Code::Blocks Manual

Introduction to Computer Programming - Part 1
(CS101.1x)

Prepared by
Sandeep Prasad

Edited by
Firuza Aibara

Guided by
Nagesh Karmali

Department of Computer Science and Engineering
Indian Institute of Technology - Bombay
Mumbai - 400076.

August 5, 2014

Contents

List of Figures

1 Introduction to Code::Blocks

"Code::Blocks is a free C++ IDE built to meet the most demanding needs of its users." [1]. Developed by 'The Code::Blocks Team', Code::Block is a free, open-source [2] and cross-platform IDE, which supports various free compilers. It is built around plugin framework, which allows functionality of Code::Block to be extended by installing appropriate plugins. Plugins required for compiling and debugging are already provided by default. This manual is prepared after installing and testing Code::Block on Ubuntu 12.04[1] and Windows 7.[2]

2 Code::Block IDE

Code::Block IDE is shown in figure 1 (Ubuntu 12.04). The main parts of Code::Block along with figures are discussed below

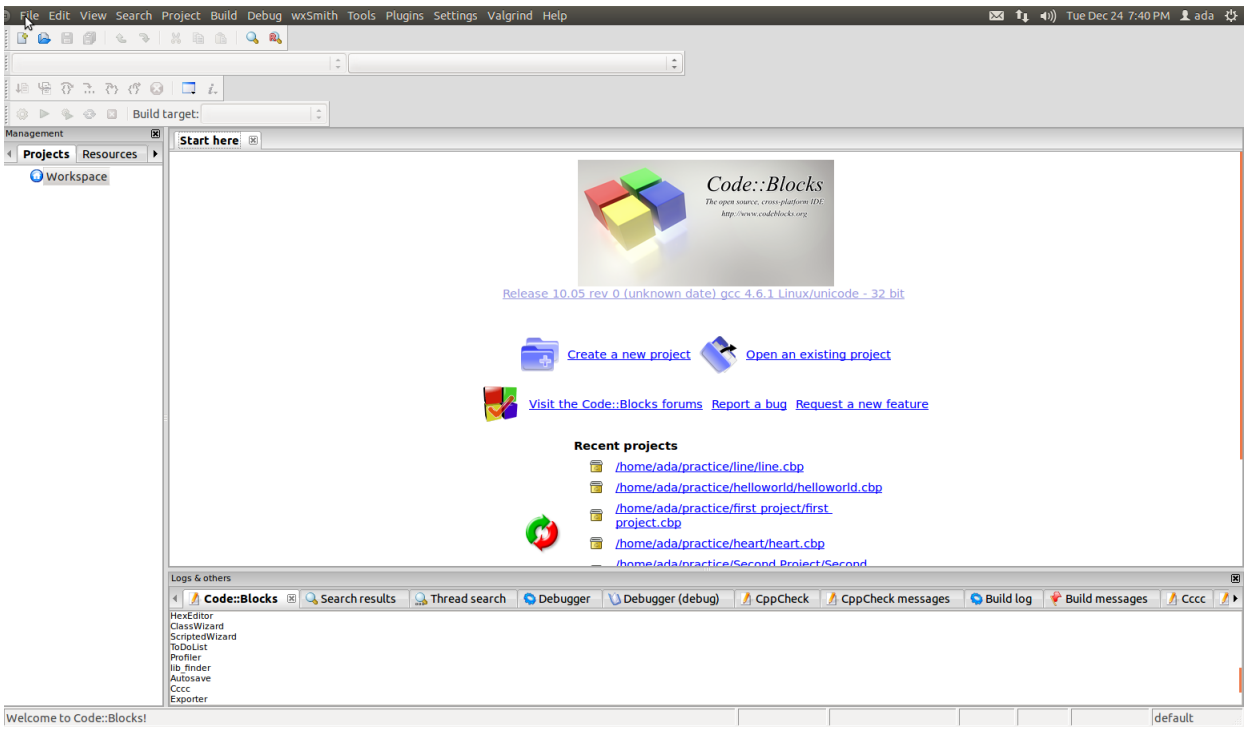

Figure 1: Code::Block IDE

1. **Menu bar:**
   Menu bar is shown in figure 2. Menu bar can be toggled using F10. Few important link in menu bar are described below (described from left to right):

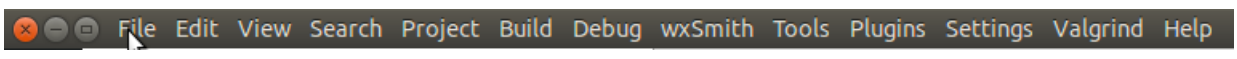

Figure 2: Menu Bar

   (a) File: File menu link contains options to create a new project, open an already existing project, save file, save project, save workspace and save everything. It also contains options for closing

[1] Ubuntu 12.04 with intel ®Core ™i3-2120 CPU @ 3.30GHzx4 processor 4 GB RAM and 32-bit architecture and 64 bit architecture.

[2] Windows 7 with intel ®Core ™i3-2120 CPU @ 3.30GHzx4 processor 4 GB RAM and 32-bit architecture.

a single file, closing a project or closing entire workspace. Other options in File are to print, export and quit the Code::Block

(b) Edit: All the editing options required for editor are provided in Edit.

(c) View: This menu link contains link for various perspectives and toolbars along with manager, logs, script console, status bar, full screen.

(d) Project: Options related to the project is provided in this link which includes configuring build options along with options for adding files, removing files and autoversioning of project.

(e) Build: Options for building the project, compiling a single file, running the project, building and running the project, rebuilding the project and cleaning the project is provided in build. Options for Building, rebuilding and cleaning the entire workspace is also provided along with options to select target (debug/release) and analysing error one by one.

(f) Debug: Various Debugging options are provided in this link.

(g) Plugins: Various plugins can be executed using this link. The link to manage the plugins is also provided here.

(h) Settings: This contains link for various settings, setting related to *Environment...*, *Editor...*, *Compiler and debugger...*, *Global Variables...* and *Scripting....* Script to be executed during Code::Block start-up can also be edited here.

(i) Help: It contains information about Code::Block version, tips which can be toggled to be displayed at start-up and information about various plugins.

2. **Main tool bar:**
Main tool bar is shown in figure 3. The buttons in Main toolbar are (from left to right):

Figure 3: Main Toolbar

(a) New File: For creating a new project.

(b) Open: For opening an already created project.

(c) Save : To save the file open in active editor (active editor means the editor tab in focus).

(d) Save all files: To save all the files for the current/selected project.

(e) Undo: To undo the executed action.

(f) Redo: To redo the undone action.

(g) Cut: To cut the selected/highlighted part in editor.

(h) Copy: To copy the selected/highlighted part in editor.

(i) Paste: To paste the cut/copy message in editor.

(j) Find: To find required text in the file in active editor.

(k) Replace: To replace required text in the file in active editor by some alternate text.

3. **Debugger tool bar:**
Debugger tool bar is shown in figure 4. Debugger tool bar is used to debug the current/selected project. The buttons in debugger toolbar are (from left to right) *Debug/Continue*, *Run to cursor*, *Next line*,*Next instruction*, *Step into*, *Step out*, *Stop debugger*, *Debugging Windows* and *Various info*. You will be able to understand the use of this buttons by rigorous practise of debugging various projects.

Figure 4: Debugger Toolbar

4. **Compiler tool bar:**
Compiler tool bar is shown in figure 5 and is used in building/compiling/running the current/selected project. The buttons in Compiler toolbar are (from left to right):

Figure 5: Compiler Toolbar

(a) Build: For building the current/selected project.

(b) Run: For running the current/selected project.

(c) Build and run: For building and running the current/selected project.

(d) Rebuild: For rebuilding the current/selected project.

(e) Abort: For aborting the build process for the current/selected project.

(f) Build target: For defining the type of build target for current/selected project, either debug or release.

5. **Manager:**
Manager is shown in figure 6. It is labelled as Management. This window provides the list of all the open projects and files for easy access to any required file of any project.

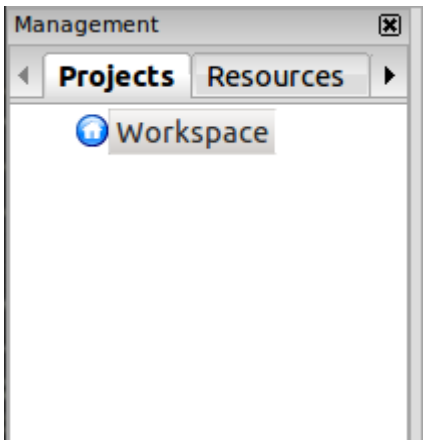

Figure 6: Manager

6. **Editor:**
Editor is shown in figure 7. Here, all the coding work will take place. It is provided in tabbed fashion to work with many files at once. When no project is open, the start page or home page is displayed in editor. The links given in start page is divided into two parts and explained below

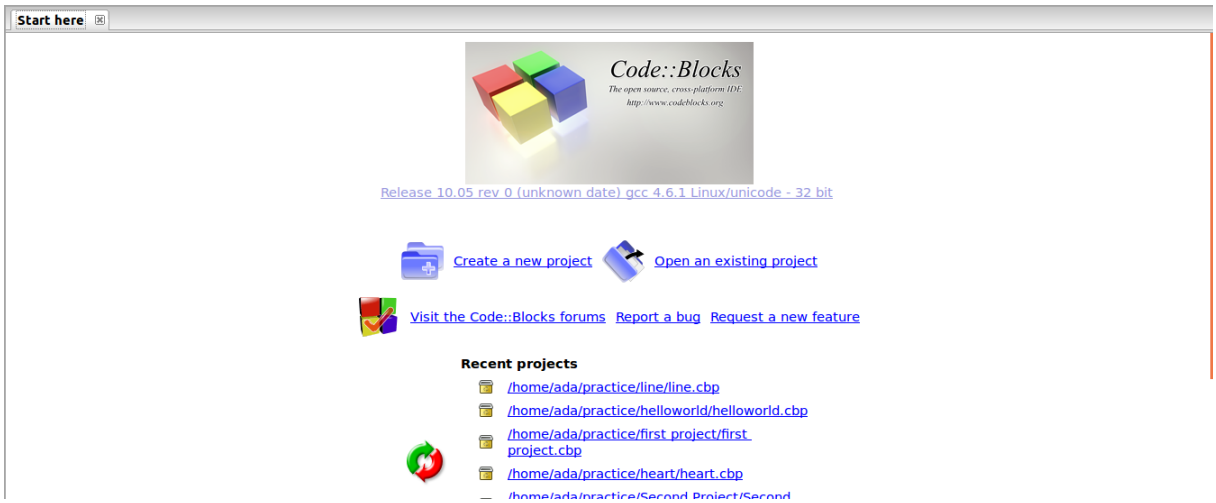

Figure 7: Editor, Start/Home Page

7. Figure 8 is short-cut on Starting page of IDE for creating a new project and opening an already created project. It also contains link for Code::Block forum where many useful resources can be found along with other useful discussions. The link points to url http://forums.codeblocks.org/. The second and third link points to BerLiOS Developer Site aims at enriching the Open Source community by providing a centralized place for Open Source Developers to control and manage Open Source Software Development.

Figure 8: Shortcut to Create New Project or Open Existing Project

8. Figure 9 is short-cut to list of projects and files already opened in the IDE. It is link to few projects and files from history of IDE.

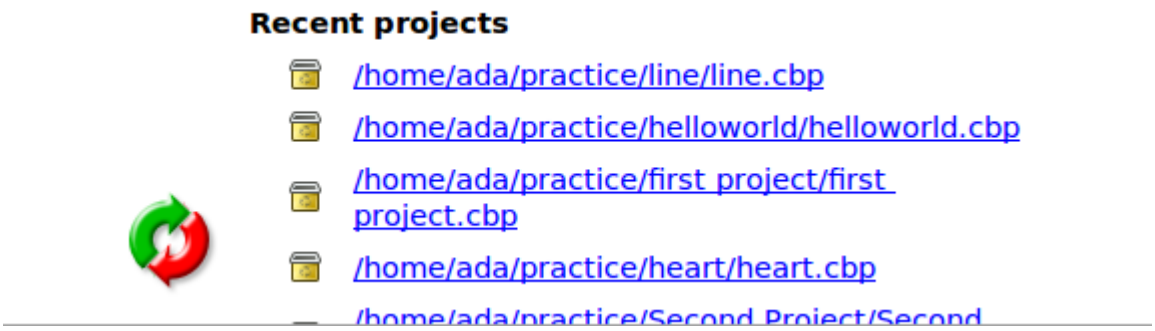

Figure 9: Shortcut to History of Projects Opened Using Code::Blocks

9. **Logs:**
   Log window is shown in figure 6. It is labelled as 'Logs & others'. This window acts as log for various actions performed in IDE. All logs related to various activities can be checked at appropriate windows.

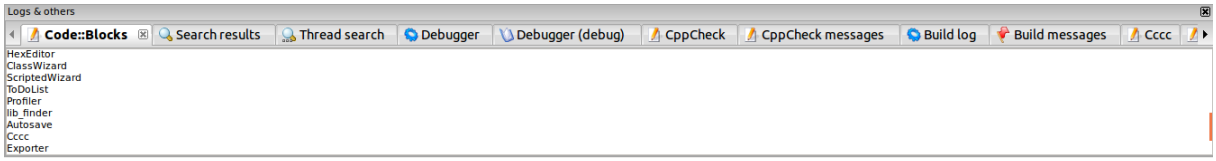

Figure 10: Logs

3 Working with Code::Block on Windows

In this section we discuss writing and building of two projects. First project (hello world.cpp) is a simple program which displays *hello world* on output. The second project (line.c) uses *graphics.h* header file and displays a line. **For Windows (Windows 7) we will be using 'CodeBlocks-EP'**. When CodeBlocks-EP is run for first time a 'file association window' is displayed as shown in figure 11. Select 3rd option "*Yes, associate Code::Blocks with c/c++ file types*" or 4th option "*Yes, associate Code::Blocks with every supported type(including project files from other IDEs)*" and click 'OK'.

