Code::Block manual

for CS101x course

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

April 9, 2014
## Contents

1 Introduction
   1.1 Code::Blocks .................................................. 1
   1.2 Simplecpp .................................................... 1

2 Code::Block IDE .............................................. 1

3 Working with Code::Block on Windows ................. 6
   3.1 Installation of Code::Blocks ................................ 6
   3.2 Installation of Simplecpp .................................... 12
   3.3 Writing a new C/C++ Program ................................ 14
      3.3.1 Non-Graphic Project .................................... 14
      3.3.2 Graphic Project using graphics.h ...................... 22
      3.3.3 Graphic Project using simplecpp ...................... 26
   3.4 Building the Project ......................................... 33
   3.5 Opening existing program/project ...................... 34

4 Working with Code::Block on Ubuntu ................ 35
   4.1 Installation of Code::Blocks ................................ 35
   4.2 Installation of packages for graphics.h header file .... 36
   4.3 Installation of Simplecpp .................................... 36
   4.4 Writing a new c/c++ program ............................... 41
   4.5 Building the Project ......................................... 45
      4.5.1 Non-Graphics Project .................................... 45
      4.5.2 Graphics Project using graphics.h ...................... 45
      4.5.3 Graphics Project using simplecpp ...................... 50
   4.6 Opening Existing Program/Project ...................... 51

A Code used for Windows ............................... 53
   A.1 helloworld.c ................................................. 53
   A.2 line.c ....................................................... 53
   A.3 3poly.cpp .................................................. 53

B Code used for Ubuntu .................................. 54
   B.1 helloworld.c ................................................. 54
   B.2 line.c ....................................................... 54
   B.3 3poly.cpp .................................................. 54

C Enabling full logging in Code::Block ............... 55
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
10. Logs ..................................................................................... 5
11. File association window ....................................................... 6
12. Click on “Run” ....................................................................... 6
14. License Agreement .............................................................. 7
15. Information window ............................................................ 8
17. Providing folder for CodeBlocks-EP shortcuts ....................... 9
20. Progress bar to show progress of installation ..................... 10
21. Completing the installation process .................................... 11
22. Code::Block IDE in Windows 7 ........................................... 11
23. Installation of simplecpp on windows ................................. 12
24. Setup - Simplecpp ............................................................... 12
26. Install ................................................................................... 13
27. Finish Simplecpp installation ............................................. 14
28. New form template .............................................................. 14
29. New console application wizard ......................................... 15
30. Selecting language for the project ..................................... 15
31. Providing title and folder for the project ......................... 16
32. Selecting compiler for the project ..................................... 16
33. Project node with no files .................................................. 17
34. Adding file to the project .................................................... 17
35. Selecting type of file to be added in the project ................. 18
36. Select checkbox to skip this window next time ................. 18
37. Select the language of the file added ................................. 19
38. Details of file to be added .................................................. 19
39. Selecting the location and file name to be added ............... 20
40. Finalize details of file to be added ..................................... 20
41. Project node with ‘+’ sign indicating it can be expanded .... 21
42. Project node expanded ....................................................... 21
43. Writing code in editor ........................................................ 21
44. New form template ............................................................. 22
45. New console application wizard ....................................... 22
46. Selecting language for the project ................................... 23
47. WinBGIm project ............................................................... 23
48. Providing title and folder for the project ......................... 24
49. Selecting compiler for the project ................................... 24
50. Finalize details of file to be added (for line project) ........... 25
51. Project node with ‘+’ sign indicating it can be expanded .... 25
52. Writing program in editor (line project) ............................ 26
53. New from template ............................................................. 26
54 Simplecpp project
55 Providing title and folder for the project
56 Selecting compiler for the project
57 Empty project node in Management window
58 Adding files to empty project node
59 Selecting c/c++ source for project
60 Selecting language for the file to be added
61 Giving location and name of the file to be added
62 Finalize details of file to be added
63 Project node with ‘+’ sign indicating it can be expanded
64 opening file in editor
65 coding
66 Output of hello world project
67 Output of line project
68 Output of 3poly project
69 Selecting Open under file in Menu Bar
70 Select file with .cbp extension to open an existing project
71 Code::Block in Ubuntu Software Center
72 Installing Code::Block using command line
73 Installing simplecpp
74 location of simplecpp/s++
75 Copying the compiler
76 Compiler and debugger settings
77 Changes to be made in Toolchain Executables
78 Simplecpp's directory
79 Additional Paths settings
80 Starting a new project
81 Selecting the language for project
82 Title for Project
83 Selecting Compiler to Compile the Program
84 Selecting simplecpp for projects including simplecpp
85 Project Node when Expanded
86 Project Node when Expanded for helloworld.c (with code shown in editor)
87 Project node when expanded for line.c (with code shown in editor)
88 Project node when expanded for 3poly.cpp (with code shown in editor)
89 Output for helloworld.c
90 Project build options
91 Linker settings (Add Libraries)
92 Interface for adding libraries
93 Files/libraries to be added for graphic projects
94 Relative or absolute path for files/libraries
95 libraries selected
96 Libraries added to project
97 Output for line.c
98 Output for 3poly.cpp
99 Select file with .cbp extension to open an existing project
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 12.04[1] 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

