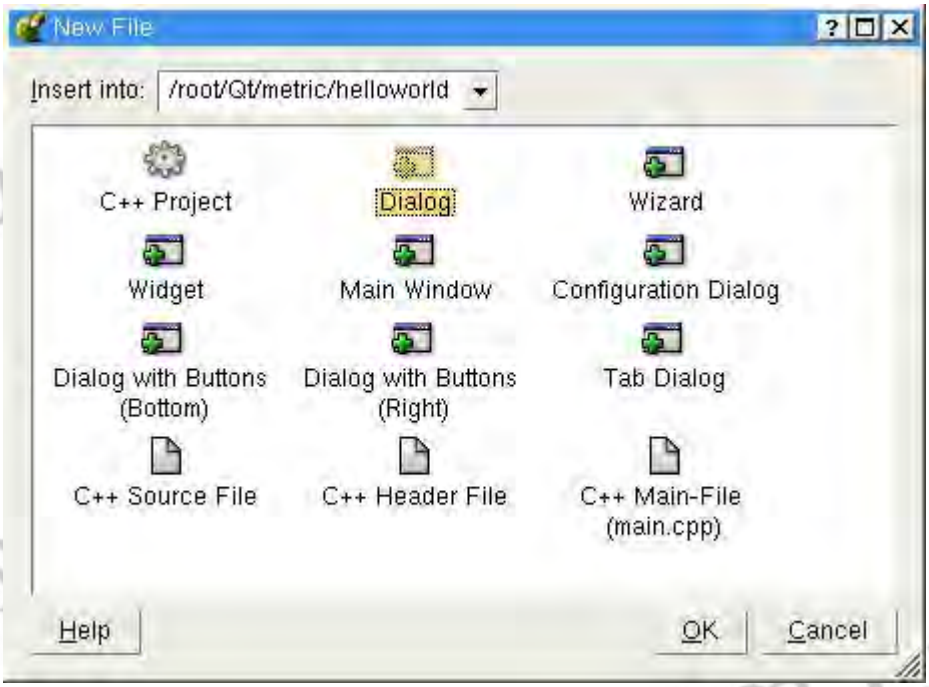
运行 Qt Designer,点击菜单 File->new，新建一个项目，



这里选择 C++ Project，确定。接下来会提示项目保存位置，



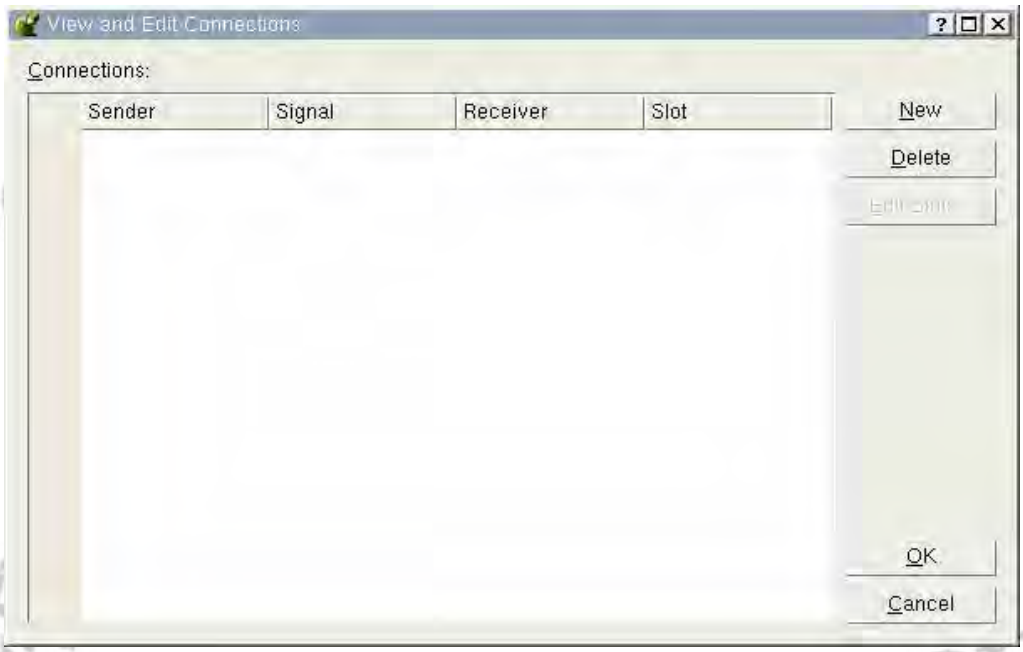
选择保存路径和文件名，确定。
选择菜单 File->new，新建一个窗口，选择 Dialog，确定



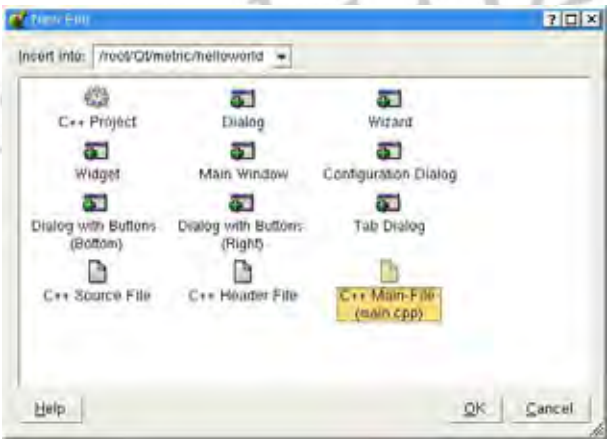
设置 Form1 的 Caption 为“我的程序”，在 Property Editor 设置窗口属性，如果你的 IDE 上看不到 Property Editor，请通过菜单 Windows->Views，将 Property Editor/Signal Handlers 选上。

接着在窗口上放一个 TextLabel，选择 Toolbox 上的 Common Widgets 上的 TextLabe，再添加一个按钮，PushButton，

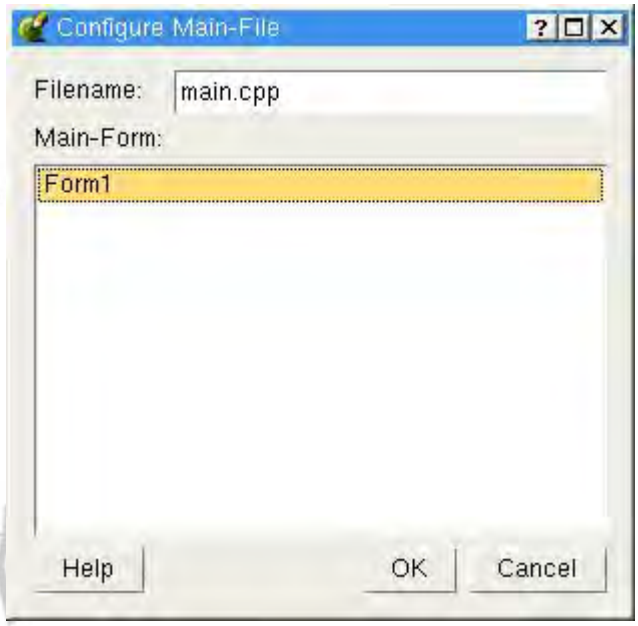
将 textLabel1 的 Text 设为“Hello World”，pushButton1 的 Text 设为“关闭”。接下来为关闭按钮添加事件，选择菜单 Edit->Connections...，弹出 View and Edit Connections 窗口，



