Programming with the Dev C++ IDE

1 Introduction to the IDE

Dev-C++ is a full-featured Integrated Development Environment (IDE) for the C/C++ programming language. As similar IDEs, it offers to the programmer a simple and unified tool to edit, compile, link, and debug programs. It also provides support for the management of the files of a program in "projects" containing all the elements required to produce a final executable program.

Dev-C++ uses Mingw port of GCC (GNU Compiler Collection) as a compiler. It can creates native Win32 executables, either console or GUI, as well as DLLs and static libraries. Dev-C++ can also be used in combination with Cygwin or any other GCC based compiler. In this session, we will use Mingw --included in the default Dev-C++ distribution-- to create console C programs.

Dev-C++ is a Free Software distributed under the terms of the GNU General Public License (GPL). The IDE can be downloaded here:

http://prdownloads.sourceforge.net/dev-cpp/devcpp-4.9.9.2_setup.exe.

Dev-C++ can be installed on any Windows machine with Windows XP and Windows 7. This tutorial uses Dev-C++ 4.9.9.2 on Windows 7 (configuration in the computer labs as of course 2012-2013). For the sake of simplicity, use the default installation options in your home computer. For Windows 8, you are recommended to install the Orwell Dev-C++ fork with bundled TDM compiler available here:

http://sourceforge.net/projects/orwelldevcpp/files/Setup%20Releases/Dev-Cpp%205.4.0%20TDM-GCC%20x64%204.7.1%20Setup.exe/download

Dev-C++ features are:

- Support GCC based compilers (Mingw included)
- Integrated debugging (with GDB)
- Support for multiple languages (localization)
- Class Browser
- Debug variable Browser
- Code Completion
- Function Listing
- Project Manager
- Customizable syntax highlighting editor
- Quickly create Windows, console, static libraries and DLL
- Support of templates for creating your own project types
- Makefile creation
- Edit and compile Resource files
- Tool Manager
- Print support

- Find and replace facilities
- Package manager, for easy installation of add-on libraries

Documentation about the C programming language and the IDE itself can be accessed from Dev-C++ by clicking on Help > Help on Dev-C++.

2 First steps

The application development process encompasses the following steps:

1. **Create a project**
   The type of application and the programming language to be used are specified.
2. **Write source code**
   Write the program in C and save the source code file.
3. **Compile and link the code**
   The source code is compiled and linked to generate a running program. Other files of the project may be created.
4. **Fix compilation errors, if any**
   If the syntax of the program is not correct, the compilation fails and the compiler generates a message related to the error/s. The programmer must correct the errors.
5. **Run the program**
   Run the program to validate the functioning.
6. **Fix execution errors, if any**
   If the actions performed by the program are not as expected, it is necessary to correct the source code. It may be also convenient to use the debugger to find complex errors.

2.1 Project creation

A project is a center for managing your different source files and options inside Dev-C++. It helps you navigate through your code, and easily set different parameters, like the type of program you are doing (GUI, console, DLL ...).

A project groups several files with a common purpose. In our programs, projects will include a file with metadata about the project (.dev), a file with C source code (.c), a file with object code (.o), and a file with linking instructions (makefile.win). Only the .c file must be explicitly created –the remaining files are automatically created by Dev-C++.

When creating a project, Dev-C++ asks the user where the files must be stored. It is convenient to use different folders for each project, since by default, the source code is named main.c and previous files can be overwritten.

Project type determines the type of application that will be developed. In this lecture, we will create Console projects. These are projects that run in a text-mode screen without windows or graphics. User interaction is performed by typing information on the keyboard (input) and printing characters on the screen (output) with proper read and write instructions.

2.1.1 Steps

a. Start Dev-C++

Start the IDE from the Program folder Dev-C++ or Bloodshed Dev-C++.

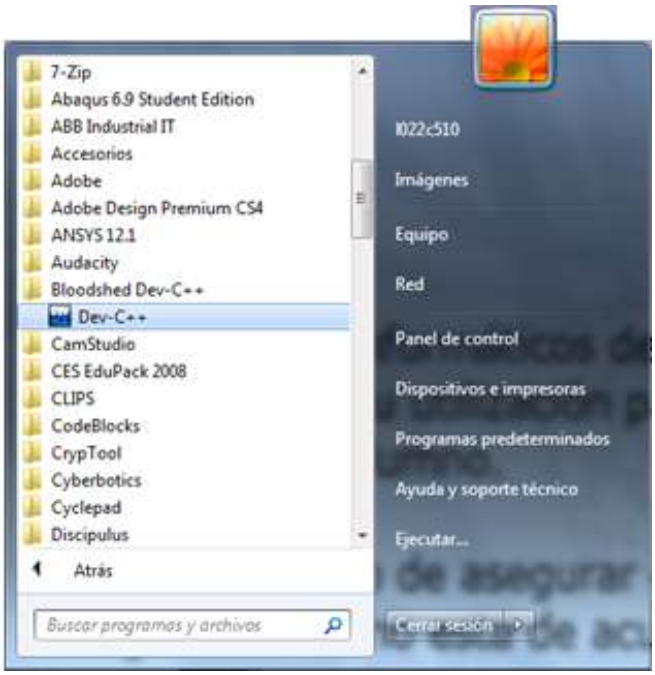

Figure 1. Running Dev-C++ in the computer lab

b. <u>Create a project</u>

Before creating the source code file, it is necessary to create a project (File > New > Project).

