APPENDIX L

Introduction to the MinGW C++ Compiler and the wxDev-C++ IDE

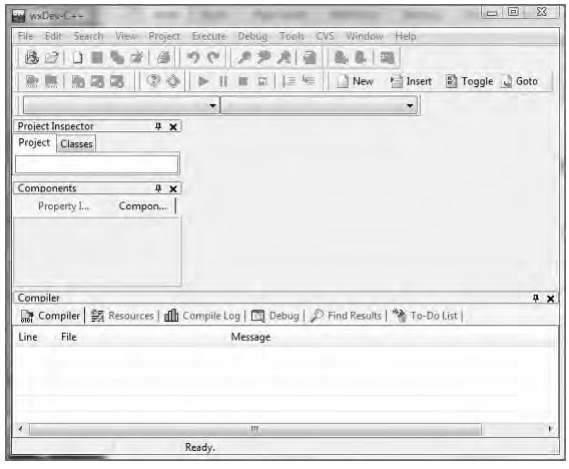
This appendix is an introduction to the use of the wxDev-C++ development environment to create C++ programs. WxDev-C++ regards a collection of files making up a C++ program as constituting a *project*. To create a program, you need to create a project and then add files containing C++ code to it. This guide will lead you through the processes of

- Creating a new project
- Editing Files
- Compiling and executing a program
- Finding and removing compile errors
- Creating multi-file projects
- Determining where data files created by a program are located

Creating a New Project

When you start wxDev-C++ , it displays a window similar to what is shown in Figure L-1.

Figure L-1

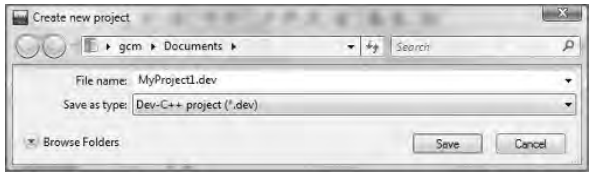


To create a program, use the main menu bar to create a new project by selecting **File**, then **New**, and then finally, **Project**. At this point you will be looking at the **New project** dialog shown in Figure L-2.

Figure L-2

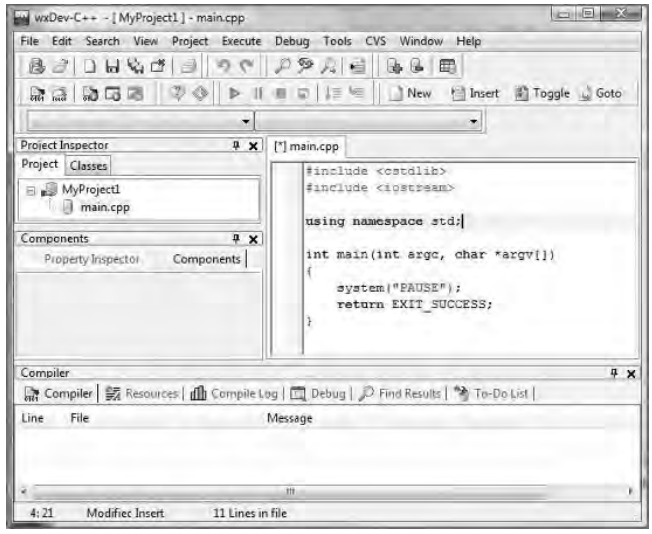
1. At the top of the **New project** dialog there are three tabs labeled **Basic**, **Introduction**, and **Multimedia**. These correspond to three different types of projects you can create. Select the **Basic** tab if it is not already selected.
2. The type of programs you will normally write while learning C++ will be console applications, so select the **Console Application** icon.
3. Next, you need to fill in the information requested by the **Project options** portion at the bottom of the dialog. Enter a name for your project in the textbox provided for that purpose, and then on the right, use the radio buttons to select C++ as the type of your project. Click **Ok**. At this point, a **Create new project** dialog similar to that shown in Figure L-3 will be displayed. By default, wxDev-C++ will store the newly created project in your Microsoft Windows Documents folder. If you prefer another location, you can use the **Browse Folders** control at the bottom left to navigate to a folder of your choice. For now, just accept the default values already entered in the dialog by clicking on the **Save** button.