点击按钮“New”在新增项中，Sender 选择 pushButton1，Signal 选择 Clicked(),Receiver 选择 Form1,Slot 选择 Close()，确定然后保存，



到这里，基本就快完成我们的 Hello World 了，我们还需要一个 main.cpp 文件，点击菜单 File->new，选择 C++ main file(main.cpp)



文件名 filename:main.cpp，程序主窗 Main-Form:Form，确定，Qt 会自动生成 main.cpp 文件代码，

```
#include <qapplication.h>
#include "form1.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    Form1 w;
    w.show();
    a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
    return a.exec();
}
```

好了，到这里我们的第二个 Hello World 程序设计完了。保存项目，然后编译程序：

```
qmake helloworld.pro
make
./helloworld
```

如果您保存的项目名称是 hellowrold.pro 就可以用上面的命令编译了。Qmake helloworld.pro 先生成 Makefile，make 开始编译程序，编译完后我们就可以运行我们的程序了。./helloworld 运行，

点击“关闭”按钮，关闭程序。

2.2 本章小节

本章通过两个简单例子带你走入 Qt 世界，如果你已经是位 C/C++程序员，你会发现 Qt 将会非常容易上手，本章的二个简单的例子，讲述了如何使用 Qt Designer 来快速的建立应用程序。其中第二个例子中介绍了如何新建窗口，添加组件，并如何设置它们。上面的两个例子，主要使用了 Qt 以下两个类(Class)：

QApplication 类

QApplication 类管理图形用户界面应用程序的控制流和主要设置，

公有成员

```
QApplication ( int & argc, char ** argv )
QApplication ( int & argc, char ** argv, bool GUIenabled )
enum Type { Tty, GuiClient, GuiServer }
QApplication ( int & argc, char ** argv, Type type )
QApplication ( Display * dpy, HANDLE visual = 0, HANDLE colormap = 0 )
QApplication ( Display * dpy, int argc, char ** argv, HANDLE visual = 0,
HANDLE colormap = 0 )
virtual ~QApplication ()
int argc () const
char ** argv () const
Type type () const
enum ColorSpec { NormalColor = 0, CustomColor = 1, ManyColor = 2 }
QWidget * mainWidget () const
virtual void setMainWidget ( QWidget * mainWidget )
virtual void polish ( QWidget * w )
QWidget * focusWidget () const
QWidget * activeWindow () const
int exec ()
void processEvents ()
void processEvents ( int maxtime )
void processOneEvent ()
bool hasPendingEvents ()
int enter_loop ()
```

```
void exit_loop ()
int loopLevel () const
virtual bool notify ( QObject * receiver, QEvent * e )
void setDefaultCodec ( QTextCodec * codec )
QTextCodec * defaultCodec () const
void installTranslator ( QTranslator * mf )
void removeTranslator ( QTranslator * mf )
enum Encoding { DefaultCodec, UnicodeUTF8 }
QString translate ( const char * context, const char * sourceText, const
char * comment = 0, Encoding encoding = DefaultCodec ) const
virtual bool macEventFilter ( EventRef )
virtual bool winEventFilter ( MSG * )
virtual bool x11EventFilter ( XEvent * )
int x11ProcessEvent ( XEvent * event )
virtual bool qwsEventFilter ( QWSEvent * )
void qwsSetCustomColors ( QRgb * colorTable, int start, int numColors )
void winFocus ( QWidget * widget, bool gotFocus )
bool isSessionRestored () const
QString sessionId () const
virtual void commitData ( QSessionManager & sm )
virtual void saveState ( QSessionManager & sm )
void wakeUpGuiThread ()
void lock ()
void unlock ( bool wakeUpGui = TRUE )
bool locked ()
bool tryLock ()
```

公有槽

```
void quit ()
void closeAllWindows ()
```

相关函数

```
void qAddPostRoutine ( QtCleanUpFunction p )
const char * qVersion ()
bool qSysInfo ( int * wordSize, bool * bigEndian )
void qDebug ( const char * msg, ... )
void qWarning ( const char * msg, ... )
void qFatal ( const char * msg, ... )
void qSystemWarning ( const char * msg, int code )
void Q_ASSERT ( bool test )
void Q_CHECK_PTR ( void * p )
QtMsgHandler qInstallMsgHandler ( QtMsgHandler h )
```

它包含主事件循环，在其中来自窗口系统和其它资源的所有事件被处理和调度。它也处理应用程序的初始化和结束，并且提供对话管理。它也处理绝大多数系统范围和应用程序范围的设置。

对于任何一个使用 Qt 的图形用户界面应用程序，都正好存在一个 QApplication 对象，而不论这个应用程序在同一时间内是不是有 0、1、2 或更多个窗口。

QApplication 对象是可以通过全局变量 qApp 访问。它的负责的主要范围有：
它使用用户的桌面设置，例如 palette()、font()和 doubleClickInterval()来初始化应用程序。如果用户改变全局桌面，例如通过一些控制面板，它会对这些属性保持跟踪。

它执行事件处理，也就是说它从低下的窗口系统接收事件并且把它们分派给相关的窗口部件。通过使用 sendEvent()和 postEvent()，你可以发送你自己的事件到窗口部件。

它分析命令行参数并且根据它们设置内部状态。
它定义了由 QStyle 对象封装的应用程序的观感。在运行状态下，可以通过 setStyle()来改变。

它指定了应用程序如何分配颜色。参考 setColorSpec()。

它定义了默认文本编码（请参考 setDefaultCodec()）并且提供了通过 translate()用户可见的本地化字符串。

它提供了一些像 desktop()和 clipboard()这样的魔术般的对象。

它知道应用程序的窗口。你可以使用 widgetAt()来询问在一个确定点上存在哪个窗口部件，得到一个 topLevelWidgets()（顶级窗口部件）的列表和通过 closeAllWindows()来关闭所有窗口，等等。