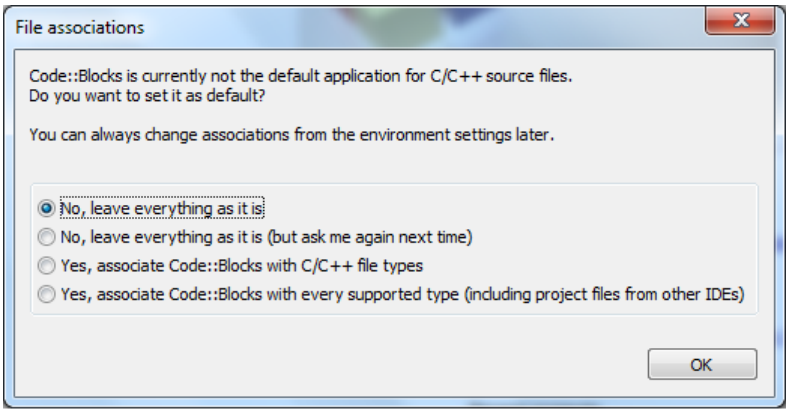

Figure 11: File association window

3.1 Installation of Code::Block

Code::Block used for windows is CodeBlocks-EP (stands for Code::Blocks - EDU Portable). WinBGIm (Borland Graphics Interface) required to run programs with graphics.h header is already integrated in CodeBlocks-EP. Download Codeblocks-EP from `http://codeblocks.codecutter.org/` The installation steps are as given below.

1. Download Code::Block-EP installer from the link given above and browse to the appropriate directory where the installer is downloaded. Click the .exe file downloaded and window as shown in figure 12 will pop up. Click 'Run'.

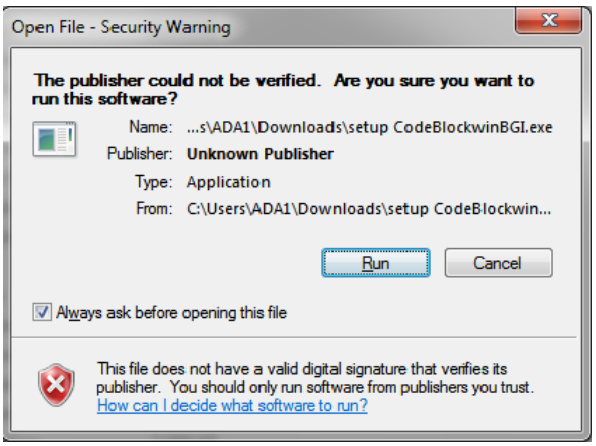

Figure 12: Click on "Run"

2. A new window appears as shown in figure 13. Click ‘Next’.

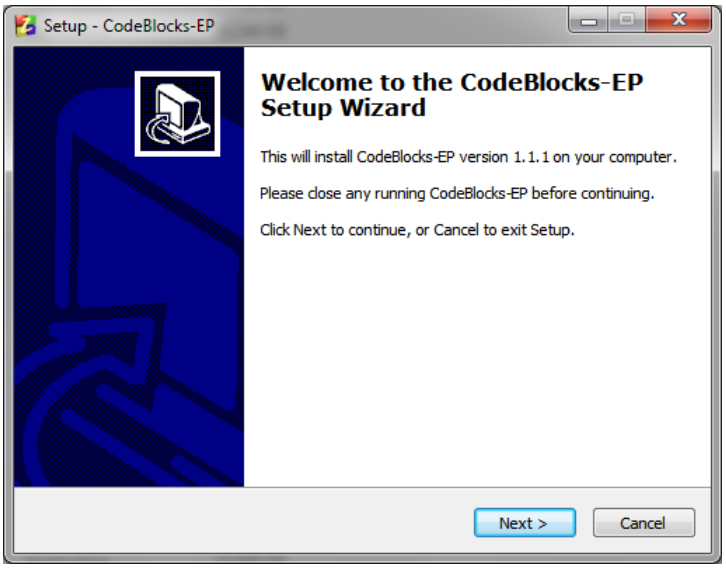

Figure 13: Welcome to the CodeBlocks-EP Setup Wizard screen

3. The third windows that appears is of license agreement as shown in figure 14. Select “I accept the agreement” and click ‘Next’.

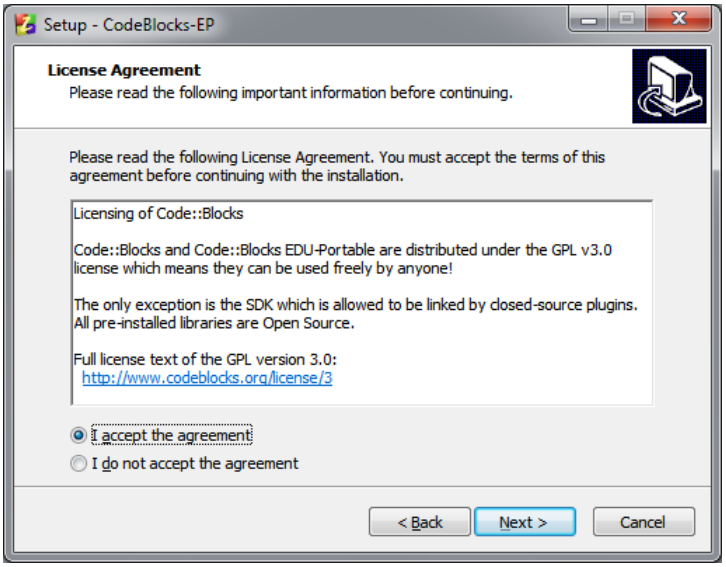

Figure 14: License Agreement

4. The next window displays some important information (shown in figure 15) regarding Code::Block-EP. Kindly go through information and click on "Next".

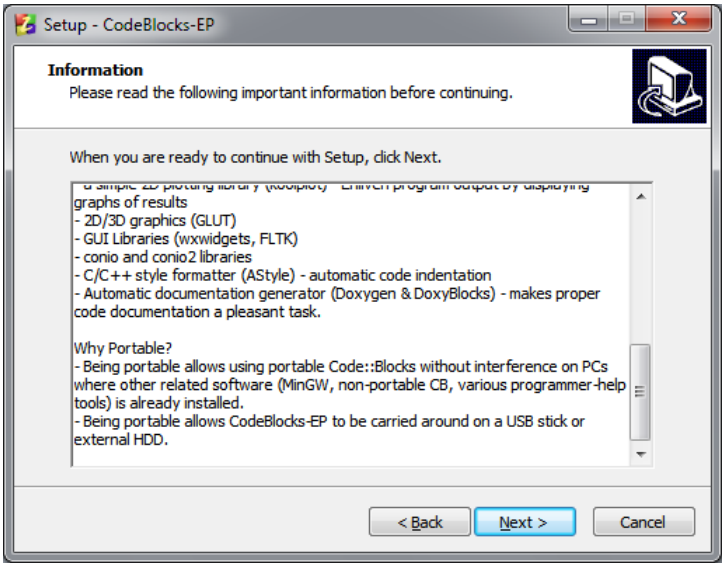

Figure 15: Information window

5. Next window asks for location where the CodeBlock-EP will be installed. The default location will be **C:\Program Files\CodeBlocks-EP**. Figure 16 shows the location provided for CodeBlocks-EP installation. Provide appropriate location and click on "Next".

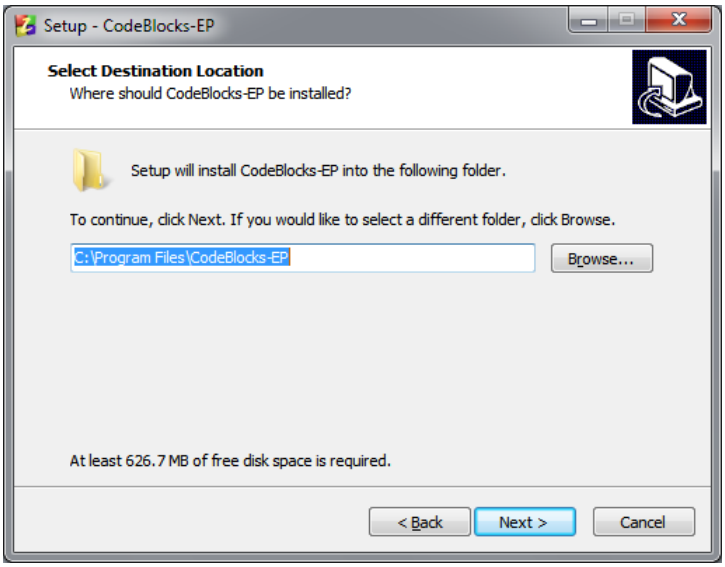

Figure 16: Providing location for CodeBlocks-EP installation

Figure 17 shows the location provided for CodeBlocks-EP shortcuts in Start Menu Bar. Click on "Next".

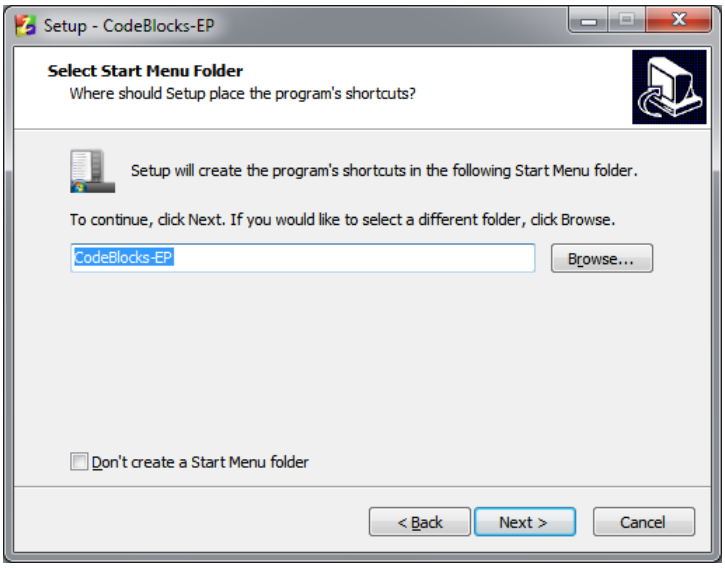

Figure 17: Providing folder for CodeBLocks-EP shortcuts

Select the checkbox 'Create a desktop icon' as shown in Figure 18. Click 'Next'.

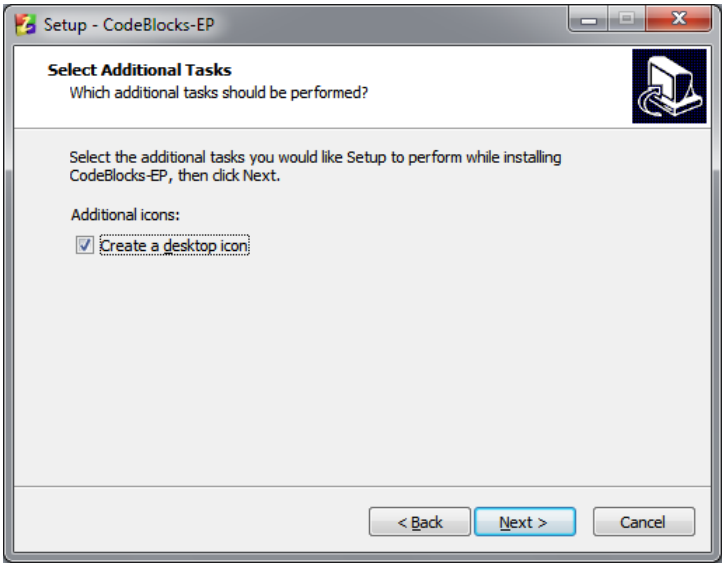

Figure 18: Creating desktop shortcut for CodeBlocks-EP

6. A new window displays stating that it is ready to install, which is shown in Figure 19. Click 'Install' to proceed with installation.

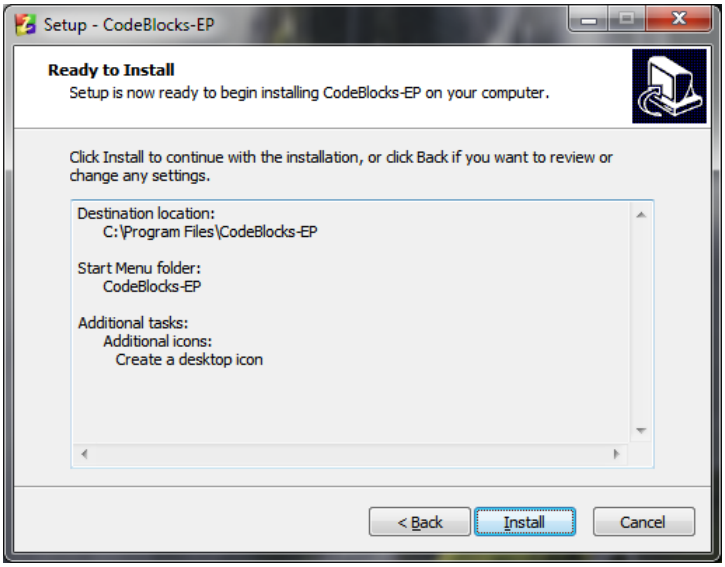

Figure 19: Installing CodeBlocks-EP

Figure 20 shows installation progress with progress bar.