Figure L-3



At the completion of the preceding steps, wxDev-C++ creates a project with the characteristics you specified and automatically adds a file called `main.cpp` to the project. The file has a skeleton C++ program that you can modify to suit your purposes. This is shown in Figure L-4.

Figure L-4



Notice that the main parts of the development environment are

1. The main menu and tool bar at the top of the wxDev-C++ main window.
2. The **Project Inspector** window on the left. This window has **Project** and **Classes** child windows that are respectively used to view the files and classes that make up your project. Selecting the **Project** tab reveals a node labeled with the name of your project: you can expand this node to view the names of files that have been added to the project. Double-clicking on a file causes it to open in a tab in the Edit window, which is explained next.
3. The Edit window to the right of the **Project Inspector**. It has tabbed pages that display files and allows you to edit them.
4. The **Compiler** window at the bottom. Among other things, it is used to show error messages during compilation.

Editing a File

The main file created by wxDev-C++ initially contains the following skeleton program

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

To edit the file, simply click with the mouse in the edit window and start typing. As an illustration, click at a point just before the statement

```
system("PAUSE");
```

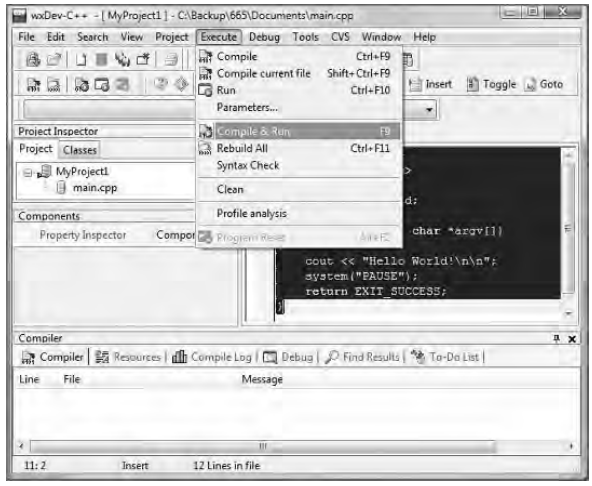
and carefully edit the program so that, with the possible exception of line spacing, it looks as shown here.

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Hello World!\n\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Compiling and Executing a Program

To execute your program, you must first compile your program to create an executable version. WxDev-C++ allows you to compile and run your program without leaving the IDE. You do this by using the **Execute** menu item as shown in Figure L-5.

Figure L-5



When you do this, the IDE runs your program and displays its output in a console window as shown in Figure L-6. The console window stays on your desktop to give you time to observe the program output and goes away when you press a key on the keyboard.

Figure L-6

You may be wondering about the two statements

```
system("PAUSE");
return EXIT_SUCCESS;
```

that appear at the end of this program. `EXIT_SUCCESS` is a return value defined in the `cstdlib` header file to be equal to zero, so the statement

```
return EXIT_SUCCESS;
```

is equivalent to

```
return 0;
```

The statement

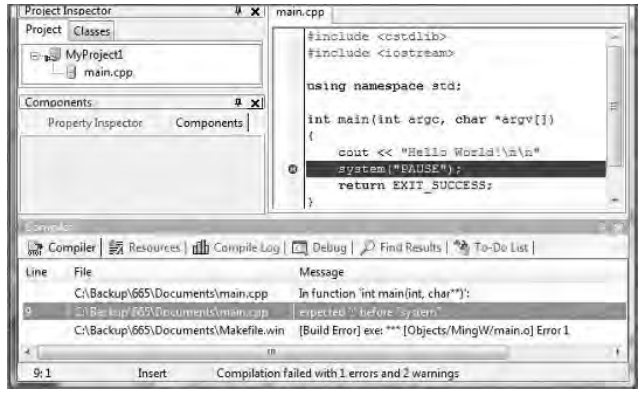
```
system("PAUSE");
```

causes the program to pause execution and continue only after the user presses a key on the keyboard. Without this statement, the program would move on to the `return` statement and terminate, causing the console window in which the program is executing to disappear before the user has had a chance to view the program's output.

