Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

GNG 1106
Fundamentals of Engineering Computation

CodeBlocks: The Project and the Debugger
Fall 2017

1. Introduction

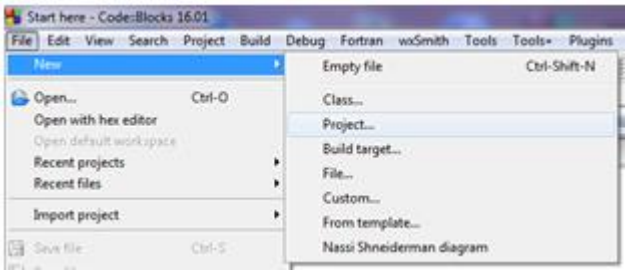
This document describes how to create projects in CodeBlocks and completes the lab manual. A project gives you access to the CodeBlocks debugger and later in the course will allow you to include the PLplot library to create plots.

2. The CodeBlocks Project

A software project may contain many C files which are treated as software modules. This concept shall not be studied in this course. Rather, we shall include a single file in the C project. As mentioned in the introduction, the project will give you access to the debugger which helps to analyze and correct logic errors (bugs) in a program. The debugger allows you to execute instructions one at a time in the program and examine current values in the variables. You will recognize a similarity between the programming model and the operation of the debugger.

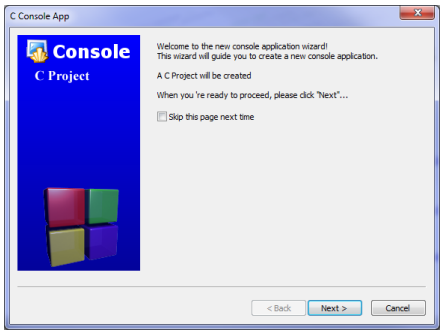
To create a CodeBlocks Project, use the following steps:

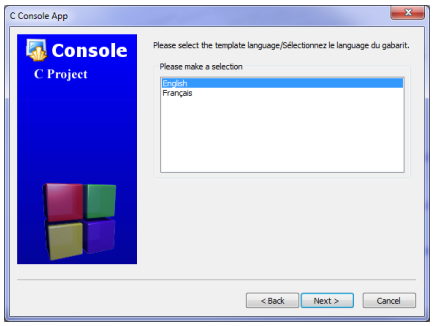
1) Select *File* | *New* | *Project* to open the "*New from template*" window.



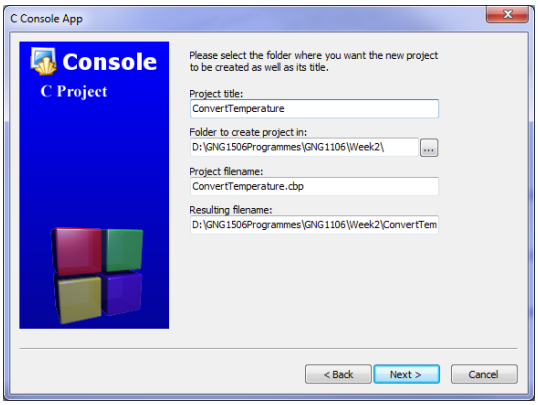
2) Click on "*C Console App*" icon:  and click on the *Go* button to start the project Wizard.

3) Click *Next* in the first window. In the second Window select your language of choice and click *Next*.