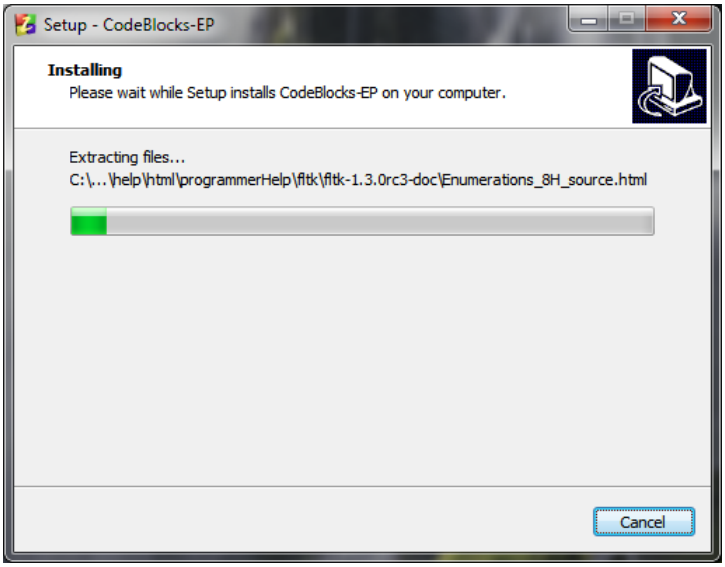

Figure 20: Progress bar to show progress of installation

7. When the installation in complete a window is displayed shown in figure 21. If you want to launch the CodeBlock select “Launch CodeBlocks-EP” and click ‘Finish’.

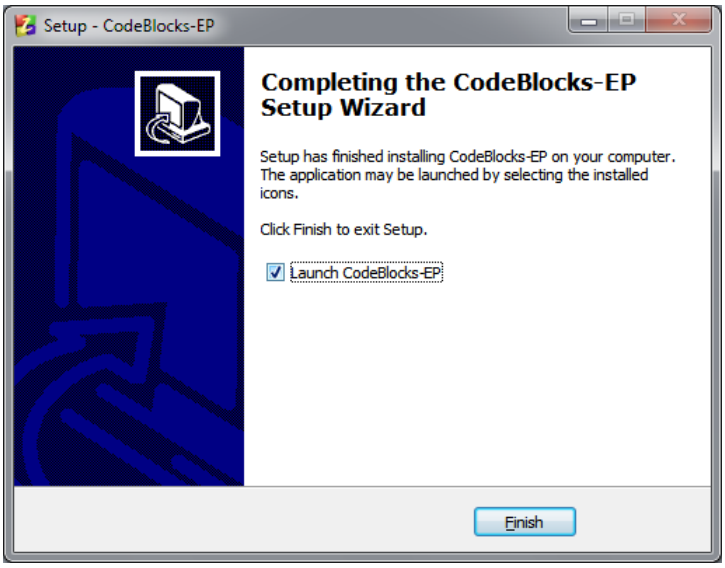

Figure 21: Completing the installation process

8. CodeBlocks IDE opens as shown in figure 22. Alternatively CodeBlocks-EP can be launched by double clicking the desktop icon created or clicking on it’s shortcut icon in Start Menu Bar.

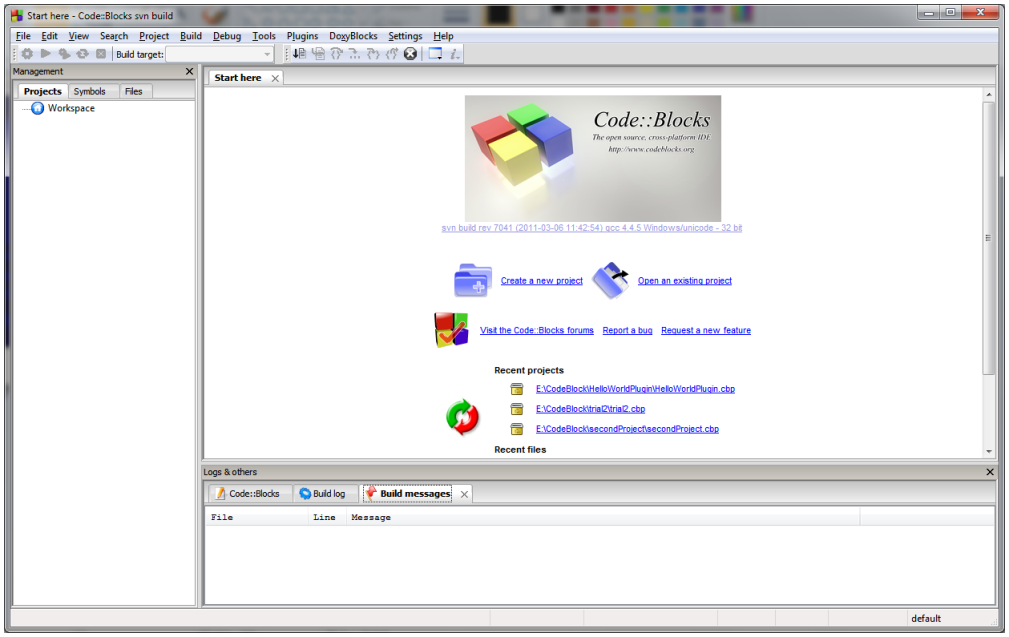

Figure 22: Code::Block IDE in Windows 7

3.2 Writing a new C/C++ Program

3.2.1 Non-Graphic Project

1. Click 'File' → 'New' → 'Project'. A new window opens as shown in figure 23. Click 'Console application' → Click *Go*.

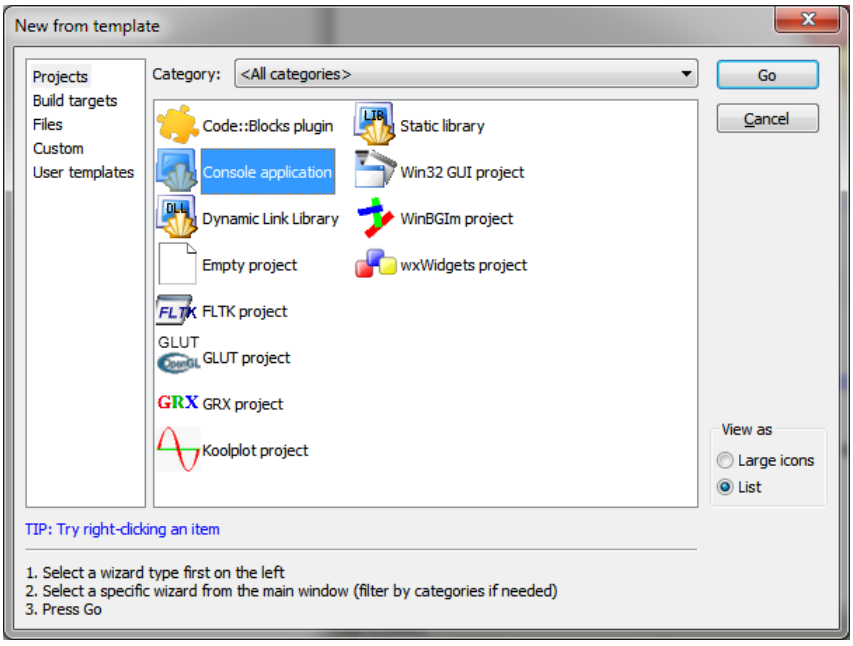

Figure 23: New form template

2. When *Go* button is clicked, a new window opens as shown in figure 24. Select checkbox 'Skip this page next time' so that the page is not displayed again. Click 'Next'.

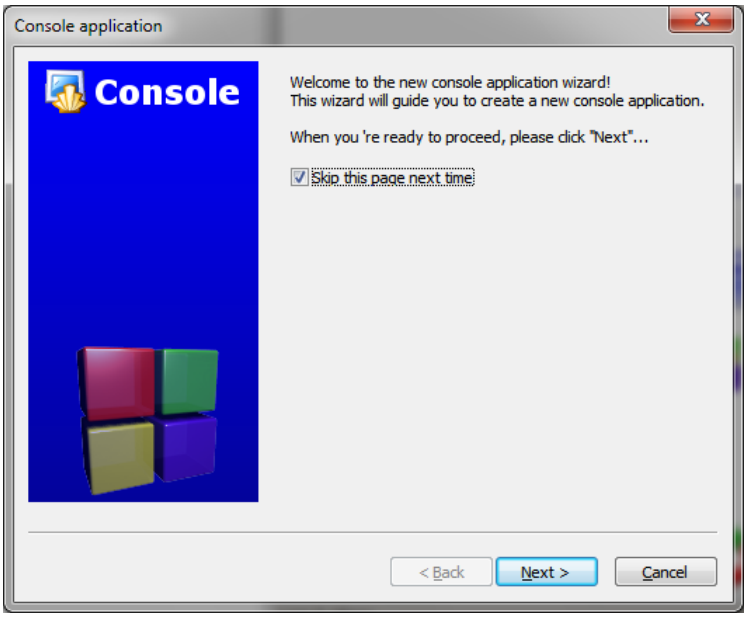

Figure 24: New console application wizard

3. Next window enables user to select the language to be used for project as shown in Figure 25. Click 'C++' → 'Next'.

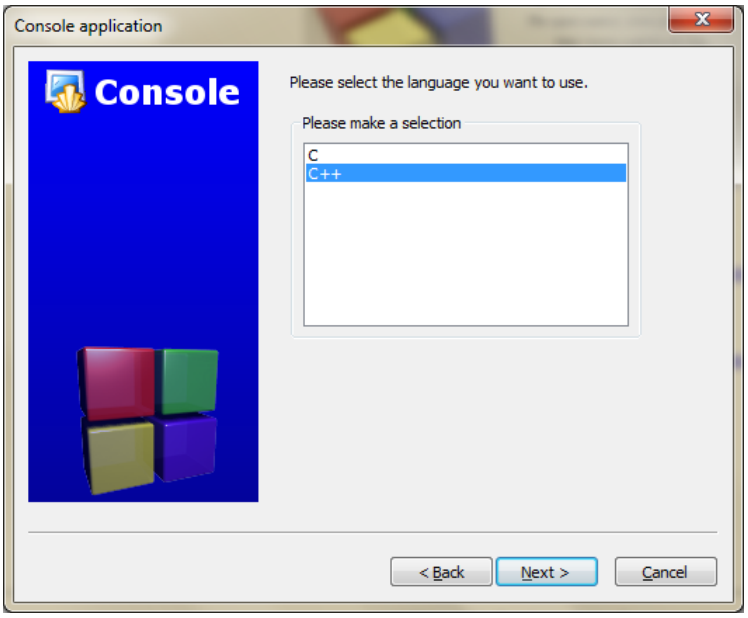

Figure 25: Selecting language for the project

4. Next windows enables user to provide title for the project and the folder where user wishes to create the project in. This is Shown in figure 26. After filling in the details click on *Next*.

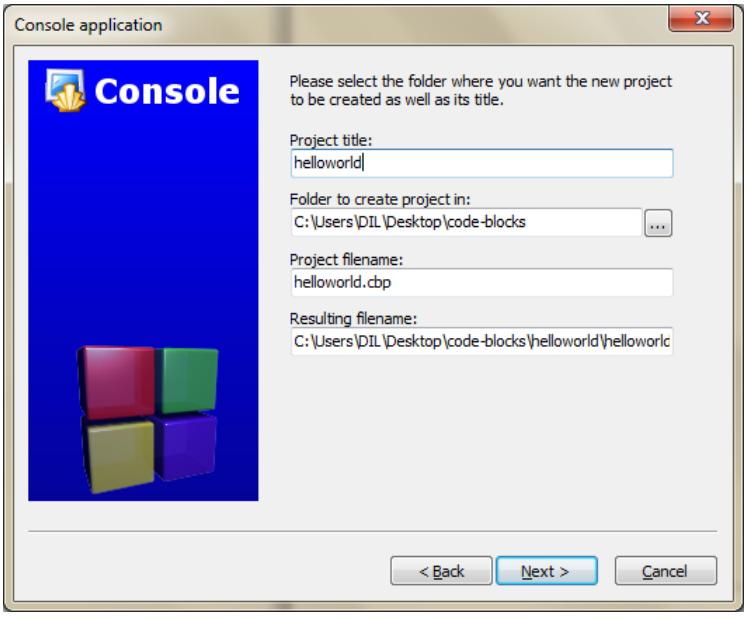

Figure 26: Providing title and folder for the project

5. Next window is used to select the compiler as shown in figure 27. By default 'GNU GCC Compiler' is selected. Click on *Finish*.

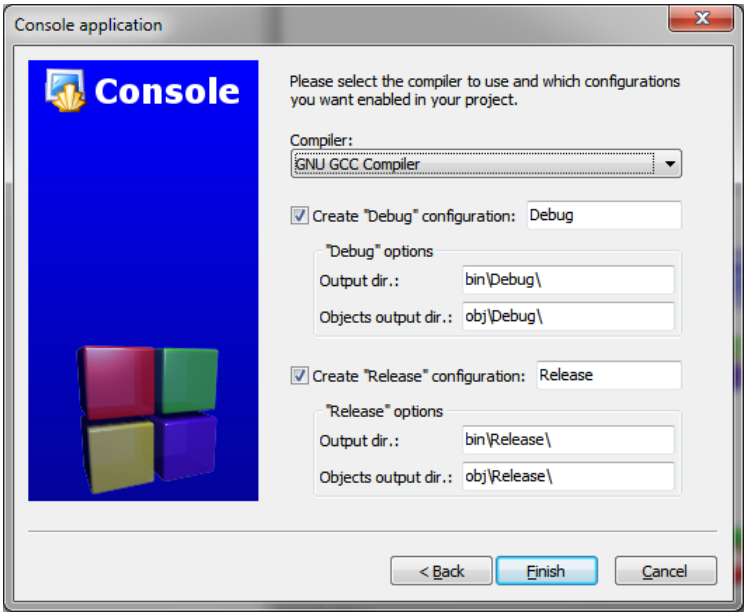

Figure 27: Selecting compiler for the project

6. The project node opens in manager window as shown in figure 28. The project node is empty and we have to add files to the project.