Finding and Removing Compile Errors

If your program has compile errors, compilation will fail and wxDev-C++ will display a list of error messages in the compile window at the bottom of the IDE screen. For example, if you omit the semicolon at the end of the `cout` statement and attempt to compile, you will get a screen similar to that shown in Figure L-7.

Figure L-7



Most programs will have more than one compile error. When this happens, wxDev-C++ will list the errors it found and highlight the line in your program that was being compiled when the first error was discovered. Normally the error itself will be just before the highlighted statement. To remove the error, simply edit the program. It is usually best to work in the order the errors were found, correcting one error at a time and recompiling the program after the correction of each individual error.

Saving Files, Closing a Project, and Exiting the IDE

The files you are editing are kept in the computer's central memory and need to be saved on disk before you exit the IDE. You can save the file you are currently working on by selecting **Save** on the **File** menu. To save all files in your project, use the **Save All** entry on the **File** menu. There are a couple of other items on the **File** menu you should know about: **Exit** allows you to exit the IDE, and **Close Project** closes the current project.

Opening an Existing Project

You will often want to continue working on a previously created project. Close the `MyProject1` project if you have not already done so, exit the IDE, and then restart wxDev-C++. Now suppose that you want to reopen `MyProject1`. To do this, use the entry **Open Project** or **File** on the **File** menu. The Microsoft Windows **Open File** dialog will open. Navigate to the folder that contains the project files: this will be whatever folder you specified when the project was created. Once there, locate the `MyProject1.dev` file, select it with the mouse, and then click **Open**. The project will open, and its files will be loaded into the Edit window. You can now continue working on it.

In general, to open an existing project, you must locate the file that represents the desired project. This file will be in the project folder, and will have the same name as the project and a `.dev` extension.

Creating a Multi-File Project

Many of the programs you write will consist of more than one file. For example, suppose your instructor has asked you to write a program that works with a simple class named `Pet`. The class will have a single member variable of type `string` that will be used to hold the name of your pet. You decide to create a project consisting of three files: a `Pet` class specification file `Pet.h`, a `Pet` class implementation file `Pet.cpp`, and a file `PrL-1.cpp` that contains the main function as follows.

Contents of `Pet.h`

```
#include <string>
using namespace std;
class Pet
{
   private:
       string name;  // To hold the pet's name
   public:
       // Constructor
       Pet(string);
       // Accessor function for name
       string getName() const;
};
```

Contents of `Pet.cpp`

```
// Pet class implementation file
#include <string>
#include "Pet.h"
using namespace std;
```

```
// Constructor
Pet::Pet(string str)
{
    name = str;
}

// getName member function
string Pet::getName() const
{
    return name;
}
```

Contents of `PrL-1.cpp`

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

int main()
{
  // Create an instance of the Pet class.
  Pet myPet("Fido");

