



EEEE1042 - Lecture 1
Introduction to Software
Engineering and Programming
Autumn Semester 2021.

Dr. Chan, Kheong Sann

kheongsann.chan@nottingham.edu.my

University of Nottingham Malaysia
Department of Electrical and Electronic Engineering

EEE1042/1032 C Programming: Scheduled classes

EEE1042: for EE students, EEEE1032: for Mecha students.

Week	Dates	Lecture	EEE1042 Practical	EEE1032 Practical	Assessment
4	Sep 26 – 30	Thu2-4pm			
5	Oct 3 – 09	Thu2-4pm	Mon3-6pm	Wed3-6pm	
6	Oct 10 – 14	Thu2-4pm	P.H.	Wed3-6pm	
7	Oct 17 – 21	Thu2-4pm	Mon3-6pm	Wed3-6pm	PT1 5%
8	Oct 24 – 28	Thu2-4pm	P.H.	Wed3-6pm	
9	Oct 31 – Nov 04		Project	Week 1	
10	Nov 07 – 11	Thu2-4pm	Mon3-6pm	Wed3-6pm	PT2 5%
11	Nov 14 – 18	Thu2-4pm	Mon3-6pm	Wed3-6pm	CW1 10%
12	Nov 21 – 25		Project	Week 2	
13	Nov 28 – Dec 04	Thu2-4pm	Mon3-6pm	Wed3-6pm	PT3 5%
14	Dec 05 – 09		Project	Week 3	
15	Dec 12 – 16	Thu2-4pm	P.H.	Wed3-6pm	PT4 5%
16	Dec 19 – 23		Study Week		CW2 30%
17-18	Dec 26 – Jan 06		Study Weeks		
19-20	Jan 09 – 21		Final Exam (40%)		

EEE1042/1032 Course Information

Targeted students

This course is targeted at first year electrical (EEE1042) and mechatronics (EEE1032) students.

Assessment

- Coursework 1: 10%
- Coursework 2: 30%
- Final Exam : 40%
- Progress Tests: 20%

Submission through upload to Moodle.
Note: coursework or final exams that are substantially similar will be penalized as evidence of plagiarism.

Learning Outcome

- Develop the ability to analyse engineering problems by selecting appropriate software and/or techniques to enable the designing, planning, developing and implementation of practical solutions.

EEE1042 C Programming: Class Outline

- ① Installation of IDE environment, test gcc on HelloWorld.c program
- ② Tokens: keywords, identifiers, data type
- ③ Casting, number systems, variables, scope, functions, libraries
- ④ Flow of control: for, if while do-while
- ⑤ Pointers, references, strings, string functions
- ⑥ File handling getchar putchar, fread fwrite, fscanf
- ⑦ Pointers, arrays, multidim arrays, pointer&functions
- ⑧ Dynamic memory allocation: malloc, calloc, realloc
- ⑨ Advanced Data structures, typedef struct, linked list, trees.

Assessment method

Assessment

- 2 Courseworks totaling: 40%
- 4 Progress Tests: 5% each, 1 hour in duration, closed book multiple choice test occurs on Friday afternoon of weeks 7, 10, 13 and 15
- 1 Final exam worth 40%

Coursework Marking Rubric

- 0-25% Codes undeveloped or did not compile properly.
- 26-50% Codes developed and compiled, but result is incorrect.
- 51-75% Codes developed and compiled, result correct, but not well commented/modular, or good programming techniques.
- 76-100% Codes developed, compiled with correct results using taught best practice standards, and modularity/good-coding rules are demonstrated/followed.

Significantly similar code as evidence of plagiarism will be penalized.

Outline EEEE1042 C Lecture 1:

1 Operating System (OS) and Integrated Development Environment (IDE)

- Introduction
- OS/IDE/compiler
- Installing Ubuntu
- Installing Code::Blocks.
- Running Code::Blocks
 - Running without project
 - Running with project
- Modular programming
- Commenting your code
 - Purpose Of Comments
 - Implementing comments
 - Use-cases examples
 - Best practices

2 Key Takeaways

Introduction to C

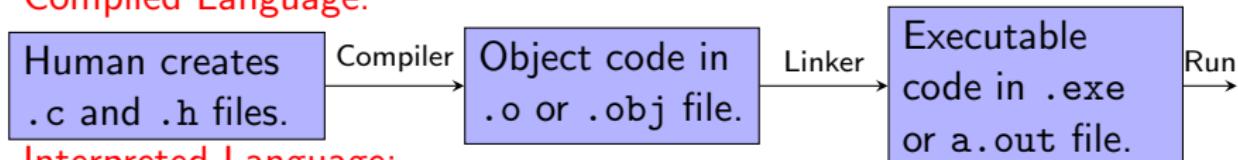
- C is a **compiled, low-level, procedural** computer programming language developed in 1972 by Dennis Ritchie at Bell Labs and it was used to write the UNIX operating system.

Compiled language takes human-readable code like `printf("hello")` and **compiles** it into **object** code that a CPU can execute.

Low-level language means that its instructions match closely the capabilities of the hardware like the **CPU** and **memory**.

Procedural language makes use of procedures, subroutines or subfunctions. It is one of the tools to help enable **modularity** and **code-reusability**.

Compiled Language:



Interpreted Language:

For interpreted Languages the interpreter (eg: Matlab) reads the commands line by line and executes them at **run-time**.

Introduction to C

- C is a **compiled, low-level, procedural** computer programming language developed in 1972 by Dennis Ritchie at Bell Labs and it was used to write the UNIX operating system.

Compiled language takes human-readable code like `printf("hello")` and **compiles** it into **object** code that a CPU can execute.

Low-level language means that its instructions match closely the capabilities of the hardware like the **CPU** and **memory**.

Procedural language makes use of procedures, subroutines or subfunctions. It is one of the tools to help enable **modularity** and **code-reusability**.

Low-level/Mid-level (C)

`z=x+y;` x, y and z must be declared and typed before use.