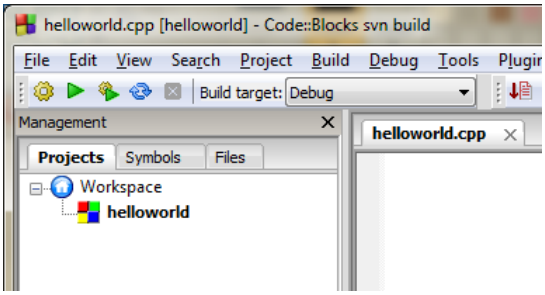

Figure 28: Project node with no files

7. To add files to the project select project node. Click ‘File’ → ‘New’ → ‘File’ as shown in figure 29.

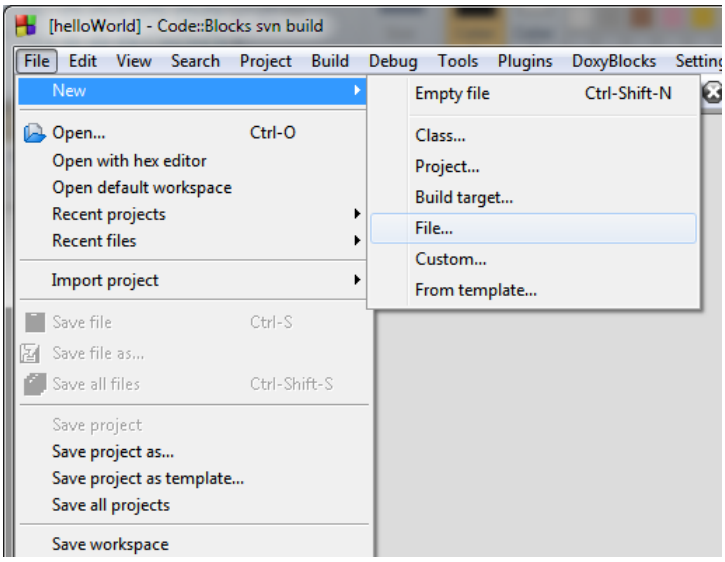

Figure 29: Adding file to the project

8. New from template opens as shown in figure 30. Click ‘C/C++ source’ → ‘Go’.

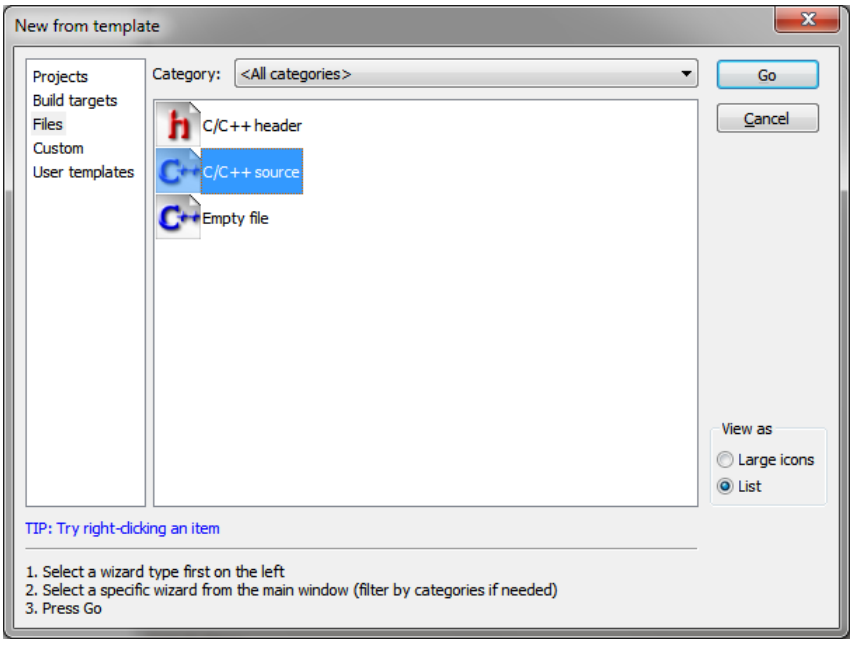

Figure 30: Selecting type of file to be added in the project

9. A new window is displayed as shown in figure 31. Select the checkbox 'Skip this page next time' so that it is not displayed again.

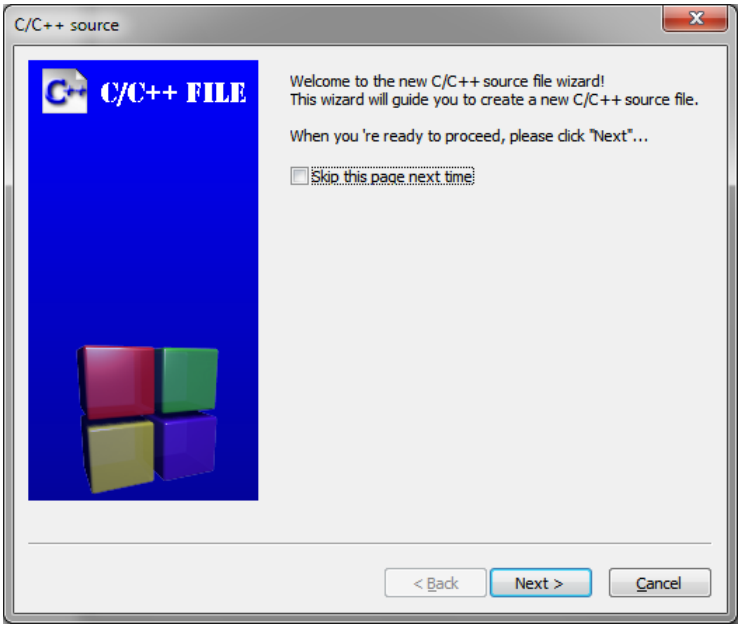

Figure 31: Select checkbox to skip this window next time

10. Click 'C++' as shown in figure 32.

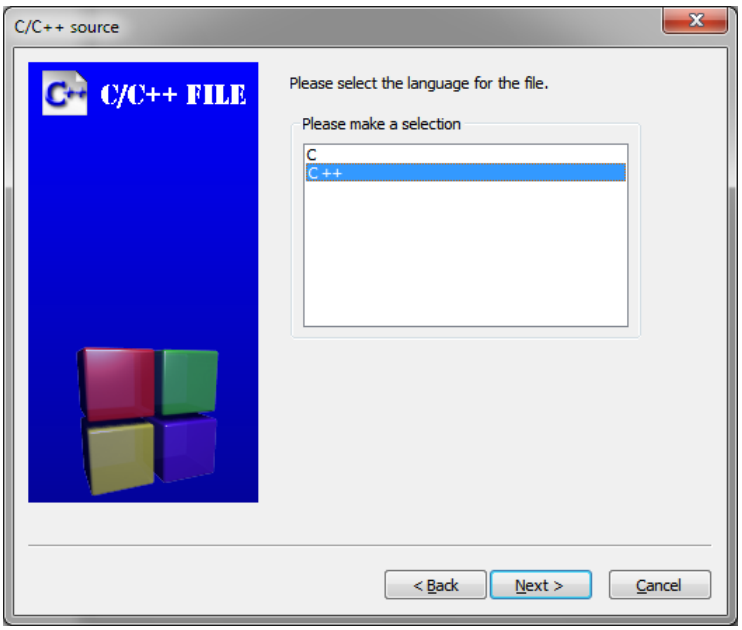

Figure 32: Select the language of the file added

11. Click on '...' besides the textbox of 'Filename with full path'. This is shown in Figure 33

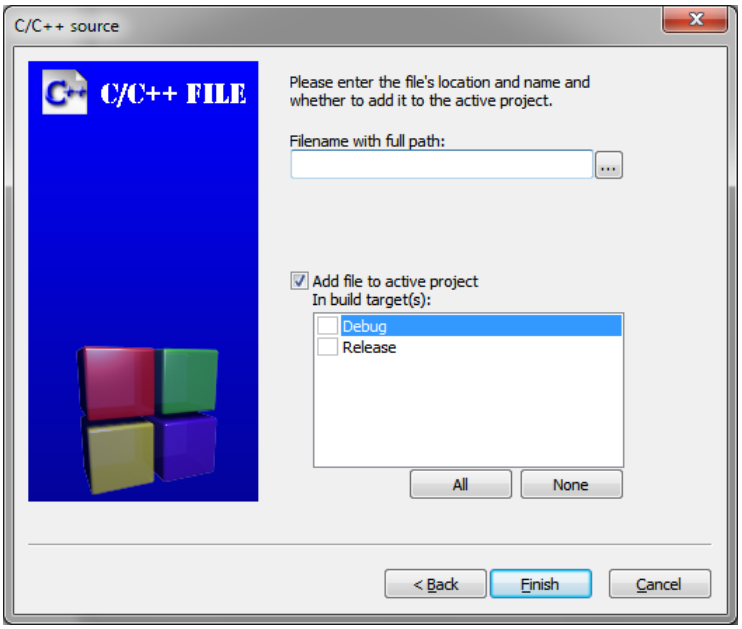

Figure 33: Details of file to be added

12. A window opens as shown in figure 34. Enter the file name (.cpp). and click 'Save'.

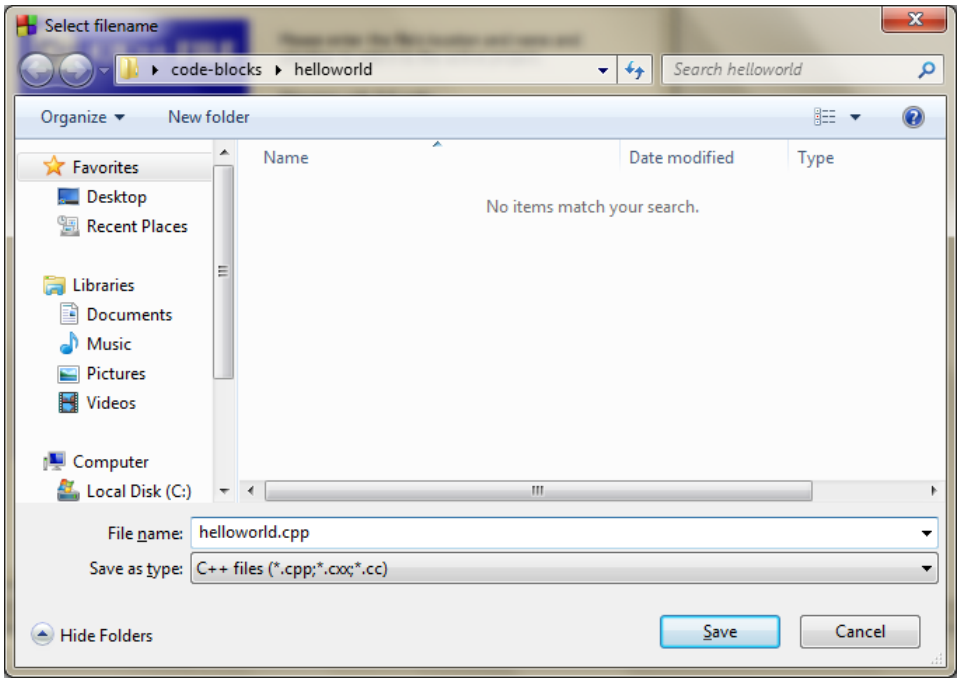

Figure 34: Selecting the location and file name to be added

13. Select the checkbox Debug and Release. Click 'Finish' as shown in figure 35.

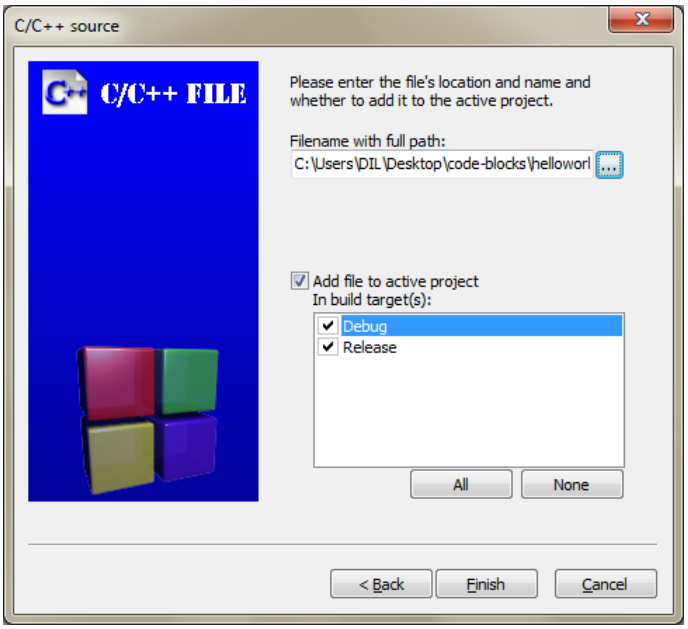

Figure 35: Finalize details of file to be added

14. Management window now shows project node which can be expanded. Double click 'Helloworld' and then double click 'Sources'. This is shown in figure 36 and figure 37 respectively.

