

Manual for Code::Blocks and Simplecpp

for CS101 course



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

October 15, 2014

Contents

1	Introduction	1
1.1	Code::Blocks	1
1.2	Simplecpp	1
2	Code::Block IDE	1
3	Working with Code::Blocks and Simplecpp on Windows	6
3.1	Installation of Simplecpp with Code::Blocks	6
3.2	Writing a new C/C++ Program	12
3.2.1	Non-Graphic Project	12
3.2.2	Graphic Project using graphics.h	20
3.2.3	Graphic Project using simplecpp	24
3.3	Building the Project	31
3.4	Opening existing program/project	32
4	Working with Code::Blocks and Simplecpp on Ubuntu	33
4.1	Installation of Code::Blocks	33
4.2	Installation of packages for graphics.h header file	34
4.3	Installation of Simplecpp	34
4.4	Integrating Simplecpp with Code::Blocks IDE	36
4.5	Writing a new c/c++ program	39
4.6	Building the Project	43
4.6.1	Non-Graphics Project	43
4.6.2	Graphics Project using graphics.h	43
4.6.3	Graphics Project using simplecpp	48
4.7	Opening Existing Program/Project	49

List of Figures

1	Code::Block IDE	1
2	Menu Bar	2
3	Main Toolbar	2
4	Debugger Toolbar	3
5	Compiler Toolbar	3
6	Manager	3
7	Editor, Start/Home Page	4
8	Shortcut to Create New Project or Open Existing Project	4
9	Shortcut to History of Projects Opened Using Code::Blocks	5
10	Logs	5
11	Welcome to Code::Blocks Simplecpp Setup	6
12	Accepting License Agreement	6
13	License Inforamtion of Code::Blocks EP	7
14	Location of Installation	7
15	Start Menu Folder	8
16	Application Shortcut Desktop Icon	8
17	Installation Step	9
18	Progress of Installation	9
19	Basic Information	10
20	Completing the Insallation	10
21	Code::Block Simplecpp IDE	11
22	New form template	12
23	New console application wizard	13
24	Selecting language for the project	13
25	Providing title and folder for the project	14
26	Selecting compiler for the project	14
27	Project node with no files	15
28	Adding file to the project	15
29	Selecting type of file to be added in the project	16
30	Select checkbox to skip this window next time	16
31	Select the language of the file added	17
32	Details of file to be added	17
33	Selecting the location and file name to be added	18
34	Finalize details of file to be added	18
35	Project node with '+' sign indicating it can be expanded	19
36	Project node expanded	19
37	Writing code in editor	19
38	New form template	20
39	New console application wizard	20
40	Selecting language for the project	21
41	WinBGIM project	21
42	Providing title and folder for the project	22
43	Selecting compiler for the project	22
44	Finalize details of file to be added (for line project)	23
45	Project node with '+' sign indicating it can be expanded	23
46	Writing program in editor (line project)	24
47	New from template	24
48	Simplecpp project	25
49	Providing title and folder for the project	25
50	Selecting compiler for the project	26
51	Empty project node in Management window	26
52	Adding files to empty project node	27
53	Selecting c/c++ source for project	27

54	Selecting language for the file to be added	28
55	Giving location and name of the file to be added	28
56	Finalize details of file to be added	29
57	Project node with ‘+’ sign indicating it can be expanded	29
58	opening file in editor	30
59	coding	30
60	Output of hello world project	31
61	Output of line project	31
62	Output of 3poly project	31
63	Selecting Open under file in Menu Bar	32
64	Select file with .cbp extension to open an existing project	32
65	Code::Block in Ubuntu Software Center	33
66	Installing Code::Block using command line	33
67	Installing simplecpp	35
68	location of simplecpp/s++	35
69	Compiler and debugger settings	36
70	Copying the compiler	36
71	Change Toolchain Executable	37
72	Changes to be made in Toolchain Executables	37
73	Simplecpp’s directory	38
74	Additional Paths settings	38
75	Starting a new project	39
76	Selecting the language for project	39
77	Title for Project	40
78	Selecting Compiler to Compile the Program	40
79	Selecting simplecpp for projects including simplecpp	41
80	Project Node when Expanded	41
81	Project Node when Expanded for helloworld.c (with code shown in editor)	42
82	Project node when expanded for line.c (with code shown in editor)	42
83	Project node when expanded for 3poly.cpp (with code shown in editor)	43
84	Output for helloworld.c	43
85	Project build options	44
86	Linker settings (Add Libraries)	44
87	Interface for adding libraries	45
88	Files/libraries to be added for graphic projects	45
89	Relative or absolute path for files/libraries	46
90	libraries selected	46
91	Libraries added to project	47
92	Output for line.c	47
93	Output for 3poly.cpp	48
94	Select file with .cbp extension to open an existing project	49

1 Introduction

1.1 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 12.04¹ and Windows 7.²

1.2 Simplecpp

Simplecpp is a package used to write and execute turtle based non-graphic/graphic programs. Simplecpp is a package developed by Prof. Abhiram Ranade at IIT Bombay and the book on ‘An Introduction to Programming through C++’ by Prof. Abhiram Ranade uses simplecpp to explain basic programming. For more information, please refer to the Chapter 1 of the book. Simplecpp is integrated with Code::Blocks and the simplecpp programs are tested with Code::Blocks on Windows 7 and Ubuntu 12.04.

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

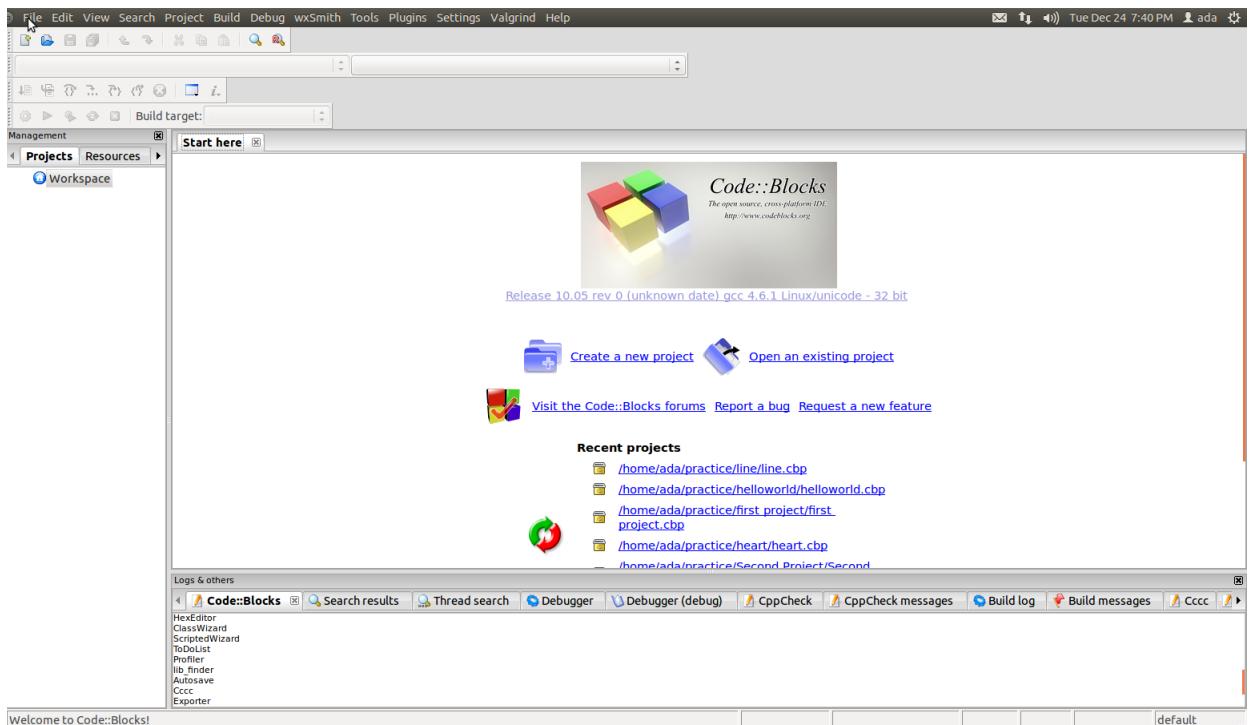


Figure 1: Code::Block IDE

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

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

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):

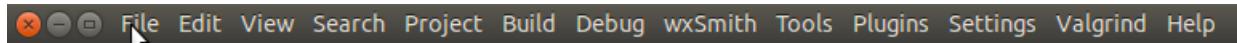


Figure 2: Menu Bar

- (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 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.

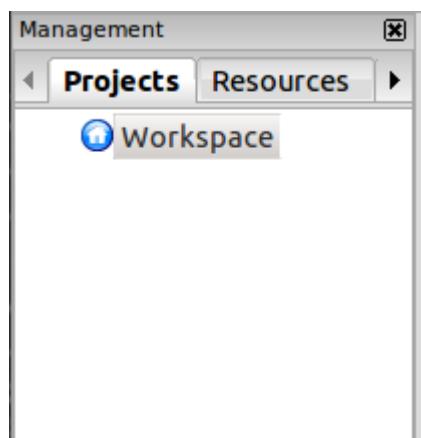


Figure 6: Manager

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

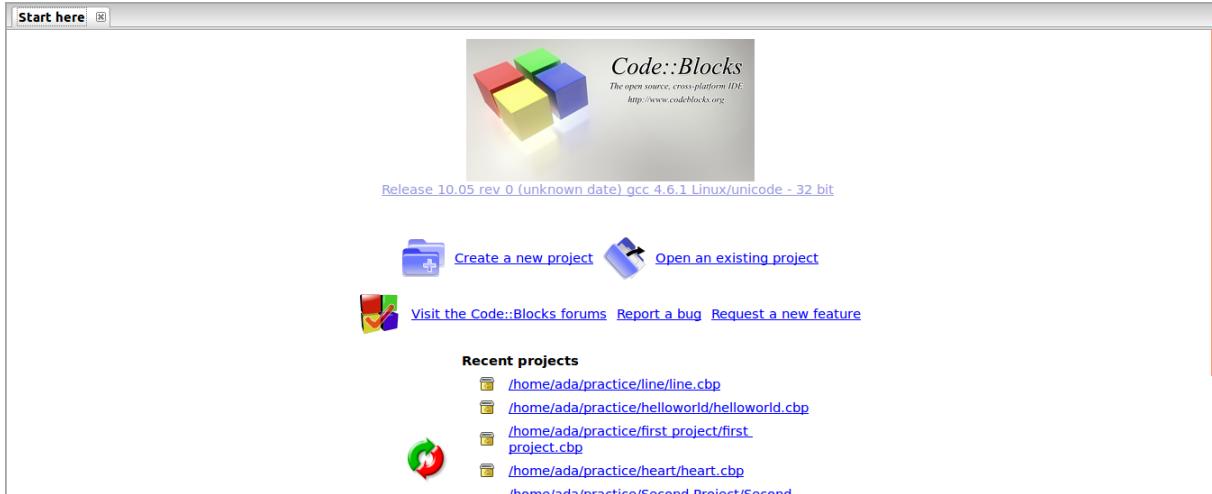


Figure 7: Editor, Start/Home Page

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.

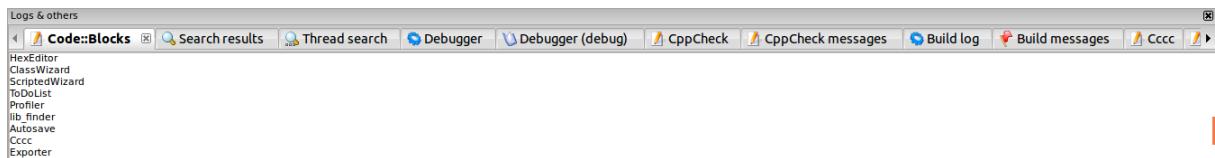


Figure 10: Logs

3 Working with Code::Blocks and Simplecpp on Windows

In this section we discuss writing and building of three projects. First project (`hello world.c`) is a simple program which displays *hello world* on output. The second project (`line.c`) uses `graphics.h` header file and displays a line. The third project (`3poly.cpp`) uses `simplecpp package` and draws three polygons on output. **The simplecpp package is integrated with Code::Blocks-EP IDE**

3.1 Installation of Simplecpp with Code::Blocks

Download the .exe file from <http://www.cse.iitb.ac.in/~cs101/Project/simplecpp/Windows/CB-Simplecpp-setup.exe> The installation steps are as given below.

1. A window as shown in Figure 11 will be displayed after double clicking on the .exe file. Click ‘Next’.

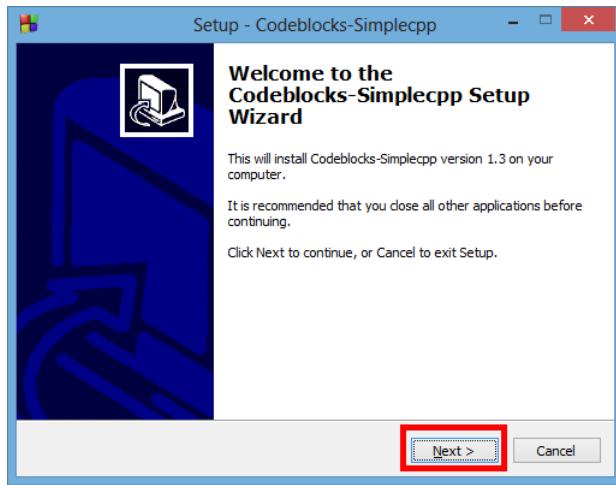


Figure 11: Welcome to Code::Blocks Simplecpp Setup

2. A new window will be displayed as shown in Figure 12 giving information on the license agreement. Select ‘I accept the agreement’. Click ‘Next’.

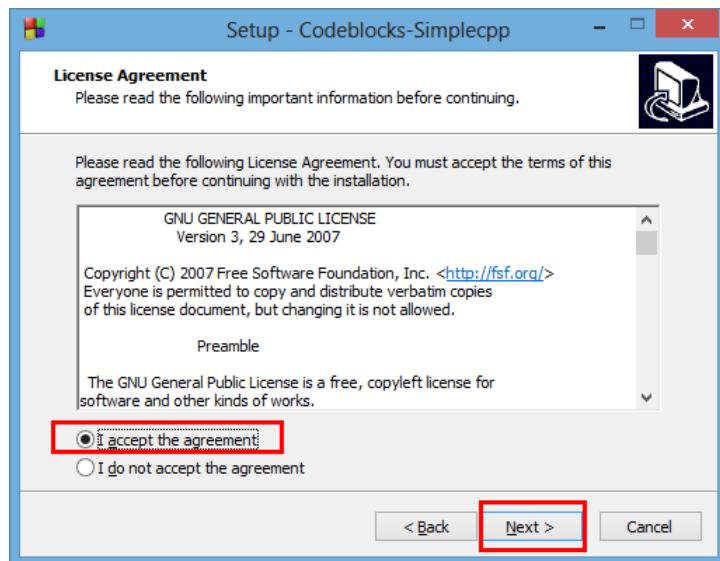


Figure 12: Accepting License Agreement

3. A new window will be displayed giving license information of Code::Blocks EP as shown in Figure 13. Click ‘Next’.

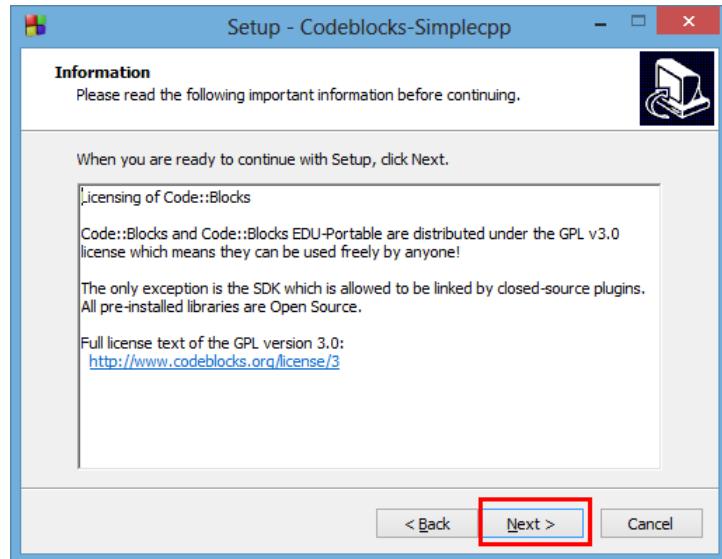


Figure 13: License Information of Code::Blocks EP

4. A new window will be displayed as shown in Figure 14. This will display the location in which Code::Blocks Simplecpp will be installed. By default it is *C:\Program Files (x86)\Codeblocks-Simplecpp*. If you want to change this default location, click browse and select the appropriate location. Click ‘Next’.