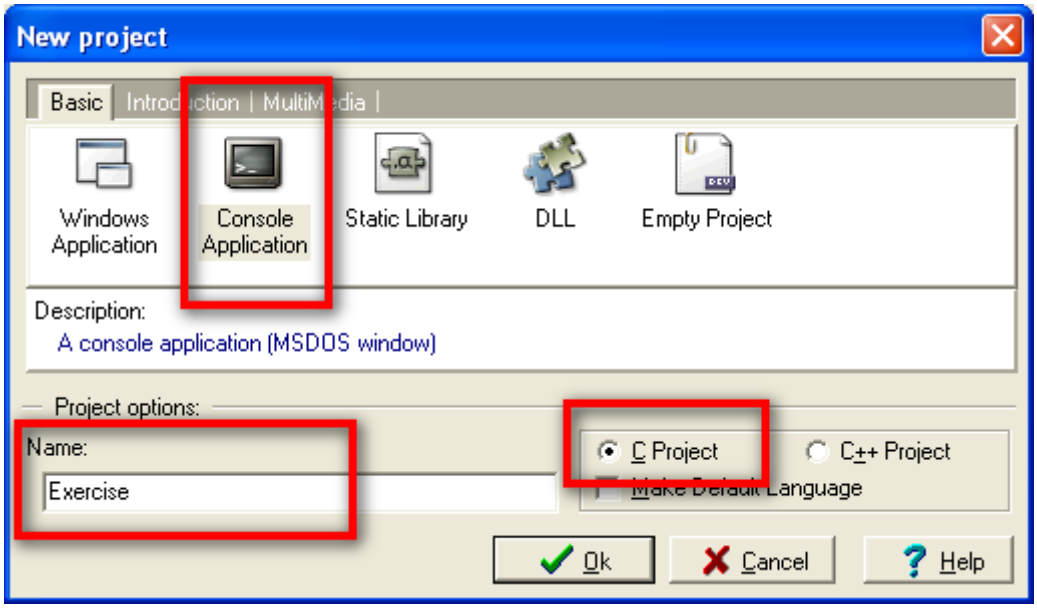

Figure 2. Project creation parameters

Options:

Console application

Name (name of the project)

C Project (for C language)

Select the folder to store the files in the next window. It is convenient to store each project in a different folder.

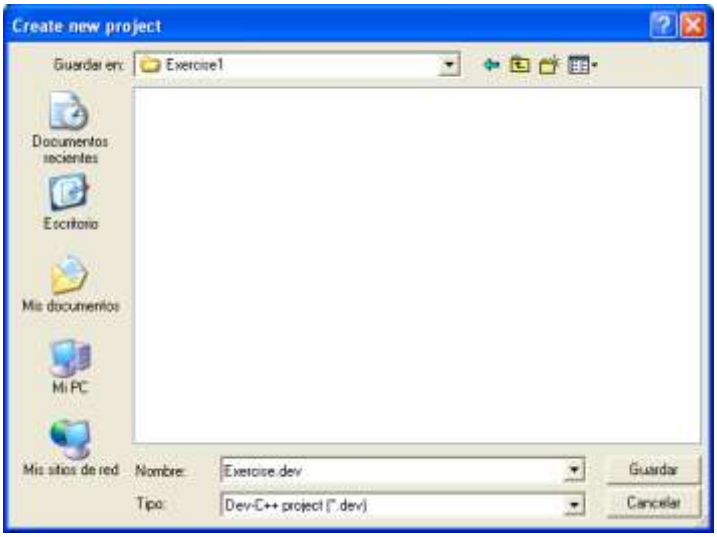

Figure 3. Saving project in a specific folder

After indicating the folder where the project configuration file (.dev) will be saved, the IDE generates a basic source code file (by default, main.c). These files are not saved in the project folder until the programmer saves or compiles the program (see below).

NOTE: The statement `system("PAUSE");` is automatically included to pause the execution of the program before closing the terminal window in order to make it possible to see the output of the program.