- Memory allocated by user
- Closer to the hardware
- Runs faster/Slower to program

High-level (Matlab)

`z=x+y;` x, y and z do not need to be pre-declared nor typed

- Memory allocated by interpreter
- Closer to the programmer
- Runs slower/Easier to program

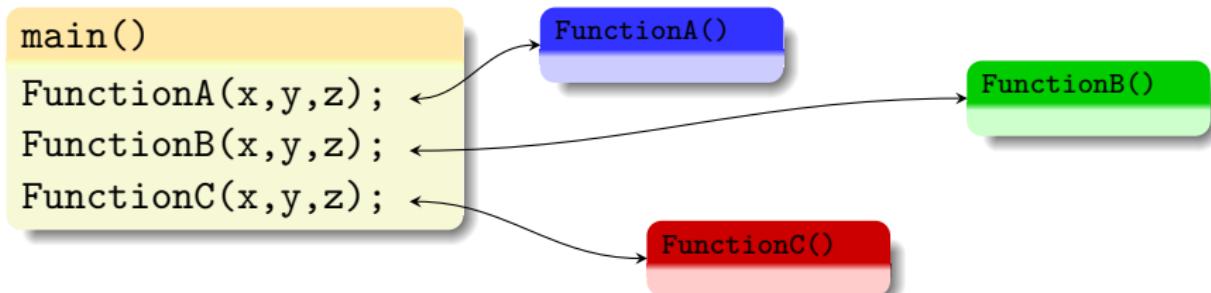
Introduction to C

- C is a **compiled, low-level, procedural** computer programming language developed in 1972 by Dennis Ritchie at Bell Labs and it was used to write the UNIX operating system.

Compiled language takes human-readable code like `printf("hello")` and **compiles** it into **object** code that a CPU can execute.

Low-level language means that its instructions match closely the capabilities of the hardware like the **CPU** and **memory**.

Procedural language makes use of procedures, subroutines or subfunctions. It is one of the tools to help enable **modularity** and **code-reusability**.



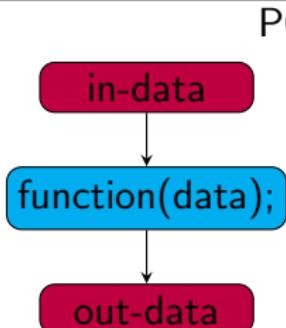
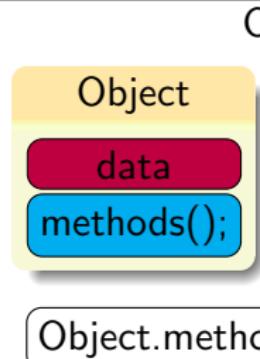
Introduction to C++

- C++ is an **object-oriented** version of C, or C with **classes**, developed by Bjarne Stroustrup from 1979-1985

Object Oriented instead of focusing on data and procedures as in C, C++ focuses on **objects**.

Classes are user defined types that **encapsulate** the data and functions (now called methods) together.

OOP Object Oriented Programming, a programming paradigm that defines/uses the object as opposed to **procedural oriented programming** (POP) that defines/uses functions

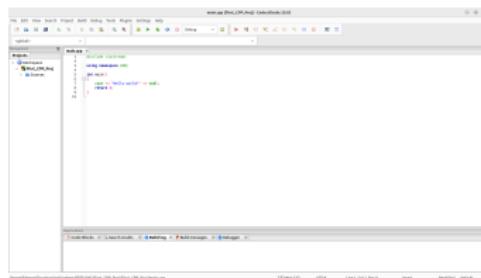
POP	OOP
 <pre> graph TD A[in-data] --> B["function(data);"] B --> C[out-data] </pre> <p>Input data exists in the calling environment</p> <p>Function is called on the input data</p> <p>Function returns modified data to the calling environment</p>	 <p>Objects are defined with a known data structure, and with methods (functions) encapsulated within. Only allowed data and methods are made available to the calling environment to process the data in the object.</p> <p><code>Object.method();</code></p>

OS, IDE and compiler

- In order to write a C/C++ program, the programmer must select an **OS** (operating system), **IDE** (Integrated Development Environment) and a **C-compiler**.
- In this course the **OS** is: Windows/Mac/Linux
IDE: CodeBlocks, **Compiler**: gcc/g++/MinGW

Code::Blocks is a free, open-source cross-platform integrated development environment (IDE) that ties in to a variety of C compiler back-ends including gcc, MinGW, MS Visual C++, amongst others.

Code::Blocks has binaries for Windows, Mac and Linux platforms.



OS, IDE and compiler

- In order to write a C/C++ program, the programmer must select an **OS** (operating system), **IDE** (Integrated Development Environment) and a **C-compiler**.
- In this course the **OS** is: Windows/Mac/Linux
IDE: CodeBlocks, **Compiler**: gcc/g++/MinGW

Compiler: gcc/g++/MinGW

gcc is the standard GNU C Compiler that comes with the standard Linux OS first released in 1987 by Richard Stallman. **g++** is the front-end for the GNU C++ Compiler.

MinGW is **Minimalist GNU for Windows** that provides GNU utilities including gcc/g++/gnu debugger for the Windows platform. You should choose to install Code::Blocks with MinGW in Windows.

Downloading and Installing Code::Blocks.

- ① Go to the download page:

<http://www.codeblocks.org/downloads/binaries/>

Downloading and Installing Code::Blocks.

- ① Go to the download page:
<http://www.codeblocks.org/downloads/binaries/>
- ② Scroll down to your relevant OS and download the appropriate binaries (installer)

Downloading and Installing Code::Blocks.

- ① Go to the download page:

<http://www.codeblocks.org/downloads/binaries/>

- ② Scroll down to your relevant OS and download the appropriate binaries (installer)

- For Windows, you are advised to get `codeblocks-20.03mingw-setup.exe`
- For Mac there is only one choice `CodeBlocks-13.12-mac.zip`
- For Linux, you can get codeblocks from the apt repository using:

```
sudo apt-get install codeblocks
```

Downloading and Installing Code::Blocks.

- ① Go to the download page:

<http://www.codeblocks.org/downloads/binaries/>

- ② Scroll down to your relevant OS and download the appropriate binaries (installer)

- For Windows, you are advised to get codeblocks-20.03mingw-setup.exe
- For Mac there is only one choice CodeBlocks-13.12-mac.zip
- For Linux, you can get codeblocks from the apt repository using:

```
sudo apt-get install codeblocks
```

In all cases, you can choose to either download from FossHUB or SourceForge.net



Microsoft Windows

File

codeblocks-20.03-setup.exe
codeblocks-20.03-setup-nonadmin.exe
codeblocks-20.03-nosetup.zip
codeblocks-20.03mingw-setup.exe
codeblocks-20.03mingw-nosetup.zip
codeblocks-20.03-32bit-setup.exe
codeblocks-20.03-32bit-setup-nonadmin.exe
codeblocks-20.03-32bit-nosetup.zip
codeblocks-20.03mingw-32bit-setup.exe
codeblocks-20.03mingw-32bit-nosetup.zip

Download from

[FossHUB](#) or [Sourceforge.net](#)
[FossHUB](#) or [Sourceforge.net](#)

Downloading and Installing Code::Blocks.