![Code::Block IDE](image)

Figure 1: Code::Block IDE

---

[1] Ubuntu 12.04 with intel \(\text{Core}\ TM\text{-}i3\text{-}2120\) CPU \(\@\) 3.30GHzx4 processor 4 GB RAM and 32-bit architecture and 64 bit architecture.

[2] Windows 7 with intel \(\text{Core}\ TM\text{-}i3\text{-}2120\) 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):

   ![Menu Bar](image2)

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

   ![Main Toolbar](image3)

   **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](image)

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):

   - **Build:** For building the current/selected project.
   - **Run:** For running the current/selected project.
   - **Build and run:** For building and running the current/selected project.
   - **Rebuild:** For rebuilding the current/selected project.
   - **Abort:** For aborting the build process for the current/selected project.
   - **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.

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

7. Figure 7 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.

8. Figure 8 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.
Recent projects
- /home/ada/practice/line/line.cbp
- /home/ada/practice/helloworld/helloworld.cbp
- /home/ada/practice/first_project/first_project.cbp
- /home/ada/practice/heart/heart.cbp
- /home/ada/practice/Second_Project/Second

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

9. **Logs:**
Log window is shown in figure 10. 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 three projects. First project (hello world.c) is a simple program which displays \textit{hello world} on output. The second project (line.c) uses \textit{graphics.h} header file and displays a line. The third project (3poly.cpp) uses \textit{simplecpp} package and draws three polygons on output. \textbf{For Windows (Windows 7) we will be using ‘CodeBlocks-EP’.} CodeBlocks-EP comes with integrated WinBGIm (stands for Windows Borland Graphics Interface). In Windows, WinBGIm is required to run the programs with \textit{graphics.h} header file. 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 \textit{c/c++ file types}” or 4th option “Yes, associate Code::Blocks with every supported type(including project files from other IDEs)” and click ‘OK’.

![File association window](image)

**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/](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’.

![Run button](image)

**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](image)

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

![Figure 14: License Agreement](image)
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](image)

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](image)
Figure 17 shows the location provided for CodeBlocks-EP shortcuts in Start Menu Bar. Click on “Next”.

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 is 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](image)

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](image)
3.2 Installation of Simplecpp

The steps to be followed to install simplecpp are as listed below. Download the tar.gz file from http://www.it.iitb.ac.in/frg/wiki/images/f/f3/Simplecpp-Windows.tar.gz. Untar the file and go into the simplecpp's folder. The steps have been tested on Windows 7.

1. Close all the running instance of Codeblock-EP

2. Double click on .exe simplecpp file. A window is displayed as shown in figure 23. Click ‘Run’.

![Figure 23: Installation of simplecpp on windows](image)

3. A new window opens which is labelled ‘Setup - Simplecpp’ as shown in figure 24. Click on ‘Next’.

![Figure 24: Setup - Simplecpp](image)
4. A new window is displayed to select path of installation as shown in figure 25. By default it is 'C:\ProgramFiles\Codeblocks-EP'. If you want to change the installation path, click 'Browse', navigate to the directory, and click 'Ok', else leave it unchanged. Click 'Next'.

![Figure 25: Selecting CodeBlocks-EP’s installation folder](image)

5. A new window stating that simplecpp is ready to install is displayed. This is shown in figure 26. Click ‘Install’.

![Figure 26: Install](image)
6. Finally, click ‘Finish’ to complete the installation as shown in figure 27.