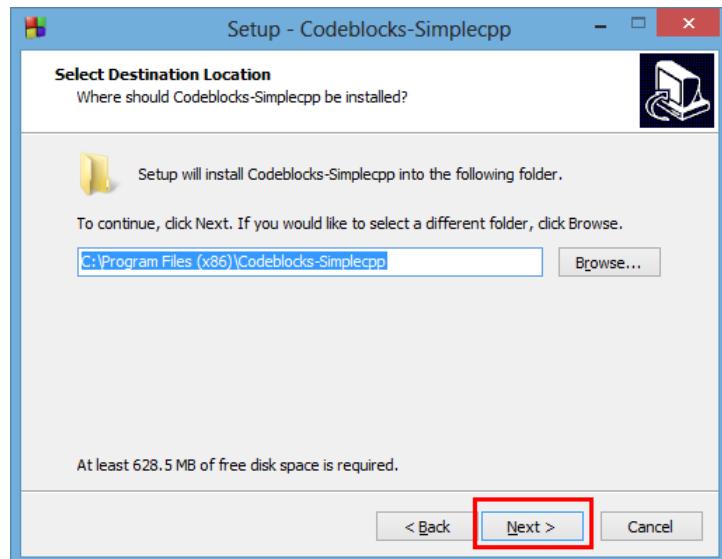


Figure 14: Location of Installation

5. A new window will be displayed as shown in Figure 15 will be displayed. Click ‘Next’.

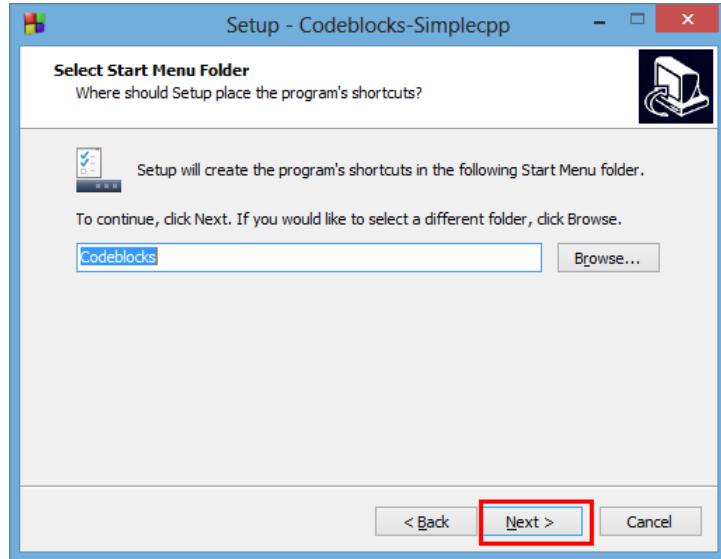


Figure 15: Start Menu Folder

6. A new window will be displayed as shown in Figure 16 will be displayed asking whether a shortcut application desktop icon should be created or not. Select ‘Create desktop icon’. Click ‘Next’.

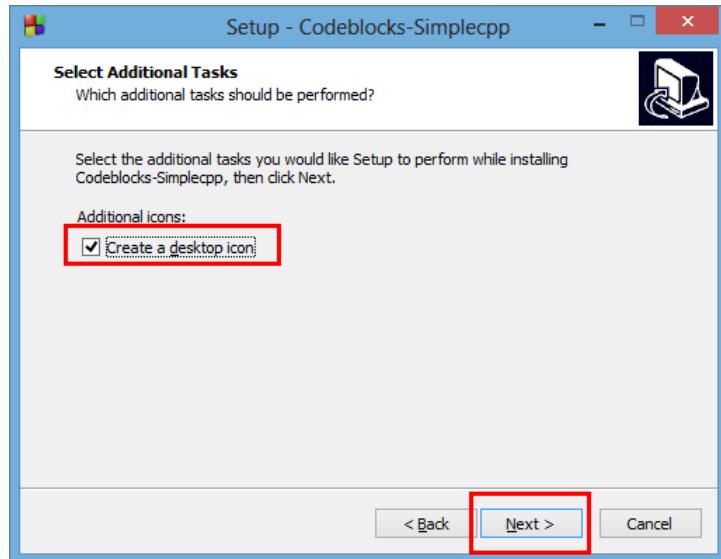


Figure 16: Application Shortcut Desktop Icon

7. A new window will be displayed as shown in Figure 17 will be displayed. Click ‘Next’.

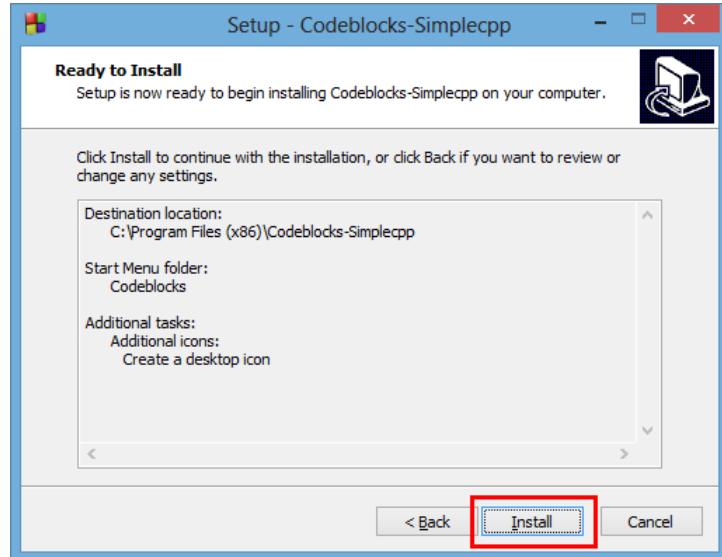


Figure 17: Installation Step

8. A new window will be displayed as shown in Figure 18 will be displayed. This window will show the progress of the installation. Click ‘Next’.

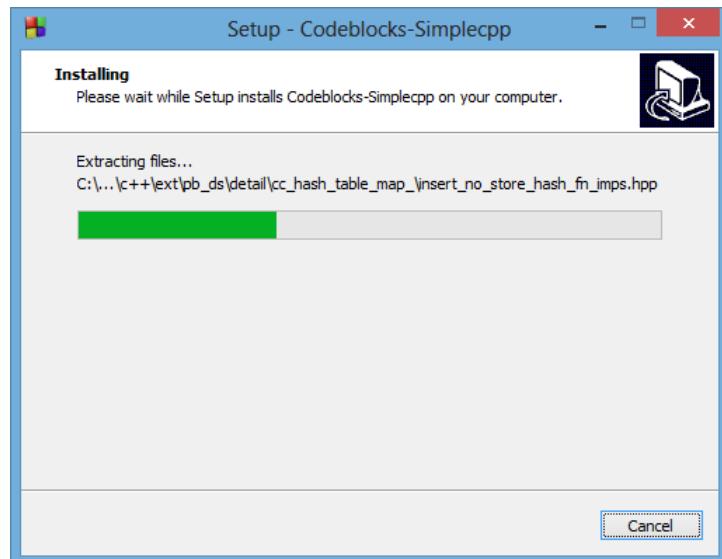


Figure 18: Progress of Installation

9. A new window will be displayed as shown in Figure 19 will be displayed . Click ‘Next’.

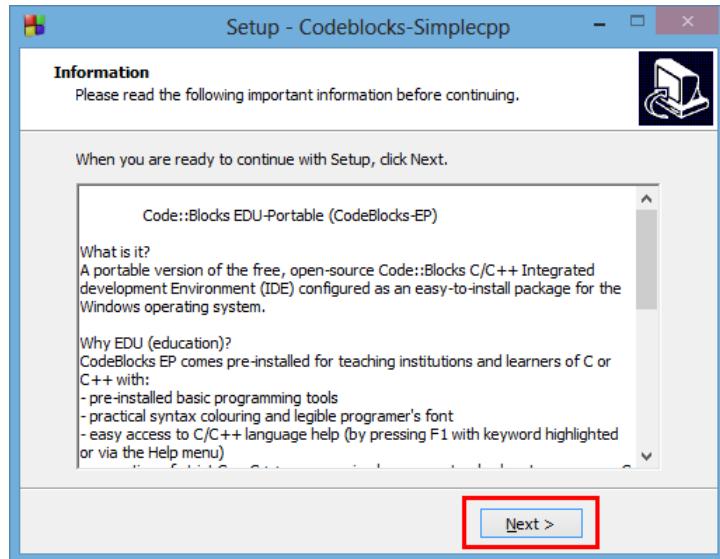


Figure 19: Basic Information

10. A new window will be displayed as shown in Figure 20 will be displayed . Select ‘Launch Codeblocks-Simplecpp’. Click ‘Finish’.

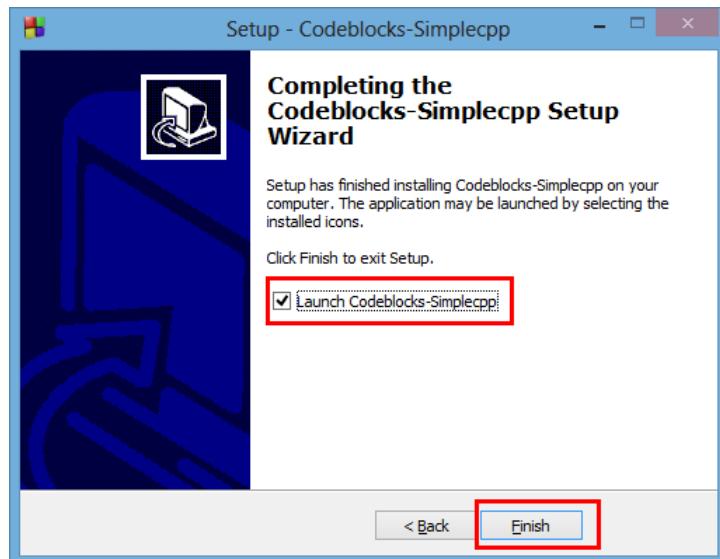


Figure 20: Completing the Installation

11. CodeBlocks IDE opens as shown in figure 21. Alternatively CodeBlocks-EP can be launched by double clicking the desktop icon created or clicking on it's shortcut icon in Start Menu Bar.

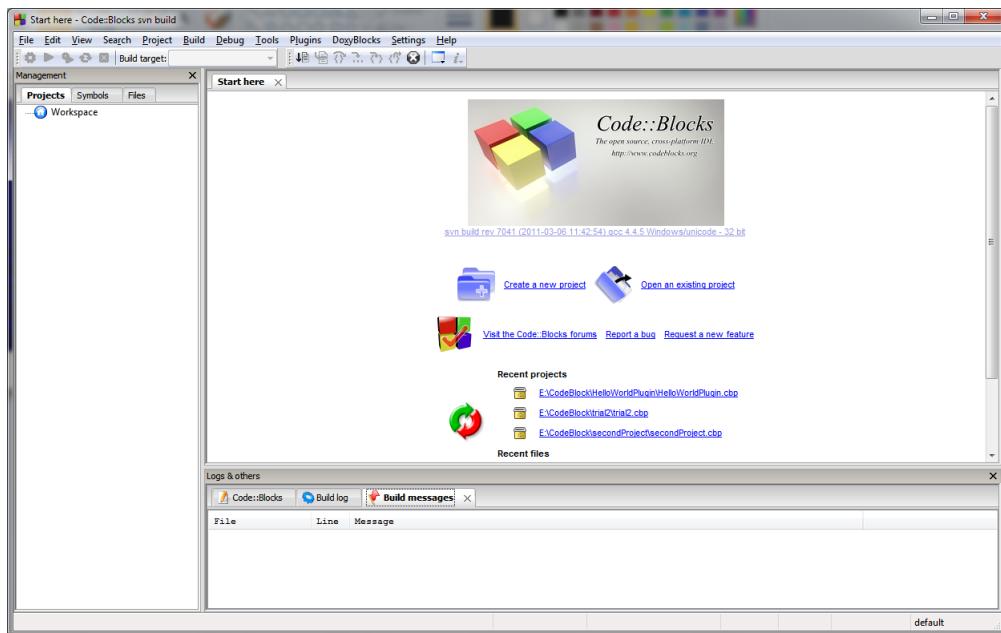


Figure 21: Code::Block Simplecpp IDE

3.2 Writing a new C/C++ Program

This section is divided into writing non-graphic program, writing graphic program using graphics.h header file and writing graphic program using simplecpp package.

3.2.1 Non-Graphic Project

1. Click on *New file* button. The ‘New from template’ window as shown in figure 22 opens. For non-graphics projects, select ‘Console application’. When the type of project is selected the *Go* button gets highlighted (top right corner). Click *Go*.

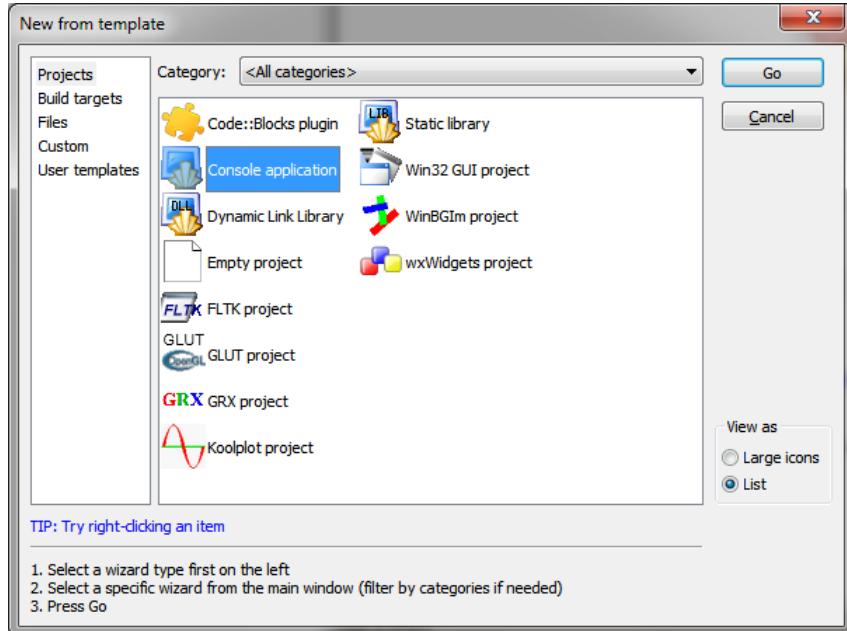


Figure 22: New form template

2. When *Go* button is clicked a new window opens as shown in figure 23. Select checkbox “Skip this page next time” so that the page is not displayed again. Click ‘Next’

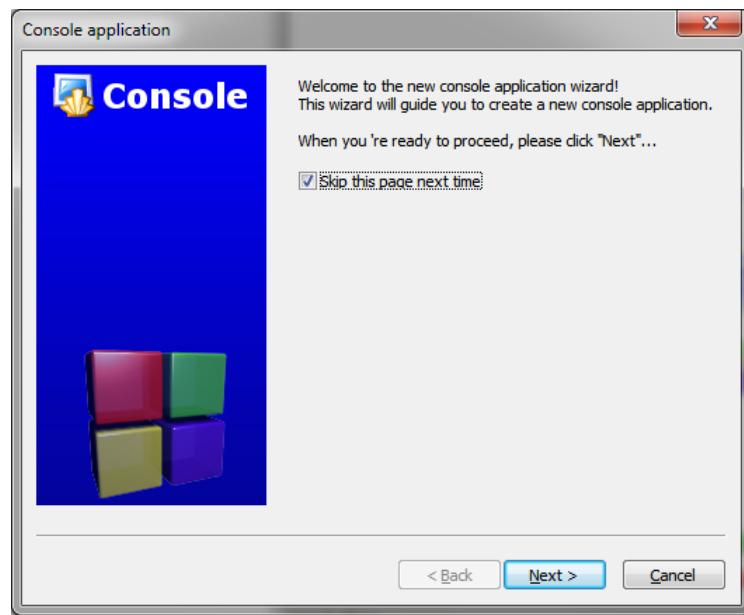


Figure 23: New console application wizard

3. Next window enables user to select the language to be used for project as shown in Figure 24. Select C/C++. Here, 'C' is selected for helloworld project. Click 'Next'.

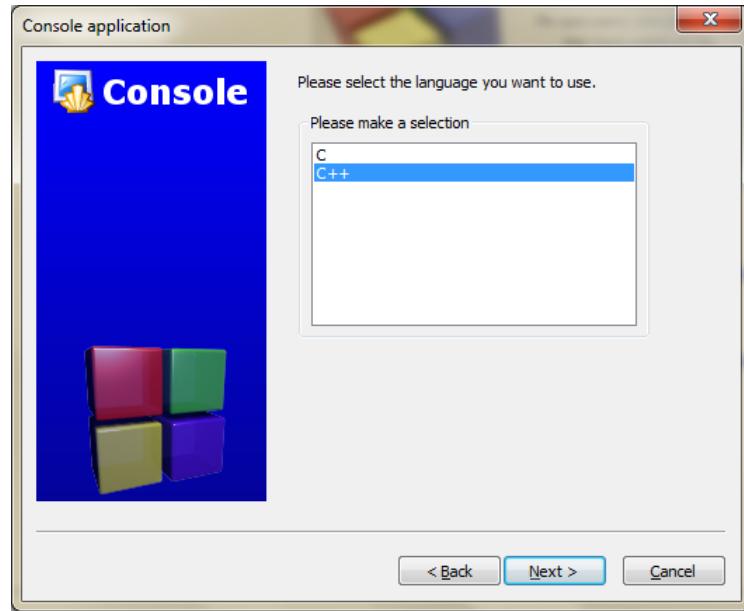


Figure 24: Selecting language for the project

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 25. After filling in the details click on Next.

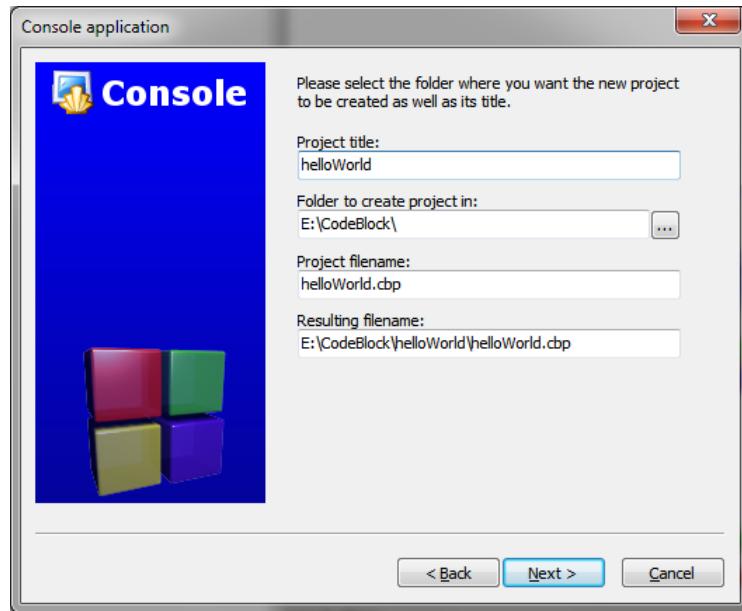


Figure 25: Providing title and folder for the project

5. Next window is used to select the compiler as shown in figure 26. By default 'GNU GCC Compiler' is selected. Click on Finish.