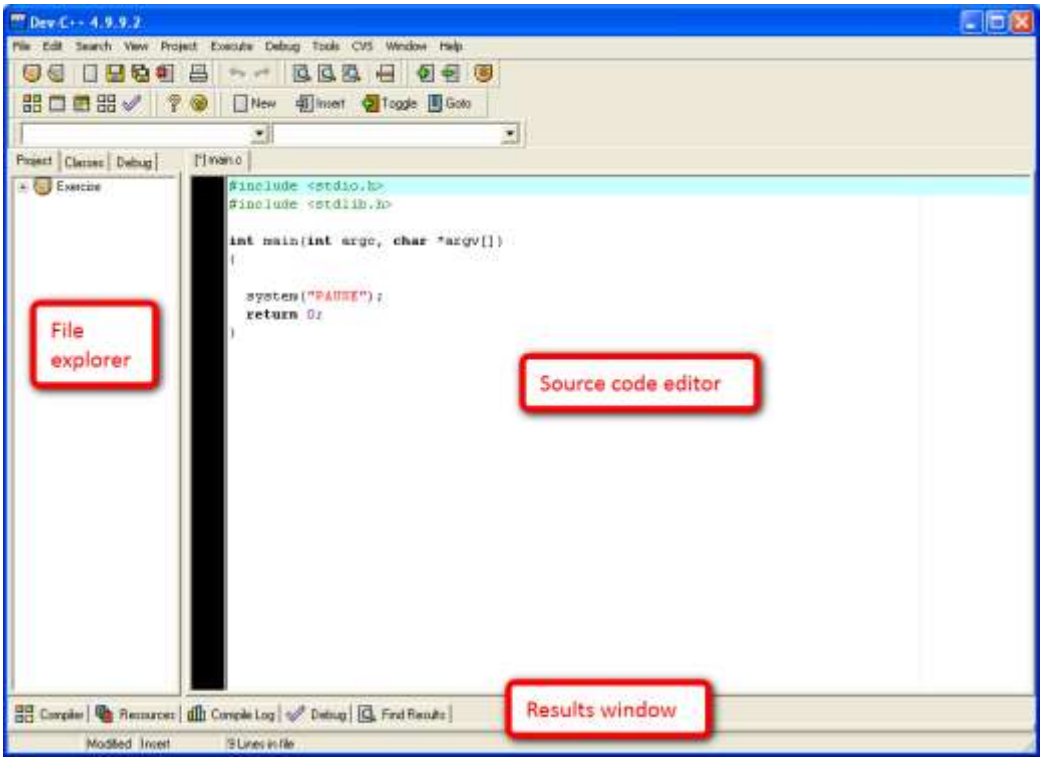

Figure 4. Dev-C++ launches after project creation

The IDE window includes three sub-windows: the *Project Files Explorer*, the *Result Tabs*, and the *Source Code Editor*. These windows can be resized and minimized.

The *Files Explorer* window shows the name of the project and the included files. The *Project* tab usually contains a single file with the source code of the program. In this pane, we can find two additional tabs: *Classes* and *Debug*. *Classes* tab shows the functions of the program. *Debug* tab shows watched variables in the debugging process.

The *Results* window is used to present the results of the actions of the IDE: compilation errors, compiling directives, debugging commands, etc.

The *Source code editor* shows the code of the program.

c. Files created

The files of the project are saved when the source code file is saved. The remaining files of the project are saved when the application is compiled and stored in the project folder.

File	Extension	Description
Project file	.dev	Project configuration data
Makefile	.win	Required for the compilation process. Manages program dependencies and includes instructions for the linker
Source code file	.c	Source code
Object file	.o	Object code resulting from the compilation of the source code. Each .c has a corresponding .o after the compilation
Executable file	.exe	Executable application

2.2 Writing source code

Once the project has been created, we can start writing our C program.

It is recommended to use the classic color configuration of the editor (Tools > Editor options > Syntax > Color Speed Settings > Classic) and to activate the support for opening and closing brackets (Tools > Editor options > General > Highlight matching braces / parenthesis).

The editor highlights with different colors keywords and other elements of the C language. The classic scheme uses:

- Light blue for comments
- Green for included libraries
- Red for text strings
- Bold black for C keywords

For instance, the *HelloWorld* program is shown as follows:

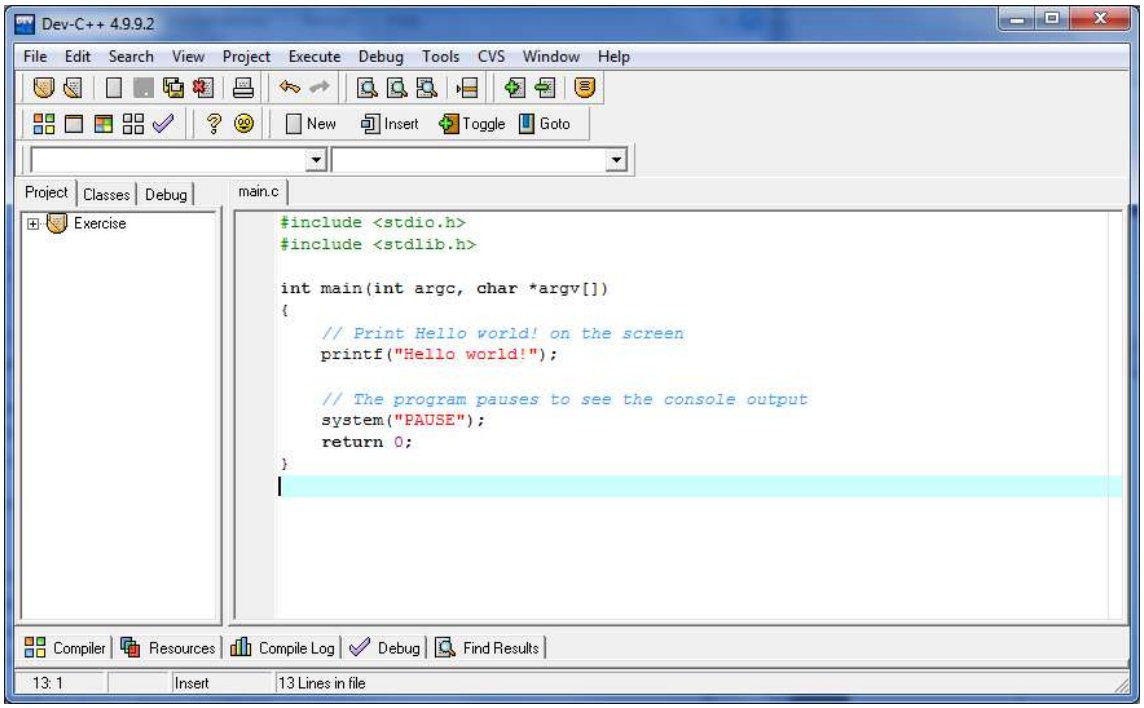

Figure 5. The *HelloWorld* program

Exercise 1. Write the source code of the program HelloWorld in C. Open the file created by the IDE (main.c) with a text editor (e.g. Notepad) and check its contents.

2.3 Compilation and link

To run a program, the source code must be compiled and linked. Dev-C++ performs the complete process by clicking the **Compile** button (or Ctrl + F9).

While the compilation and link process is being performed, the IDE shows a dialog with related information. If the process is successful, the window shows the message **Done**.

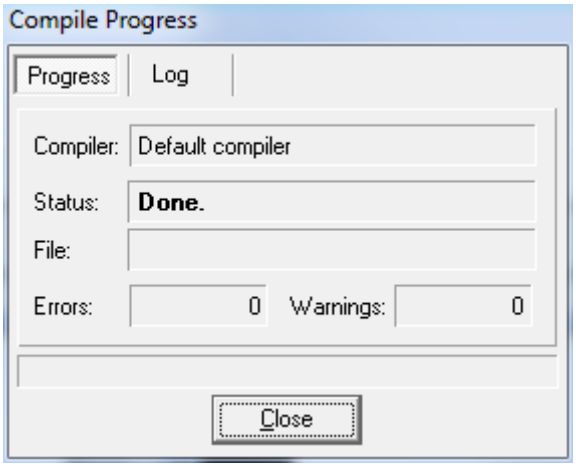

Figure 6. Compilation window (success)

The *Compile Log* tab shows information about the process.

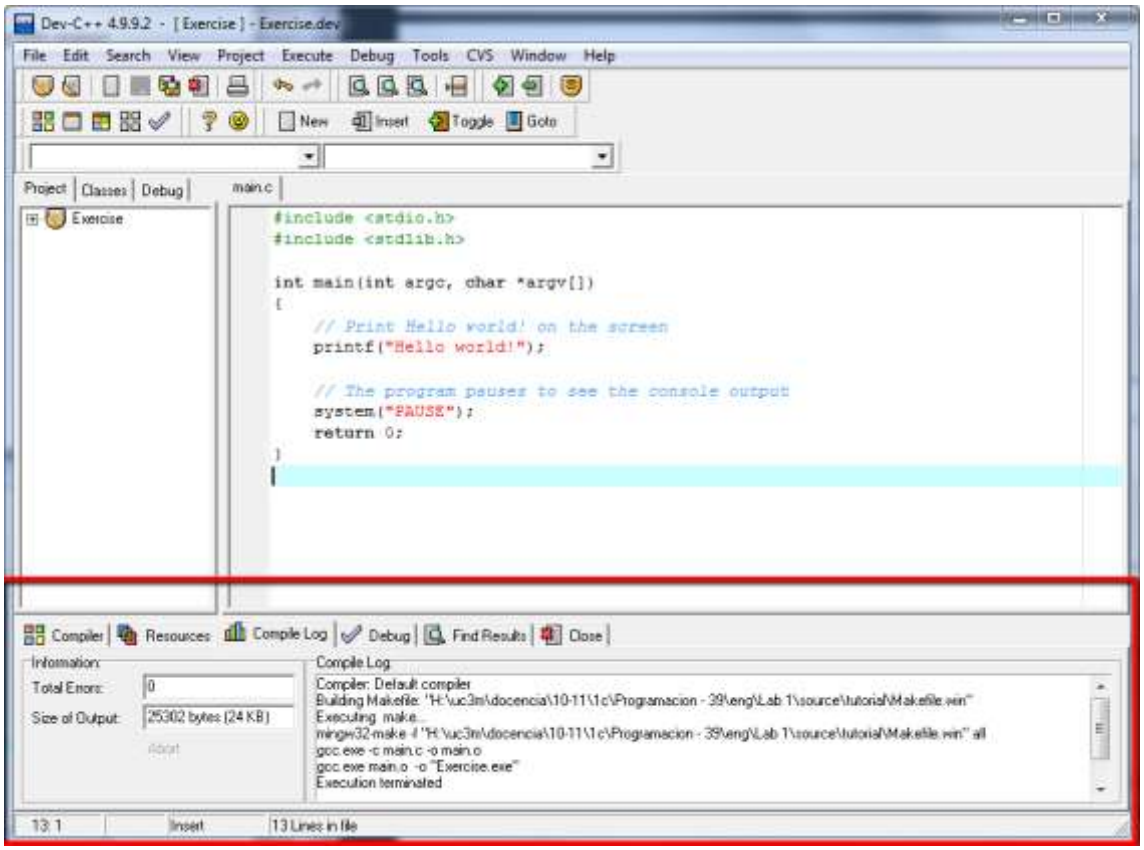

Figure 7. Compilation process output

After the process, we can check that the files described in the previous table have been created.