- ① Go to the download page:

<http://www.codeblocks.org/downloads/binaries/>

- ② Scroll down to your relevant OS and download the appropriate binaries (installer)

- For Windows, you are advised to get codeblocks-20.03mingw-setup.exe
- For Mac there is only one choice CodeBlocks-13.12-mac.zip
- For Linux, you can get codeblocks from the apt repository using:

```
sudo apt-get install codeblocks
```

In all cases, you can choose to either download from FossHUB or SourceForge.net

Choose to download from either FossHUB or SourceForge



Microsoft Windows

File

codeblocks-20.03-setup.exe
codeblocks-20.03-setup-nonadmin.exe
codeblocks-20.03-nosetup.zip

codeblocks-20.03mingw-setup.exe

codeblocks-20.03mingw-nosetup.zip

codeblocks-20.03-32bit-setup.exe

codeblocks-20.03-32bit-setup-nonadmin.exe

codeblocks-20.03-32bit-nosetup.zip

codeblocks-20.03mingw-32bit-setup.exe

codeblocks-20.03mingw-32bit-nosetup.zip

Download from

[FossHUB](#) or [Sourceforge.net](#)

Downloading and Installing Code::Blocks.

- ① Go to the download page:

<http://www.codeblocks.org/downloads/binaries/>

- ② Scroll down to your relevant OS and download the appropriate binaries (installer)

- For Windows, you are advised to get `codeblocks-20.03mingw-setup.exe`
- For Mac there is only one choice `CodeBlocks-13.12-mac.zip`
- For Linux, you can get codeblocks from the apt repository using:

```
sudo apt-get install codeblocks
```

In all cases, you can choose to either download from [FossHUB](#) or [SourceForge.net](#)

Choose with MinGW to install the files that include the Windows version of the GNU C compiler (gcc)



Microsoft Windows

File

[codeblocks-20.03-setup.exe](#)
[codeblocks-20.03-setup-nonadmin.exe](#)
[codeblocks-20.03-nosetup.zip](#)
[codeblocks-20.03mingw-setup.exe](#)
[codeblocks-20.03mingw-nosetup.zip](#)
[codeblocks-20.03-32bit-setup.exe](#)
[codeblocks-20.03-32bit-setup-nonadmin.exe](#)
[codeblocks-20.03-32bit-nosetup.zip](#)
[codeblocks-20.03mingw-32bit-setup.exe](#)
[codeblocks-20.03mingw-32bit-nosetup.zip](#)

Download from

FossHUB	Sourceforge.net

Downloading and Installing Code::Blocks.

- ③ In Windows, run the installer you just downloaded.

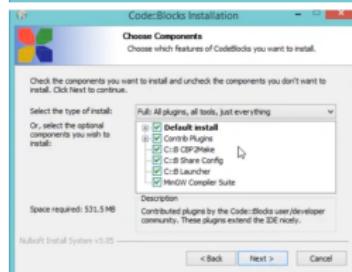


Downloading and Installing Code::Blocks.

- ③ In Windows, run the installer you just downloaded.



- ④ You can choose which components to include in the install. Be sure you include MinGW to have the compiler

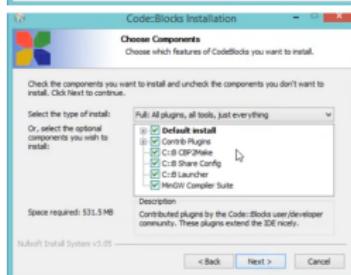


Downloading and Installing Code::Blocks.

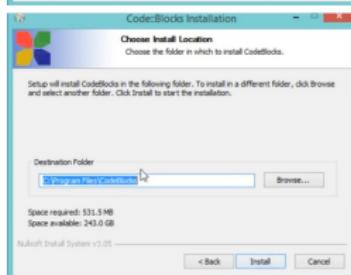
- ③ In Windows, run the installer you just downloaded.



- ④ You can choose which components to include in the install. Be sure you include MinGW to have the compiler



- ⑤ This screen allows you to choose where you want to install Code::Blocks, you can just leave it at the default.

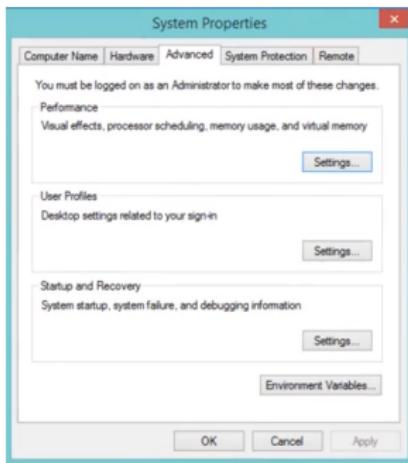


Set MinGW to the system path.

- ⑥ After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at `C:\ Program Files\CodeBlocks\MinGW\bin`

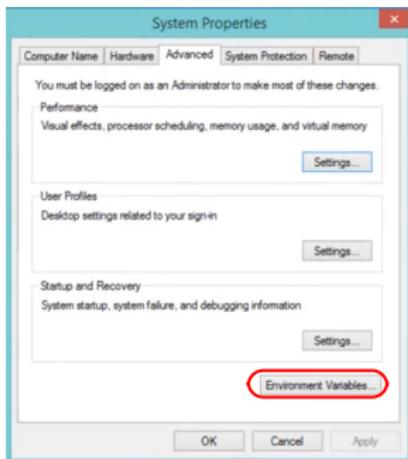
Set MinGW to the system path.

- ⑥ After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at C:\ Program Files\CodeBlocks\MinGW\bin
- ⑦ Add this to the path in the system properties



Set MinGW to the system path.

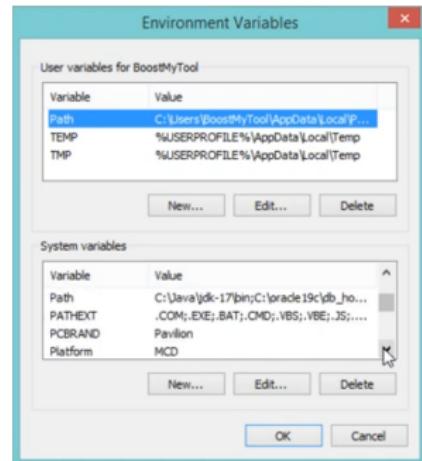
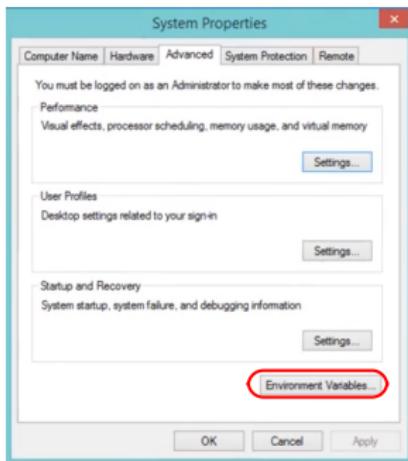
- ⑥ After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at C:\ Program Files\CodeBlocks\MinGW\bin
- ⑦ Add this to the path in the system properties



Click Environment variables.