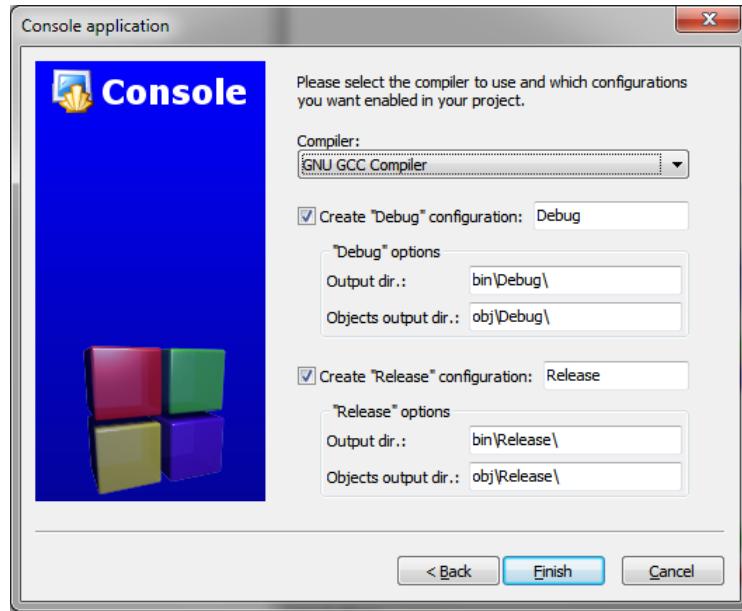


Figure 26: Selecting compiler for the project

6. The project node opens in manager window as shown in figure 27. The project node is empty and we have to add files to the project.

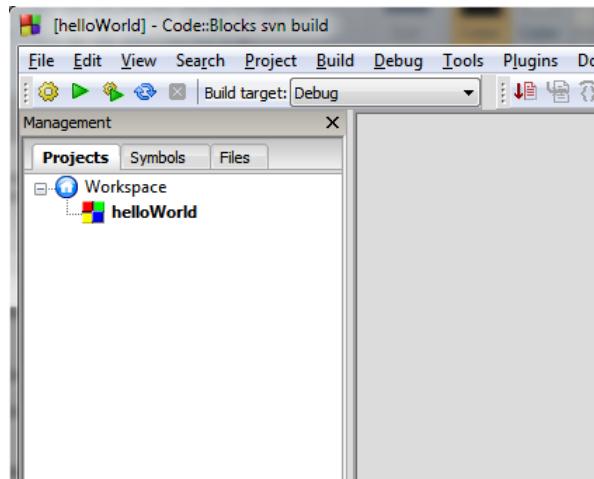


Figure 27: Project node with no files

7. To add files to the project select project node and click on File in menu bar, then click on 'File...' in options in 'New'. The process is shown in figure 28.

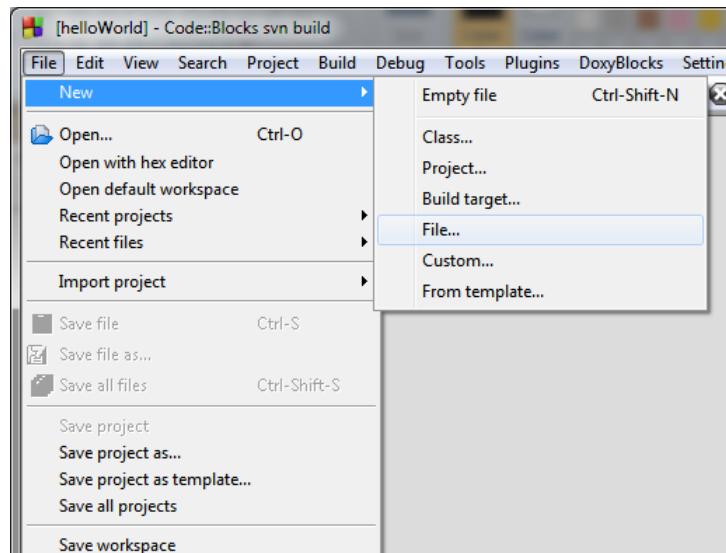


Figure 28: Adding file to the project

8. New from template opens as shown in figure 29. For our example select ‘C/C++ source’ and click on Go.

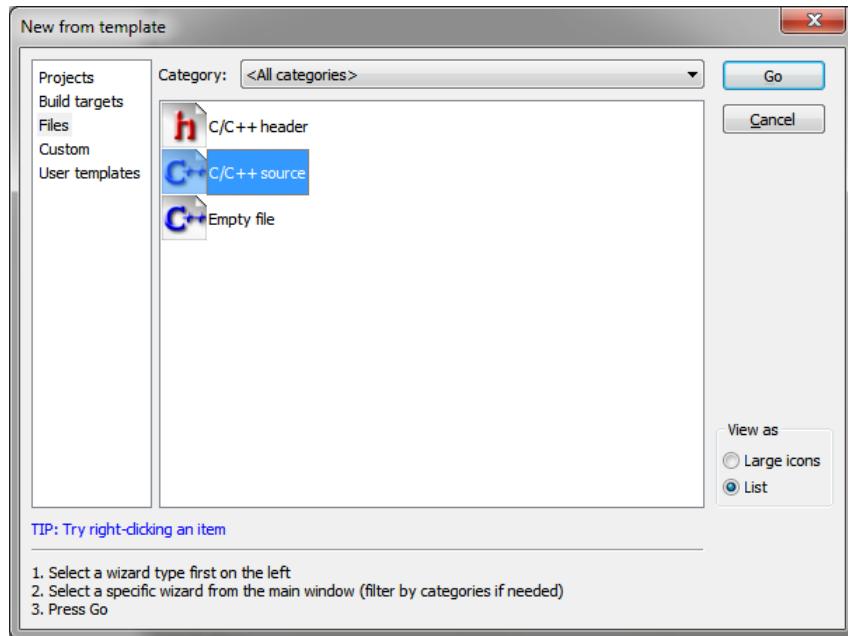


Figure 29: Selecting type of file to be added in the project

9. A new window is displayed as shown in figure 30. Select the checkbox ‘Skip this page next time’ so that it is not displayed again.

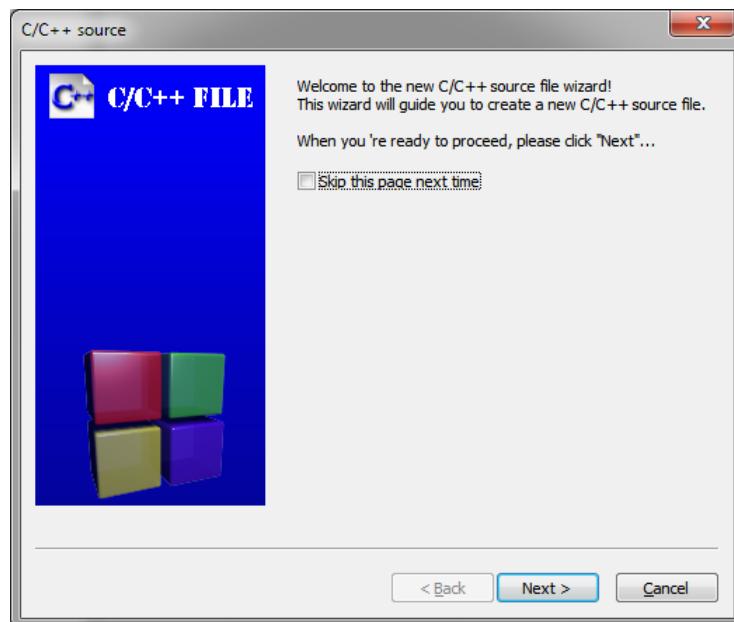


Figure 30: Select checkbox to skip this window next time

10. Select the preferred language as shown in figure 31. We have selected ‘C’ for our helloworld example.

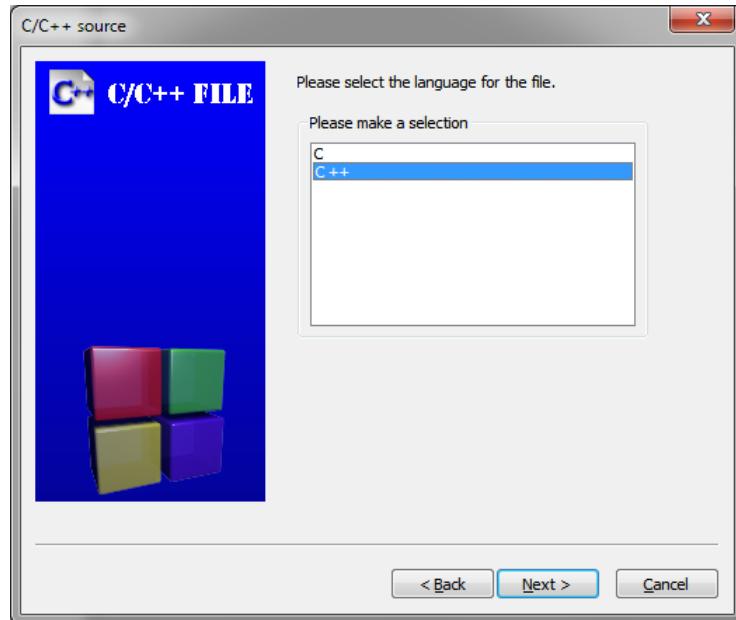


Figure 31: Select the language of the file added

11. Figure 32 shows the window that opens to add the file. click on ‘...’ beside ‘Filename with full path’.

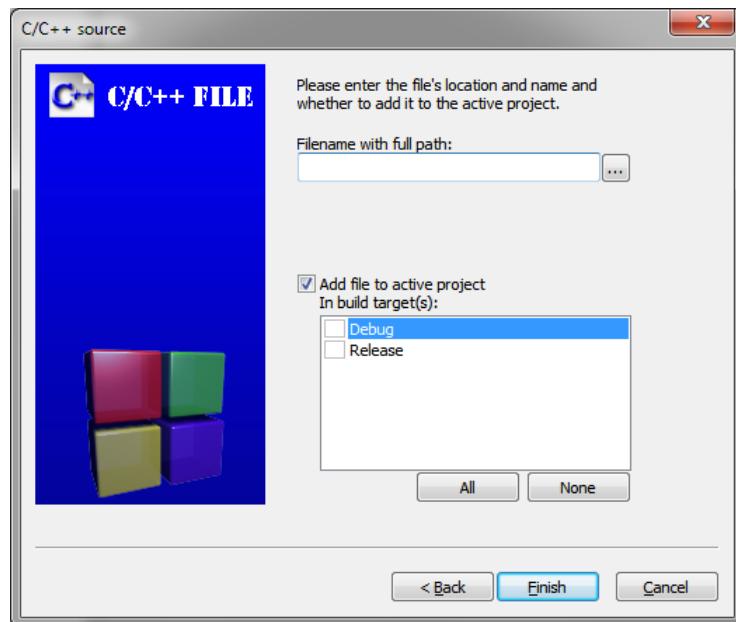


Figure 32: Details of file to be added

A window as shown in figure 33 opens. Select the folder of the project and enter file name to be added. Click on ‘Save’.

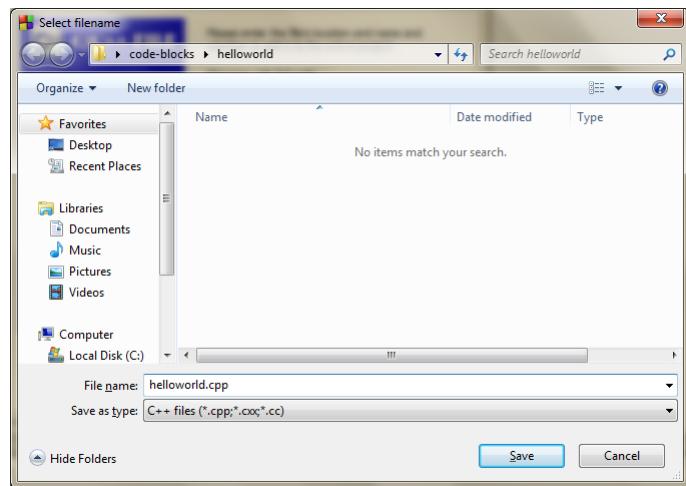


Figure 33: Selecting the location and file name to be added

When ‘Save’ button is clicked the window in figure 32 opens again with full path and name of the file. Select Debug and Release. Shown in figure 34. Click on Finish.

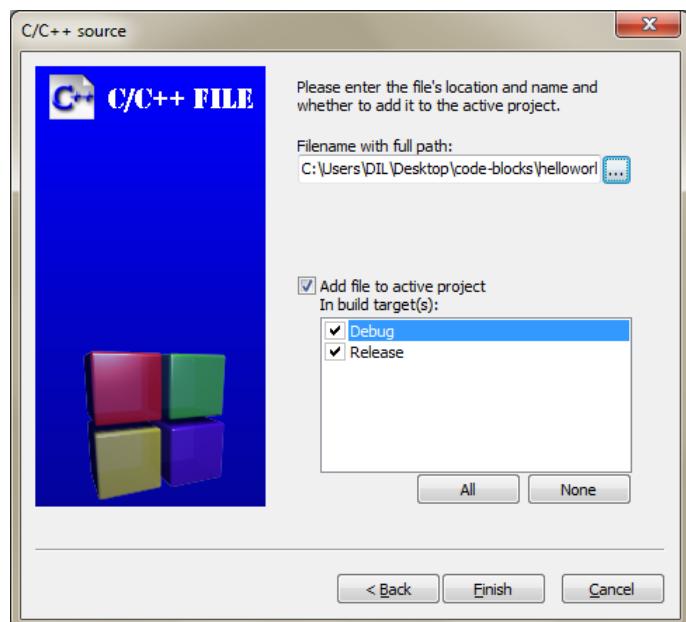


Figure 34: Finalize details of file to be added

12. Management window now shows project node which can be expanded. Click on project node and double click on 'hello.c' to open the file in editor shown in figure 35 and figure 36

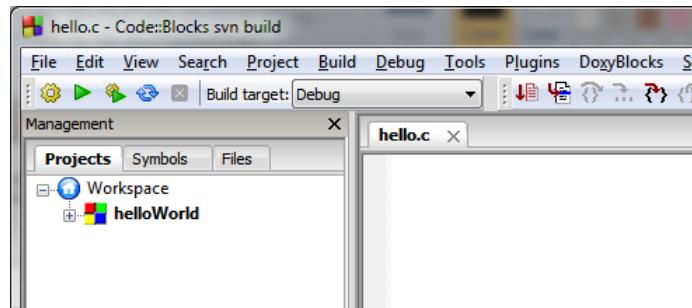


Figure 35: Project node with '+' sign indicating it can be expanded

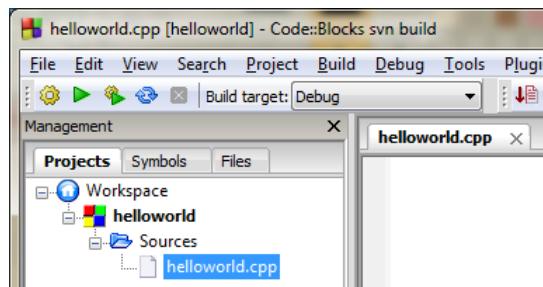


Figure 36: Project node expanded

13. When the main.cpp file opens in editor, user can start coding. Code is shown in figure 37.

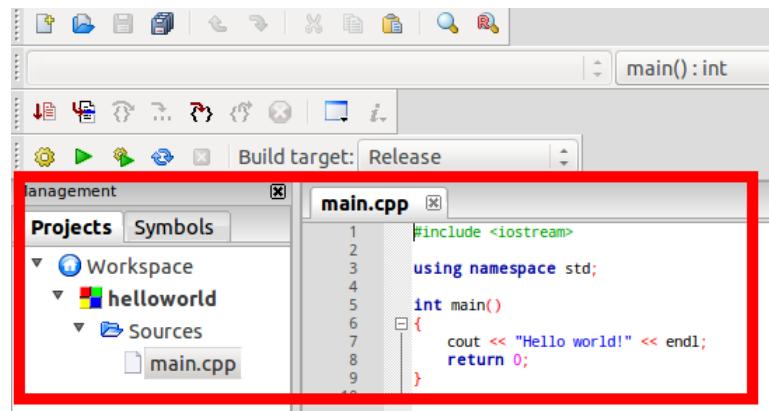


Figure 37: Writing code in editor

3.2.2 Graphic Project using graphics.h

1. Click on *New file* button. The ‘New form template’ window as shown in figure 22 opens. For graphics projects, select ‘WinBGI project’. *Go* button gets highlighted (top right corner). Click on *Go*.

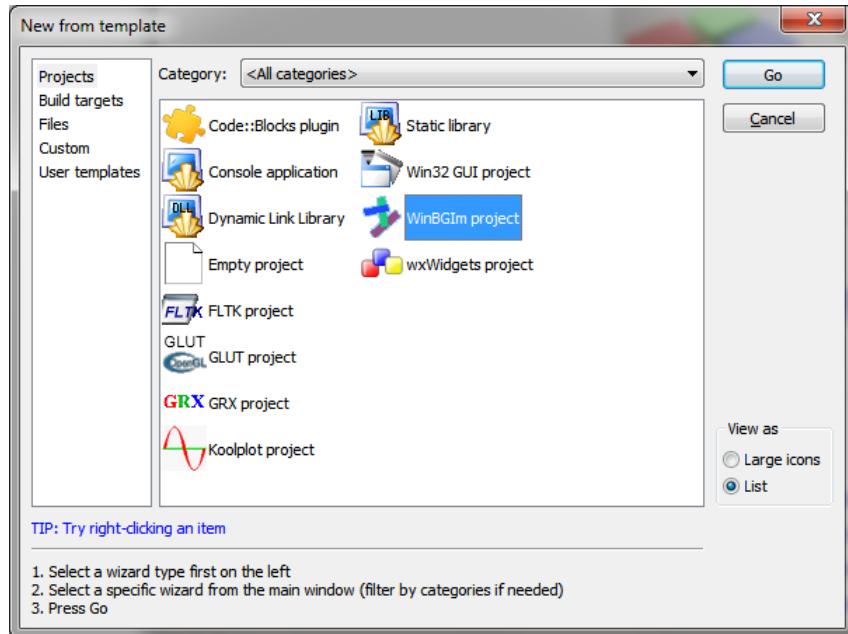


Figure 38: New form template

2. When *Go* button is clicked a new window opens as shown in Figure 39. 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.