Exercise 2. Compile the *HelloWorld* program and check that the corresponding files have been created, including the .exe program. Check that it is possible to run the program out from the Windows Explorer by double-clicking on it.

2.4 Fixing compilation errors

Compilation errors are errors that are detected by the compiler. They are also named design-time errors. To correct compilation errors, it is recommended to solve the first one and then to re-compile the program again, since next errors are frequently wrong sections of the code that the compiler cannot interpret as a result of the first error.

Dev-C++ underlines in red the line of code where the compilation error has been detected. The *Compile* tab of the *Results* window provides a detailed description of the error. The *Compile Log* shows the error message issued by the compiler program.

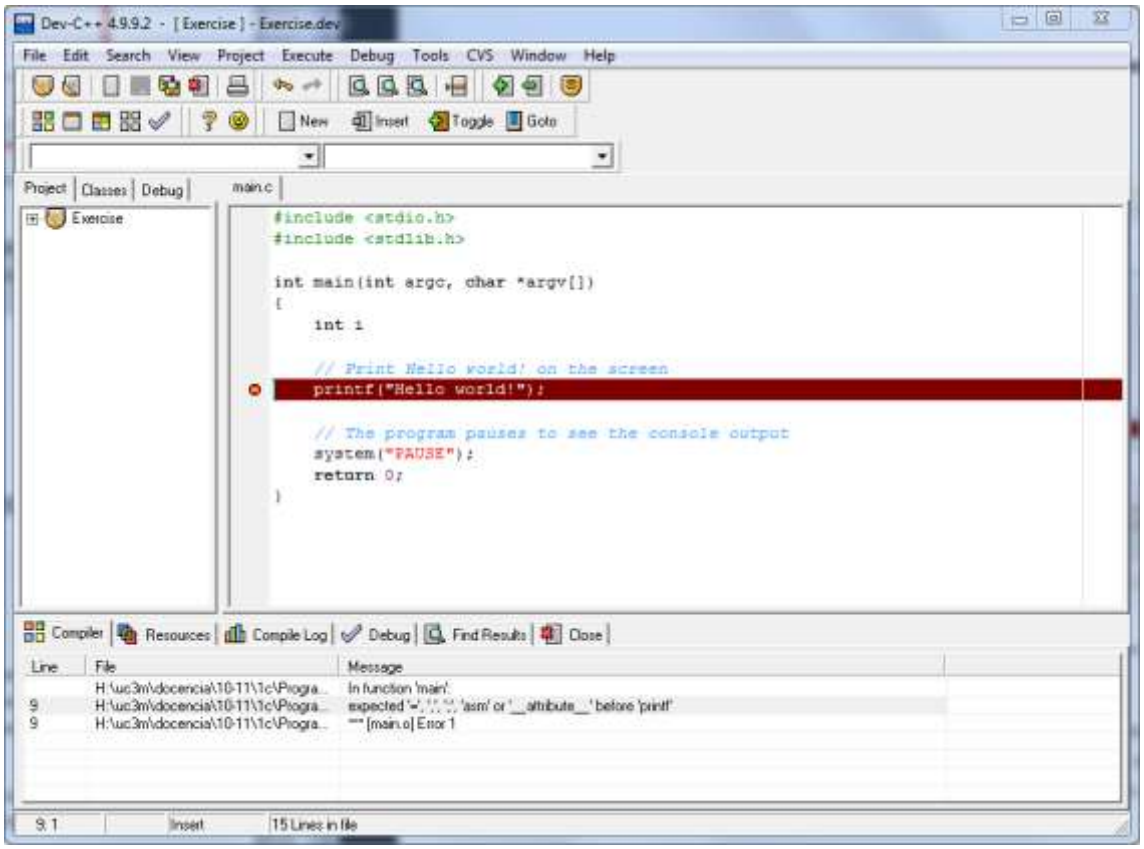

Figure 8. Compilation errors

Common errors are: selecting a wrong project type (instead of Console), selecting a wrong language type (instead of C), or library inclusion errors (wrong syntax, library missing, etc.).

2.5 Running the program

As a result of the link process, an executable program is created. To run this program, we have to click the Run button (Ctrl + F10). Alternatively, we can find the executable program (.exe) in the project folder and double-click on it.

NOTE: It is possible to compile and run the program in a single step by clicking the **Compile & Run** button (F9).

Exercise 3. Change the *HelloWorld* program to introduce a compilation error; for instance, we can mistype the `printf` function or remove one of the double-quote marks in the character string. Compile and run the program (use the **Compile & Run button**). Check the error messages.