它管理应用程序的鼠标光标处理，参考 setOverrideCursor()和 setGlobalMouseTracking()。

在 X 窗口系统上，它提供刷新和同步通讯流的函数，可以参考 flushX()和 syncX()。

它提供复杂的对话管理支持。这使得当用户注销时，它可以让应用程序很好地结束，如果无法终止，撤消关闭进程并且甚至为未来的对话保留整个应用程序的状态。

QPushButton 类

QpushButton 主要用于命令按钮，
#include <qpushbutton.h>
继承了 QButton。

公有成员

- ✧ QPushButton (QWidget * parent, const char * name = 0)

- QPushButton (const QString & text, QWidget * parent, const char * name = 0)
- QPushButton (const QIconSet & icon, const QString & text, QWidget * parent, const char * name = 0)
- ~QPushButton ()
- void setToggleButton (bool)
- bool autoDefault () const
- virtual void setAutoDefault (bool autoDef)
- bool isDefault () const
- virtual void setDefault (bool def)
- virtual void setIsMenuButton (bool enable) （废弃）
- bool isMenuButton () const （废弃）
- void setPopup (QPopupMenu * popup)
- QPopupMenu * popup () const
- void setIconSet (const QIconSet &)
- QIconSet * iconSet () const
- void setFlat (bool)
- bool isFlat () const

公有槽

virtual void setOn (bool)

属性

- bool autoDefault - 推动按钮是否是自动默认按钮
- bool autoMask - 按钮中自动面具特征是否有效 （只读）
- bool default - 推动按钮是否是默认按钮
- bool flat - 边缘是否失效
- QIconSet iconSet - 推动按钮上的图标
- bool menuButton - 推动按钮是否有一个菜单按钮在上面 （废弃）
- bool on - 推动按钮是否被切换
- bool toggleButton - 按钮是不是切换按钮

推动按钮或者命令按钮或许是任何图形用户界面中最常用到的窗口部件。推动(点击)按钮来命令计算机执行一些操作，或者回答一个问题。典型的按钮有确定（OK)、应用（Apply)、撤销（Cancel)、关闭（Close)、是（Yes)、否（No）和帮助（Help)。

命令按钮是矩形的并且通常显示一个文本标签来描述它的操作。标签中有下划线的字母（在文本中它的前面被“&”标明）表明快捷键，例如：

```
QPushButton *pb = new QPushButton( "&Download", this );
```

在这个实例中加速键是 Alt+D，并且文本标签将被显示为 Download。

第三章 C/C++ 基础

我们知道 Qt 是一个多平台的 C++图形用户界面应用程序框架，所以，要想更好的使用它，学好 C++是基础，因为本书不是一篇重点介绍 C++的文章，接下来就简单的介绍一个 C++ 基础知识，让读者熟悉一下 C++的基础知识。

C++是建立在 C 语言之上的，称为“带类的 C 语言”。这个 C 语言基础在当今的 C++程序中仍然很重要。C++并不是取代 C，而是补充和支持 C。

C++可以充分地利用面向对象编程(OOP)的优势。Qt 提供大量的应用程序框架，可以用模块化方法进行编程，从而避免每次从头开始，可以生成可复用的对象。

3.1 变量

变量(variable)实际上是赋予内存地址的名称。声明变量后，就可以用它操作内存中的数据。下面举几个例子进行说明。下列码段用了两个变量，每条语句末尾用说明语句描述执行该语句时发生的情况：

```
int x;  // 定义一个整型的变量 x
x = 10;//  x 是 10
x +=5;//   x 是 15
int y = 15;// 定义一个变量 y，同时赋值 15
x += y;// x 是 30
x++;//  x 是 31
```

变量(variable)是留作存放某个数值的计算机内存地址。注意 x 的值在变量操作时会改

变，稍后会介绍操作变量的 C++操作符。注：声明而未初始化的变量包含随机值。由于变量所指向的内存还没有初始化，所以不知道该内存地址包含什么值。

例如，下列代码

```
int k;
int y;
x=y+10;
```

本例中变量 y 没有事先初始化，所以 x 可能取得任何值。例外的情况是全局变量和用 static 修饰声明的变量总是初始化为 0。而所有其它变量在初始化或赋值之前包含随机值。变量名可以混合大写、小写字母和数字与下划线（_），但不能包含空格和其它特殊字符。变量名必须以字母或下划线开始。一般来说，变量名以下划线或双下划线开始不好。变量名允许的最大长度随编译器的不同而不同。如果变量名保持在 32 个字符以下，则绝对安全。实际中，任何超过 20 个字符的变量名基本上是不实用的。

下例是有效变量名的例子：

```
int aVeryLongVariableName;
int my_variable;
int _x;
int MyName;
int GetName();
```

注：C++中的变量名是考虑大小写的，下列变量是不同的：int X;int x;如果你原先所用语言不考虑大小写（如 Basic、Pascal 等），则开始接触考虑大小写的语言可能不太适应。

3.2 C++数据类型

C++数据类型定义编译器在内存中存放信息的方式。有些编程语言中，可以向变量赋予任何数值类型。例如，下面是 BASIC 代码的例子：

```
x = 100;
x = 1;
x = 3.1412596;
x = 10000000;
```

在 BASIC 中，翻译器能考虑根据数字长度和类型分配空间。而在 C++，则必须先声明变量类型再使用变量：int x1 = 100;int x = 1;float y = 3.1412596;long z = 10000000;这样，编译器就可以进行类型检查，确保程序运行时一切顺利。数据类型使用不当会导致编译错误或警告，以便分析和纠正之后再运行。有些数据类型有带符号和无符号两种。带符号（signed）数据类型可以包含正数和负数，而无符号（unsigned）数据类型只能包含正数。下列出了 C++中的数据类型、所要内存量和可能的取值范围。

C++数据类型(32 位程序)：

数据类型	字节	数取值范围
char	1	-128~126
unsigned char	1	0~255
short	2	-32,768~32,767

unsigned short	2	0~65,535
long	4	-2,147,483,648~2,147,483,648
unsigned long	4	0~4,294,967,295
int	4	同 long
unsigned int	4	同 unsigned long
float	4	1.2E-38~3.4E381
double	8	2.2E-308~1.8E3082
bool	1	true 或 false

(表 3.1)

从表 3.1 可以看出，int 与 long 相同。那么，为什么 C++还要区分这两种数据类型呢?实际上这是个遗留问题。在 16 位编程环境中，int 要求 2 个字节而 long 要求 4 个字节。而在 32 位编程环境中，这两种数据都用 4 个字节存放。从表 3.1 可以看出，短整型的最大取值为 32767,在最大值之上加 1 会怎么样呢?得到的是 32768。这实际上与汽车里程计从 99999 回到 00000 的道理一样。为了说明这点，请看下面的程序：

```
#include <iostream.h>
#include <conio.h>
main(int argc,char **argv)
{
    short x = 32767;
    cout << " x = " << x << endl;
    x++;
    cout << " x = " << x << endl;
    getch();
    return 0;
}
```