![Finish Simplecpp installation](image)

**Figure 27: Finish Simplecpp installation**

### 3.3 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.

#### 3.3.1 Non-Graphic Project

1. Click on *New file* button. The ‘New from template’ window as shown in figure 28 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*.

![New form template](image)

**Figure 28: New form template**
2. When Go button is clicked a new window opens as shown in figure 29. Select checkbox “Skip this page next time” so that the page is not displayed again. Click ‘Next’.

Figure 29: New console application wizard

3. Next window enables user to select the language to be used for project as shown in Figure 30. Select C/C++. Here, ‘C’ is selected for helloworld project. Click ‘Next’.

Figure 30: Selecting language for the project
4. Next window enables user to provide title for the project and the folder where user wishes to create the project in. This is shown in figure 31. After filling in the details click on Next.

Figure 31: Providing title and folder for the project

5. Next window is used to select the compiler as shown in figure 32. By default ‘GNU GCC Compiler’ is selected. Click on Finish.

Figure 32: Selecting compiler for the project
6. The project node opens in manager window as shown in figure 33. The project node is empty and we have to add files to the project.

Figure 33: 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 34.

Figure 34: Adding file to the project
8. New from template opens as shown in figure 35. For our example select ‘C/C++ source’ and click on Go.

![Figure 35: Selecting type of file to be added in the project](image)

9. A new window is displayed as shown in figure 36. Select the checkbox ‘Skip this page next time’ so that it is not displayed again.

![Figure 36: Select checkbox to skip this window next time](image)
10. Select the preferred language as shown in figure 37. We have selected ‘C’ for our helloworld example.

![Figure 37: Select the language of the file added](image)

11. Figure 38 shows the window that opens to add the file. Click on ‘...’ beside ‘Filename with full path’.

![Figure 38: Details of file to be added](image)
A window as shown in figure 39 opens. Select the folder of the project and enter file name to be added. Click on ‘Save’.

![Figure 39: Selecting the location and file name to be added](image)

When ‘Save’ button is clicked the window in figure 38 opens again with full path and name of the file. Select Debug and Release. Shown in figure 40. Click on Finish.

![Figure 40: Finalize details of file to be added](image)
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 41 and figure 42.

Figure 41: Project node with ‘+’ sign indicating it can be expanded

Figure 42: Project node expanded

13. When the hello.c file opens in editor, user can start coding. Code is shown in figure 43.

Figure 43: Writing code in editor
3.3.2 Graphic Project using graphics.h

1. Click on New file button. The ‘New form template’ window as shown in figure 28 opens. For graphics projects, select ‘WinBGIm project’. Go button gets highlighted (top right corner). Click on Go.

![New form template](image)

Figure 44: New form template

2. When Go button is clicked a new window opens as shown in Figure 45. 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.

![New console application wizard](image)

Figure 45: New console application wizard

3. Next window enables user to select the language to be used for project as shown in figure 46. For
the example hello world used in this manual select ‘C’ and click on Next.

Figure 46: 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 47. Select ‘Graphics only’ and click on Next.

Figure 47: 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 48. After filling in the details click on Next.

![Figure 48: Providing title and folder for the project](image)

6. Next window is used to select the compiler as shown in figure 49. By default ‘GNU GCC Compiler’ is selected. Click on Finish.

![Figure 49: Selecting compiler for the project](image)
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 35. 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 38. A window as shown in figure 39 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 50. Click on Finish.

![Image 1](image1.png)

Figure 50: Finalize details of file to be added (for line project)

8. Management window now shows project node which can be expanded (figure 51).

![Image 2](image2.png)

Figure 51: 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 52.

![Figure 52: Writing program in editor (line project)](image)

### 3.3.3 Graphic Project using simplecpp

1. Click on New file button. The ‘New form template’ window as shown in figure 53 opens. For graphics projects using simplecpp, select ‘Simplecpp project’. Go button gets highlighted (top right corner). Click on Go.

![Figure 53: New from template](image)
2. Next window asks the user to select type of project. The options are ‘Add Console’ and ‘Graphics only’ as shown in figure 54. Select ‘Graphics only’ and click on Next.

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 55. After filling in the details click on Next.