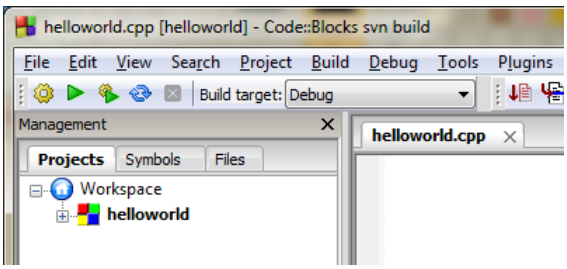

Figure 36: Project node with '+' sign indicating it can be expanded

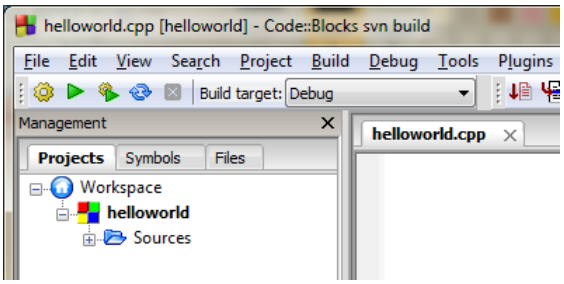

Figure 37: Project node with '+' sign indicating it can be expanded

15. Double click on ‘helloworld.cpp’ to open the file in editor. This is shown in figure 38

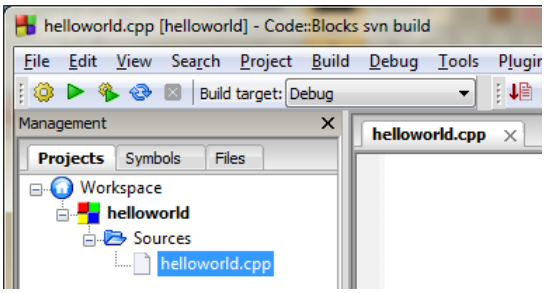

Figure 38: Project node expanded

16. Now, start writing your code in the editor.

3.2.2 Graphic Project using graphics.h

1. Click on *New file* button. The 'New form template' window as shown in figure 23 opens. For graphics projects, select 'WinBGIm project'. *Go* button gets highlighted (top right corner). Click on *Go*.

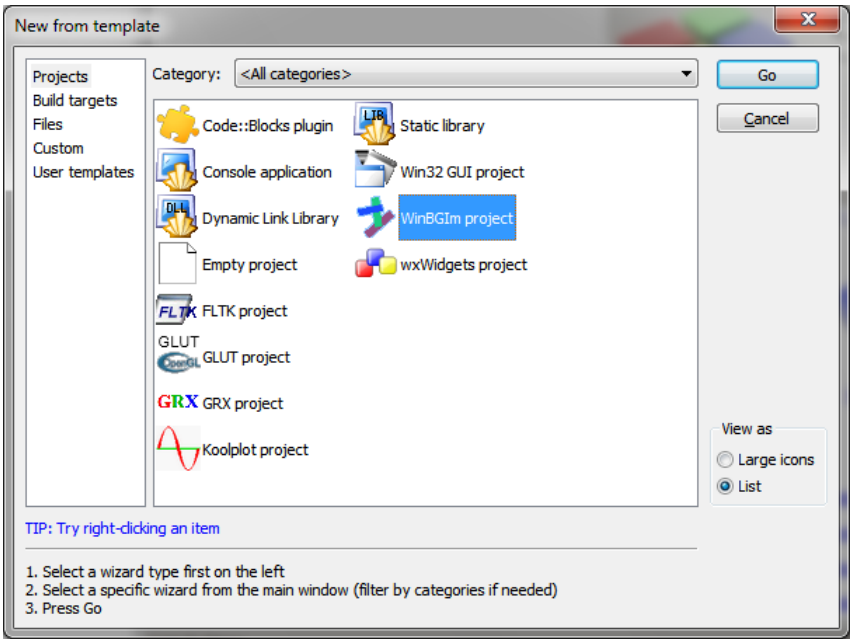

Figure 39: New form template

2. When *Go* button is clicked a new window opens as shown in Figure 40. Select checkbox "Skip this page next time" so every time new project is created this window should not come. Click on *Next*. If this step has been performed earlier, this window will not be displayed.

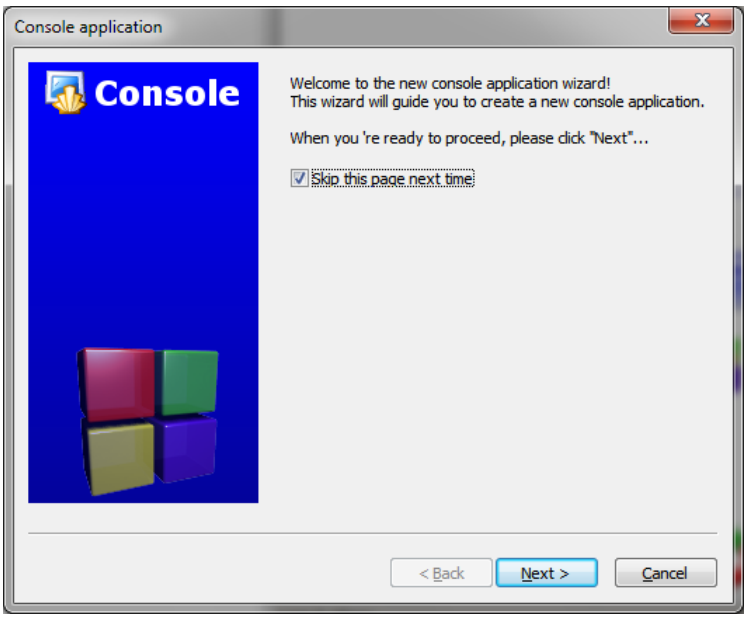

Figure 40: New console application wizard

3. Next window enables user to select the language to be used for project as shown in figure 41. For

the example hello world used in this manual select '*C*' and click on *Next*.

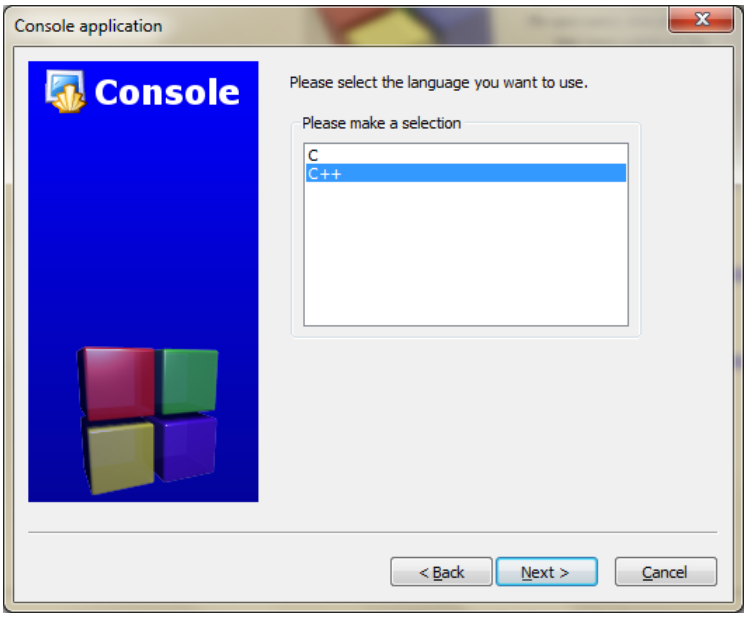

Figure 41: Selecting language for the project

4. Next window asks the user to select type of project. The options are 'Add Console' and 'Graphics only' as shown in figure 42. Select 'Graphics only' and click on *Next*.

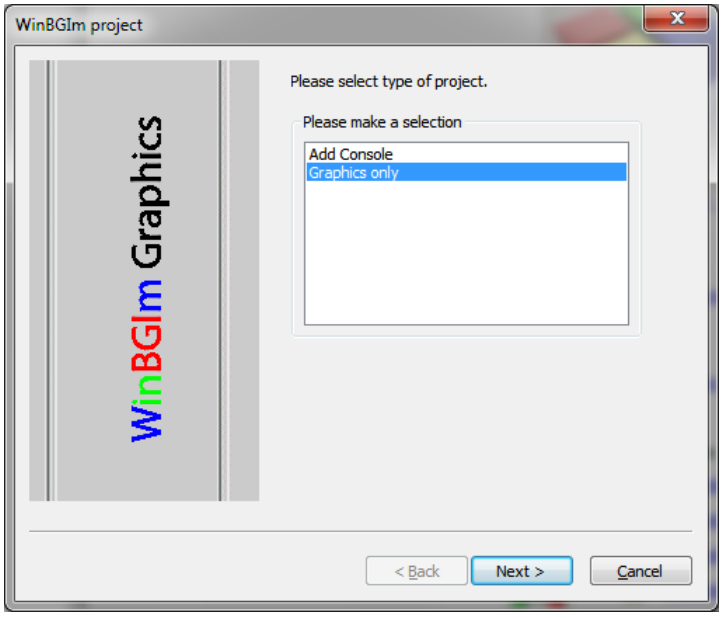

Figure 42: WinBGIm project

5. Next windows enables user to provide title for the project and the folder where user wishes to create the project. This is Shown in figure 43. After filling in the details click on *Next*.

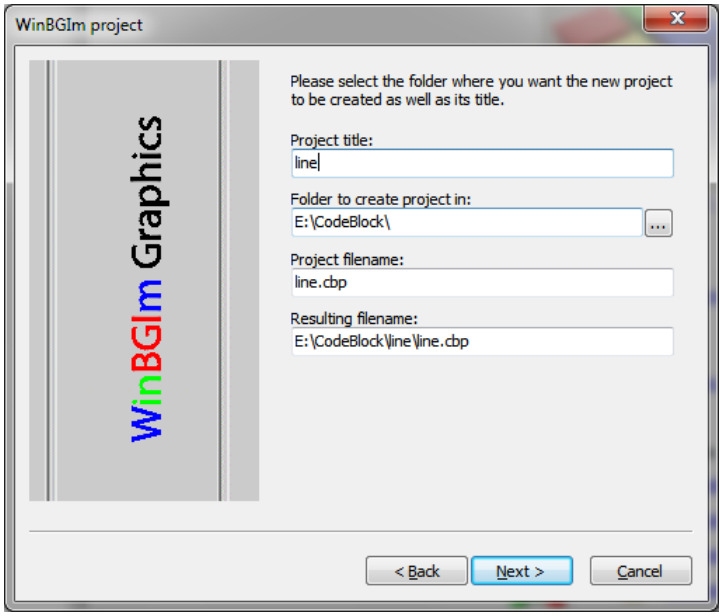

Figure 43: Providing title and folder for the project

6. Next window is used to select the compiler as shown in figure 44. By default 'GNU GCC Compiler' is selected. Click on *Finish*.

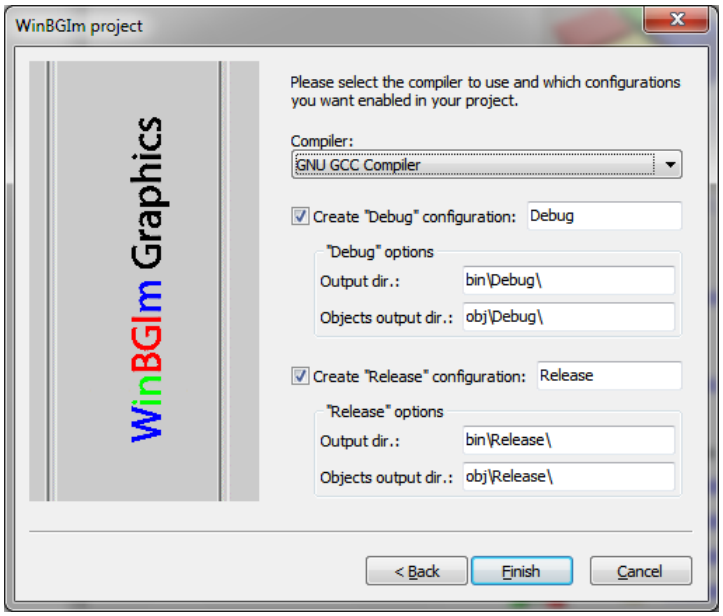

Figure 44: Selecting compiler for the project

7. The project node opens in manager window. The project node is empty and we have to add files to the project. To add files to the project select project node and click on File in menu bar, then click on 'File...' in options in 'New'. New from template opens as shown in figure 30. For our example select 'C/C++ source' and click on Go. A new window pops out which have a checkbox 'Skip this page next time'. Select the checkbox so this window should not open every time a new file is added to the project. Select the preferred language. For our example select 'C'. A new window opens which allows user to add the files to the project. Click on '...' beside 'Filename with full path'. This is shown in Figure 33. A window as shown in figure 34 opens. Select the folder of the project and enter file name to be added. Click on 'Save' (see steps 6-11 of section 5.2.1.1). Select Debug and Release. Shown in figure 45. Click on Finish.

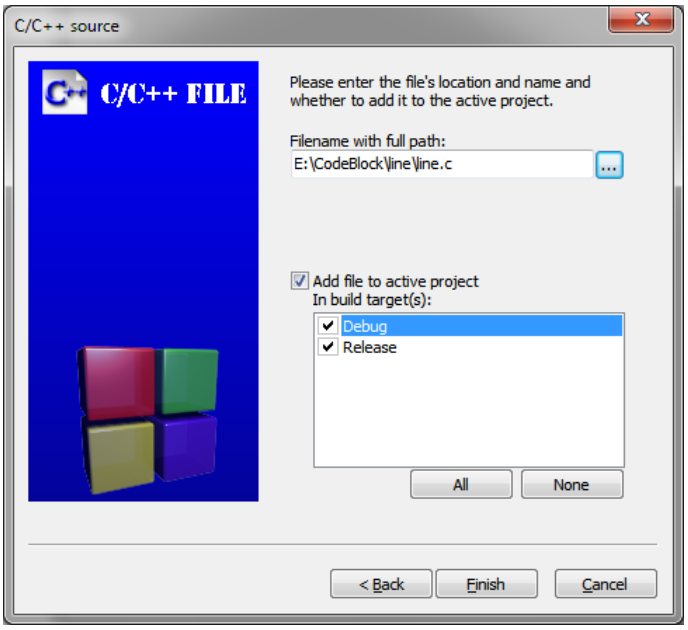

Figure 45: Finalize details of file to be added (for line project)

8. Management window now shows project node which can be expanded (figure 46).

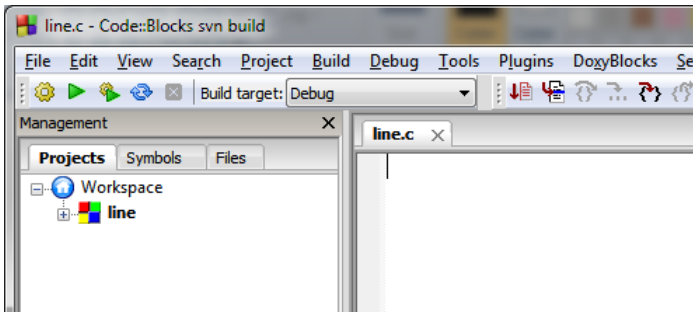

Figure 46: Project node with '+' sign indicating it can be expanded

9. Click on project node and double click on 'line.c' to open the file in editor. When the line.c file opens in editor, user can start coding. Code is shown in figure 47.