2.6 Fixing execution errors

Developing a syntactically-correct program does not implies that it performs the required procedures. Complex programs are error-prone and usually the logic implemented in the source code does not lead to the expected behavior. These errors are named execution or

runtime errors, since they are detected when the program is running. Therefore, it may be necessary to change the source code and repeat the compilation and link procedure.

The IDE includes a debugger, which is a supporting tool to find runtime errors. To use the debugger, it is necessary to modify the compiler options to include debugging information in the object and the executable files (**Tools > Compiler Options > Compiler**). This is done by adding the –g parameter to the calls to the compiler and the linker.

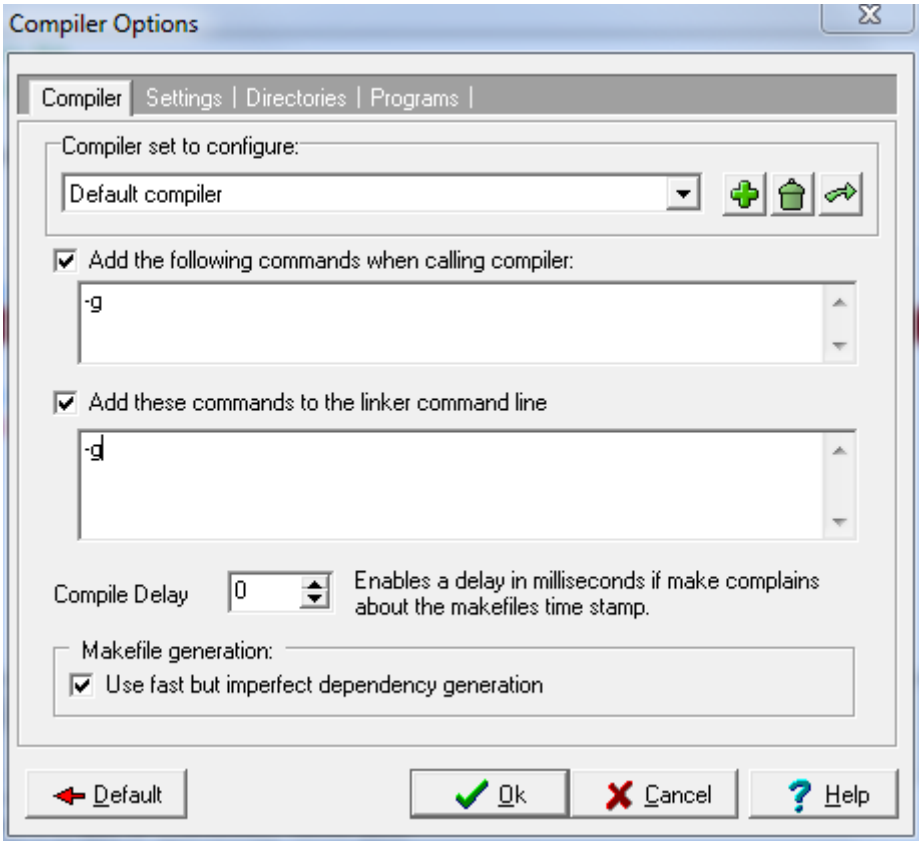

Figure 9. Compiler options for activating the debugger

When the program is compiled and link with these options, we can run the program in debug mode by clicking on the **Debug** button (F8). This mode allows running the program step by step (instructions are executed one-by-one), stopping the execution at breakpoints, or watch the value of a variable at any time of the execution.

Breakpoints

To create a breakpoint, we can (alternatively): (a) select the line of the code and type Ctrl+F5; (b) right-click on the line and select **Toggle Breakpoint**; (c) click on the gray vertical bar on the left of the code. The instruction will be highlighted in red.