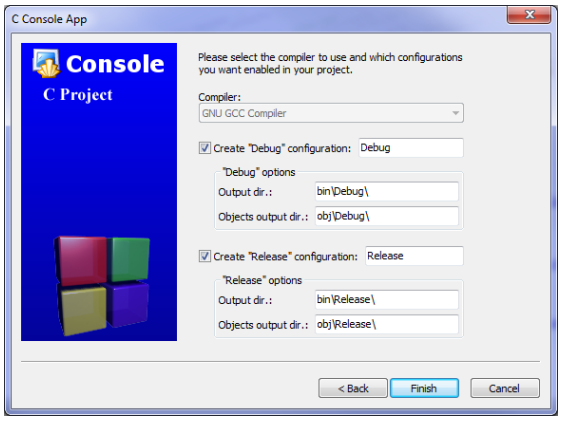




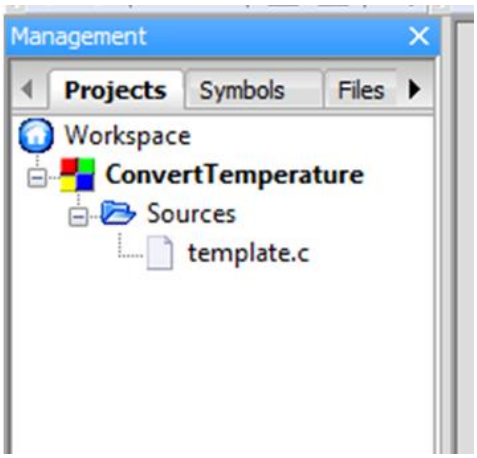
4) In the third window, enter a *Project title*. Then click on the "…" button to Browse to a folder on your hard drive where the project will be created. Note that the other two fields are filled in automatically. Click *Next*.

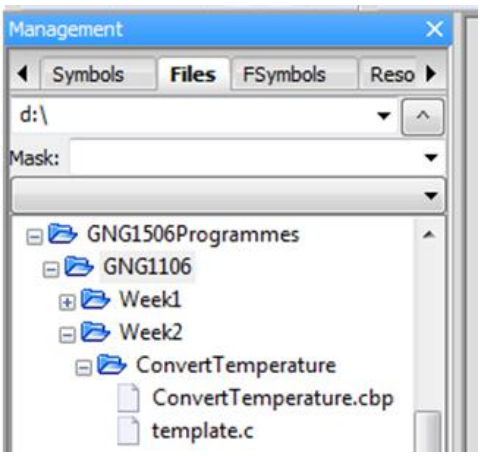


5) In the final window of the wizard ensure that both "*Debug*" and "*Release*" configuration have been checked off. Then click *Finish*.



6) In the *Projects* tab, you will see the newly created project. You can also see the contents of the project directory using the *Files* tab. Note the template file (template.c) provided.



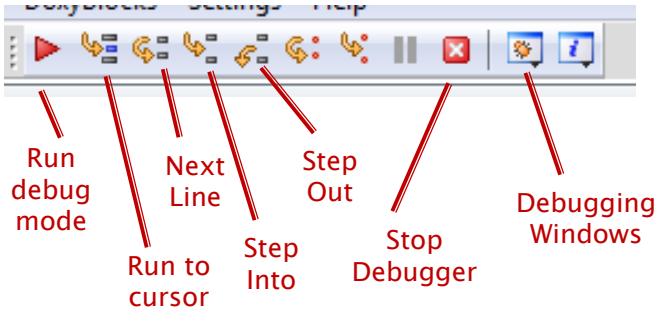


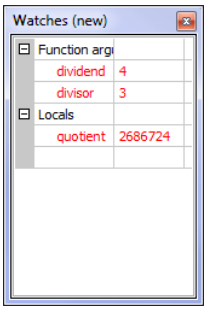
7) From the *Projects* tab:
    a. You can rename the file by doing a right-click on the file name and select "*Rename file…*".
    b. You can remove the file from the project by right clicking on the file and selecting "*Remove file from the project*". Note that the file is not removed from the hard disk (see the *Files* tab).
    c. You can format the source code in the file by right clicking on the file and selecting "*Format this file (Astyle)*". Do this on a regular basis.
    d. Double clicking on the file will open the file for editing in the editor.
    e. To add a new file, select *File* | *New* | *Empty file* and follow instructions.
    f. Close the project by right clicking on the Project name and select "*Close Project*".
    g. To open a closed project, select *File* | *Open* and browse to your project directory and select the *cbp* file. You can also use *File* | *Recent Projects* that gives a list of your recent projects.
    h. The *Build* and *Run* icons apply to the active project for compilation and executing the project.
8) From the *Files* tab
    a. The *Files* tab allows you to add existing files to your project. Find the file you which to copy into your project, right click on the file name and select "*Copy to*". Use the window that appears to find your *Project* folder. Note to add the file to the project you must also go to the *Project* tab, right click on the project name and select "*Add files…*" and then select the file you copied to your project.
    b. To delete a project you must first close it in the *Projects* tab, then in the *Files* tab delete the project by right clicking on the file and select "Delete".
    c. To delete files removed from the project, in the Files tab, go to the project directory, right click on the file to be deleted and select "Delete".