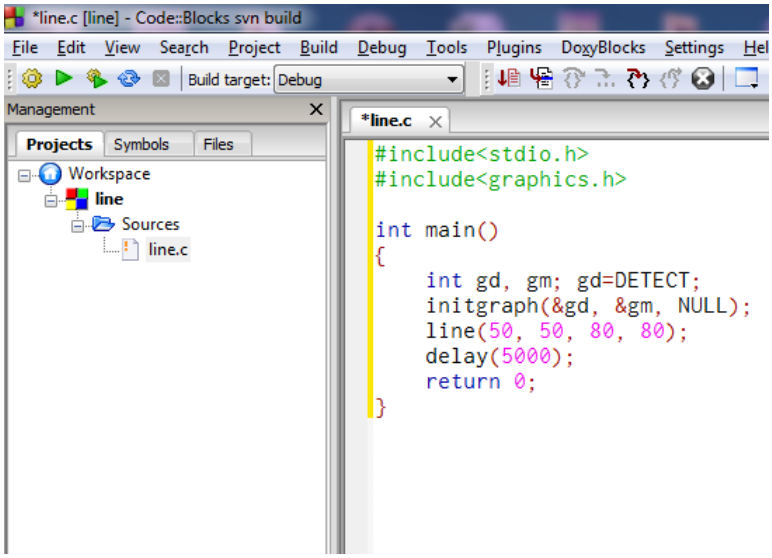

Figure 47: Writing program in editor (line project)

3.3 Building the Project

The process to build the graphics and non-graphics project is same, just Click on '*Build*' and then '*Run*' (or directly on '*Build and run*'). The output for the program used is shown in figure 48 for hello world project, figure 49 for line project.

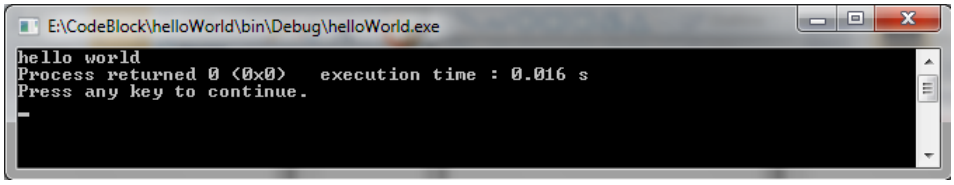

Figure 48: Output of hello world project

Figure 49: Output of line project

3.4 Opening existing program/project

Click on *Open* button 50. Browse to desired directory and open the file with *.cbp* extension as shown in figure 51.

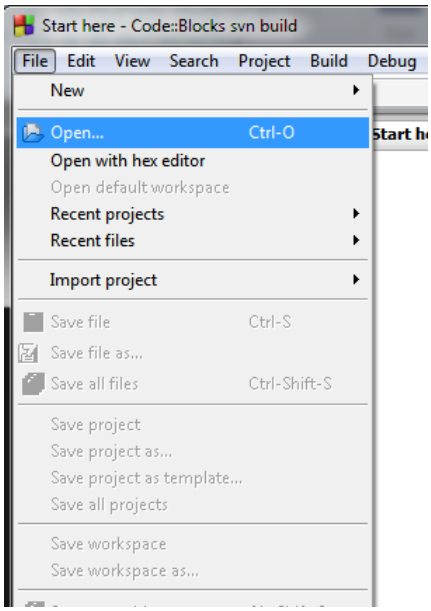

Figure 50: Selecting Open under file in Menu Bar

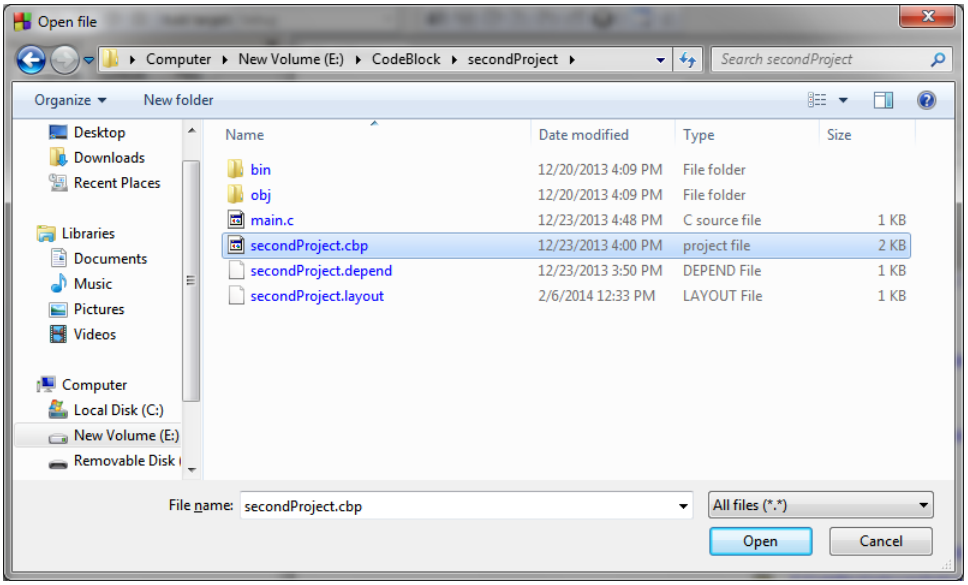

Figure 51: Select file with .cbp extension to open an existing project

4 Working with Code::Block on Ubuntu

In this section we discuss writing and building of two projects. First project (hello world.cpp) is a simple program which displays *hello world* on output. The second project (line.c) uses *graphics.h* header file and displays a line. *graphics.h* is not supported by *gcc*, which is the default C/C++ compiler on Ubuntu. We have to install some packages, include few libgraph libraries during building the project with graphics.h header file.

4.1 Installation of Code::Blocks

Pre-requisite for installing Code::Block is 'libwxgtk' which is available in ubuntu software center. This package will be already installed in your system[3]. It is also advised to install 'build-essential' package and update repository list. In case the libwxgtk is not installed, it can be installed from command line using command given in listing 1 [3]. Code::Block is available in Ubuntu's repository. It can be installed using Ubuntu Software Center or it can also be installed using command line as given in listing 2.

```
1 $ sudo apt-get install libwxgtk2.8-0
2 $ sudo apt-get install build-essential
3 $ sudo apt-get update
```

Listing 1: Installing libwxgtk2.8-0 using command line

```
1 $ sudo apt-get install codeblocks
```

Listing 2: Installing Code::Blocks using command line

Figure 52: Code::Block in Ubuntu Software Center

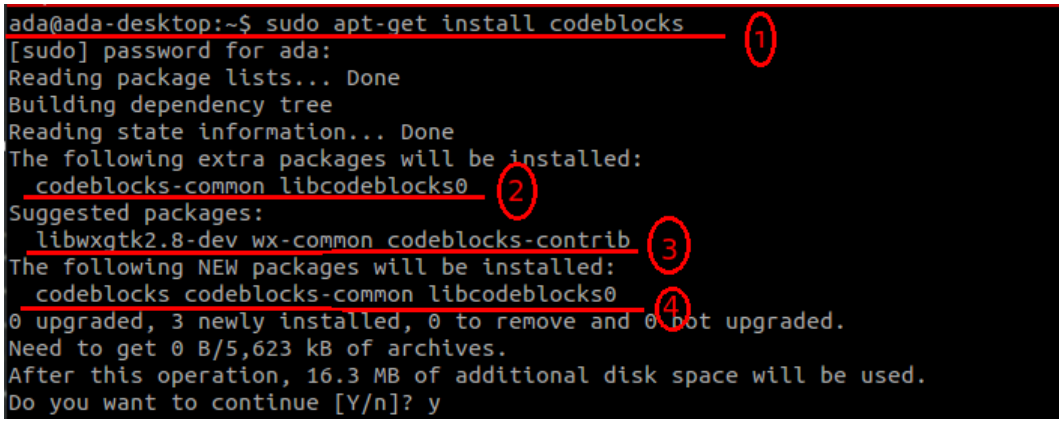

Figure 53: Installing Code::Block using command line

[3]Check for the libwxgtk version available for your Ubuntu, minimum required version for Code::Block to run is 2.0, version available on Ubuntu 12.04 is 2.8

Figure 52 shows Code::Block along with it's logo in Ubuntu Software Center, while figure 53 shows installation of Code::Block using command line. In figure 53 four underlined lines are:

line 1: Command to install the Code::Block.

line 2: Packages installed along with Code::Block.

line 3: Packages suggested along with Code::Block installation. *libwxgtk2.8*[4] package as mentioned above is already installed in your system. *libwxgtk2.8-dev* package is not required.

line 4: Final list of packages that will be installed in your system

When command line prompts for [Y/n] enter 'y' and press enter. When Code::Block is run for first time, It asks for default compiler. Select the appropriate compiler to proceed further. We have used 'GNU GCC Compiler'.

4.2 Installation of packages for graphics.h header file

GCC compiler does not support *graphics.h*, *conio.h*, *windows.h* and few other header files that works on Turbo C or Borland C. *graphics.h* header files enables programmer to write simple c/c++ graphics programs. In Ubuntu, gcc is default c/c++ compiler, thus, we have to make some settings, for gcc to support graphics[4]. We will start with installing some packages from command line as given in listing 3

```
1 $ sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev guile-1.8 guile-1.8-dev
    libsdl1.2debian libart-2.0-dev libaudiofile-dev libesd0-dev libdirectfb-dev
    libdirectfb-extra libfreetype6-dev libxext-dev x11proto-xext-dev libfreetype6
    libaa1 libaa1-dev libslang2-dev libasound2 libasound2-dev
```

Listing 3: Installing required packages to support graphics.h

After the above mentioned packages are installed, download the **libgraph** package (download link given in footnote[5]) and untar it in home directory. For this untarring tool must be installed on system. Open the command line and follow the instructions given in listing 4.

```
1 $ cd libgraph-1.0.2
2 $ ./configure
3 $ sudo make
4 $ sudo make install
5 $ sudo cp /usr/local/lib/libgraph.* /usr/lib
```

Listing 4: Installing libgraph package using command line

[4]Details about above mentioned packages can be found at http://packages.ubuntu.com/precise/allpackages.
[5]http://download.savannah.gnu.org/releases/libgraph/libgraph-1.0.2.tar.gz

4.3 Writing a new c/c++ program

1. Click on *New file* button. The 'New form template' window opens as shown in figure 54. Click 'Console application' → 'Go'.

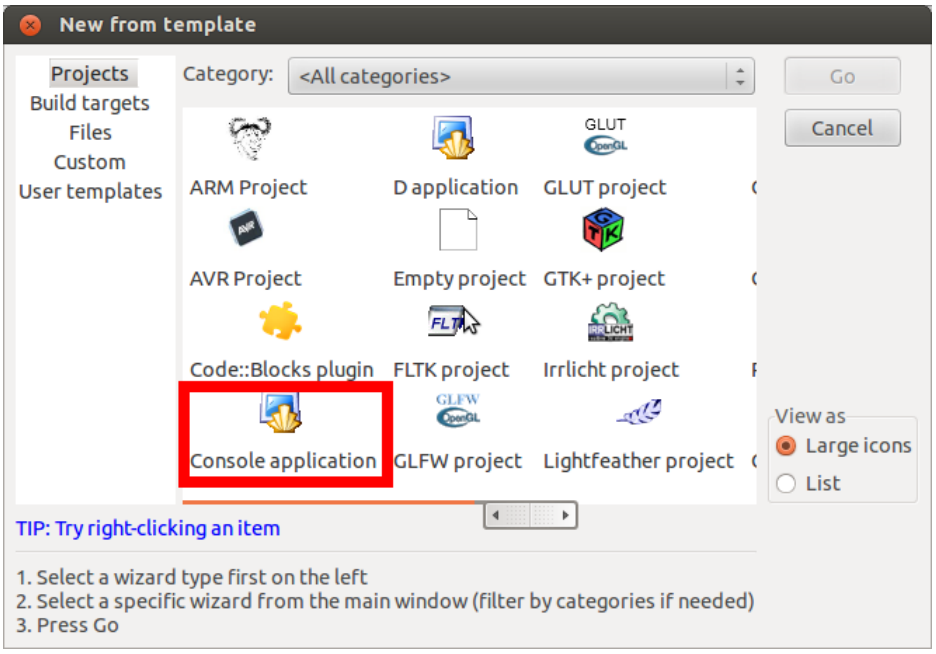

Figure 54: Starting a new project

2. A new window opens as shown in figure 55. Click 'C++' → *Next*.

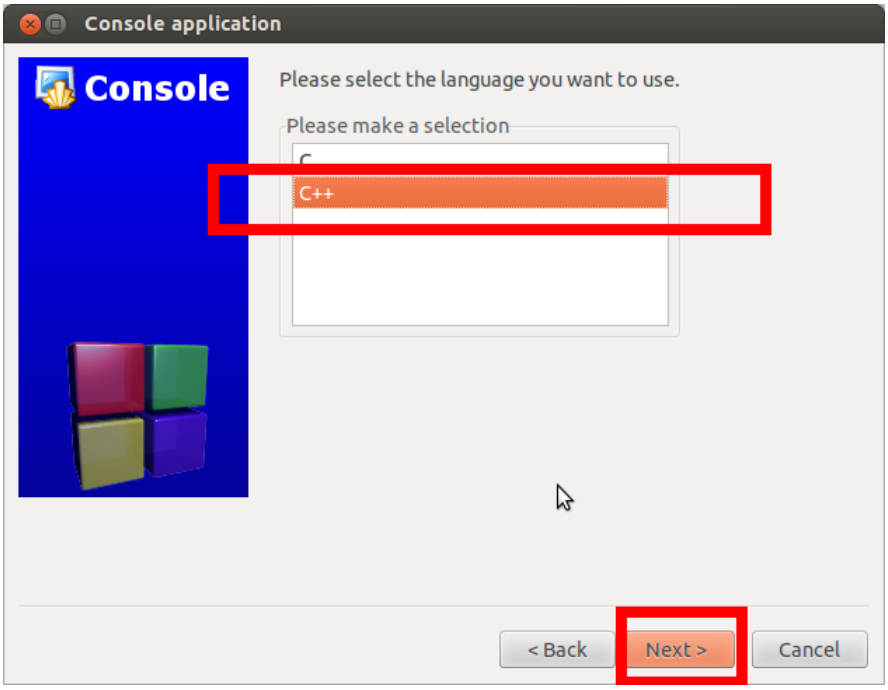

Figure 55: Selecting the language for project

3. The subsequent windows enable the user to provide title for the project and the folder where user wishes to create the project in. This is shown in figure 56. After filling in the details click 'Next'.