结查显示是：

x=32767
x=-32768

3.3 C++操作符

操作符(operator)用于操作数据。操作符进行计算、检查等式、进行赋值、操作变量和进行其它更奇怪的工作。C++中有许多操作符，这里我们只列出最常用的操作符，如下所示：

算术运算符

+ 加 x=y+z;
- 减 x=y-z;
* 乘 x=y*z;
/ 除 x=y/z;

赋值运算符

= 赋值 x=10;

+= 赋值与和 x+=10;(等于 x=x+10;)
-= 赋值与减 x-=10;
= 赋值与乘 x=10;
\= 赋值与除 x\=10;
&= 赋值位与 x&=0x02;
|= 赋值位或 x|=0x02;

逻辑操作符

&& 逻辑与 if(x && 0xFF) {...}
|| 逻辑或 if(x || 0xFF) {...}
等式操作符
== 等于 if(x == 10) {...}
!= 不等于 if(x != 10) {...}
< 小于 if(x < 10) {...}
> 大于 if(x > 10) {...}
<= 小于或等于 if(x <= 10) {...}
>= 大于或等于 if(x >= 10) {...}

一元操作符

* 间接操作符 int x=*y;
& 地址操作符 int* x=&y;
～ 位非 x &=～0x02;
! 逻辑非 if(!valid) {...}
++ 递增操作符 x++（等于 x=x+1;）
-- 递减操作符 x--;

类和结构操作符

:: 范围解析 MyClass :: MyFunction();
-> 间接成员 MyClass-> MyFunction();
• 直接成员 MyClass . MyFunction();

C++操作符较多，使用 C++时，你会慢慢熟悉这些操作符的。这里要注意，递增操作符既可用作前递增(++x)，也可用作后递增(x++)。前递增操作符告诉编译器先递增再使用变量，而后递增操作符则让编译器先使用变量值再递增。例如下列代码：

```
#include <iostream.h>
#include <conio.h>
main(int argc,char **argv)
{
     int x = 1;
     cout << "x = " << x++ <<endl;
     cout << "x = " << x <<endl;
     cout << "x = " << x <<endl;
     cout << "x = " << ++x <<endl;
     return 0;
}
```

输出结果如下：

x=1

x=2
x=2
x=3
递减操作符也是这样，这里就不再一一介绍。

注：在 C++中操作符可以重载(overload)。编程人员可以通过重载标准操作符让它在特定类中进行特定运行。例如，可以在一个类中过载递增操作符，让它将变量递增 10 而不是递增 1。操作符过载是个高级 C++技术，本书不准备详细介绍。读者不必死记每个操作符的作用，你可以在学习中通过程序和码段去理解其作用。

3.4 C++中的函数

函数是与主程序分开的码段。这些码段在程序中需要进行特定动作时调用（执行）。例如，函数可能取两个值并对其进行复杂的数学运算。然后返回结果，函数可能取一个字串进行分析，然后返回分析字串的一部分。函数是各种编程语言的重要部分，C++也不例外。最简单的函数不带参数，返回 void（表示不返回任何东西），其它函数可能带一个或几个参数并可能返回一个值。函数名规则与变量名相同。参数(parameter)是传递给函数的值，用于改变操作或指示操作程度。

```
返回类型   函数名    参数表
↓          ↓         ↓
int SomeFunction(int x, int y){
函数体→int z = (x * y);
return z;
  ↑返回语句
}
```

函数的构成部分使用函数前，要先进行声明。函数声明或原型(prototype)告诉编译器函数所取的参数个数、每个参数的数据类型和函数返回值的数据类型。原型(prototype)是函数外观的声明或其定义的说明。

```
#include <iostream.h>
#include <conio.h>

int multiply(int x,int y)
{
   return x * y;
}

void showResult(int res)
{
  cout << "The result is: " << res <<endl;
```

```
}


int main(int argc,char **argv)
{
  int x,y,result;
  cout << endl << "Enter the first value:";
  cin >> x;
  cout << "Enter the second value: ";
  cin >> y;
  result=multiply(x,y);
  showResult(result);
  cout << endl << endl << "Press any key to continue...";
  getch();
  return 0;
}
```

这个程序标准输入流 cin 向用户取两个数字，调用 multiply()函数将两个数相乘，调用 showResult()函数显示相乘的结果。函数可以调用其它函数，甚至可以调用自己，这种调用自己的调用称为递归(recursion)。这在 C++编程中是个较复杂的问题，我们这不介绍。

注：递归(recursion)就是函数调用自己的过程。

本节介绍的函数指的是 C 或 C++程序中的独立函数（独立函数不是类的成员）。C++中的独立函数可以和 C 语言中一样使用。

函数规则

- 函数可以取任意多个参数或不取参数。
- 函数可以返回一个值，但函数不强求返回一个值。
- 如果函数返回 void 类型，则不能返回数值。

如果要让返回 void 类型的函数返回数值，则会发生编译错误。返回 void 类型的函数不需包含 return 语句，但也可以包含这个语句。如果没有 return 语句，则函数到达末尾的结束大括号时自动返回。

- 如果函数原型表示函数返回数值，则函数体中应包含返回数值的 return 语句，如果函数不返回数值，则会发生编译错误。
- 函数可以取任意多个参数，但只能返回一个数值。
- 变量可以按数值、指针或引用传递给函数（将在稍后介绍）。

语法：函数语句的声明（原型）格式如下：

ret_type function_name(argtype_1 arg_1,argtype_2 arg_2,...,argtype_n arg_n);

函数声明表示代码中要包括的函数，应当显示函数的返回数据类型(ret_type)和函数名(function_name)，表示函数所要数据变元的顺序

（arg_1,arg_2,...,arg_n)和类型(argtype_1,argtype_2,...argtype_n)。

函数语句的定义格式如下：

ret_type function_name(argtype_1 arg_1,argtype_2 arg_2,...,argtype_narg_n);

{ statements;

return ret_type; }

函数定义表示构成函数的代码块(statements),应当显示函数的返回数据类型(ret type)和函数名(function_name)，包括函数所要数据变元（arg_1,arg_2,...,arg_n）和类型(argtype_1,argtype_2,...argtype_n)。