Set MinGW to the system path.

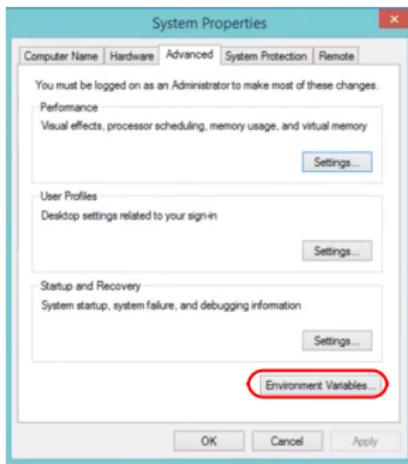
- ⑥ After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at `C:\ Program Files\CodeBlocks\MinGW\bin`
- ⑦ Add this to the path in the system properties
- ⑧ Add the path to MinGW to the environment variable.



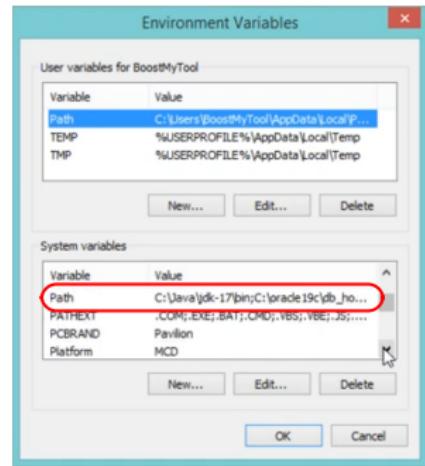
Click Environment variables.

Set MinGW to the system path.

- ⑥ After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at `C:\ Program Files\CodeBlocks\MinGW\bin`
- ⑦ Add this to the path in the system properties
- ⑧ Add the path to MinGW to the environment variable.



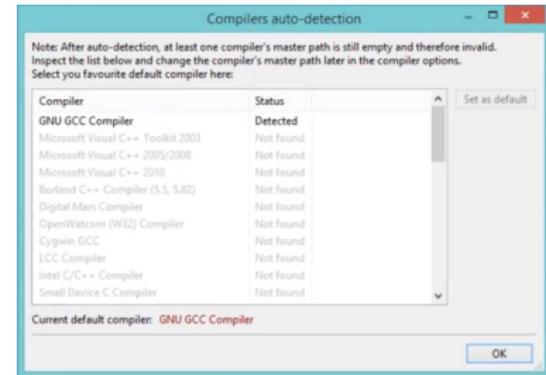
Click Environment variables.



Add MinGW to the system path here.

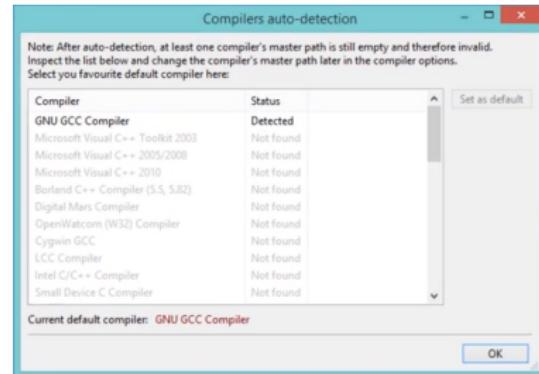
Run Code::Blocks for the first time.

- 1 Executing Code::Blocks for the first time, it auto-detects the MinGW GCC compiler. You can set it as the default.

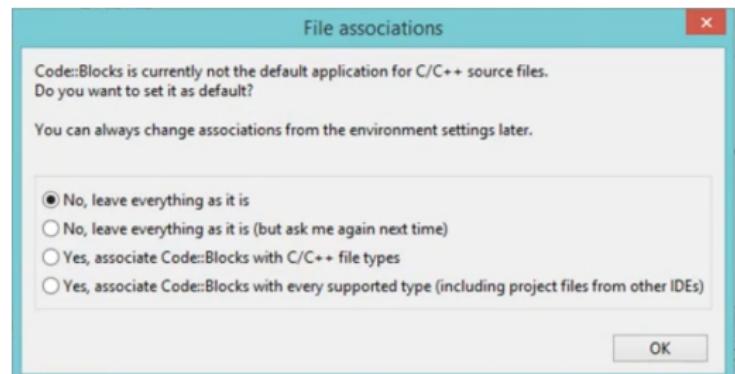


Run Code::Blocks for the first time.

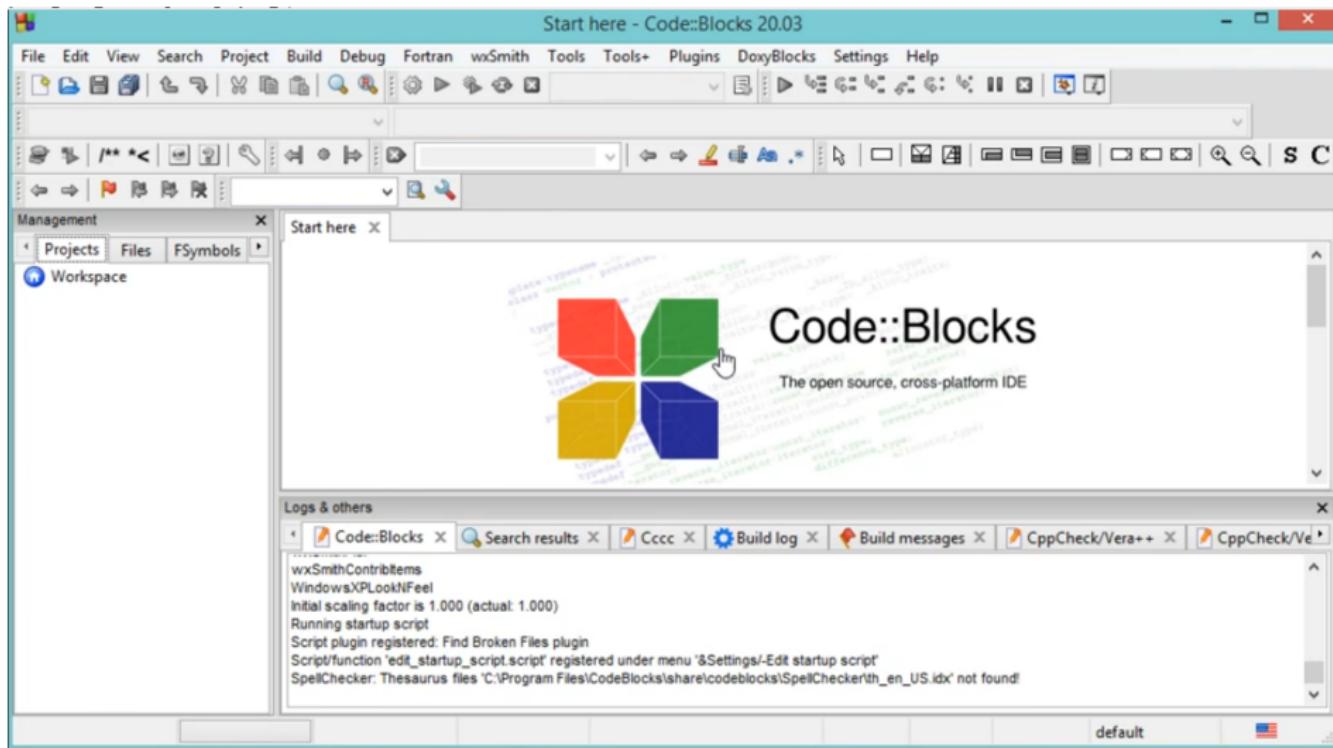
- ① Executing Code::Blocks for the first time, it auto-detects the MinGW GCC compiler. You can set it as the default.