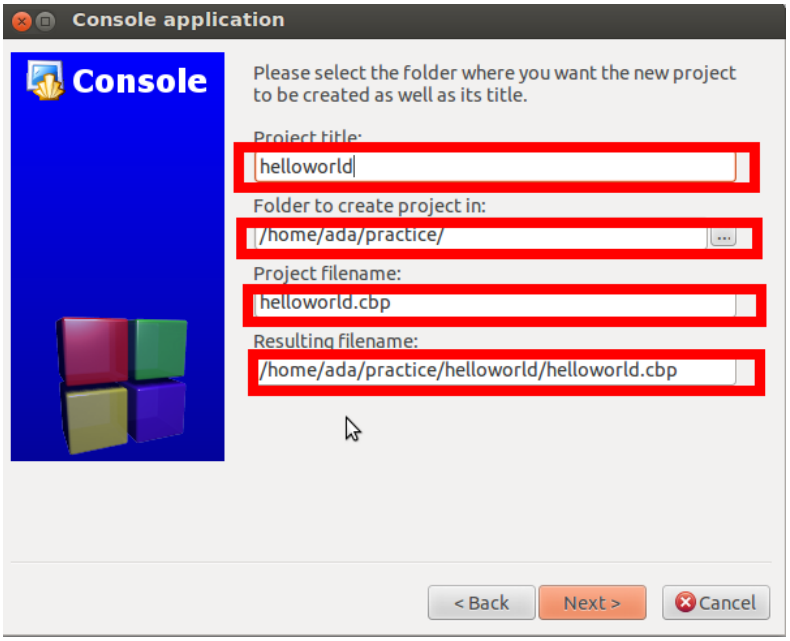

Figure 56: Title for Project

4. Next window is to select the compiler. By default 'GNU GCC Compiler' is selected. Click *'Finish'*.

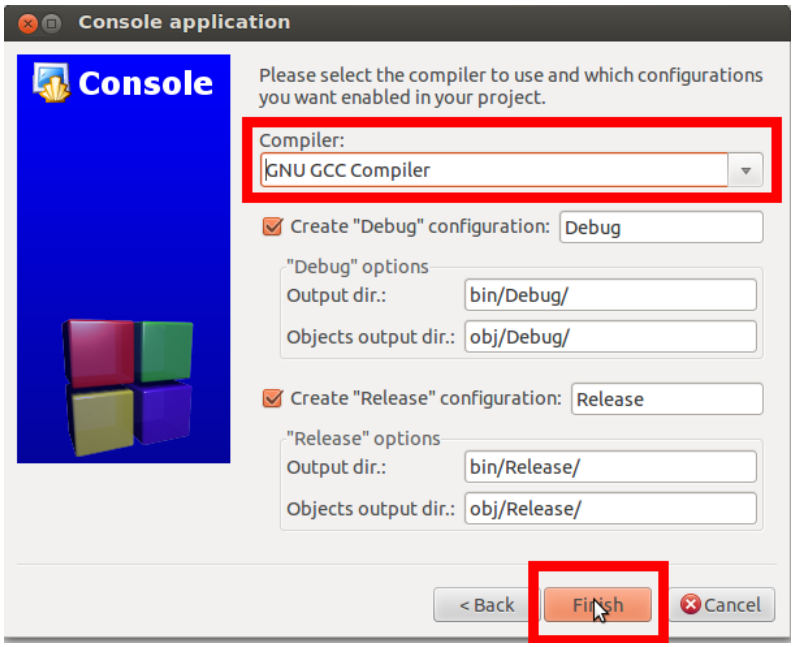

Figure 57: Selecting Compiler to Compile the Program

5. Now, the project node opens in manager window as shown in figure 58. Project node can be expanded to see the main.c file.

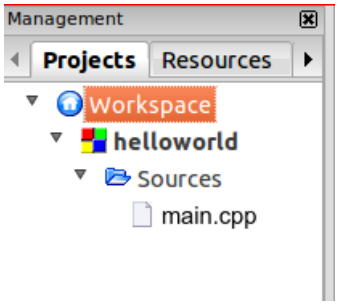

Figure 58: Project Node when Expanded

When main.c file is clicked, it opens in editor as shown in figure 59 for hello world project.

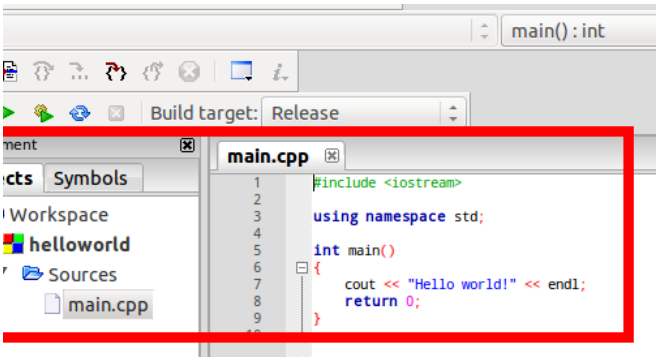

Figure 59: Project Node when Expanded for helloworld.c (with code shown in editor)

6. Code used for *graphics* program (line) is shown in figure 60.

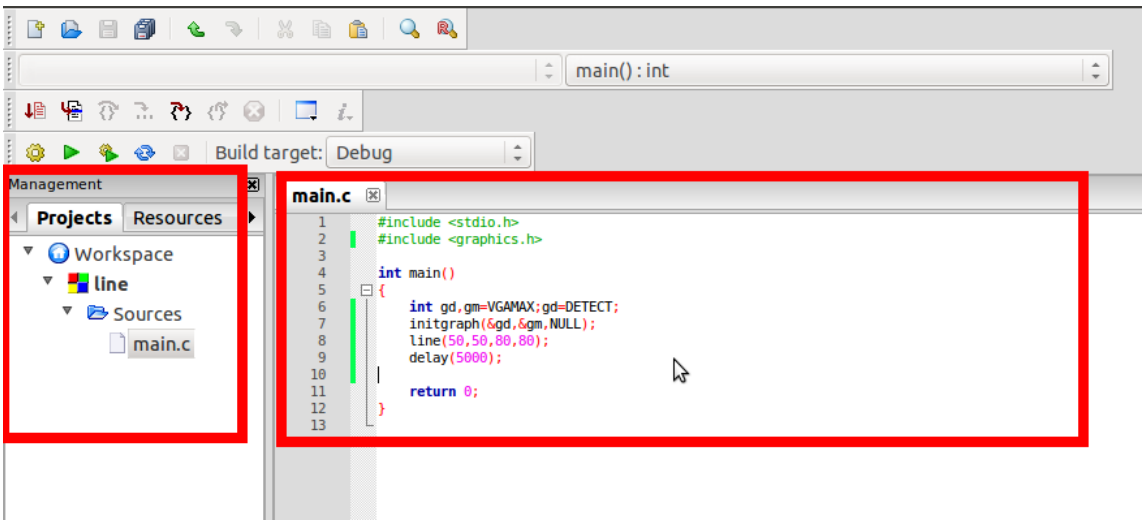

Figure 60: Project node when expanded for line.c (with code shown in editor)

7. While using Code::Blocks for the first time, some extra windows will be displayed. In this manual only the important windows are shown.

4.4 Building the Project

4.4.1 Non-Graphics Project

After the code is written, project needs to be built. Click *'Build'* → *'Build and Run'*. Output is as shown in figure 61.

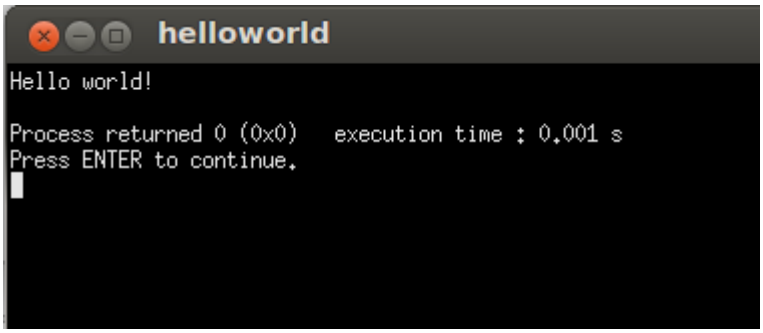

Figure 61: Output for helloworld.c

4.4.2 Graphics Project using graphics.h

The steps to link the libraries and build the project is given below.

1. Right click on project node in Manager box and select *build options...* A new window 'Project build options' as shown in figure 62 pops up. Option to change the compiler selected for the project is also available in this window.

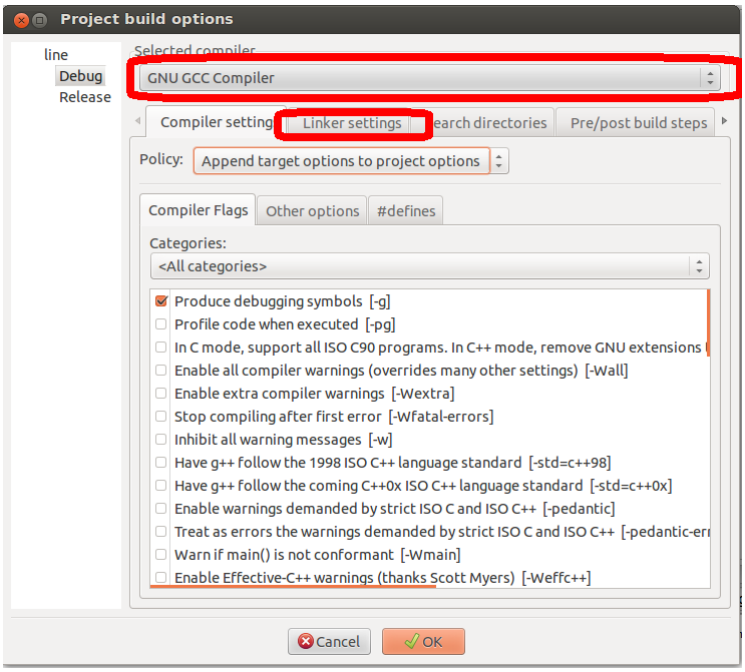

Figure 62: Project build options

2. In 'Project build options' window, click on *linker settings* tab. This tab is shown in figure 63.

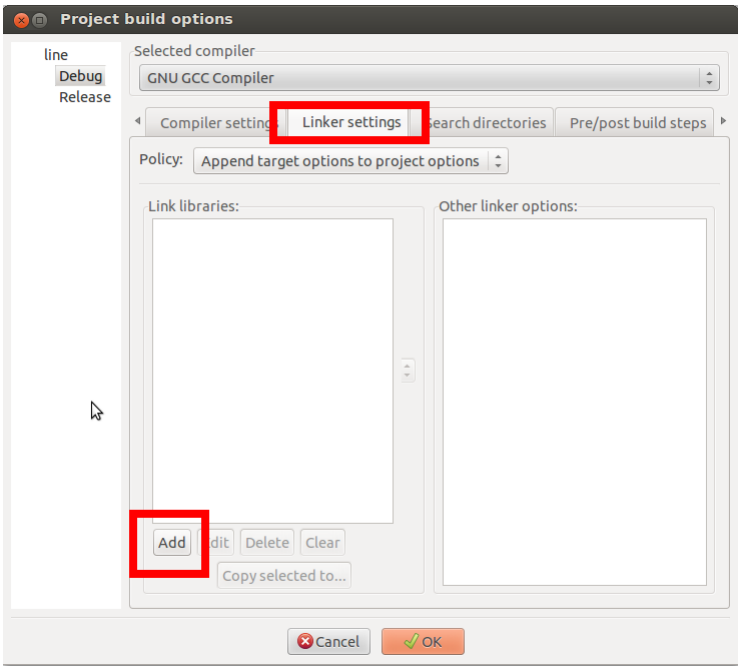

Figure 63: Linker settings (Add Libraries)

3. In linker settings tab click on *Add* button under *Link libraries* box. When Add button is clicked a small window titled *Add library* opens as shown in figure 64.