3.5 main()函数

C++程序必须有 main()函数。main()函数是程序的入口点。Qt Designer 的菜单 File->New 弹出的 New File 对话框就有一项专门用来生成 main.cpp 文件的模板，main.cpp 里面主要是一个 main()函数。我们前面介绍的每个样本程序都有 main()函数。但是要注意一点，并非所有 C++程序都有传统的 main()函数。VC 中用 C 或 C++写成的 Windows 程序入口点函数称为 WinMain()，而不是传统的 main()函数。main()函数和其它函数一样是函数，有相同的构成部分。Qt Designer 可以生成具有下列原型的缺省 main()函数：int main(int argc, char **argv);这个 main()函数形式取两个参数并返回一个整型值。前面说过，数值在调用函数时传递给函数。但对于 main()函数，没有直接调用，而是在程序运行时自动执行。那么，main()函数如何取得参数呢?办法是从命令行取得。

现说明如下：

假设有个命令行下用命令行执行：mypro a b c

这里就是要用命令行变量 a、b 和 c 启动程序 mypro,我们要演示如何在 main()函数中将其变为 argc 和 argv.首先，整型变量 argc 包含命令行中传递的参数个数，至少为 1，因为程序名也算作参数。变量 argv 是个数组，包含字串的指针。这个数组包含命令行中传递的每个字串。本例中：

argc 包含 4

argv[0] 包含 mypro

argv[1] 包含 a

argv[2] 包含 b

argv[3] 包含 c

下面用一个小程序验证这个事实

程序如下：

```
#include <iostream.h>
#include <conio.h>

int main(int argc,char **argv)
{
  cout << "argv = "<<argc << endl;
  for (int i=0;i<argc;i++)
  cout << "Parameter " << i << ": " << argv[i]<< endl;
  cout << endl << "Press any key to continue...";
  getch();
  return 0;
}
```

将这个项目存为 mytest，编译运行试一下，

大多数程序中 main()函数的返回值并不重要，因为通常不使用返回值。事实上，可以不要求 main()函数返回数值。main()函数的形式有多种，下列声明均有效：main();int main();

```
int main(void);
int main(int argc,char** argv);
void main();
void main(int argc, char** argv);
```

还有更多的形式。如果不想使用命令行变量，则可以用第一种 main()函数形式，其不取参数（括号内为空的）并返回一个 int(不指定时返回缺省返回值)。换句话说 main()函数最基本的形式不取参数并返回一个 int。

3.6 数组

任何 C++固有数据类型都可以放进数组中。数组(array)就是数值的集合。例如，假设要保存一个整型数组，放五个整型值。可以声明数组如下：int myArray[5];这里编译器会为数组分配内存空间。由于每个 int 要 4 个字节存储，所以整个数组占用 20 字节的内存空间。

mArray[0]mArray[1]mArray[2]mArray[3]

mArray[4]

baseAddrbasseAddr+4baseAddr+8

baseAddr+12baseAddr+16

声明数组后，就可以用如下脚标操作符([])填入数值：

```
myArray[0] = -100;
myArray[1] = -200;
myArray[2] = 0;
myArray[3] = 100;
myArray[4] = 200;
```

由上可见，C++中的数组是以 0 为基数的。后面程序中可以用脚标操作符访问数组的各个元素：

```
int result=myarray[3]+myArray[4]; // 结查应该是 300
```

还有一次声明和填入整个数组内容的简捷方法如下：

```
int myArray[5] = {-100, -200,0,100,200};
```

进一步说，如果知道数组的元素个数，并在声明数组时填充数组，则声明数组时连数组长度都可以省略。例如：int myArray[] = {-100, -200,0,100,200 };这是可行的，因为编译器从赋予的数值表可以判断出数组中元素的个数和分配给数组的内存空间。

数组可以是多维的。为了生成两维整型数组，可用下列代码：

```
int mdArray[3][5];
```

这样就分配 15 个 int 空间(共 60 字节)。数组的元素可以和一维数组一样访问，只是要提供两个脚标操作符：int x = mdArray[1][1]+mdArray[2][1];

C++一个强大的特性是能直接访问内存。由于这个特性，C++无法阻止你写入特定内存地址，即使这个地址是程序不让访问的。下列代码是合法的，但会导致程序或系统崩溃：int array[5];array[5]=10;这是常见的错误，因为数组是以 0 为基数的，最大脚标应是 4 而不是 5。如果重载数组末尾，则无法知道哪个内存被改写了，使结果难以预料，甚至会导致程

序或系统崩溃。这类问题很难诊断，因为受影响的内存通常要在很久以后才访问，这时才发生崩溃（让你莫名其中之妙）。所以写入数组时一定要小心。

数组规则

• 数组是以 0 为基数。数组中的第一个元素为 0，第二个元素为 1，第三个元素为 2，等等。

• 数组长度应为编译常量。编译器在编译时必须知道为数组分配多少内存空间。不能用变量指定数组长度。所以下列代码不合法，会导致编译错误：

```
int x = 10;int myArray[x]; // 这种写法是不合法的，编译会报错。
```

小心不要重载数组末尾。

• 大数组从堆叠（heap）而不是堆栈(stack)中分配(详见稍后)。

• 从堆叠分配的数组可以用变量指定数组长度。例如：int x = 10;int* myArray = new int[x]; // 这种写法是正确的

字符数组

C++不支持字串变量（放置文本的变量），C++程序中的字串是用 char 数据类型的数组表示的。例如，可以将变量赋予 char 数组如下：

```
char text[] = "This is a string.";
```

这就在内存中分配 18 字节的内存空间用于存放字串。根据你的领悟能力，也许你会发现该字串中只有 17 个字符。分配 18 个字节的原因是字串要以终止 null 结尾，C++在分配内存空间时把终止 null 算作一个字符。

注：终止 null 是个特殊字符，用\0 表示，等于数值 0。程序遇到字符数组中的 0 时，表示已经到字串末尾。为了说明这点，输入并运行下列应用程序。

```
#include <iostream.h>
#include <conio.h>

int main(int argc,char **argv)
{
   char str[]="This is a string.";
   cout << str << endl;
   str[7]= '\0';
   cout << str << endl;
   cout << endl << "Press any key to continue...";
   getch();
   return 0;
}
```

说明：最初，字符数组包含字符串 This is a string 和一个终止 null，这个字串通过 cout 送到屏幕上。下一行将数组的第 7 个元素赋值为|0，即终止 null。字串再次发送到屏幕上，但这时只显示 This is。原因是计算机认为数组中字串在第 7 个元素上终止，余下字串仍然在内存空间中，但不显示，因为遇到了终止 null。