Figure 54: Simplecpp project

Figure 55: Providing title and folder for the project
4. Next window is used to select the compiler as shown in figure 56. By default ‘GNU GCC Compiler’ is selected. Click on Finish.

Figure 56: Selecting compiler for the project

5. The project node opens in manager window. The project node is empty as shown in figure 57. We have to add files to the project.

Figure 57: 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 58.

![Figure 58: Adding files to empty project node](image)

7. New from template opens as shown in figure 59. For our example select ‘C/C++ source’ and click on **Go**.

![Figure 59: Selecting c/c++ source for project](image)
8. Select the preferred language. For our example select ‘C++’ (figure 60).

![Figure 60: Selecting language for the file to be added](image)

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 38. A window as shown in figure 61 opens. Select the folder of the project and enter file name to be added. Click on ‘Save’.

![Figure 61: Giving location and name of the file to be added](image)
In next window select Debug and Release. Shown in figure 62. Click on Finish.

Figure 62: Finalize details of file to be added

9. Management window now shows project node which can be expanded (figure 63).

Figure 63: 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 64), user can start coding. Code is shown in figure 65.

Figure 64: opening file in editor

Figure 65: coding
3.4 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 66 for hello world project, figure 67 for line project and figure 68 for 3poly project.

Figure 66: Output of hello world project

Figure 67: Output of line project

Figure 68: Output of 3poly project
3.5 Opening existing program/project

Click on Open button. Browse to desired directory and open the file with .cbp extension as shown in figure 70.

Figure 69: Selecting Open under file in Menu Bar

Figure 70: 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 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 *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. 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. 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

![Code::Blocks IDE](image)

Figure 71: Code::Block in Ubuntu Software Center

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

Figure 72: 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 71 shows Code::Block along with its logo in Ubuntu Software Center, while figure 72 shows installation of Code::Block using command line. In figure 72, 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

libwxgtk2.8-dev
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.h. We will start with installing some packages from command line as given in listing 3.

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 untarrring tool must be installed on system. Open the command line and follow the instructions given in listing 4.

Listing 4: Installing libgraph package using command line

4.3 Installation of Simplecpp

Download the simplecpp’s tar package and untar it. The tar file can be downloaded from the link given in footnote 6. Change to the simplecpp directory and run the command ‘sh configure.sh’. Shown in figure 73.

Listing 5: Installing simplecpp package using command line

4Details about above mentioned packages can be found at http://packages.ubuntu.com/precise/allpackages.
5http://download.savannah.gnu.org/releases/libgraph/libgraph-1.0.2.tar.gz
6http://www.it.iitb.ac.in/frg/wiki/images/4/45/Simplecpp.tar
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 74.

For the project using simplecpp package some setting has to be changed. The steps to make the required changes are as given below.

1. Open Code::Blocks and click on ‘Setting’ and select ‘Complier and Debugger...’. A window labelled Compiler and debugger settings will open as shown in Figure 76. By default the selected compiler is ‘GNU GCC Compiler’. Click ‘Copy’ highlighted with red box in figure 76. A window as shown in figure 75 will open. Write ‘simplecpp’ and click ‘Ok’.

Figure 74: location of simplecpp/s++

Figure 75: Copying the compiler
2. A message stating 'change the appropriate setting in Toolchain executable' is displayed, which is shown in figure 76.

3. Click on ‘Toolchain executables’ tab highlighted with red box in figure 76.

![Figure 76: Compiler and debugger settings](image-url)
4. Change the compiler path of Code::Blocks to that of simplecpp. The steps are given below.

5. Click all ‘...’ buttons highlighted with red box in figure 77 on by one. Change all the settings as shown in figure. When ‘...’ is clicked, browse to the ‘/simplecpp’ directory. This is shown in figure 78. ‘s++’ and ‘makes++’ are available in ‘/simplecpp/’ folder. ‘libsprite.a’ is available in ‘/simplecpp/lib/’ folder.

6. 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++/

7. When all the required fields are changed, click on ‘Additional Paths’ shown in red block in figure 77.
8. In Additional Paths click on Add shown in red block in figure 79. 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.
4.4 Writing a new c/c++ program

1. Click on New file button. The ‘New from template’ window opens as shown in figure 80. 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 80: Starting a new project

2. When Go button is clicked, a new window opens as shown in figure 81. This window enables the user to select the required language to proceed. Select the language and click on next.