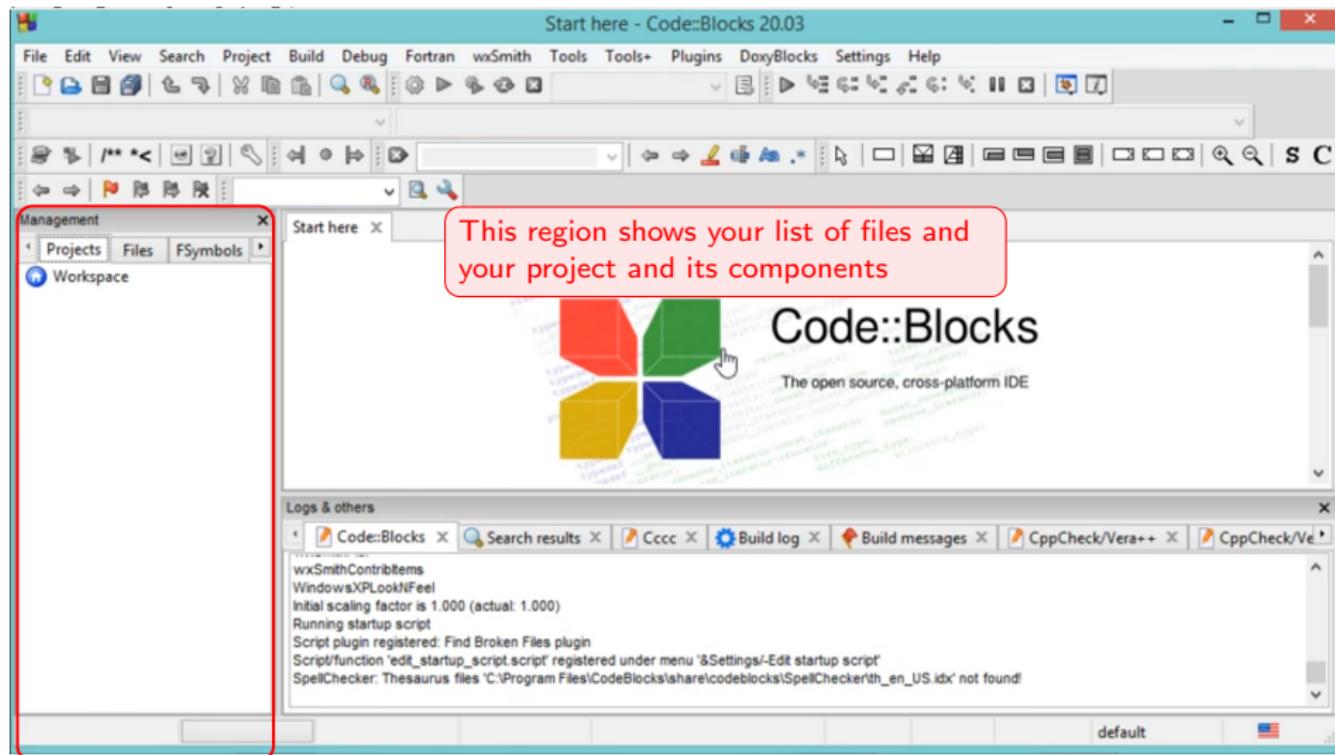
- ② You can choose to associate Code::Blocks with .c and .cpp files so that it is the chosen application to run whenever you click them.



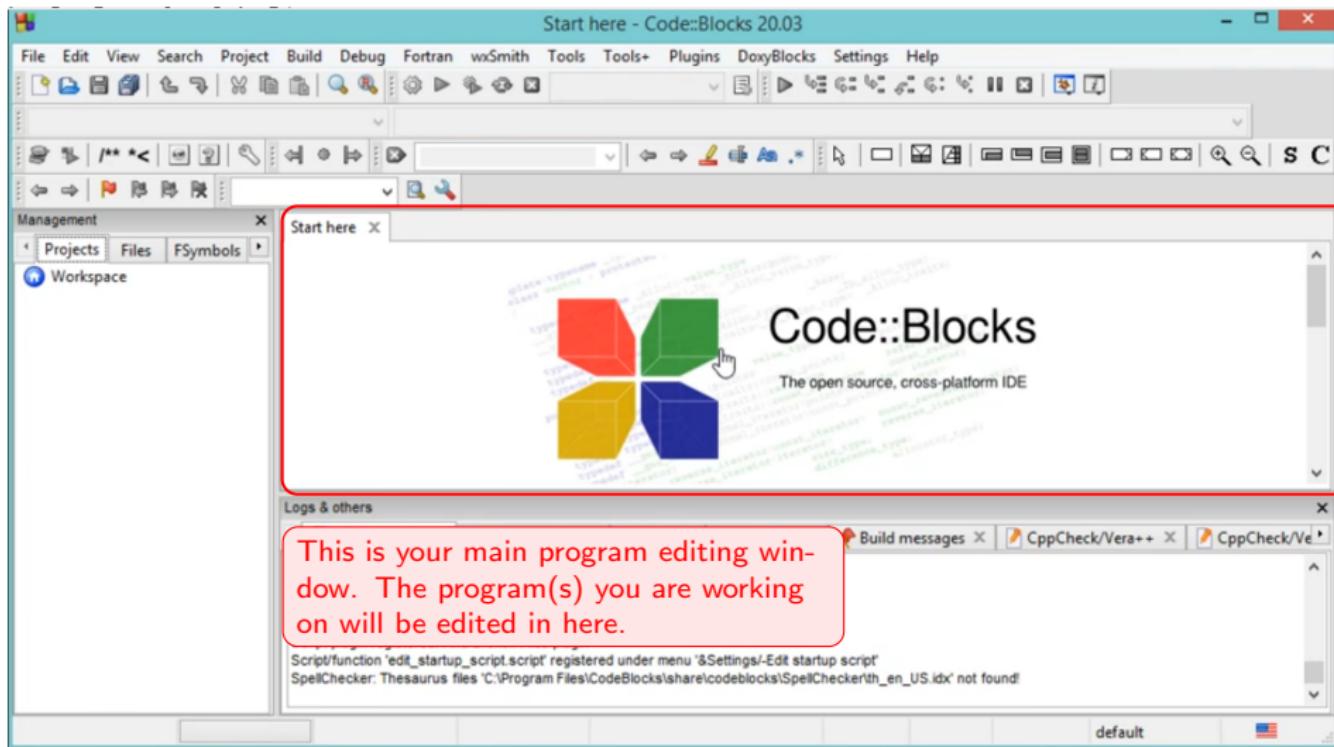
Run Code::Blocks for the first time.