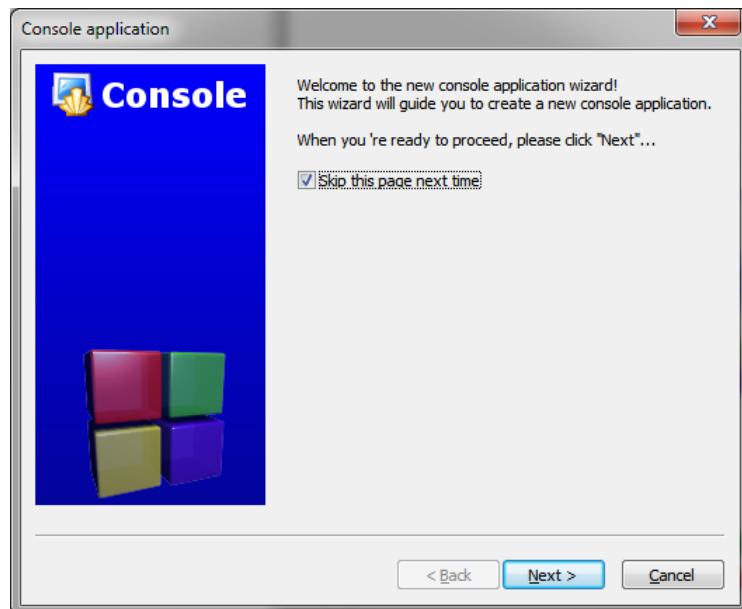


Figure 39: New console application wizard

3. Next window enables user to select the language to be used for project as shown in figure 40. For

the example hello world used in this manual select ‘C’ and click on Next.

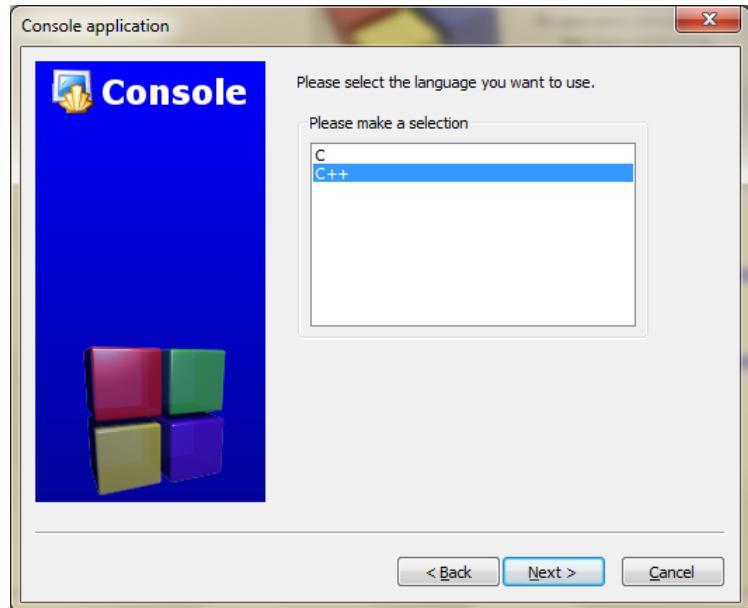


Figure 40: Selecting language for the project

4. Next window asks the user to select type of project. The options are ‘Add Console’ and ‘Graphics only’ as shown in figure 41. Select ‘Graphics only’ and click on *Next*.

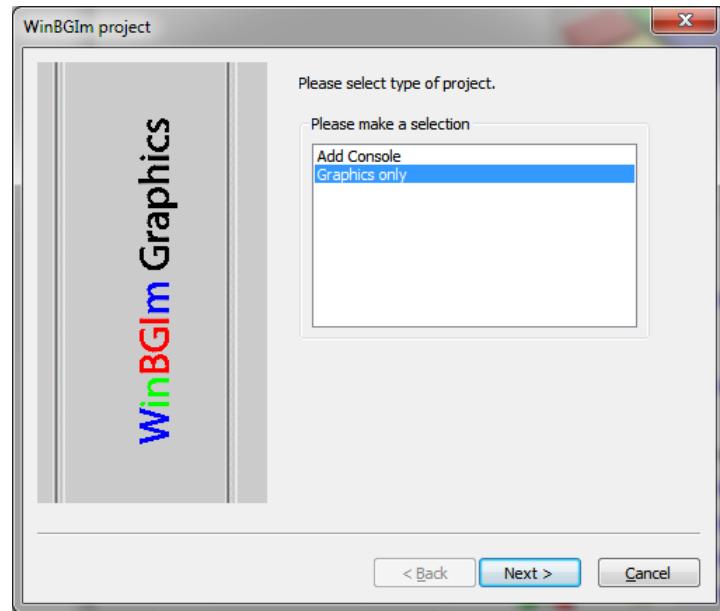


Figure 41: WinBGIm project

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 42. After filling in the details click on Next.

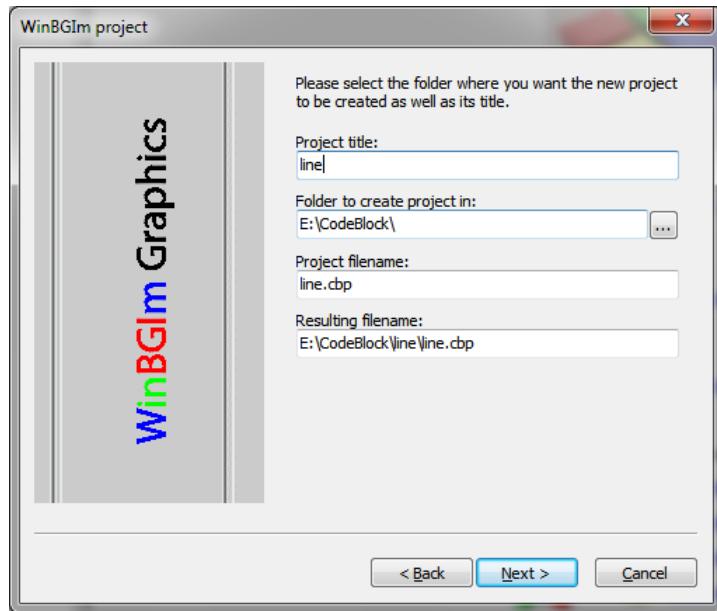


Figure 42: Providing title and folder for the project

6. Next window is used to select the compiler as shown in figure 43. By default 'GNU GCC Compiler' is selected. Click on Finish.

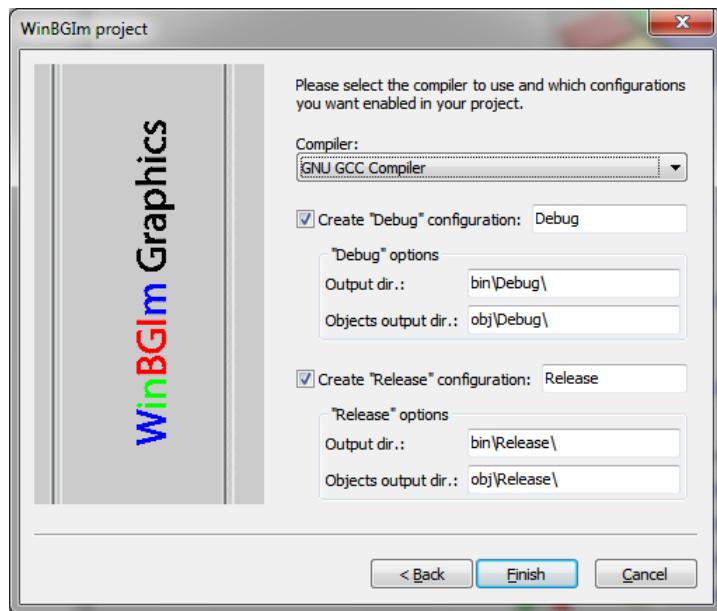


Figure 43: Selecting compiler for the project

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 29. 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 32. A window as shown in figure 33 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 44. Click on Finish.

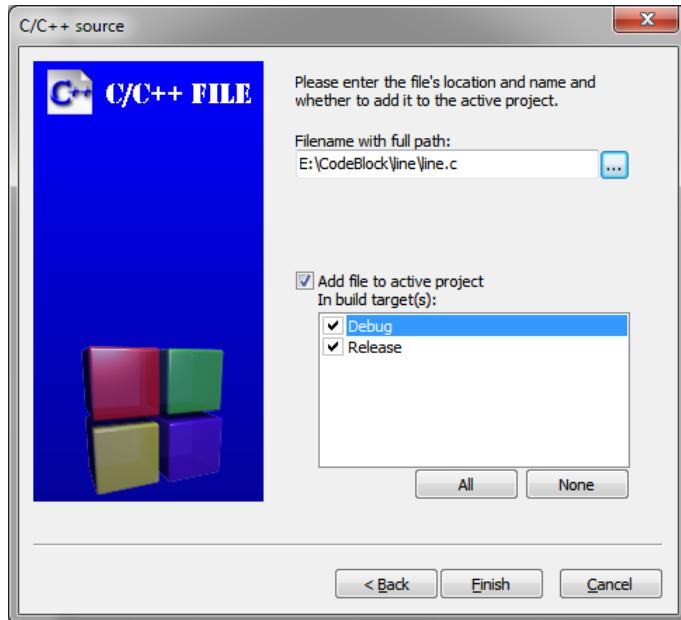


Figure 44: Finalize details of file to be added (for line project)

8. Management window now shows project node which can be expanded (figure 45).

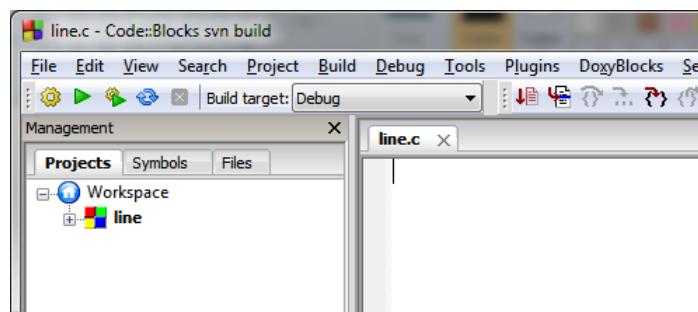
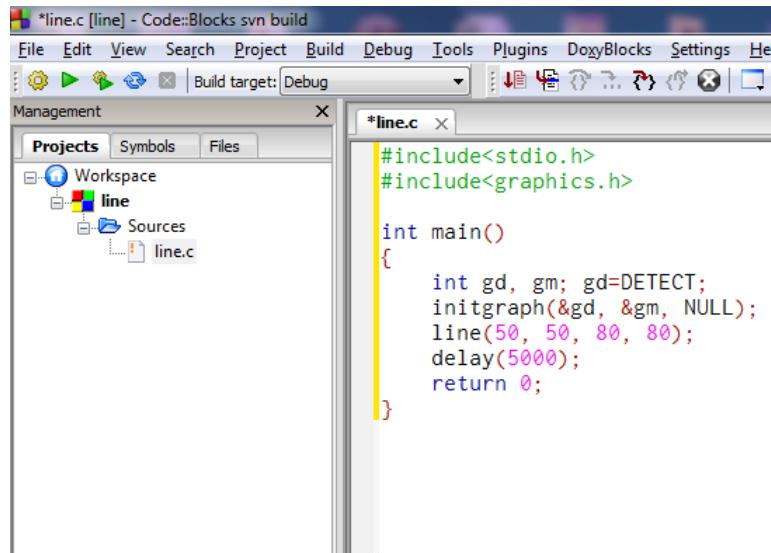


Figure 45: Project node with ‘+’ sign indicating it can be expanded

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 46.



```
#include<stdio.h>
#include<graphics.h>

int main()
{
    int gd, gm; gd=DETECT;
    initgraph(&gd, &gm, NULL);
    line(50, 50, 80, 80);
    delay(5000);
    return 0;
}
```

Figure 46: Writing program in editor (line project)

3.2.3 Graphic Project using simplecpp

1. Click on *New file* button. The ‘New form template’ window as shown in figure 22 opens. For graphics projects using simplecpp, select ‘Simplecpp project’. *Go* button gets highlighted (top right corner). Click on *Go*.

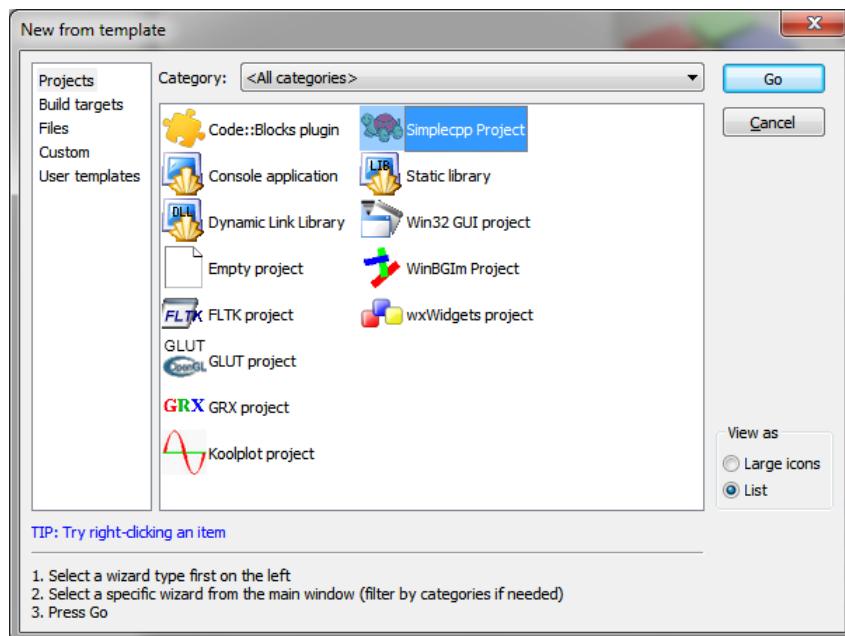


Figure 47: New from template

2. Next window asks the user to select type of project. The options are ‘Add Console’ and ‘Graphics only’ as shown in figure 48. Select ‘Graphics only’ and click on *Next*.

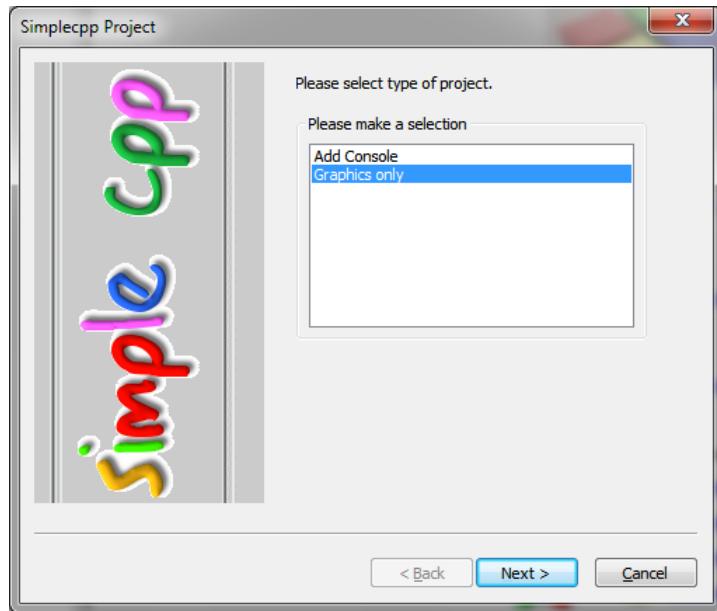


Figure 48: Simplecpp project

3. 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 49. After filling in the details click on *Next*.

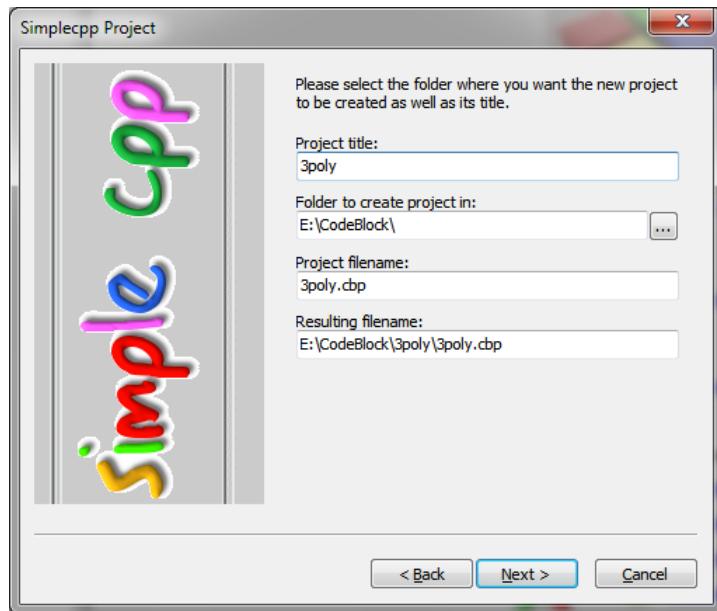


Figure 49: Providing title and folder for the project

4. Next window is used to select the compiler as shown in figure 43. By default ‘GNU GCC Compiler’ is selected. Click on Finish.