Figure 81: 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 82. After filling in the details click ‘Next’.

![Figure 82: Title for Project](image)

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 83).

**Simplecpp programs:** Select ‘Simplecpp’ (figure 84). Click on Finish.

![Figure 83: Selecting Compiler to Compile the Program](image)
Figure 84: Selecting simplecpp for projects including simplecpp

5. Now, the project node opens in manager window as shown in figure 85. Project node can be expanded to see the main.c file.

Figure 85: Project Node when Expanded
When main.c file is clicked, it opens in editor as shown in figure [86] for hello world project.

![Image of main.c file](image1)

Figure 86: Project Node when Expanded for helloworld.c (with code shown in editor)

6. Code used for *graphics* program (line) is shown in figure [87]

![Image of line.c file](image2)

Figure 87: Project node when expanded for line.c (with code shown in editor)
7. Code used for *simplecpp* program (3poly) is shown in figure 88. Every project involving simplecpp has to include simplecpp package as seen in first line of the code.

![Figure 88: Project node when expanded for 3poly.cpp (with code shown in editor)](image)

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

4.5 Building the Project

4.5.1 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 89.

![Figure 89: Output for helloworld.c](image)

4.5.2 Graphics Project using graphics.h

To build a program involving *graphics.h*, it is recommend to enable full logging in Code::Block. Full logging also helps in better debugging. Steps to enable full Full logging in Code::Block is given in appendix C.
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 90 pops up. Option to change the compiler selected for the project is also available in this window.

   ![Figure 90: Project build options](image)

2. In ‘Project build options’ window, click on *linker settings* tab. This tab is shown in figure 91.

   ![Figure 91: Linker settings (Add Libraries)](image)
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 92.

![Figure 92: Interface for adding libraries](image)

4. Click on dotted button to right of box. A new window opens as shown in figure 93. 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 93.

![Figure 93: Files/libraries to be added for graphic projects](image)
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 94. Click on No.

![Figure 94: Relative or absolute path for files/libraries](image)

7. The libraries will be linked using absolute path as shown in figure 95. Click on ‘Ok’.

![Figure 95: libraries selected](image)
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 [96] 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 96: Libraries added to project](image)

9. Now click the *build and run* button from compiler bar and output will be displayed as shown in figure [97].

![Figure 97: Output for line.c](image)
4.5.3 Graphics Project using simplecpp

To build a program involving *simplecpp*, it is recommend to enable full logging in Code::Block. Full logging also helps in better debugging. Steps to enable full Full logging in Code::Block is given in appendix C. Click the *build and run* button from compiler bar and output will be displayed as shown in figure 98.

Figure 98: Output for 3poly.cpp
4.6 Opening Existing Program/Project

Click File and select ‘Open’. Browse to desired directory and select the file with .cbp extension as shown in figure 99 and click on Open.

Figure 99: Select file with .cbp extension to open an existing project
References

   http://www.codeblocks.org/

   http://www.codeblocks.org/license

[3] Installing code::block on ubuntu

[4] Eternal thinker: How to use graphics.h in ubuntu?

[5] Code::block faq’s
A Code used for Windows

A.1 helloworld.c

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Listing 6: helloworld.c

A.2 line.c

```c
#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;
}
```

Listing 7: line.c

A.3 3poly.cpp

```c
#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);
}
```

Listing 8: 3poly.cpp
B  Code used for Ubuntu

B.1  helloworld.c

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Listing 9: helloworld.c

B.2  line.c

```c
#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;
}
```

Listing 10: line.c

B.3  3poly.cpp

```c
#include <simplecpp>

turtle 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);
}
```

Listing 11: 3poly.cpp
C Enabling full logging in Code::Block

Logging can be used extensively for many purposes including debugging and profiling. Steps to enable full logging in Code::Block are as mentioned below.

Settings → Compiler and debugger settings → Global compiler settings → check Selected compiler → Other settings → Compiler logging → Full command line

1. Click on Setting in Menu bar and select Compiler and debugger settings.

2. Select Global compiler settings from list on left panel. The default compiler selected is shown above in right panel. Here we can change the default compiler.

3. Click on right arrow beside tabs to select ‘Other settings’.

4. The first option is of Compiler logging, select ‘Full command line’.