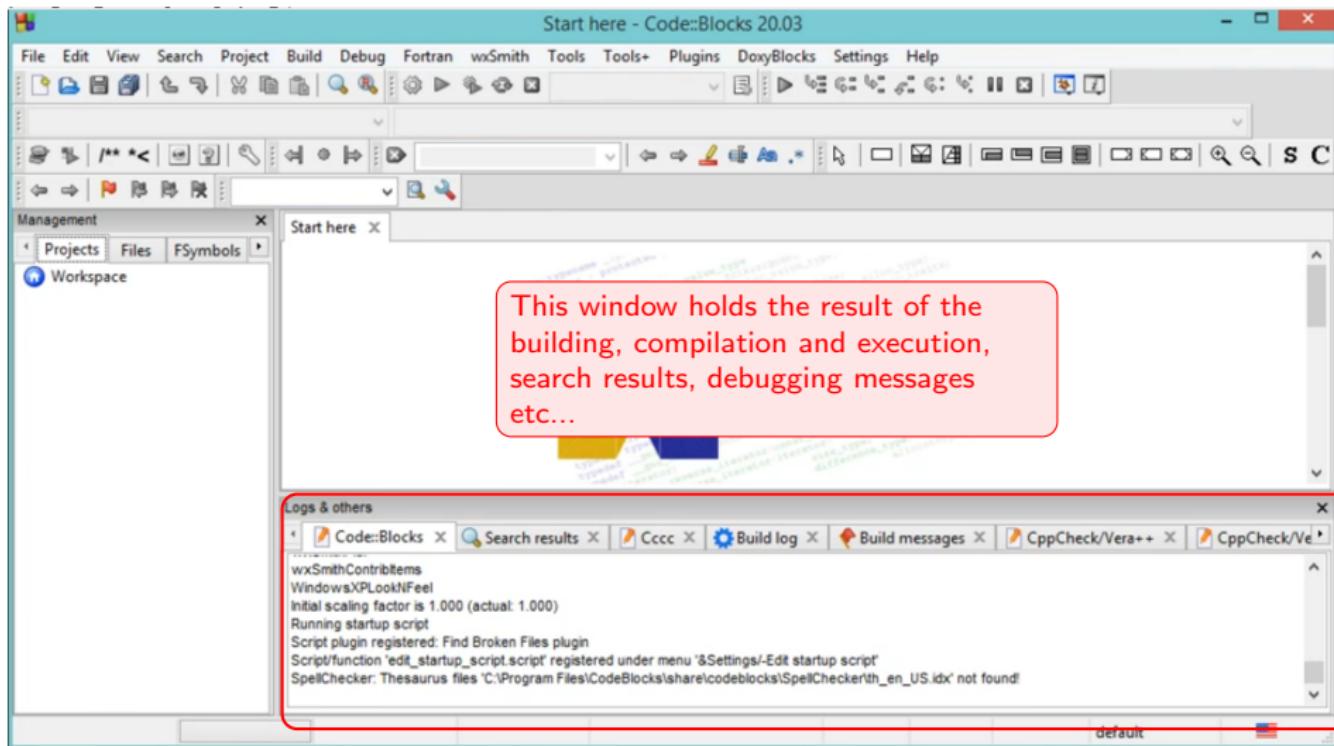
Run Code::Blocks for the first time.



Run Code::Blocks for the first time.



Run Code::Blocks for the first time.



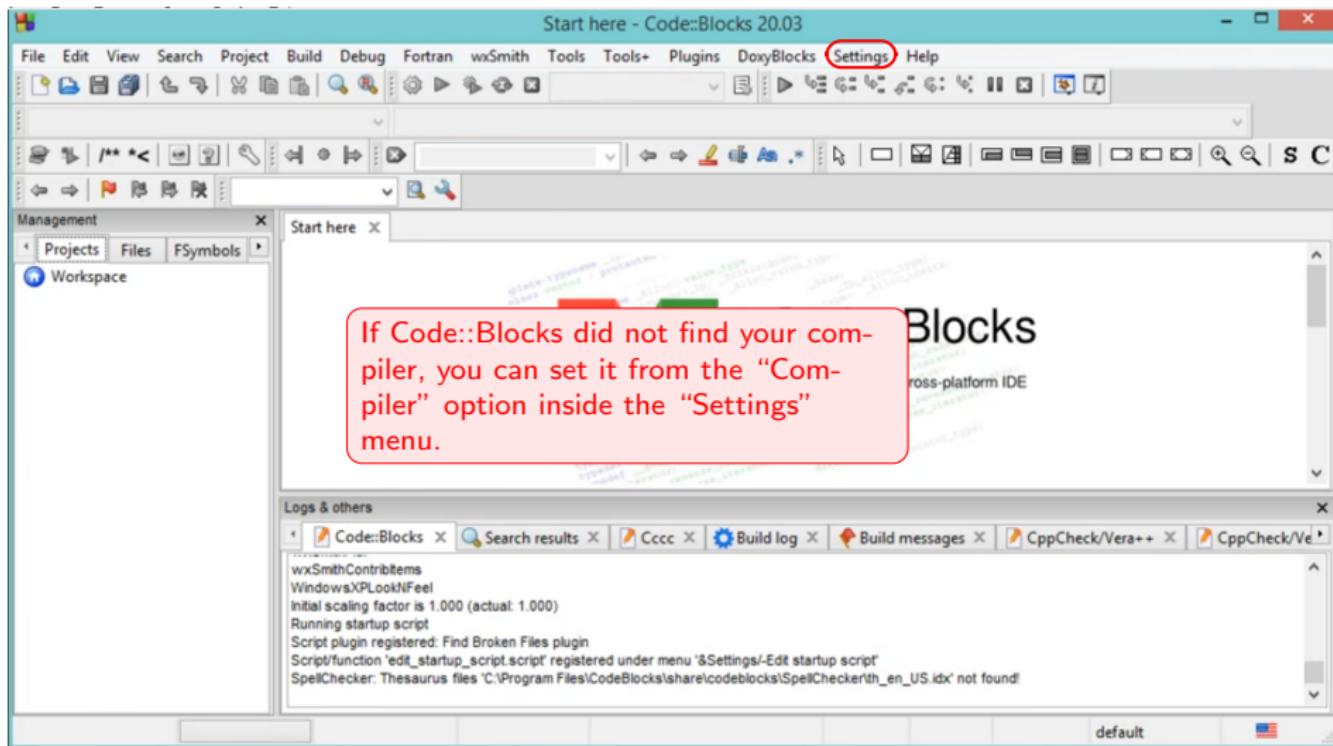
This window holds the result of the building, compilation and execution, search results, debugging messages etc...