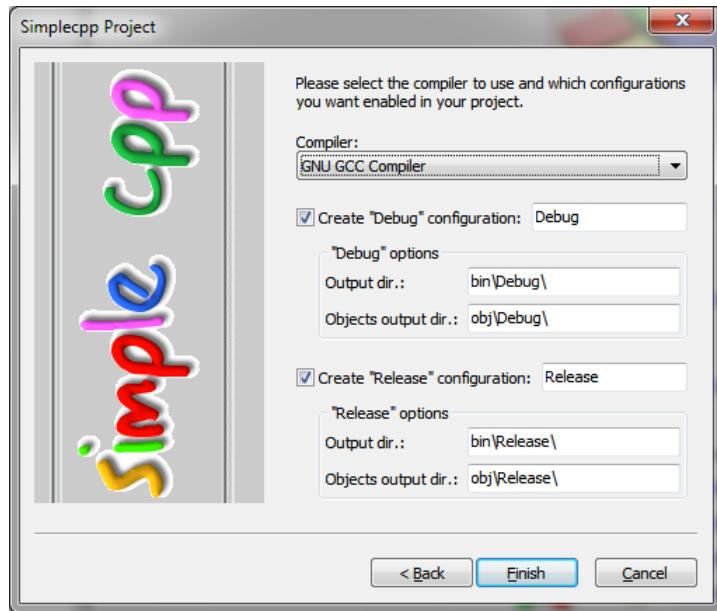


Figure 50: Selecting compiler for the project

5. The project node opens in manager window. The project node is empty as shown in figure 51. We have to add files to the project.

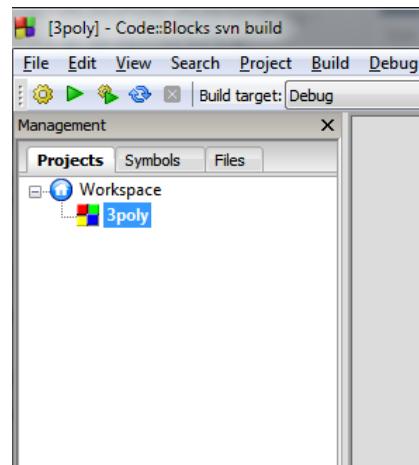


Figure 51: Empty project node in Management window

6. To add files to the project select project node and click on File in menu bar, then click on ‘File...’ in options in ‘New’. Shown in figure 52.

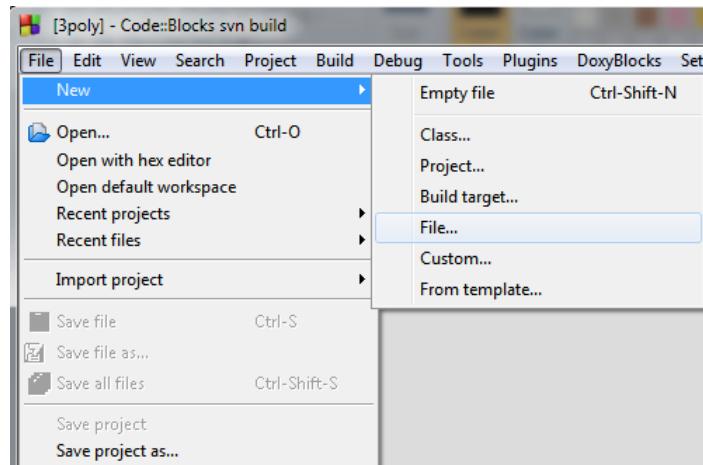


Figure 52: Adding files to empty project node

7. New from template opens as shown in figure 53. For our example select ‘C/C++ source’ and click on Go.

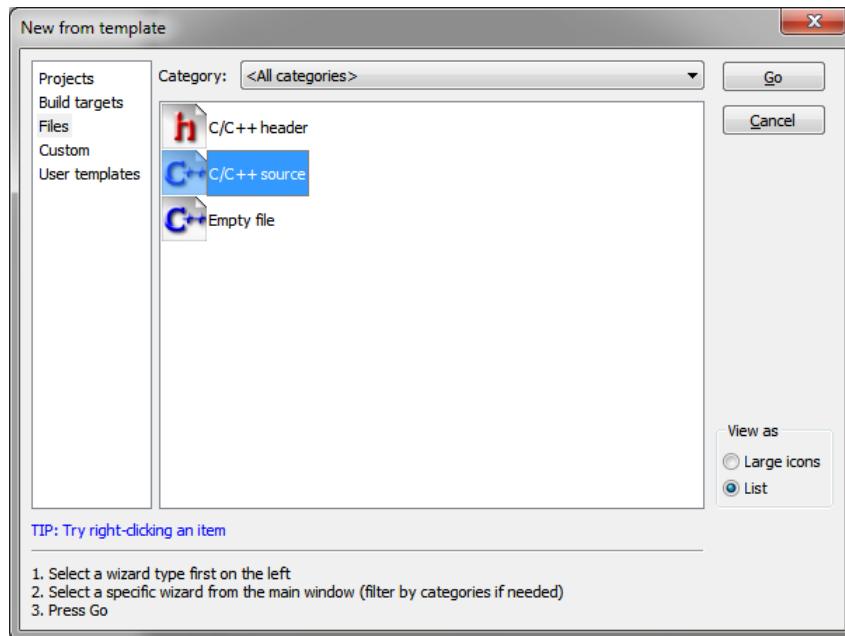


Figure 53: Selecting c/c++ source for project

8. Select the preferred language. For our example select ‘C++’ (figure 54).

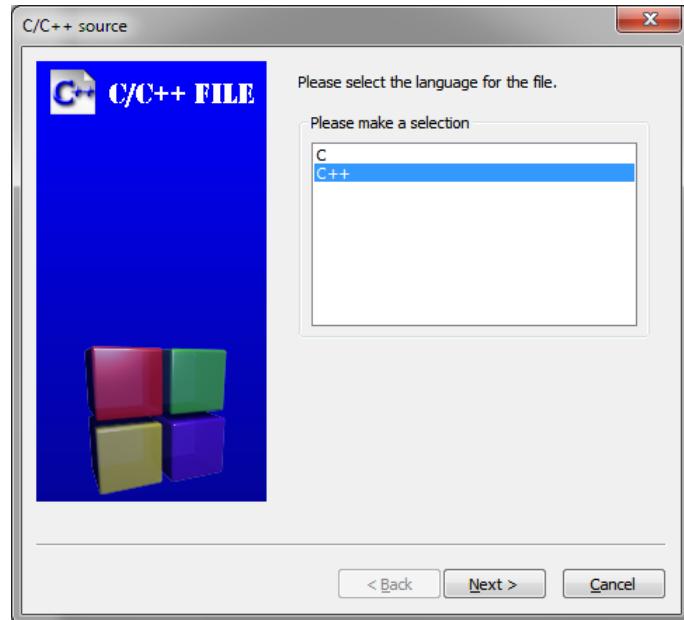


Figure 54: Selecting language for the file to be added

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 32. A window as shown in figure 55 opens. Select the folder of the project and enter file name to be added. Click on ‘Save’.

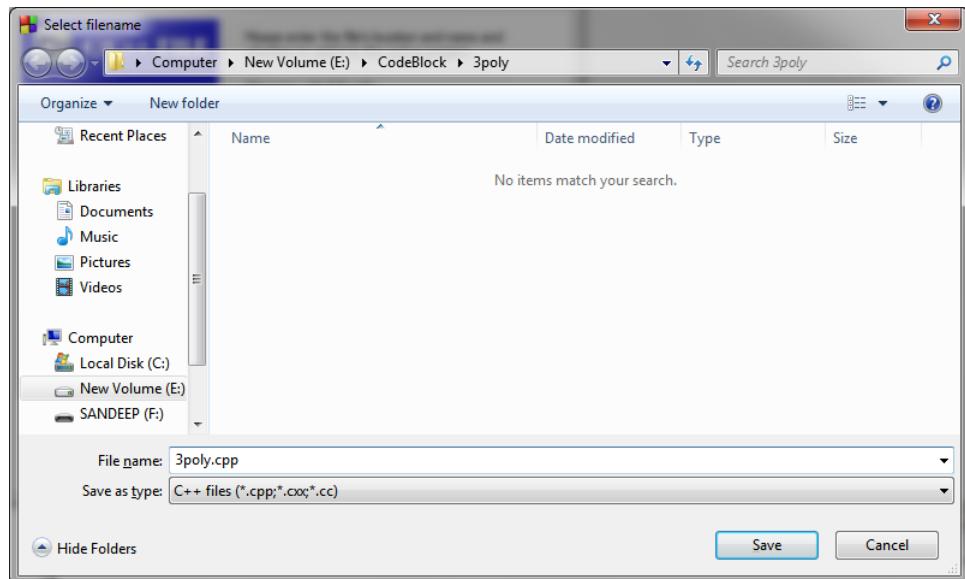


Figure 55: Giving location and name of the file to be added

In next window select Debug and Release. Shown in figure 56. Click on Finish.

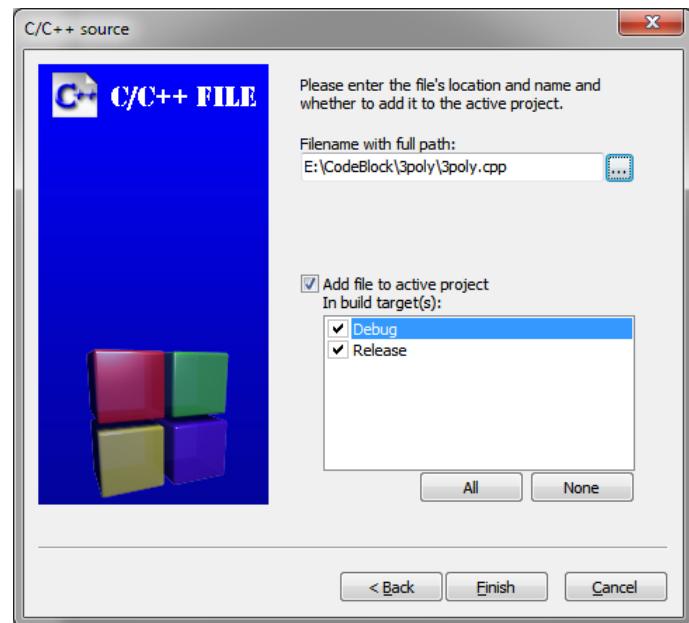


Figure 56: Finalize details of file to be added

9. Management window now shows project node which can be expanded (figure 57).

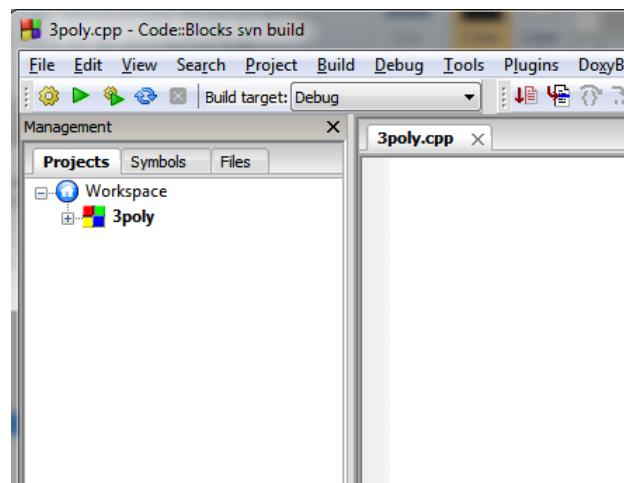


Figure 57: Project node with ‘+’ sign indicating it can be expanded

10. Click on project node and double click on ‘3poly.cpp’ to open the file in editor. When the 3poly.cpp file opens in editor (figure 58), user can start coding. Code is shown in figure 59.

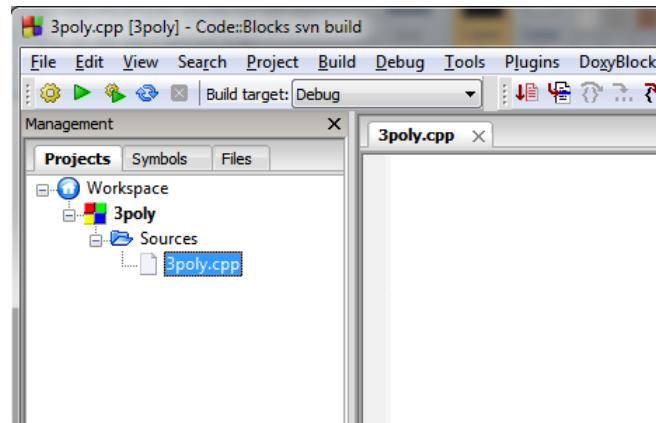


Figure 58: opening file in editor

A screenshot of the Code::Blocks IDE showing the "3poly.cpp" file content. The code uses the SimpleC++ library to draw a three-pointed star (triangle) with three turtles. It includes an include directive for "simplecpp", defines a main program with three turtles, and uses a repeat loop to draw the star. A green vertical line highlights the closing brace of the repeat loop.

Figure 59: coding

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 60 for hello world project, figure 61 for line project and figure 62 for 3poly project.

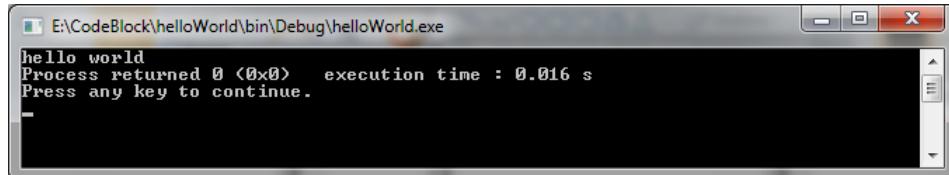


Figure 60: Output of hello world project

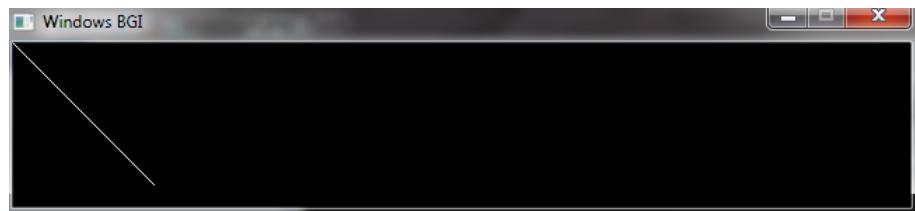


Figure 61: Output of line project

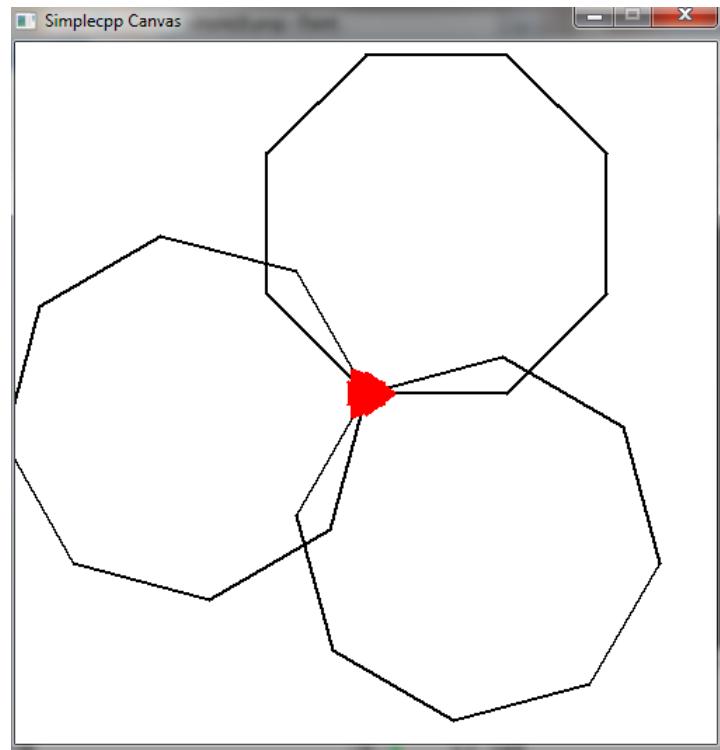


Figure 62: Output of 3poly project

3.4 Opening existing program/project

Click on *Open* button 63. Browse to desired directory and open the file with *.cbp* extension as shown in figure 64.