3. The CodeBlocks Debugger

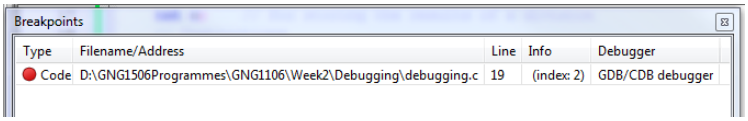
The debugger is a practical tool that allows the control of the execution of a program to see the effect of its execution on the variables used in the program and thus follow the logic of the program. Its main role is to investigate and correct logic errors in a program. As a teaching tool, it will allow you to better understand the execution logic of a program.

The following is a summary of the debugging icons available in CodeBlocks.



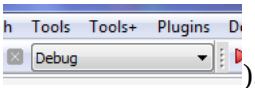


Watches window shows content of working memory



Breakpoints window allows management of breakpoints

Download the file "debugging.zip", available from lab 2, and unzip the file contents (a CodeBlocks project with a program that contains some logic errors, i.e. bugs). Open the project with CodeBlocks. The program is a simple program that requests two integer values from the user, divides the integers, and prints the quotient result.

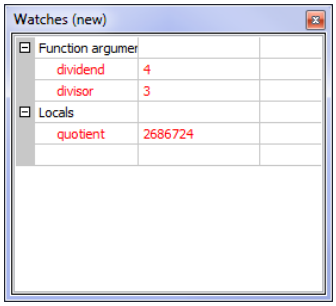
Compile the project (ensure that in the tool bar you see Debug 

). Open the source file *debugging.c*. Run the project with the input values 4 and 3 (i.e. divide 4/3). The program gives the value 4, which is incorrect. The following steps demonstrate the use of the debugger to discover the bug or bugs.

1) Run to cursor: This feature starts the programs and runs it to the cursor. In the source code, click anywhere on the line containing the first `printf` in the `main` function. Then click on the "*Run to cursor*" icon. The program will start its execution and stop at that line. You will see a yellow arrow pointing to the next line to be executed, in this case the call to `printf`.
2) Next Line:
   a. Click on this icon to run the program until the next line. This is practical when the current instruction to be executed contains a function call. In the debugging example, the `printf` function call shall be executed and you should see the line printed on the console. The next instruction to run is the call to `scanf`.
   b. Click on the *Next line* icon again. Note that this time the program is waiting for input into the console. Type in "4 3" followed by return. You will see the program stop at the next line which calls the `divide` function.
3) Step into: This icon is used to step into a function. Click on it to find yourself in the `divide` function. Note that declarations are automatically executed and the next instruction to be executed is the arithmetic expression in the `divide` function.
4) Step out: Clicking on this icon has the debugger complete the execution of the currently executing function and return to the calling function. Click on this icon and you will find yourself back in the `main` function. Note that the debugger points to the instruction with the call to the `divide` function since the next operation is to assign the return value to the variable `c`. Click on the next line

icon to move to the `printf` instruction. Another click on the *Next line* icon and the debugger moves to the end of the program and you will see the output (incorrect) on the console.

5) Breakpoints: It's great to trace through the program, but breakpoints provides the means to indicate to the debugger a line (point) to stop (break).
   a. Go into the `divide` function and right click on line 37 and select "*Toggle breakpoint*".
   b. You will see a red dot appear next to line 37.
   c. Stop the debugger by clicking on the *Stop Debugger* icon (red x). Your console should close.
   d. Now click on the *Run Debug Mode* icon. The debugger will start from the beginning of the program and stop in the divide function line 37. Note that you will need to enter "4 3" again.
   e. Click on the *Debugging Windows* icon and select *breakpoints*. A window shows all breakpoints that are set. This window allows you manage breakpoints, including enabling/disabling and removing breakpoints. Right click on the breakpoint and select the desired function.
6) Watches: Nice to trace through the program, but it is not very useful if the values of variables cannot be examined. Again click on the *Debugging Windows* icon and select "*Watches*". A window will appear to show the contents of the working memory allocated to the function. Note that the function parameters (called arguments by CodeBlocks) are shown along with the local variable quotient. Note the value of quotient which is an arbitrary value.



   a. Click on *Next line* and see what happens to the values in the *Watches* window. We see that quotient is set to 1, which is the correct value.
   b. Click again on *Next line* until you go back to the main function. When you return to the main function, note that the *Watches* window contains the working memory of the `main` function. When you reach the `printf` statement, note that `c` contains the value 4 which is incorrect. So where is the bug? In the return statement. It is returning the value of dividend, not quotient.

You have explored the use of an important tool to verify the execution of a program and to help correct logic errors.