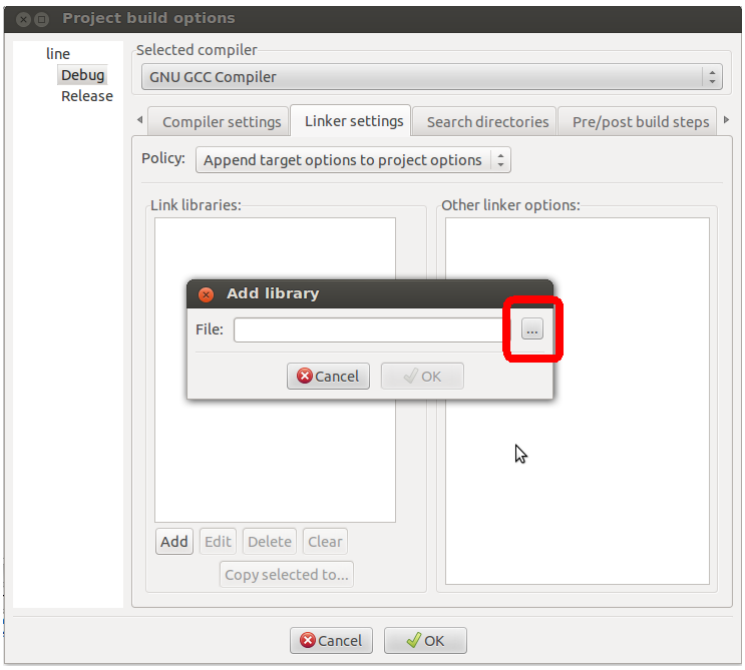

Figure 64: Interface for adding libraries

4. Click on dotted button to right of box. A new window opens as shown in figure 65.
   This window enables user to browse to appropriate folder and to select required library. Browse to the */usr/lib* directory. All the required library files are not visible.
   Select *all files* in dropdown placed at the bottom, to enable visibility of all types of files.
   Select all the libgraph .* files, except libgraph.la file. Files to be selected are shown in the figure 65.

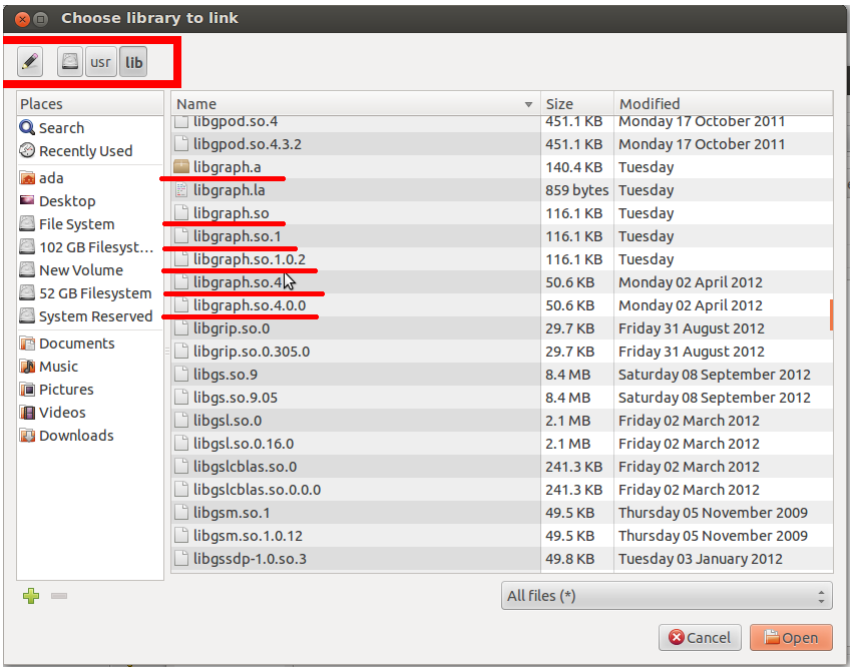

Figure 65: Files/libraries to be added for graphic projects

5. Libraries shown in Ubuntu 12.04, 32-bit OS are *libgraph.a*, *libgraph.so*, *libgraph.so.1*, *libgraph.so.1.0.2*, *libgraph.so.4* and *libgraph.so.4.0.0*. In Ubuntu 12.04, 64-bit OS libraries *libgraph.so.4* and *libgraph.so.4.0.0* are not available. Select all the libgraph.* files except libgraph.la

6. After selecting all the required libraries click on Open. A new window labelled 'Question' will open asking 'Keep this as relative path' as shown in figure 66. Click on No.

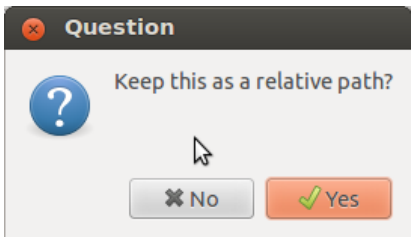

Figure 66: Relative or absolute path for files/libraries

7. The libraries will be linked using absolute path as shown in figure 67. Click on 'Ok'.

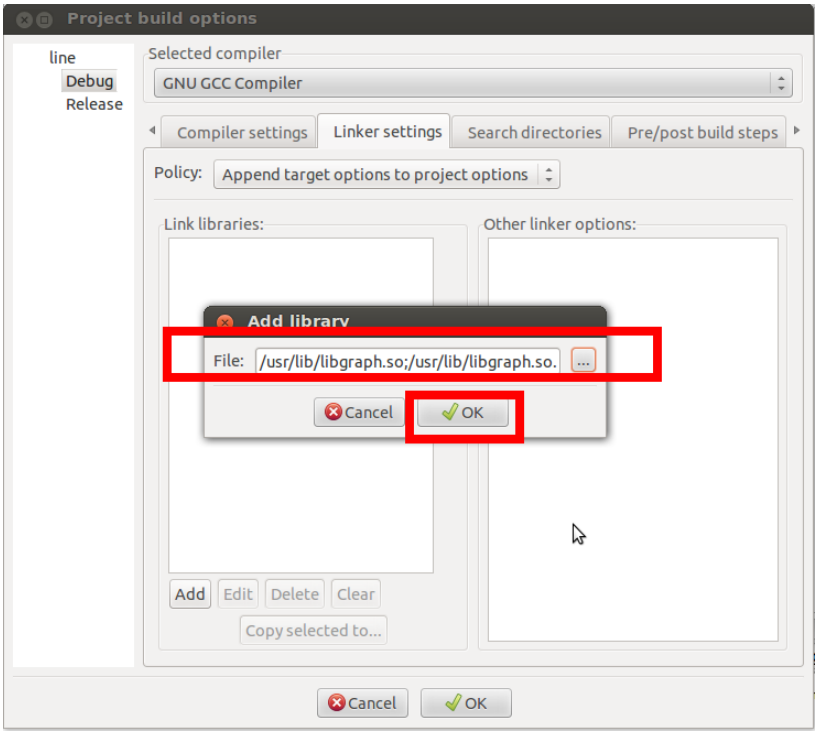

Figure 67: libraries selected

8. We are now re-directed to *linker settings* tab. The added libraries are shown. In *Other linker options* window write '-lgraph' . The final setup is shown in figure 68 with all the required 6 libraries and '-lgraph'. Click on 'Ok' to get back to editor. Now we are ready to build the project with *graphics.h* header file

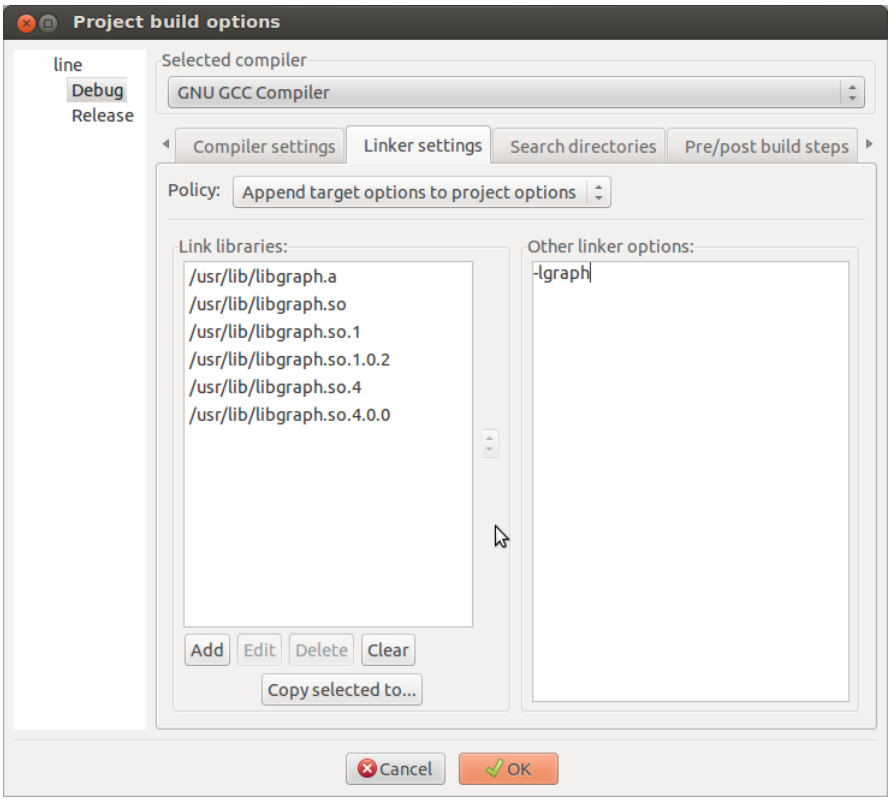

Figure 68: Libraries added to project

9. Now click the *build and run* button from compiler bar and output will be displayed as shown in figure 69.

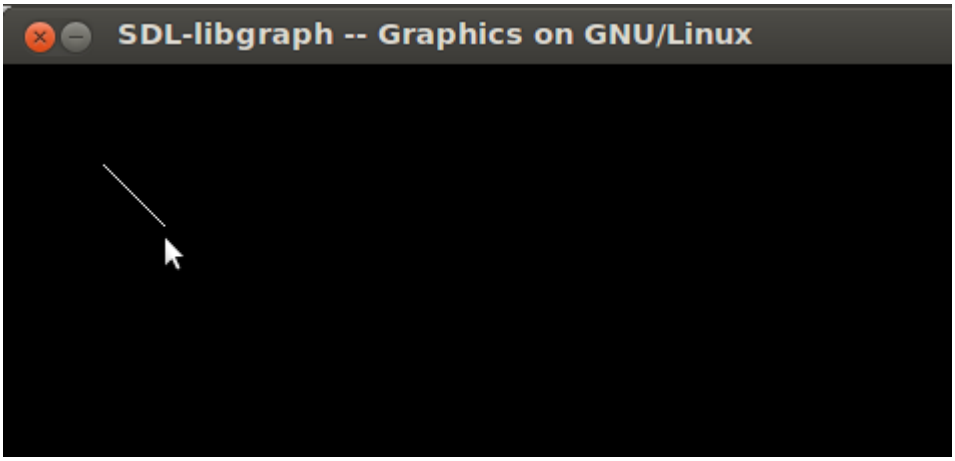

Figure 69: Output for line.c

4.5 Opening Existing Program/Project

Click 'File' → 'Open'. Browse to desired directory and select the file with .cbp extension as shown in figure 70 and click Open.

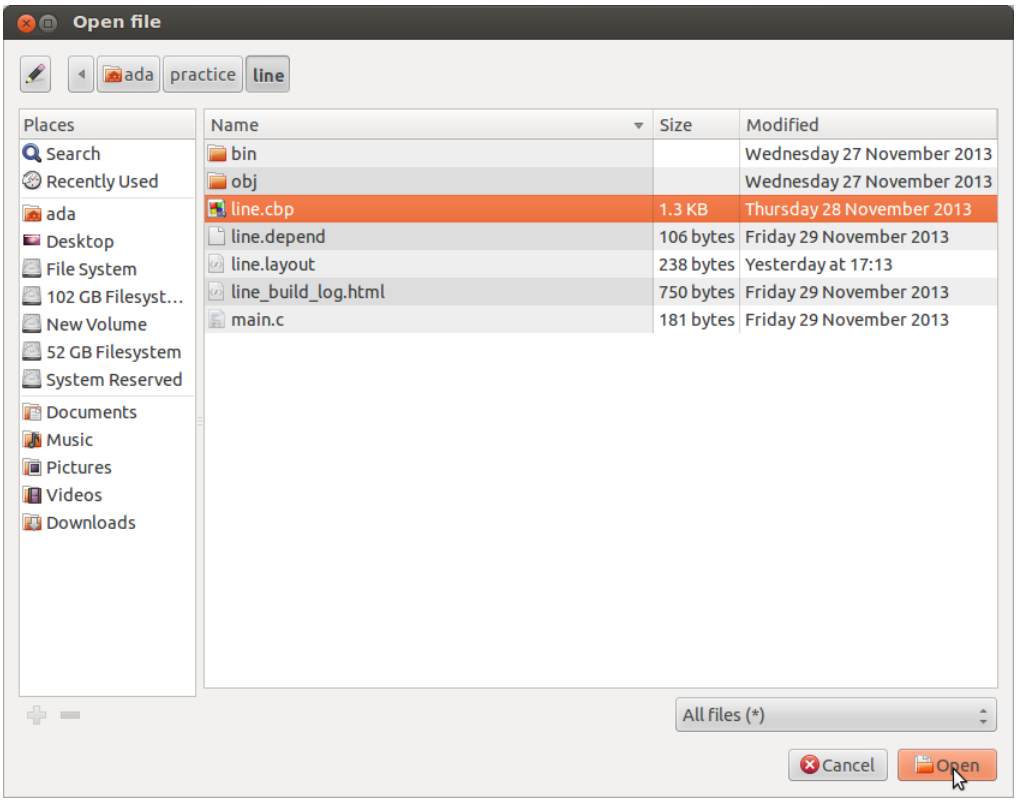

Figure 70: Select file with .cbp extension to open an existing project

References

[1] The Code::Block Team. Code::block homepage
http://www.codeblocks.org/.

[2] The Code::Block Team. Gpl v3.0 license
http://www.codeblocks.org/license.

[3] Installing code::block on ubuntu
http://wiki.codeblocks.org/index.php?title=Installing_Code::Blocks_from_source_on_Linux.

[4] Eternal thinker: How to use graphics.h in ubuntu?
http://blog.eternal-thinker.com/2010/09/how-to-use-graphicsh-in-ubuntu.html.

[5] Code::block faq's
http://wiki.codeblocks.org/index.php?title=FAQ.