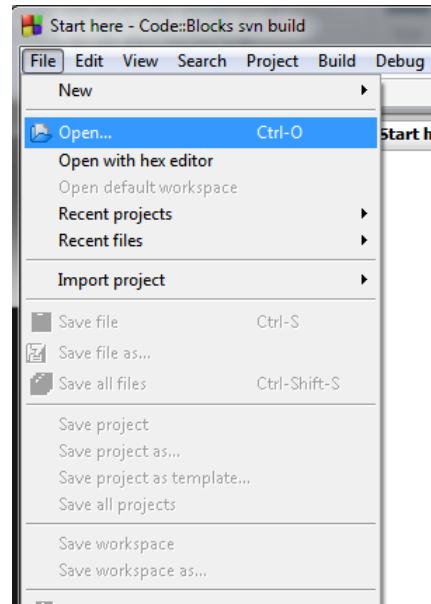


Figure 63: Selecting Open under file in Menu Bar

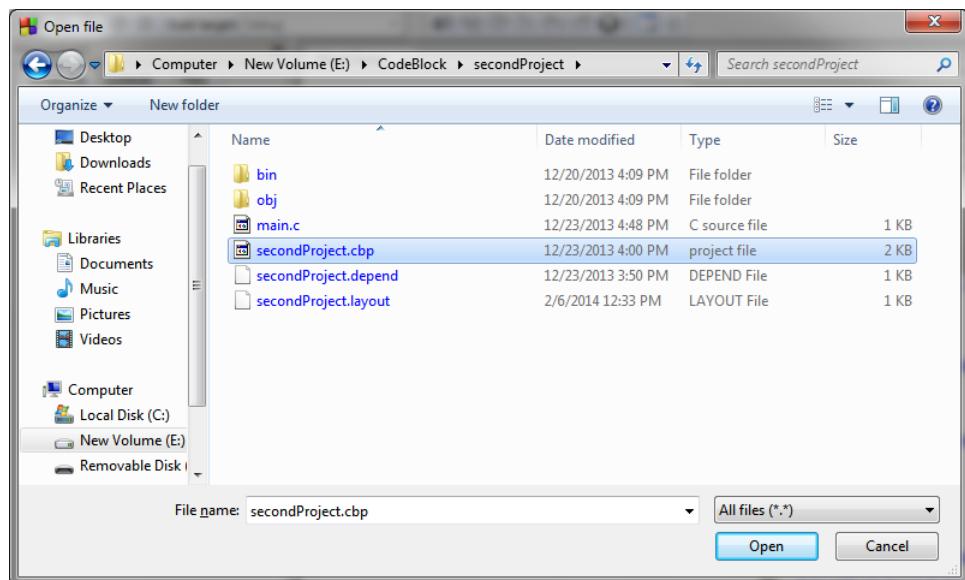


Figure 64: Select file with *.cbp* extension to open an existing project

4 Working with Code::Blocks and Simplecpp on Ubuntu

In this section we discuss writing and building three projects. First project (hello world.c) is simple program which displays *hello world* on output. The other 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. The third project (3poly.cpp) uses *simplecpp package* and draws three polygons on output. We have to install some packages, include few libgraph libraries during building the project with *graphic.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 system³. 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 65: Code::Block in Ubuntu Software Center

```
ada@ada-desktop:~$ sudo apt-get install codeblocks
[sudo] password for ada: ①
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  codeblocks-common libcodeblocks0 ②
Suggested packages:
  libwxgtk2.8-dev wx-common codeblocks-contrib ③
The following NEW packages will be installed:
  codeblocks codeblocks-common libcodeblocks0 ④
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/5,623 kB of archives.
After this operation, 16.3 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

Figure 66: Installing Code::Block using command line

³Check 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 65 shows Code::Block along with its logo in Ubuntu Software Center, while figure 66 shows installation of Code::Block using command line. In figure 66 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. *libuxgk2.8⁴* package as mentioned above is already installed in your system. *libuxgk2.8-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
  libaal libaal-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 footnote⁵) 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

4.3 Installation of Simplecpp

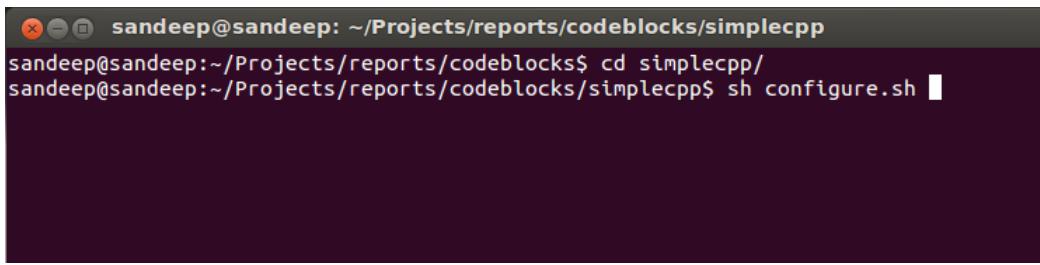
Download the simplecpp’s tar package from <http://www.cse.iitb.ac.in/~cs101/Project/simplecpp/> Ubuntu/**simplecpp.tar**. Untar the tar file and change to the simplecpp directory and run the command ‘sh configure.sh’. Shown in figure 67.

```
1 $ cd DIRECTORY_WHERE_Simplecpp.tar.IS.DOWNLOADED
2 $ tar -xvf Simplecpp.tar
3 $ cd simplecpp/
4 $ sh configure.sh
```

Listing 5: Installing simplecpp package using command line

⁴Details about above mentioned packages can be found at <http://packages.ubuntu.com/precise/allpackages>.

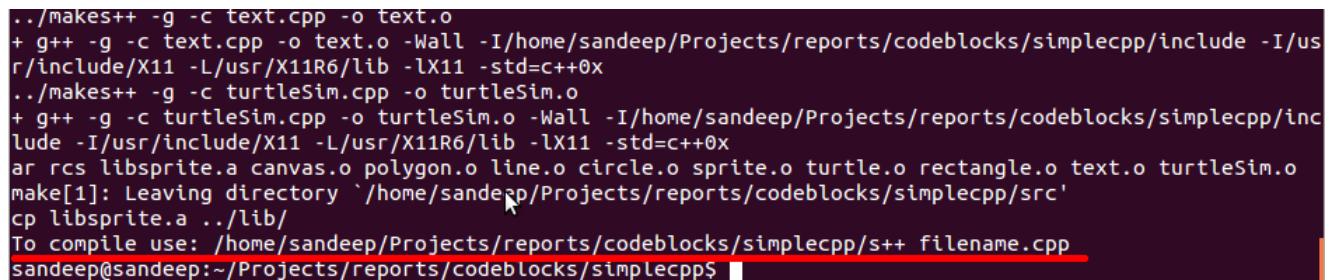
⁵<http://download.savannah.gnu.org/releases/libgraph/libgraph-1.0.2.tar.gz>



```
sandeep@sandeep: ~/Projects/reports/codeblocks/simplecpp
sandeep@sandeep:~/Projects/reports/codeblocks$ cd simplecpp/
sandeep@sandeep:~/Projects/reports/codeblocks/simplecpp$ sh configure.sh
```

Figure 67: Installing simplecpp

The last line in installation process mentions *To compile use: /home/sandeep/Projects/reports/codeblocks/simplecpp/s++ filename.cpp*. The path to simplecpp/s++ will vary from user to user as per user's machine, this path is location of simplecpp/s++ on the machine. Shown in figure 68.