Logs & others

- Code::Blocks
- Search results
- Cccc
- Build log
- Build messages
- CppCheck/Vera++
- CppCheck/Ve

```
wxSmithContribitems
WindowsXPLookAndFeel
Initial scaling factor is 1.000 (actual: 1.000)
Running startup script
Script plugin registered: Find Broken Files plugin
Script/function 'edit_startup_script.script' registered under menu '&Settings/Edit startup script'
SpellChecker: Thesaurus files 'C:\Program Files\CodeBlocks\share\codeblocks\SpellCheck\th_en_US.idx' not found!
```

Run Code::Blocks for the first time.

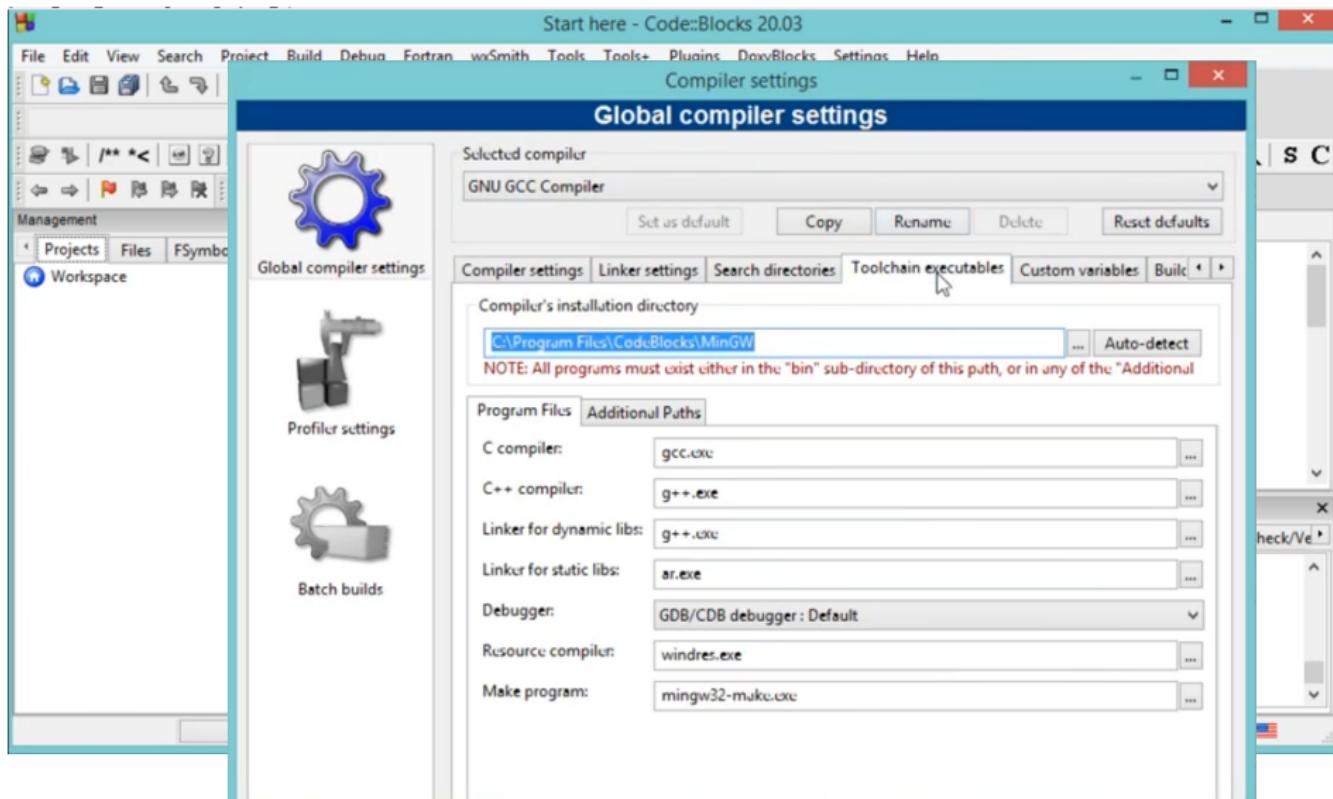


If Code::Blocks did not find your compiler, you can set it from the "Compiler" option inside the "Settings" menu.