字串操作函数

如果你用过具有 string 数据类型的编程语言，你可能很不习惯，很多人有这样的同

感，所以标准 C 语言库中提供了几个字串操作函数。下面列出了最常用的字串操作函数及其用法说明。

函数	说明
strcat()	将字串接合到目标字串的末尾
strcmp()	比较两个字串是否相等
strcmpi()	比较两个字串是否相等，不考虑大小写
strcpy()	将字串内容复制到目标字串中
strstr()	扫描字串中第一个出现的字串
strlen()	返回字串长度
strupr()	将字串中的所有字符变成大写
sprintf()	根据几个参数建立字串

说明：这里介绍的字串操作是 C 语言中的字串处理方法。大多数 C++编译器提供了 cstring 类，可以简化字串的处理。尽管 C 语言中的字串处理方法比较麻烦，但并不过时，C++编程人员经常在使用 cstring 类和 AnsiString 类等字串类的同时使用 C 语言中的字串处理方法。这里不想对表中的每个函数进行举例说明，只想举两个最常用的函数。

注： 如果经常用到字串数组，应当看看标准模板库(STL).STL 提供了比用 C 语言式字符数组更方便地存放和操作字串数组的方法.STL 中还有个 string 类。

3.7 if 语句

if 语句用于测试条件并在条件为真时执行一条或几条语句。

说明:if 表达式后面不能带分号，否则它本身表示代码中的空语句，使编译器将空语句解释为在条件为真时执行的语句。

```
if (x == 10);
DoSomething(x);
```

这里 DoSomething()函数总会执行，因为编译器不把它看成在条件为真时执行的第一条语句。由于这个代码完全合法（但无用），所以编译器无法发出警告。

假设要在条件为真时执行多行语句，则要将这些语句放在大括号内：

```
if (x > 10) {
   cout << "The number is greater than 10" << endl;
   DoSomethingWithNumber(x);
}
```

条件表达式求值为 false 时，与 if 语句相关联的码段忽略，程序继续执行该码段之后的第一条语句。

说明 ：

C＋＋中包含许多快捷方法，其中一个是用变量名测试 true，例如：

if (fileGood) ReadData();

这个方法是下列语句的速写方法：

```
if (fileGood == true) ReadData();
```

本例用了 bool 变量，也可以用其它数据类型。只要变量包含非零数值，表达式即求值为 true，对变量名加上逻辑非(!)操作符可以测试 false 值：

```
bool fileGood = OpenSomeFile();
if (!fileGood) ReportError();
```

学会 C＋＋快捷方法有助于写出更精彩的代码。 有时要在条件表达式求值为 true 时进行某个动作，在条件表达式求值为 false 时进行另一动作，这时可以用 else 语句如下：

```
if (x == 20) {DoSomething(x);}
else {DoADifferentThing(x);}
```

else 语句和 if 语句一起使用，表示 if 语句失败时（即在条件表达式求值为 false 时）执行的码段。

if 语句形式之二：

```
if (cond_expr_1) {
true_statements_1;
}
else if (cond_expr_2)
{ true_statements_2; }
else {
false_statements;
}
```

如果条件表达式 cond_expr 为 1 真（非零），则执行 true_statements1 码段;如果条件表达式 cond_expr 为 1 为假而如果条件表达式 cond_expr 为 2 真(非零),则执行 true_statements 2 码段;如果两个表达式均为假，执行 false_statements 码段。

3.8 循环语句

循环是所有编程语言共同的要素。循环可用于对数组重复或对某个动作重复进行指定次数，如从磁盘中读取文件，等等。

循环有:for 循环、while 循环和 do while 循环。

这几个循环基本相同，所有循环都有下列共同要素：

- 起点
- 循环体，通常放在大括号内，包含每次循环要执行的语句
- 终点
- 确定循环终止的测试条件
- 可选使用 break 和 continue 语句

for 循环是最常用的循环，取三个参数：起始数，测试条件和增量表达式。

for 循环语句：

```
for(initial; cond_expr;adjust)
{ statements;
```

```
}
```

for 循环重复执行 statements 码段，直到条件表达式 cond_expr 不为真。循环状态由 initial 语句初始化，执行 statements 码段后，这个状态用 adjust 语句修改。下面举一个 for 循环的典型例子进行说明：

```
for (int i=0;i<10;i＋＋){
cout << "This is iteration" << i << endl;
}
```

while 循环与 for 循环的差别在于它只有一个在每次循环开始时检查的测试条件。只要测试条件为 true，循环就继续运行。

```
int x;
while (x < 1000) {
x = DoSomeCalculation();
}
```

本例中我调用一个函数，假定它最终会返回大于或等于 1000 的值。只要这个函数的返回值小于 1000，while 循环就继续运行。变量 x 包含大于或等于 1000 的值时，测试条件变成 false，程序转入 while 循环闭括号后面的第一条语句。while 循环通常用 bool 变量进行测试。测试变量状态可以在循环体中进行设置：

```
bool done = false;
while (!done) {
//some code here
done = SomeFunctionReturningABool();
//more code
}
```

do while 循环与 while 循环基本相同，但有两点差别。

，while 循环测试发生在循环体开头，而 do while 循环测试则发生在循环结束时：

```
bool done = false;
do {
// some code
done =SomeFunctionReturningABool();
// more code
} while (! done);
```

使用 dowhile 循环还是 while 循环取决于循环本身的作用。语法中 do while 循环语句：

```
do {
tatements;
} while (cond_expr);
```

只要条件表达式 cond_expr 为真(非零)，do 循环重复 statements 码段。循环状态必须在 do 语句之前初始化，并在码段中显式修改。条件表达式 cond_expr 为假时，循环终止。

goto 语句可以将程序转入前面用标号和冒号声明的标号处。

下列代码演示了这个语句：

```
bool done = false;
```

```
startPoint:
// do some stuff
if (!done) goto(startPoint);// loop over, moving on...
```

这里不需要大括号，因为 goto 语句与标号之间的所有代码均会执行。

goto 语句被认为是 C＋＋程序中的不良语句。用 goto 语句能做的任何工作都可以用 While 和 dowhile 循环进行。一个好的 C＋＋编程人员很少在程序中使用 goto 语句。如果你从别的语言转入 C＋＋，你会发现 C＋＋的基本结构使 goto 语句显得多余。