```
.../makes++ -g -c text.cpp -o text.o
+ g++ -g -c text.cpp -o text.o -Wall -I/home/sandeep/Projects/reports/codeblocks/simplecpp/include -I/usr/include/X11 -L/usr/X11R6/lib -lX11 -std=c++0x
.../makes++ -g -c turtleSim.cpp -o turtleSim.o
+ g++ -g -c turtleSim.cpp -o turtleSim.o -Wall -I/home/sandeep/Projects/reports/codeblocks/simplecpp/include -I/usr/include/X11 -L/usr/X11R6/lib -lX11 -std=c++0x
ar rcs libsprite.a canvas.o polygon.o line.o circle.o sprite.o turtle.o rectangle.o text.o turtleSim.o
make[1]: Leaving directory '/home/sandeep/Projects/reports/codeblocks/simplecpp/src'
cp libsprite.a ..//lib/
To compile use: /home/sandeep/Projects/reports/codeblocks/simplecpp/s++ filename.cpp
sandeep@sandeep:~/Projects/reports/codeblocks/simplecpp$
```

Figure 68: location of simplecpp/s++

4.4 Integrating Simplecpp with Code::Blocks IDE

To create a project using simplecpp package, we need to make some changes in the compiler and debugger settings. The steps to make the required changes are as given below.

1. Open Code::Blocks and click on ‘Settings’ → ‘Compiler and Debugger...’.
2. A window labelled Compiler and debugger settings will open as shown in Figure 69.

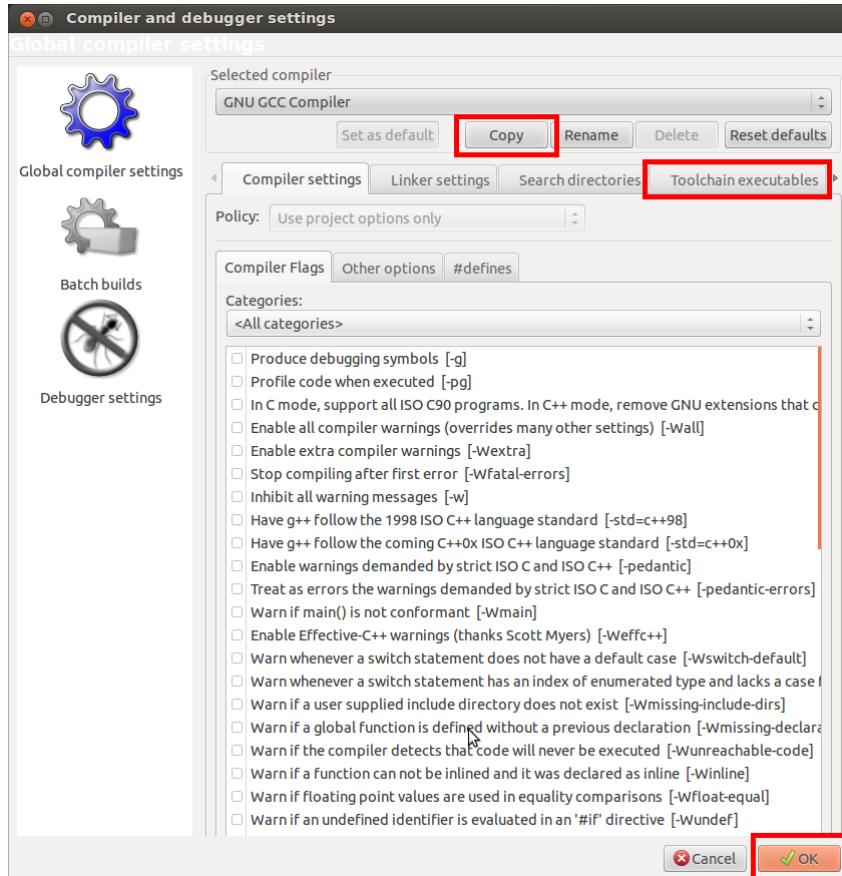


Figure 69: Compiler and debugger settings

3. By default the selected compiler is ‘GNU GCC Compiler’. Click ‘Copy’ highlighted with red box in figure 69.
4. A window as shown in figure 70 will open. Write ‘simplecpp’ and click ‘Ok’.

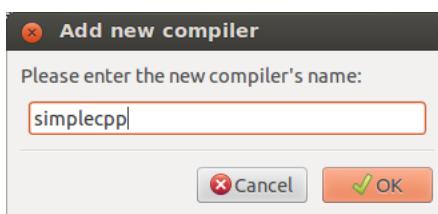


Figure 70: Copying the compiler

5. A message stating 'change the appropriate setting in Toolchain executable' is displayed, which is shown in figure 71.

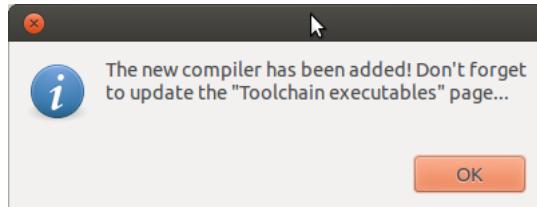


Figure 71: Change Toolchain Executable

6. Click on 'Toolchain executables' tab highlighted with red box in figure 69.
7. Change the compiler path of Code::Blocks to that of simplecpp. The steps are given below.
8. Click all '...' buttons highlighted with red box in figure 72) on by one. Change all the settings as shown in figure.
9. When '...' is clicked, browse to the '/simplecpp' directory as shown in figure 73. 's++' and 'makes++' are available in '/simplecpp/' folder. 'libsprite.a' is available in '/simplecpp/lib/' folder.

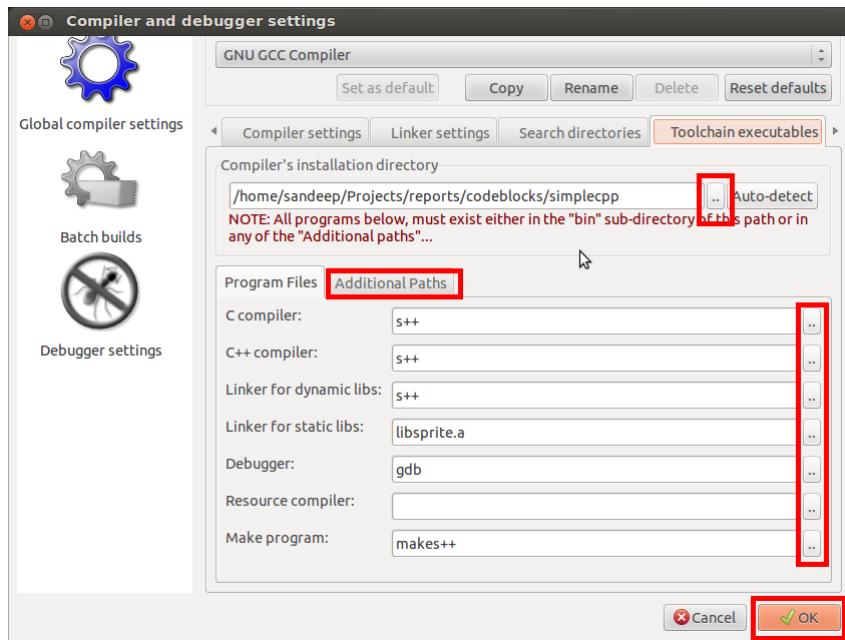


Figure 72: Changes to be made in Toolchain Executables

10. To summarize, the list of all the fields that need to be changed is given below
 - (a) **Compiler's installation directory:** /simplecpp/
 - (b) **C compiler:** /simplecpp/s++/
 - (c) **C++ compiler:** /simplecpp/s++/
 - (d) **Linker for Dynamic libs:** /simplecpp/s++/
 - (e) **Linker for static libs:** /simplecpp/s++/libsprite.a
 - (f) **Make program:** /simplecpp/makes++/

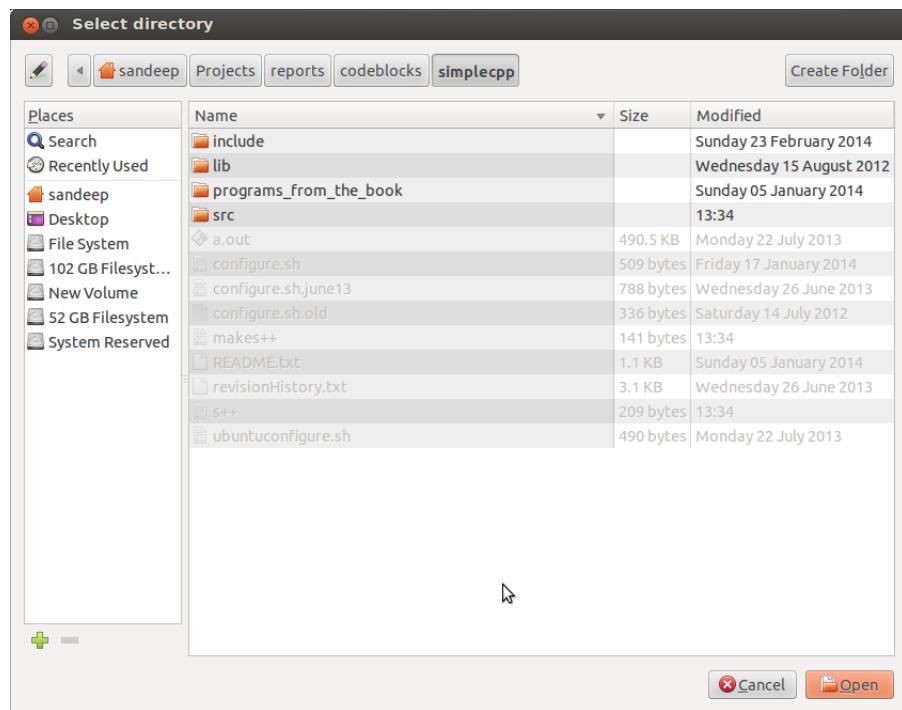


Figure 73: Simplecpp's directory

11. When all the required fields are changed, click on ‘Additional Paths’ shown in red block in figure 72.
12. In Additional Paths click ‘Add’ as shown in Figure 74. Browse to /simplecpp/ directory and click on Ok. User will return on Compiler and debugger setting window. Click Ok again to exit settings and return to editor.

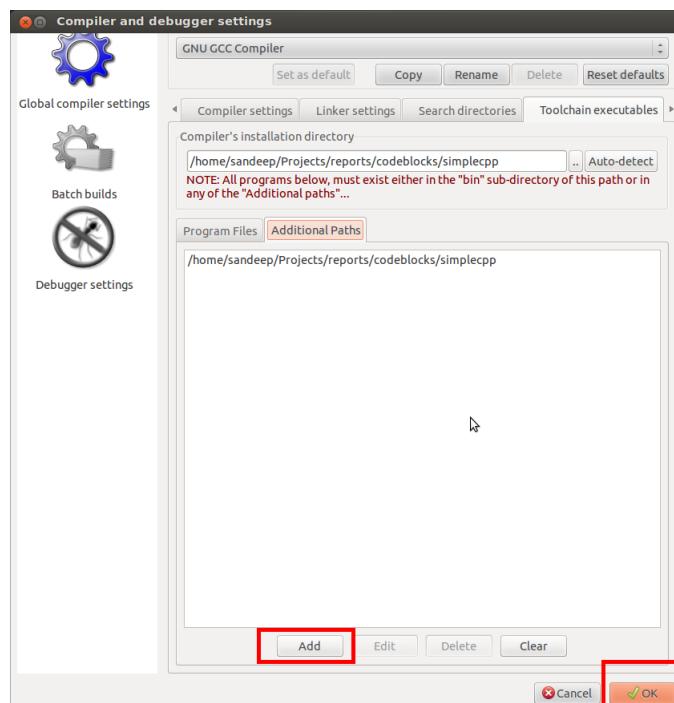


Figure 74: Additional Paths settings

4.5 Writing a new c/c++ program

1. Click on *New file* button. The ‘New from template’ window opens as shown in figure 75. Select the type of project you want to program in. When the type of project is selected the *Go* button gets highlighted (top right corner). Select ‘Console application’ and click on ‘Go’.

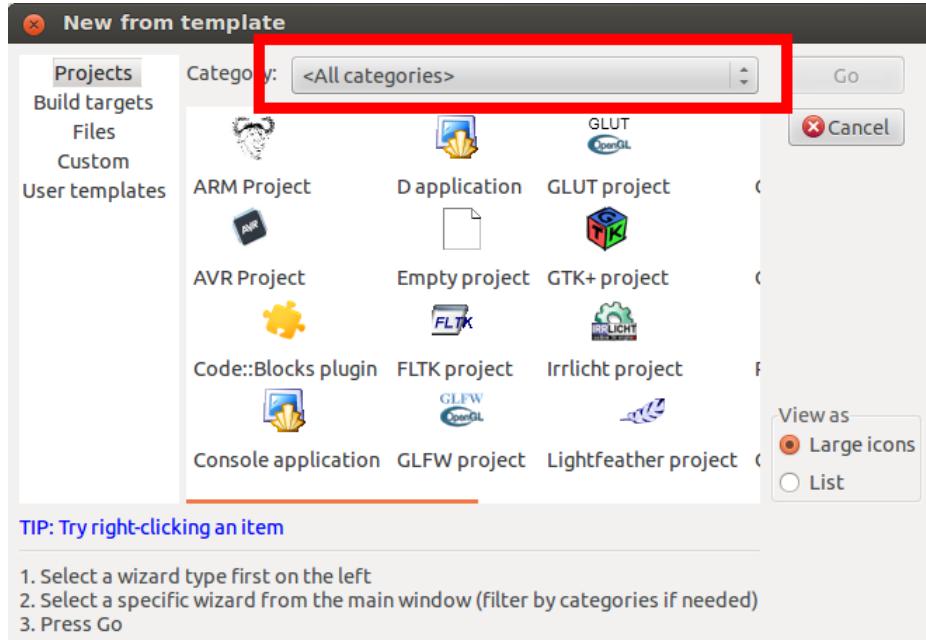


Figure 75: Starting a new project

2. When *Go* button is clicked, a new window opens as shown in figure 76. This window enables the user to select the required language to proceed. Select the language and click on *next*.

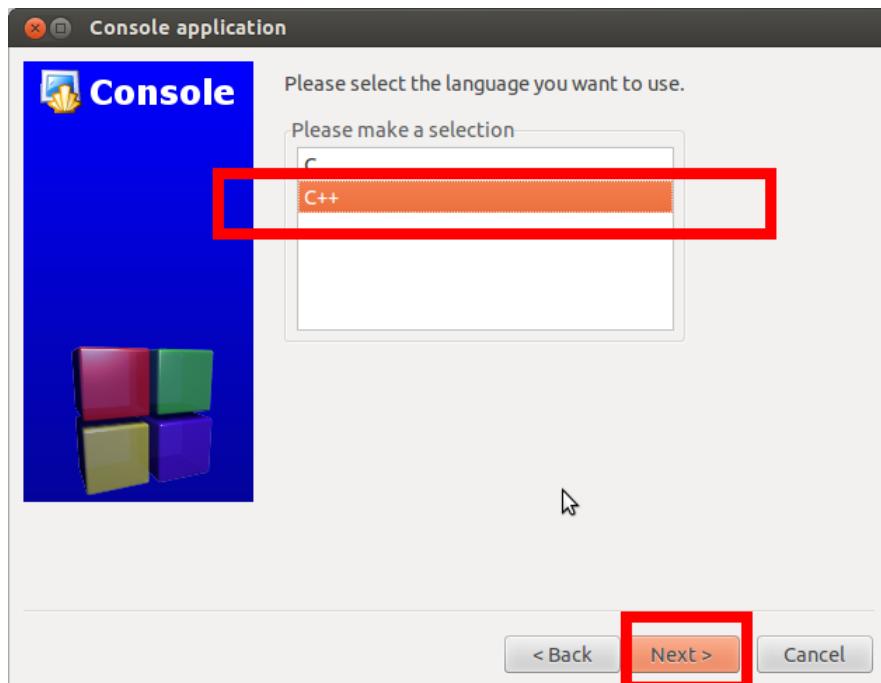


Figure 76: Selecting the language for project

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 77. After filling in the details click ‘Next’.

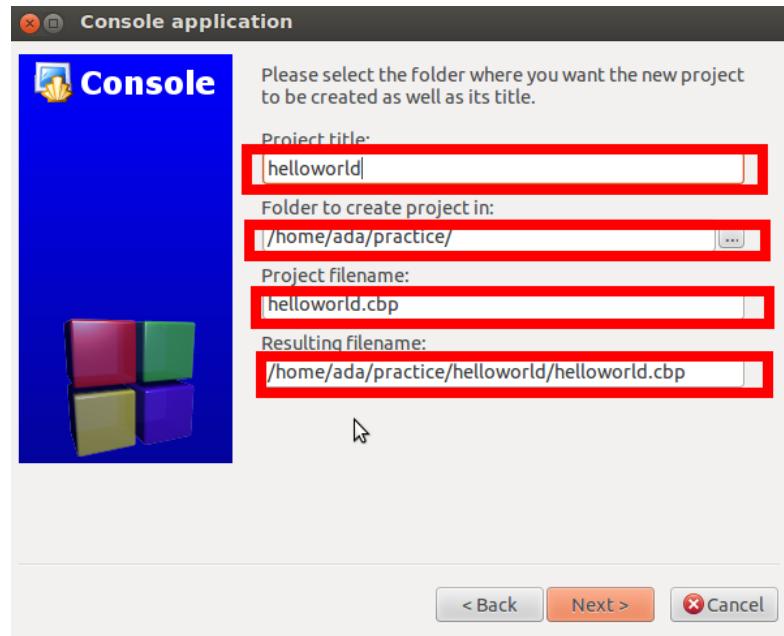


Figure 77: Title for Project

4. Next window is to select the compiler. By default ‘GNU GCC Compiler’ is selected. In this window, the user can change the compiler to be used as per the use and requirements.

Non-graphic and Programs including graphics.h header file: Select ‘GNU GCC Compiler’ (figure 78)

Simplecpp programs: Select ‘Simplecpp’ (figure 79) . Click on Finish.

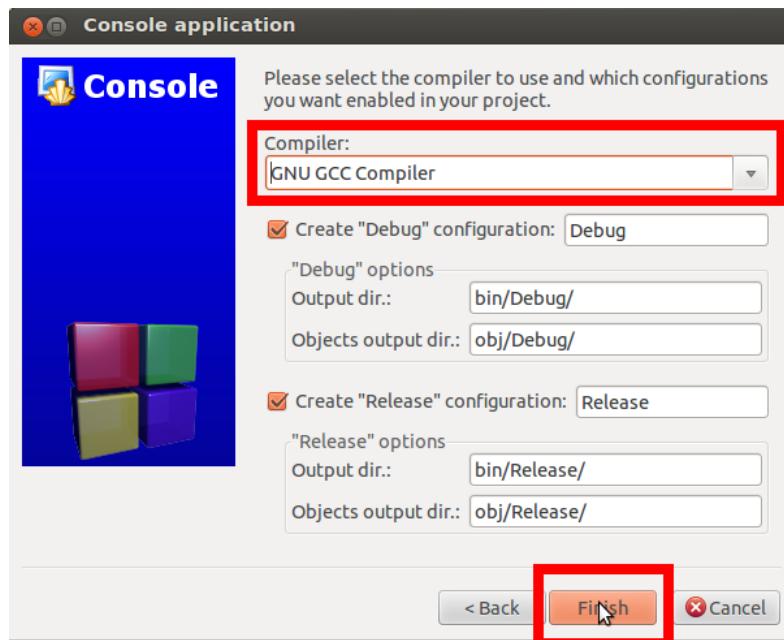


Figure 78: Selecting Compiler to Compile the Program

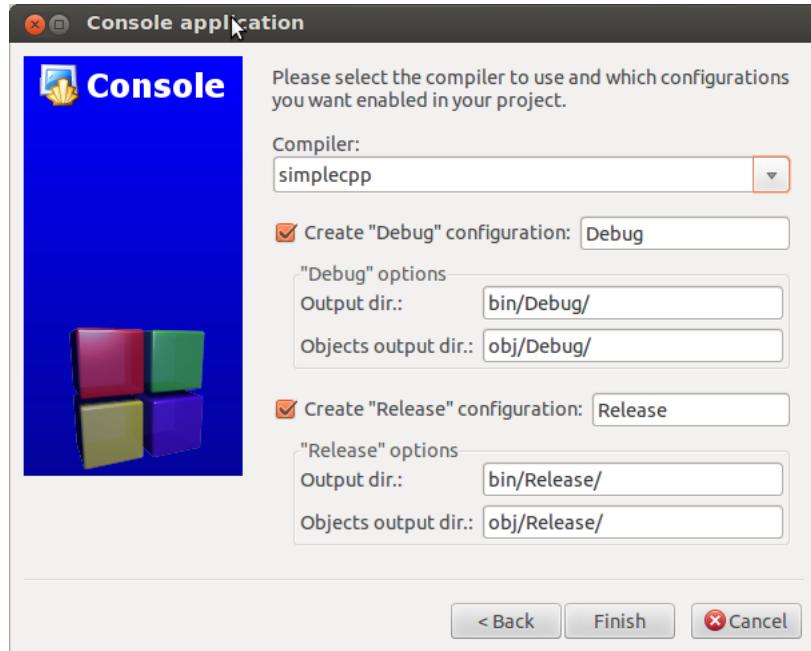


Figure 79: Selecting simplecpp for projects including simplecpp

5. Now, the project node opens in manager window as shown in figure 80. Project node can be expanded to see the main.c file.

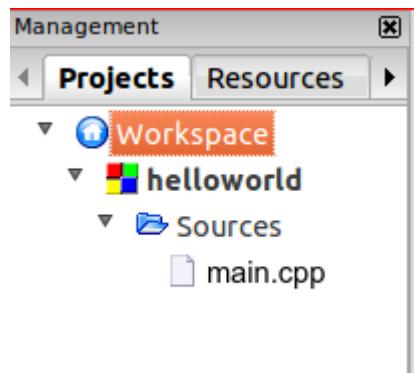
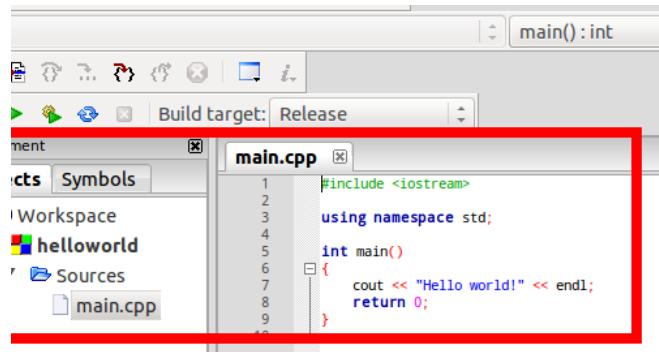


Figure 80: Project Node when Expanded

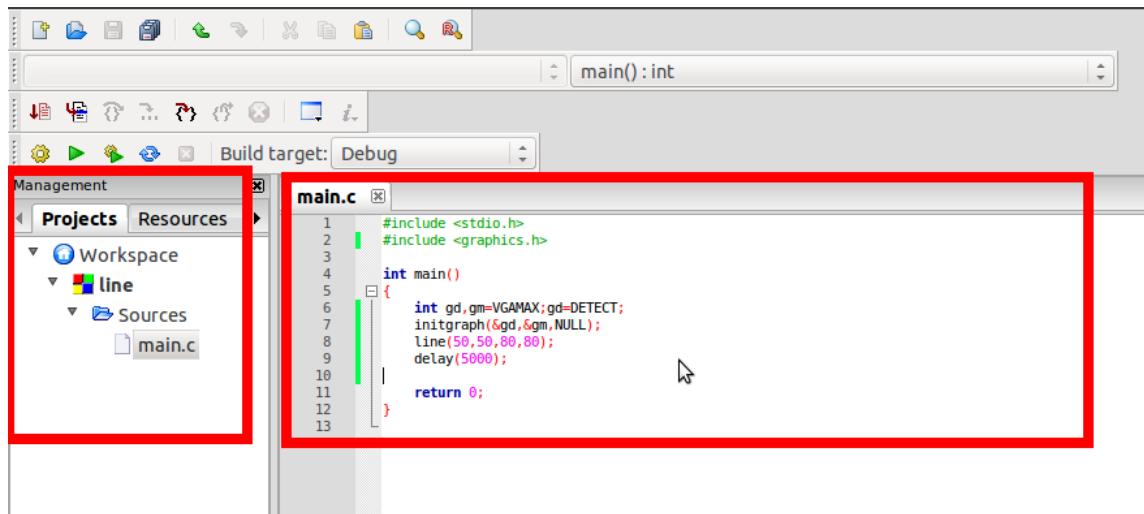
When main.c file is clicked, it opens in editor as shown in figure 81 for hello world project.



```
main() : int
Build target: Release
main.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Hello world!" << endl;
7     return 0;
8 }
```

Figure 81: Project Node when Expanded for helloworld.c (with code shown in editor)

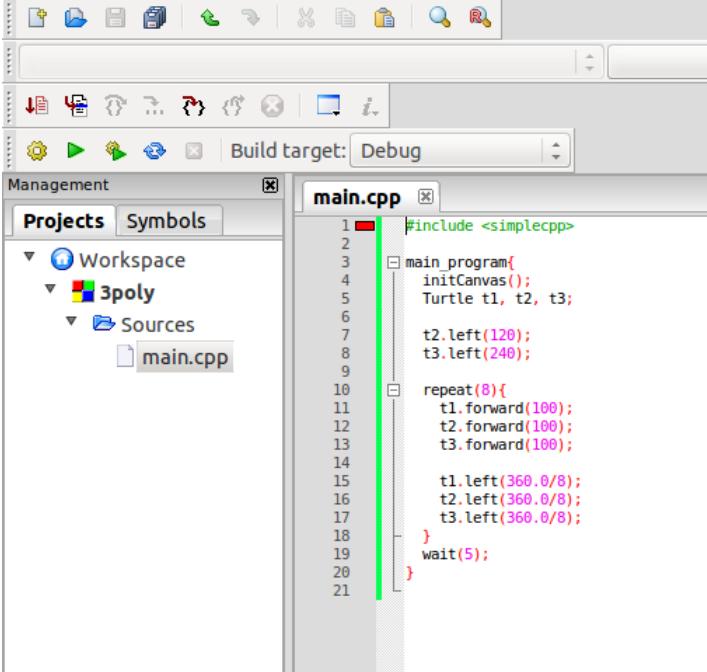
6. Code used for *graphics* program (line) is shown in figure 82.



```
main() : int
Build target: Debug
Management
Projects Resources
Workspace
  line
    Sources
      main.c
main.c
1 #include <stdio.h>
2 #include <graphics.h>
3
4 int main()
5 {
6     int gd,gm=VGAMAX;gd=DETECT;
7     initgraph(&gd,&gm,NULL);
8     line(50,50,80,80);
9     delay(5000);
10
11
12
13 }
```

Figure 82: Project node when expanded for line.c (with code shown in editor)

7. Code used for *simplecpp* program (3poly) is shown in figure 83. Every project involving simplecpp has to include simplecpp package as seen in first line of the code.



```
#include <simplecpp>
main_program{
    initCanvas();
    Turtle t1, t2, t3;
    t2.left(120);
    t3.left(240);
repeat(8){
    t1.forward(100);
    t2.forward(100);
    t3.forward(100);
    t1.left(360.0/8);
    t2.left(360.0/8);
    t3.left(360.0/8);
}
wait(5);
}
```

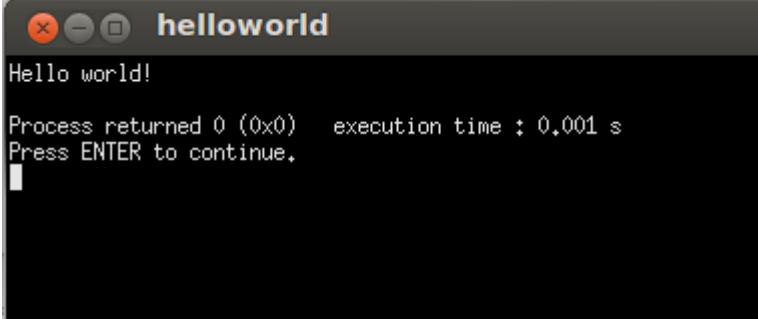
Figure 83: Project node when expanded for 3poly.cpp (with code shown in editor)

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

4.6 Building the Project

4.6.1 Non-Graphics Project

After the code is written, project needs to be built. Click on ‘build and run’ from compiler bar. Output is as shown in figure 84.



```
Hello world!
Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

Figure 84: Output for helloworld.c

4.6.2 Graphics Project using graphics.h

To build a program involving *graphics.h*, it is recommended to enable full logging in Code::Block. Full logging also helps in better debugging. Steps to enable full logging in Code::Block is given in appendix ??

Now the project using *graphics.h* (line) has to be linked with libraries copied in */usr/lib* while installing **libgraph** package. 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 85 pops up. Option to change the compiler selected for the project is also available in this window.