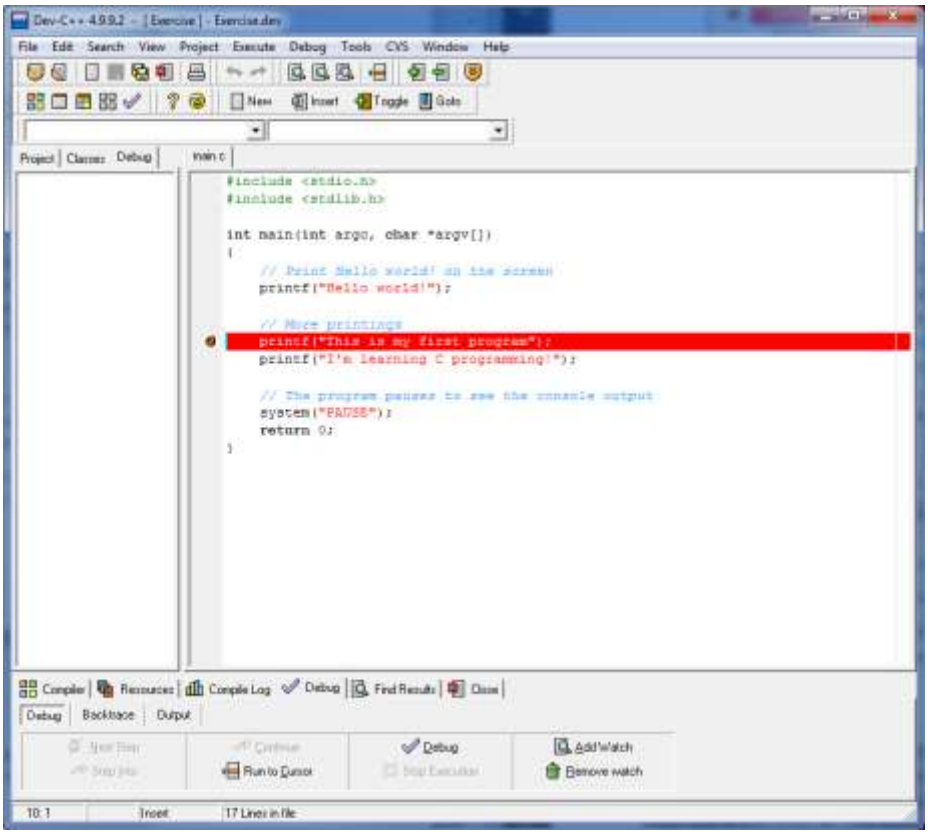

Figure 10. Setting a breakpoint

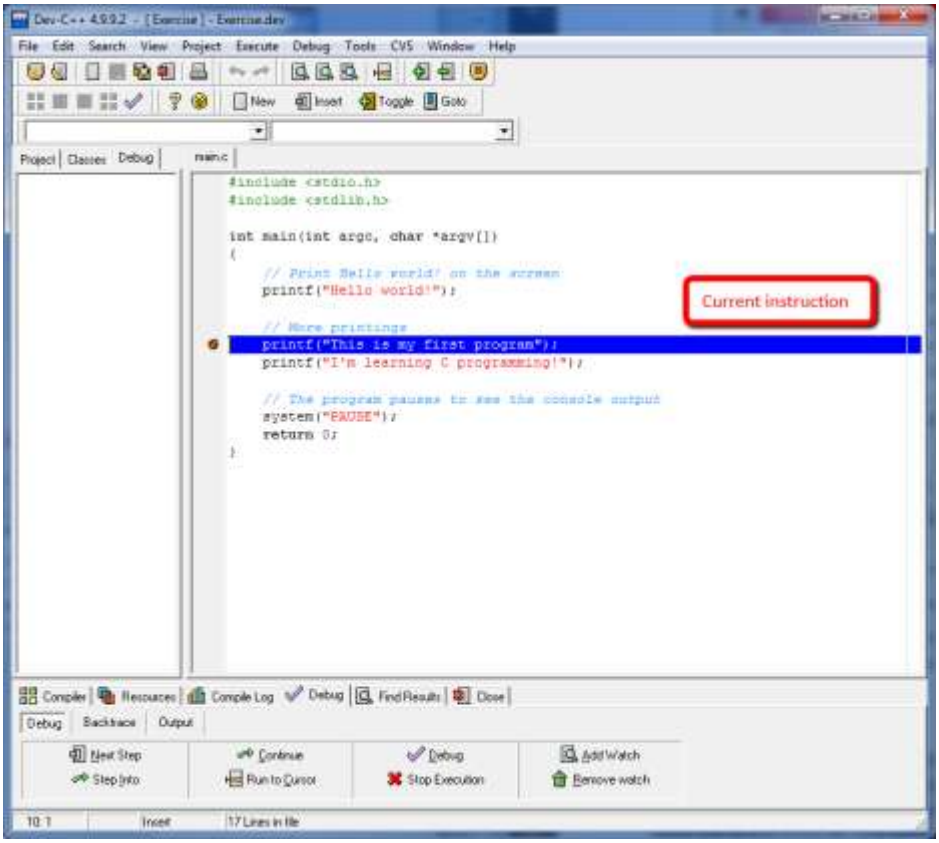

Figure 11. Pausing the program at a breakpoint

When the program runs in *Debug* mode (F8), the execution is paused at this line. From this point, the execution can continue normally or step by step. The current instruction is highlighted in blue.

The *Debug* tab of the *Results* window shows the debugging commands of Dev-C++. As depicted in the following picture, it is possible to continue the execution, run the program step by step, run to the cursor, etc.

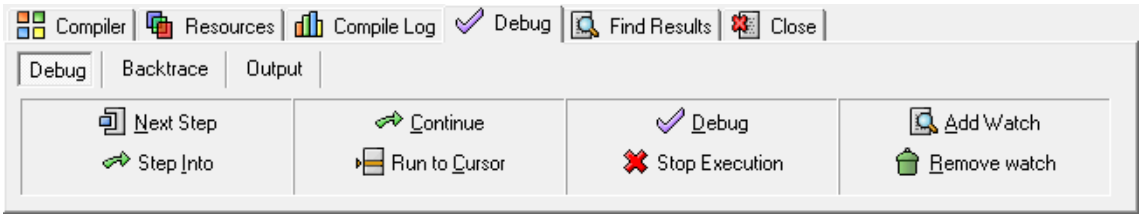

Figure 12. Debug mode options to run a program

NOTE: It is possible to start the execution of a program in Debug mode without defining any breakpoint by using **Run to Cursor**.