循环中有两个关键字必须介绍，那就是控制循环中程序执行的 continue 和 break。continue 语句强制程序转入循环底部，跳过 continue 语句之后的任何语句。例如，某个测试为真时，循环的某个部分可能不需要执行。这时可以用 continue 语句跳过 continue 语句之后的任何语句：

```
bool done = false;
while (!done) {
// some codebool error =SomeFunction();
if (error) continue;
// jumps to the top of the loop
// other code that will execute only if no error occurred
}
```

break 语句用于在循环正常测试条件符合之前终止循环执行。例如，可以在 ints 数组中搜索某个元素，找到数字后可以终止循环执行，取得该数字所在的索引位置：

```
int index=1
int searchNumber=50;
for (int i=0;i<numElements;i＋＋) {
if (myArray [i] ==searchNumber) {
index=i;break;}
}
if(index !=1)
cout << "Number found at index " << index << endl;
else
cout <<"Number not found in array." << endl;
```

continue 和 break 语句在许多情况下有用。和其它要介绍的知识一样，continue 和 break 语句也要在实践中不断熟悉。

3.9 switch 语句

switch 语句是高级 if 语句，可以根据表达式的结果执行几个码段之一。表达式可以是变量、函数调用结果或其它有效 C＋＋表达式。下面举一个 switch 语句例子：switch (amountOverSpeedLimit) {

```
case 0 :
{fine =0;
break;
```

```
}
case 10 :
{fine = 20;
break;
}
case 15 :
{fine =20;
break;
}
case 20 :
case 25 :
case 30 :
 {
fine=amountOverSpeedLimit * 10;
break;
}
default :
{fine =GoToCourt();
jailTime=GetSentence();
}
}
```

switch 语句分为几个部分。首先有一个表达式，本例中是 amountOverSpeedLimit 变量(够长的变量名!)，然后用 case 语句测试表达式，如果 amountOverSpeedLimit 等于 0(case 0:)，则向变量 fine 赋值 0,如果 amountOverSpeedLimit 等于 10，则向变量 fine 赋值 20，等等。在前三个 case 中都有 break 语句。break 语句用于转出 switch 块，即找到了符合表达式的情况，switch 语句的余下部分可以忽略了。最后有个 default 语句，如果没有符合表达式的情况，则程序执行 default 语句。

3.10 结构

“结构”是一种构造类型，它是由若干“成员”组成的。 每一个成员可以是一个基本数据类型或者又是一个构造类型。 结构既是一种“构造”而成的数据类型， 那么在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

结构的定义

定义一个结构的一般形式为：

```
struct 结构名
{
成员表列
};
```

成员表由若干个成员组成， 每个成员都是该结构的一个组成部分。对每个成员也必须作

类型说明，其形式为：
类型说明符 成员名;
成员名的命名应符合标识符的书写规定。例如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
};
```

在这个结构定义中，结构名为 stu，该结构由 4 个成员组成。 第一个成员为 num，整型变量；第二个成员为 name，字符数组；第三个成员为 sex，字符变量；第四个成员为 score，实型变量。 应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。 凡说明为结构 stu 的变量都由上述 4 个成员组成。由此可见， 结构是一种复杂的数据类型，是数目固定，类型不同的若干有序变量的集合

例子：给结构变量赋值并输出其值。

```
main(){
struct stu
{
int num;
char *name;
char sex;
float score;
} boy1,boy2;
boy1.num=102;
boy1.name="Zhang ping";
printf("input sex and score\n");
scanf("%c %f",&boy1.sex,&boy1.score);
boy2=boy1;
printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
struct stu
{
int num;
char *name;
char sex;
float score;
}boy1,boy2;
boy1.num=102;
boy1.name="Zhang ping";
```

```
printf("input sex and score\n");
scanf("%c %f",&boy1.sex,&boy1.score);
boy2=boy1;
printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
```

本程序中用赋值语句给 num 和 name 两个成员赋值，name 是一个字符串指针变量。用 scanf 函数动态地输入 sex 和 score 成员值，然后把 boy1 的所有成员的值整体赋予 boy2。最后分别输出 boy2 的各个成员值。本例表示了结构变量的赋值、输入和输出的方法。

3.11 指针

什么是指针

X, Y 是存储在内存单元 1010, 1012 中的两个变量。若我们写 X=100,则整数 100 被放入内存单元 1010 中。

指针可以指向 C/C++ 语言中的所有基本数据类型。它还可以指向由基本数据类型导出的数组, 结构等其它类型

指针定义的方法

type * var;

类型说明表示了指针所指向的变量的类型。星号 * 表示变量本身是一个指针。

在此举例指针是怎样说明的:

```
int *pi;
char *pc;
float *pf, *pg;
```

指针的运算方法

两个重要的运算符: & 和 *。取地址运算符 & 给出变量的地址。间访运算符 *

给出指针所指向的地址中的内容。

看下例:

```
int x,ball=100; 说明了二个变量。
int *ptr; 说明了指针 ptr。
ptr=&ball; 使指针 ptr 指向变量 ball。
x=*ptr; *ptr 是指针所指地址内的内容。
x 被赋值为 100。
```

看一个程序吧:

```
main()
{
    char ch_var = 'A';
    char *ch_pointer = &ch_var;
```

```
    printf("%c %c\n",ch_var,*ch_pointer);
    ch_var = 'd';
    printf("%c %c\n",ch_var,*ch_pointer);
    *ch_pointer = 'a';
    printf("%c %c\n",ch_var,*ch_pointer);
}
```

解释:

1. 可以用普通的方法对指针赋值。
2. 语句 *ch_pointer='a' 把 'a' 赋值给 ch_pointer 所指向的变量 ch_var。因此, ch_var 的值也变为 'a' 了。
3. 该程序的结果如下:

```
A A
d d
a a
```

3.12 类

在面向对象的程序设计中，有经常接触类、对象等专业名词；到底什么是类、什么是对象呢？在程序又是怎样运用呢？类是面向对象程序设计的核心，它实际是一种新的数据类型,也是实现抽象类型的工具,因为类是通过抽象数据类型的方法来实现的一种数据类型。类是对某一类对象的抽象；而对象是某一种类的实例，因此，类和对象是密切相关的。没有脱离对象的类，也没有不依赖于类的对象。

什么是类

类是一种复杂的数据类型,它是将不同类型的数据和与这些数据相关的操作封装在一起的集合体。这有点像 C 语言中的结构，唯一不同的就是结构没有定义所说的“数据相关的操作”，“数据相关的操作”就是我们平常经常看到的“方法”，因此，类具有更高的抽象性，类中的数据具有隐藏性，类还具有封装性。

类的结构(也即类的组成)是用来确定一类对象的行为的,而这些行为是通过类的内部数据结构和相关的操作来确定的。这些行为是通过一种操作接口来描述的(也即平时我们所看到的类的成员函数),使用者只关心的是接口的功能(也就是我们只关心类的各个成员函数的功能)，对它是如何实现的并不感兴趣。而操作接口又被称为这类对象向其他对象所提供的服务。

类的定义格式

类的定义格式一般地分为说明部分和实现部分。说明部分是用来说明该类中的成员，包含数据成员的说明和成员函数的说明。成员函数是用来对数据成员进行操作的，又称为“方法”。实现部分是用来对成员函数的定义。概括说来，说明部分将告诉使用者“干什么”，而实现部分是告诉使用者“怎么干”。

类的一般定义格式如下：