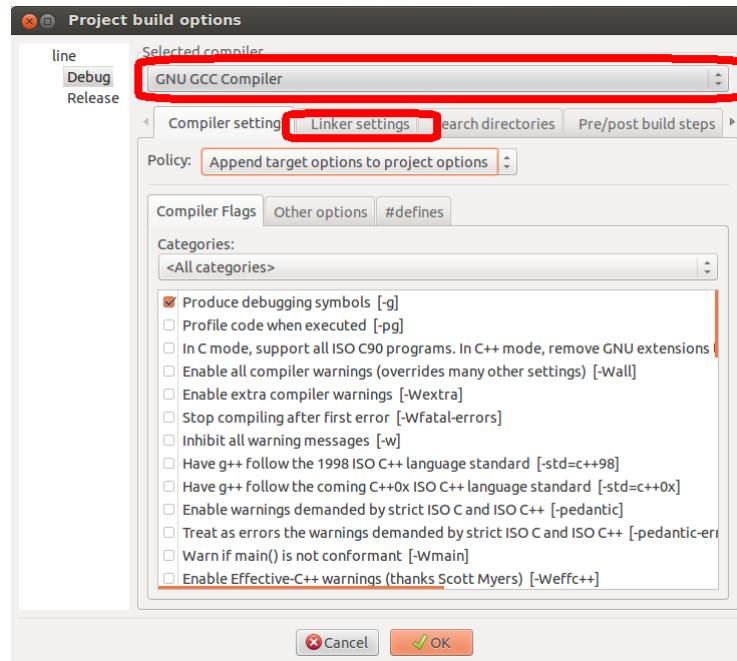


Figure 85: Project build options

2. In ‘Project build options’ window, click on *linker settings* tab. This tab is shown in figure 86.

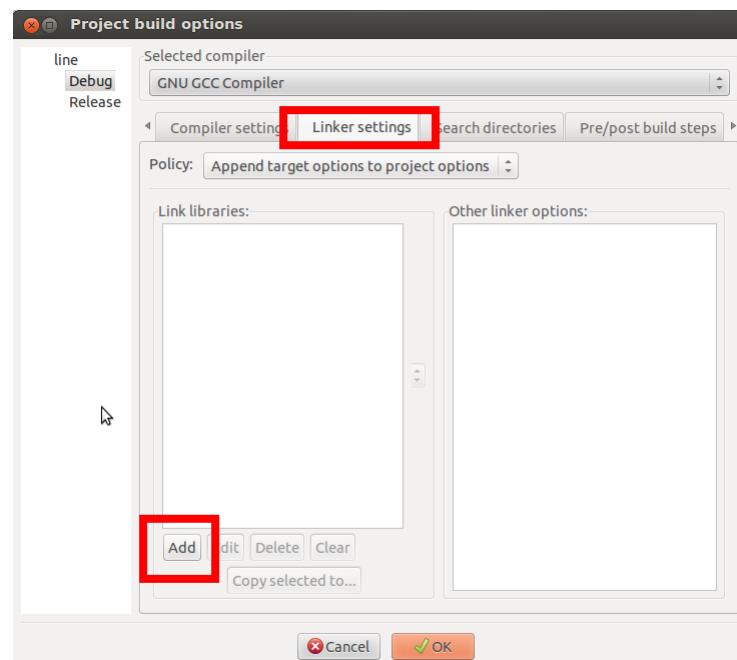


Figure 86: Linker settings (Add Libraries)

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 87.

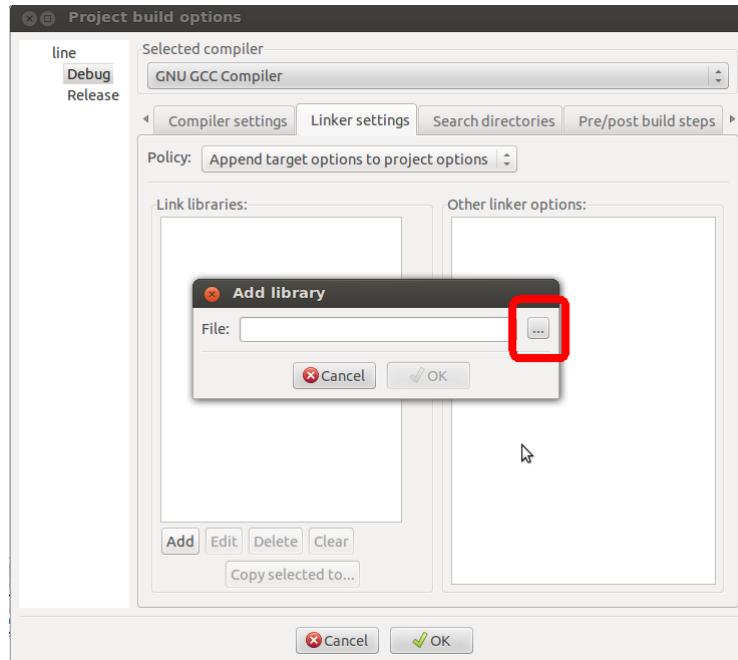


Figure 87: Interface for adding libraries

4. Click on dotted button to right of box. A new window opens as shown in figure 88. 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.a file. Files to be selected are shown in the figure 88.

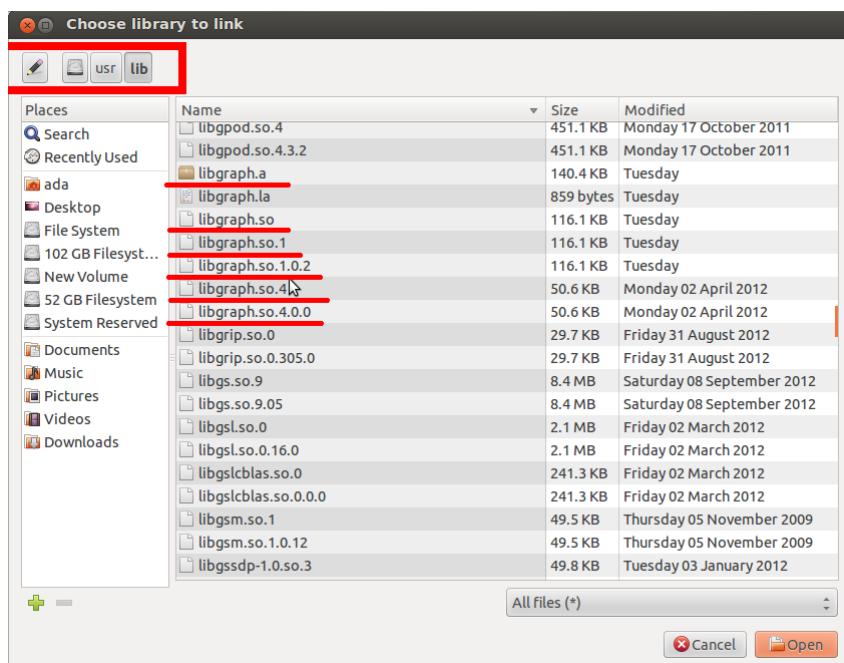


Figure 88: Files/libraries to be added for graphic projects

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 89. Click on No.

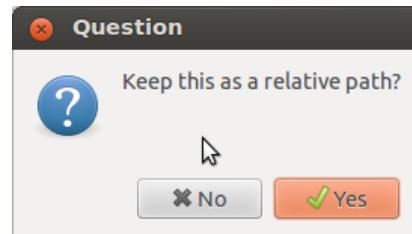


Figure 89: Relative or absolute path for files/libraries

7. The libraries will be linked using absolute path as shown in figure 90. Click on ‘Ok’.

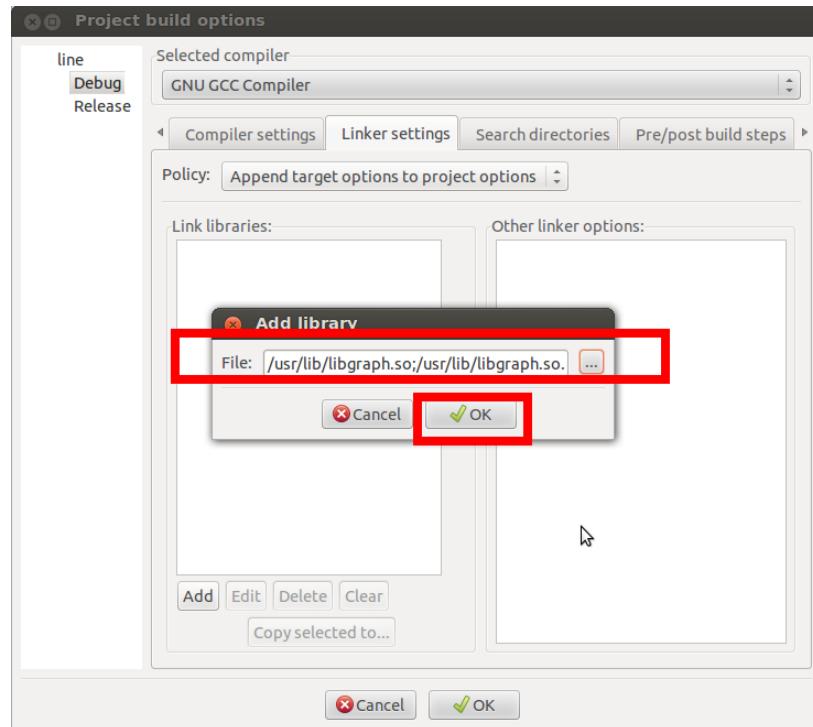


Figure 90: libraries selected

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 91 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

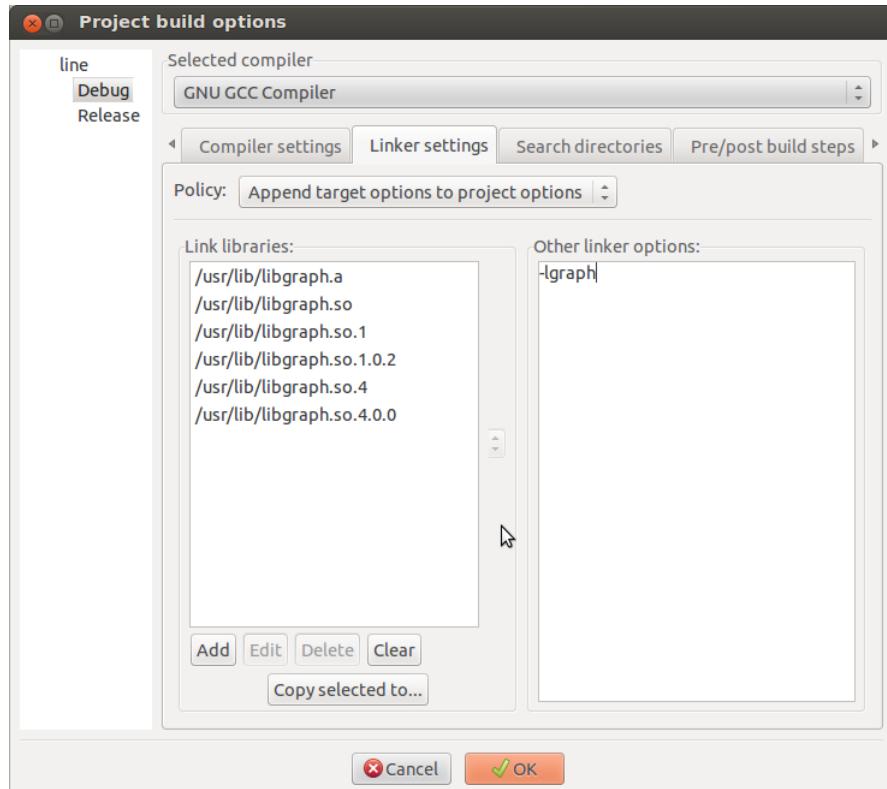


Figure 91: Libraries added to project

9. Now click the *build and run* button from compiler bar and output will be displayed as shown in figure 92.

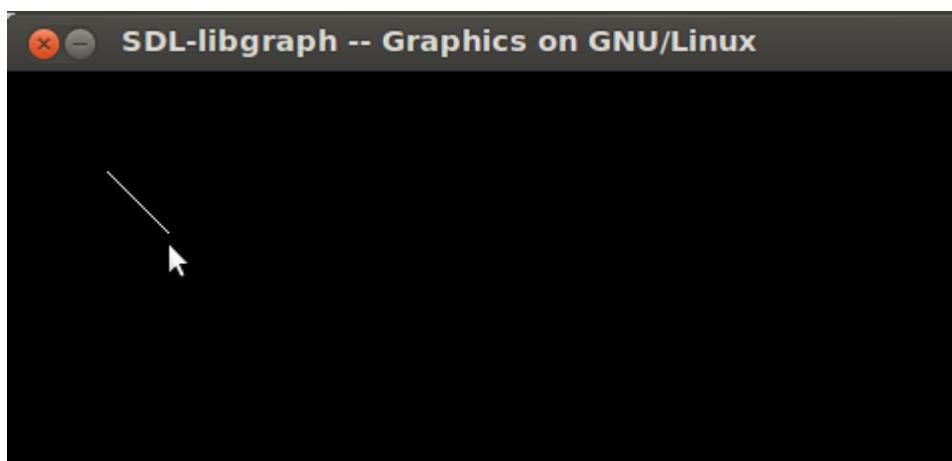


Figure 92: Output for line.c

4.6.3 Graphics Project using simplecpp

To build a program involving *simplecpp*, it is recommended to enable full logging in Code::Blocks. Full logging also helps in better debugging. Steps to enable full logging in Code::Blocks is given in appendix ??

Click the *build and run* button from compiler bar and output will be displayed as shown in figure 93.

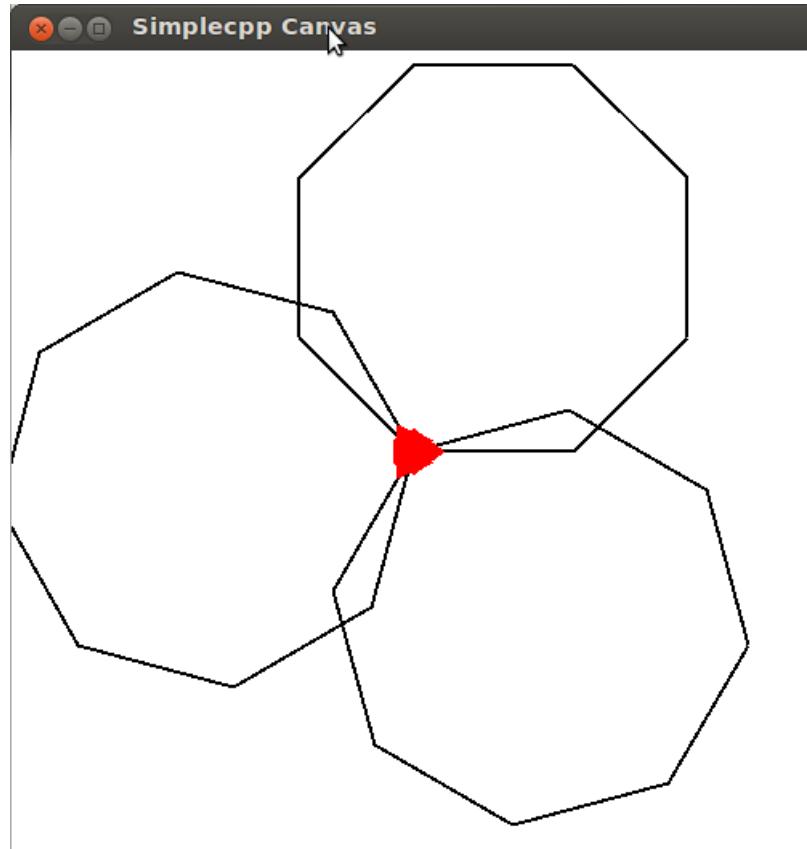


Figure 93: Output for 3poly.cpp

4.7 Opening Existing Program/Project

Click File and select ‘Open’. Browse to desired directory and select the file with .cbp extension as shown in figure 94 and click on Open.

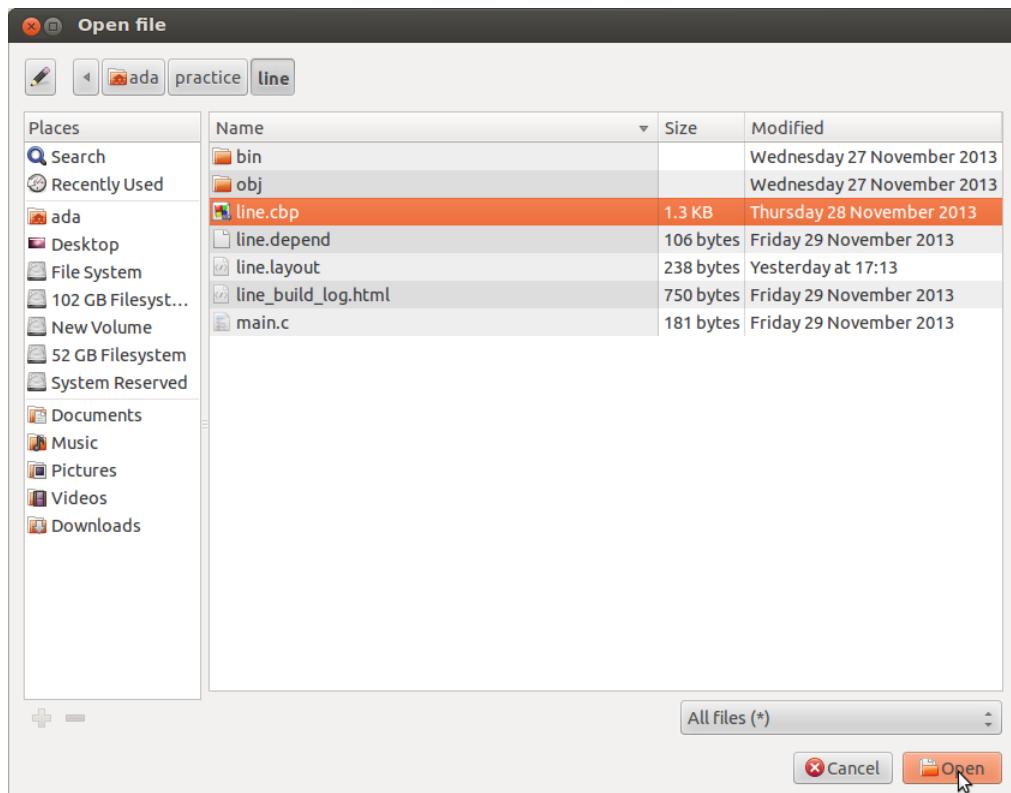


Figure 94: Select file with .cbp extension to open an existing project

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>.