Exercise 4. Modify the *HelloWorld* program by adding two additional printing instructions. Toggle a breakpoint in the first one of them. Run the program in debug mode and check that the execution is stopped at the breakpoint. Continue the execution step by step. Check that the printings are performed after the execution of the instructions.

Watches

The Debug mode allows us to consult or watch the value of a variable at any time during the execution of the program. To watch a variable, the program must be stopped (with a breakpoint or after using Run to Cursor). In this situation, we can select **Add Watch** in the *Debug* tab (alternatively, F4 or **Debug > Add Watch**) and type the variable. The values of the watched variables are listed in the *Debug* tab of the *File Explorer* window; if these values are changed in the program, the change is shown.

Exercise 5. Extend the *HelloWorld* with the code of the previous figure. Watch the value of the integer variable n during the execution of the program.

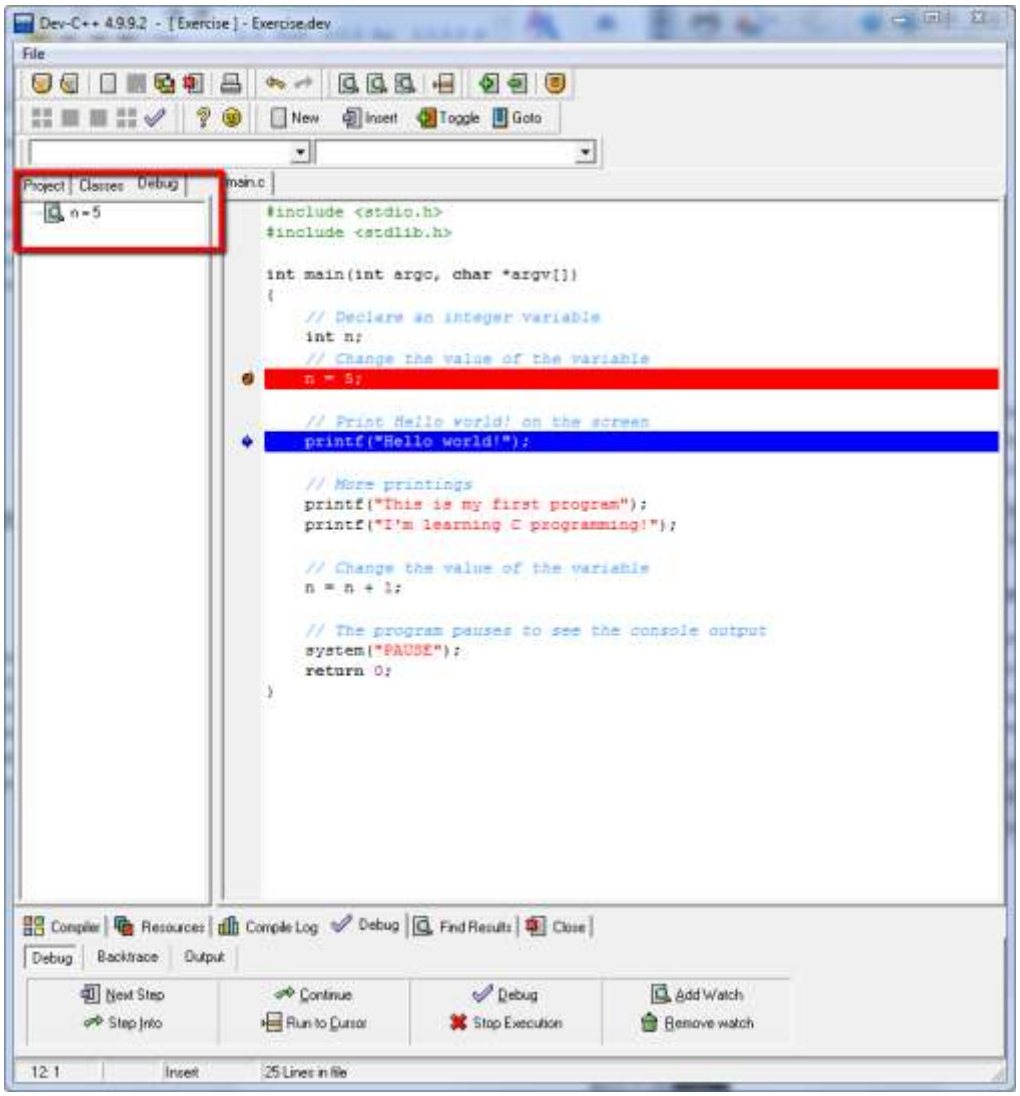

Figure 12. Watching variables in debug mode

2.7 Start over: create a new program

To create a new program, we have to repeat the steps 2.1-2.6 and create a new project, or alternatively, replace the previous source code with the new code. We cannot have two files with a `main` function in the same project. Therefore, it is convenient to create a different project for each program.

Exercise 6. Develop a C program that asks the user for two integer values, multiplies them, and prints the result on the screen. Add watches to all the variables of the program. Run the program step by step and check how the values are changed by the instructions. What is the value of the variables before reading them from the keyboard?