```
class <类名>
{
  public:     <成员函数或数据成员的说明>
  private:      <数据成员或成员函数的说明>
};      <各个成员函数的实现>
```

下面简单地对上面的格式进行说明：class 是定义类的关键字，<类名>是种标识符，通常用 T 字母开始的字符串作为类名。一对花括号内是类的说明部分(包括前面的类头)说明该类的成员。类的成员包含数据成员和成员函数两部分。从访问权限上来分，类的成员又分为：公有的(public)、私有的(private)和保护的(protected)三类。公有的成员用 public 来说明，公有部分往往是一些操作(即成员函数)，它是提供给用户的接口功能。这部分成员可以在程序中引用。私有的成员用 private 来说明，私有部分通常是一些数据成员，这些成员是用来描述该类中的对象的属性的，用户是无法访问它们的，只有成员函数或经特殊说明的函数才可以引用它们，它们是被用来隐藏的部分。保护类(protected)将在以后介绍。

关键字 public,private 和 protected 被称为访问权限修饰符或访问控制修饰符。它们在类体内(即一对花括号内)出现的先后顺序无关，并且允许多次出现，用它们来说明类成员的访问权限。

其中，<各个成员函数的实现>是类定义中的实现部分，这部分包含所有在类体内说明的函数的定义。如果一个成员函数的类体内定义了，实现部分将不出现。如果所有的成员函数都在类体内定义，则实现部分可以省略。

下面给出一个日期类定义的例子：

```
class Tdate
{
  public:
  void SetDate(int y, int m, int d);
  int IsLeapYear();
  void Print();
  private:
  int year, month, day;
};        //类的实现部分
void Tdate::SetDate(int y, int m, int d)
{
  year = y;
  month = m;
  day = d;
}
int Tdate::IsLeapYear()
{
  return(year%4==0 && year%100!=0) || (year%400==0);
```

```
}
void Tdate::Print();
{
  cout< }
```

这里出现的作用域运算符::是用来标识某个成员函数是属于哪个类的。

该类的定义还可以如下所示：

```
class Tdate
{
  public:
  void SetDate(int y, int m, int d)
   {year=y; month=m; day=d;}
   int IsLeapYear()
   {return(year%4==0 && year%100!=0) || (year%400==0);}
void Print()
   {cout<    private:
  int yeay, month, day;
}
```

这样对成员函数的实现(即函数的定义)都写在了类体内，因此类的实现部分被省略了。如果成员函数定义在类体外，则在函数头的前面要加上该函数所属类的标识，这时使用作用域运算符::。

定义类时应注意的事项

1、在类体中不允许对所定义的数据成员进行初始化。

2、类中的数据成员的类型可以是任意的，包含整型、浮点型、字符型、数组、指针和引用等。也可以是对象。另一个类的对象，可以作该类的成员，但是自身类的对象是不可以的，而自身类的指针或引用又是可以的。当一个类的对象用为这个类的成员时，如果另一个类的的定义在后，需要提前说明。

3、一般地，在类体内先说明公有成员，它们是用户所关心的，后说明私有成员，它们是用户不感兴趣的。在说明数据成员时，一般按数据成员的类型大小，由小至大说明，这样可提高时空利用率。

4、经常习惯地将类定义的说明部分或者整个定义部分(包含实现部分)放到一个头文件中。

3.13 预处理

宏定义

这里介绍两种宏定义类型。它们是:

- 符号常量
- 带参数的宏

现在, 让我们看看符号常量。

#define 符号名 字符串常量

#define 预处理语句象其它的预处理语句一样,以 # 符号作为开头。它可以出现在源文件的任何地方。用 #define 定义的名字的 "作用域" 是从它定义点开始到源文件结尾。

例子 #define ONE 1 在源文件经过预处理后, ONE 将被 1 代替。但是引号内并不进行这样的替代。

例子

```
#define TWO 2
#define MSG "I love c"
#define FMT "x is %d\n"
main()
{
    int x=TWO;
    printf(FMT,x);
    printf("%s\n",MSG);
}
```

这个例子的输出是:

```
x is 2
I love c
```

#define 中的名字同 C 语言中的标识符形式一样, 置换的文本是任意的。一个长的定义应该在行结尾处加一个续行符 \, 以示续行。

在这个例子中: TWO 被 2 置换, MSG 被 "I love c" 置换, FMT 被 "x is %d\n" 置换。如果你不用 #define, 那么:

```
main()
{
    int x=2;
    printf("x is %d\n",x);
    printf("%s\n","I love c");
}
```

这个程序和上边的那个相同。

符号常量很简单是吧? 让我们更进一步, 来学习带参数的宏定义。

#define 也可以用来定义带参量的宏, 正象下面这样

#define max(a,b) a>b?a:b
这个宏被用来得到 a 和 b 中较大的数。只要 max(x,y) 出现在程序中的某处，预处理程序就把它置换为: x>y?x:y

宏的定义是很简单的，但是如果你使用时如果不小心，也许你会出错误。请看:
在 main() 之前你定义了: #define A(x) x*x
在程序中，你想计算 (y+1)*(y+1)，那么可能你会用 A(y+1) 来求得结果。但这是错的，因为 A(y+1)=y+1*y+1。你可以使 A(y+1) 等于 (y+1)*(y+1) 的，只要简单地加上括号:

```
#define A(x) (x)*(x)
```

文件包括

它使编译器把那些指定文件的原始代码包括进来。

它的格式是:
#include "filename" OR #include <filename>

```
/* the name of this program is test */
#include "mytest.h"
main()
{
    float litex, gallons;
    printf("**Liters to Gallons**\n");
    printf("Enter number of liters:\n");
    scanf( "%f", &liters );
    gallons= liters*QTS_PER_LITER/4.0;
    printf("%.2f liters=%.2f gallons\n",liters, gallons );
}
```

程序的输出:

```
**Liters to Gallons**
Enter number of liters:
55.75
55.75 liters=14.73 gallons
```

```
/* Content of the file "mytest.h" */
#define INCHES_PER_CM 0.394
#define CMS_PER_INCH 1/INCHES_PER_CM
#define QTS_PER_LITER 1.057
```

3.14 本章小节

本章主要介绍了 C/C++语言基础，学习完本章后，你会对 C/C++有个大概的了解。要想学好 C/C++，还需要慢慢的学习积累。