  // Display my pet's name.
  cout << "My pet's name is " << myPet.getName() << endl;
  system("Pause");
  return 0;
}
```

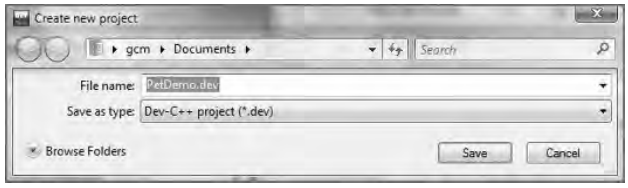
You can use the following steps to create the project.

1. Start wxDev-C++ and select the options for creating a new project named `PetDemo`, as show in Figure L-8.

Figure L-8

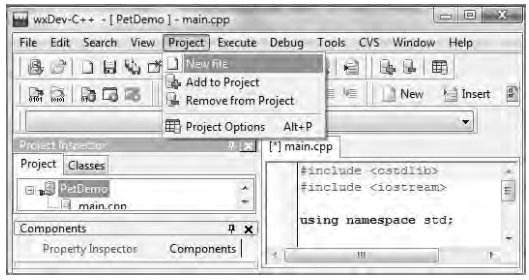
2. Confirm that you want to close whatever project may already be open, if any, and press **Save** on the **Create new project** dialog, as shown in Figure L-9. You are now looking at a project containing a single file `main.cpp`.

Figure L-9



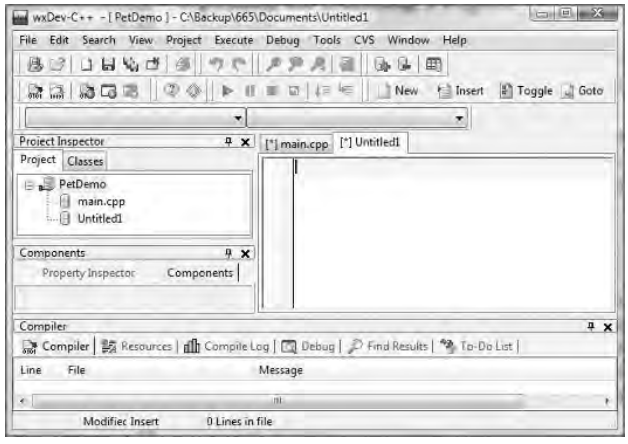
3. We now want to add the class specification file `Pet.h` to the project. On the main menu, select **Project**, and then **New File** as shown in Figure L-10.

Figure L-10



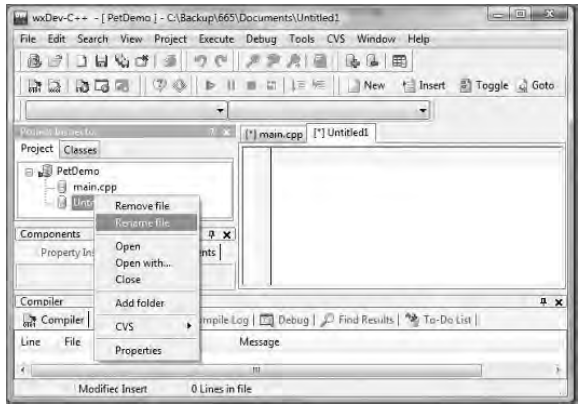
This will add a new file, named `Untitled1`, to the project. The IDE will now appear as in Figure L-11.

Figure L-11



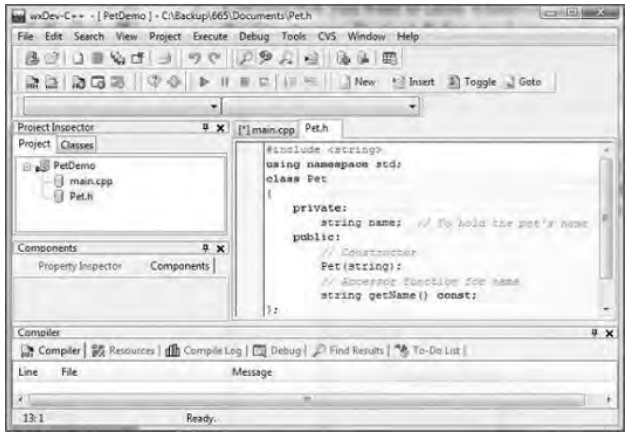
4. Right-click on the `Untitled1` file icon in the **Project** page of the **Project Inspector**, as shown in Figure L-12, and rename the file to `Pet.h` by typing the new name in the small dialog that pops up.

Figure L-12



5. Edit the contents of the new Edit window to contain the code shown above for `Pet.h`, and save the file to disk using the **File** menu to save the file. The situation should now be as in Figure L-13.

Figure L-13



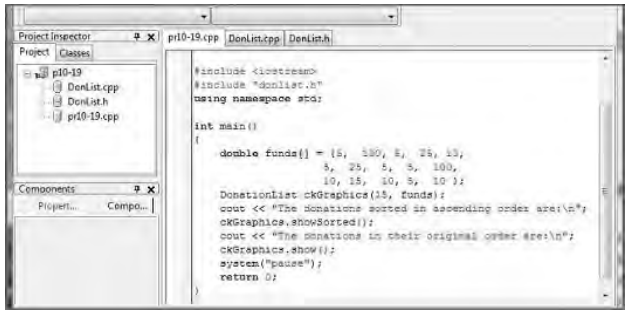
6. Repeat steps 3, 4, and 5 to add a new `Untitled` file to the project, rename it to `Pet.cpp`, edit the file to have the contents of `Pet.cpp` shown above, and then save the file to disk.
7. We are now ready to add the last file, `PrL-1.cpp`, to the project. Right-click on the `main.cpp` file in the **Project** page of the **Project Inspector** and select **Remove File** to remove `main.cpp` from the project. (You may have to save the file before it can be removed from the project). Repeat steps 3, 4, and 5 to add a new `Untitled` file to the project, rename it to `PrL-1.cpp`, and edit it so it contains the code for `PrL-1.cpp` shown earlier.
8. We are now ready to execute the program. Go to the **Execute** menu and select **Compile and Run**.

Executing the Example Programs from the Book

If you wish to compile and execute any of the example programs that come with the book, you will have to create a wxDev-C++ project and add the program files to it. The following steps lead you through the process, using program `Pr10-19` from the book as an example. Begin by copying the files `donlist.h`, `donlist.cpp`, and `Pr10-19.cpp` from the student CD into a folder on your hard drive.

1. Start the IDE and create a console application project named `P10-19`. As usual, the project will be created with a file `main.cpp`. Right click on this file and remove it from the project.
2. Using the **Project** menu, select **Add to project**. (This is the menu item you use to add an existing file to a project.) The **Open File** dialog comes up: use it to navigate to the folder that holds the three files you want to use in your project, select the `DonList.h` file, and add it to the project. Repeat this procedure to add the other two files, `DonList.cpp` and `pr10-19.cpp`, to the project.
3. Edit the `Pr10-19.cpp` file to add the `system("pause");` statement just before the return statement in the main function. The view in the IDE should now look as in Figure L-14.
4. When you are working on a project with several files, it is advisable to compile one file at a time. Select the editor tab that contains `Donlist.cpp`, and then, using the **Execute** menu, select **Compile current file**. After the file compiles with no errors, select the `Pr10-19.cpp` tab and compile the file in that tab by selecting **Compile current file** from the **Execute** menu.
5. Execute the program by selecting **Run** from the **Execute** menu.

Figure L-14



The **Execute** menu has several other items that are useful. Among these is the **Rebuild All**, which causes all files in the current project to be compiled and then creates an executable program.

Where Files are Stored

The **Create new project** dialog box shown in Figure L-3 has a **Browse Folders** control that lets you specify a *project folder*, which is the location where the files that make up your project will be stored. As you can see in Figure L-13, wxDev-C++ displays the location of source files in the title bar of its main window.

When you invoke **Compile and Run** on the **Execute** menu, wxDev-C++ creates an executable file that is used to run the program. This file has the same name as the project, but has an `.exe` extension. The executable file is not stored in the same location as the other project files. Instead, wxDev-C++ creates two subfolders named `output` and `mingw`. The `output` folder is a subfolder of the project folder, and `mingw` is a subfolder of `output`. The executable program is stored in the `mingw` folder.

It is sometimes necessary to execute programs that read or write files. The location of a file can be specified by a full pathname, as in `c:\temp\myfile.txt`, or by relative path name that omits the path to the containing folder and just gives the name of the file, as in `myfile.txt`. In this latter case, programs executed from inside of wxDev-C++ will look for the input or output files in the same folder than contains the executable. Thus for example, a program whose project folder is `c:\temp`, and which reads and writes a file named `file1.txt` will look for the file in the folder `c:\temp\output\mingw`. The upshot of all this is that, when you are running programs that read file inputs, you should place the input files in the `output\mingw` folder of the project folder if you want the program to access them without using full pathnames.