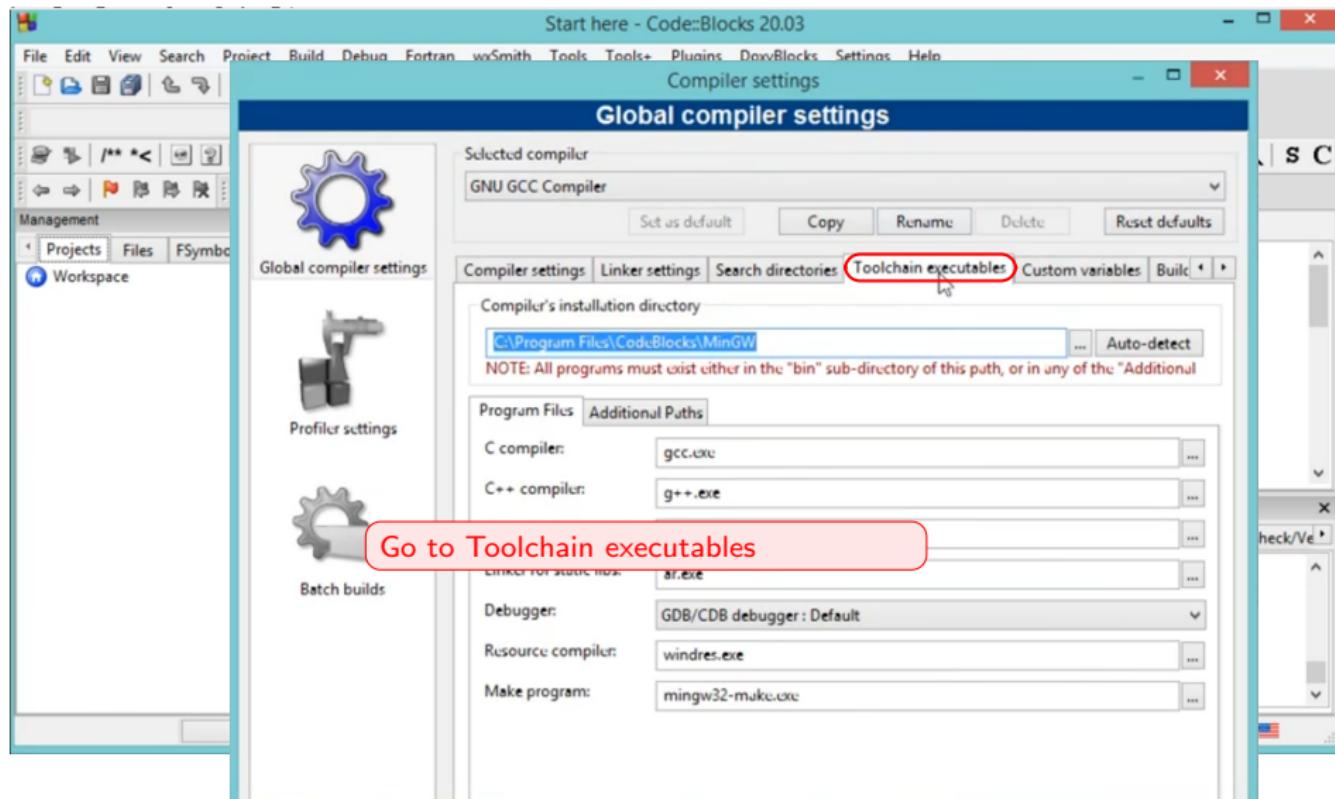
Logs & others

```
Code::Blocks x Search results x Cccc x Build log x Build messages x CppCheck/Vera++ x CppCheck/Vera++ x
wxSmithContribItems
WindowsXPLookAndFeel
Initial scaling factor is 1.000 (actual: 1.000)
Running startup script
Script plugin registered: Find Broken Files plugin
Script/function 'edit_startup_script.script' registered under menu '&Settings/Edit startup script'
SpellChecker: Thesaurus files 'C:\Program Files\CodeBlocks\share\codeblocks\SpellCheck\th_en_US.idx' not found!
```

Run Code::Blocks for the first time.

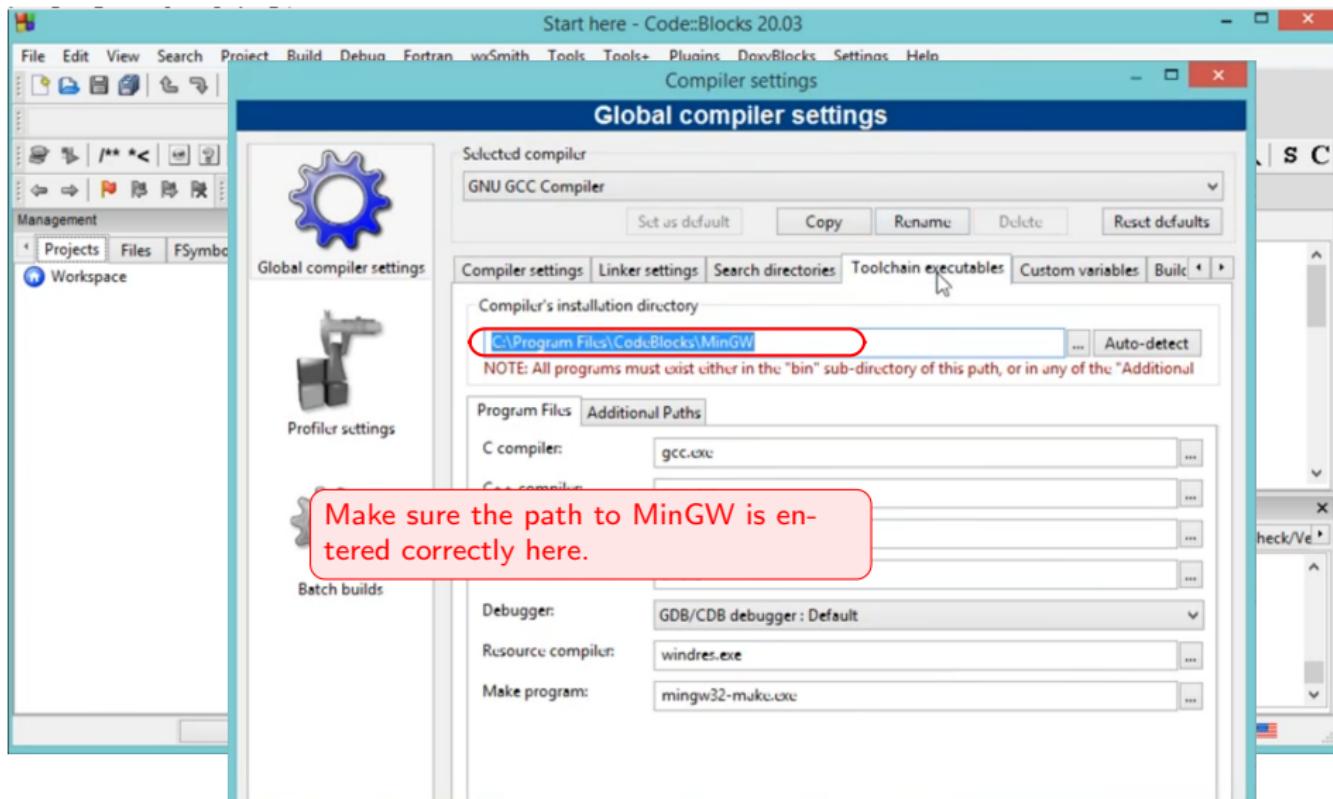


Run Code::Blocks for the first time.



The screenshot shows the Code::Blocks 20.03 interface. The main window title is "Start here - Code::Blocks 20.03". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DocsBlocks, Settings, Help. A toolbar with various icons is visible. On the left, there's a sidebar with Management, Projects, Files, FSymbols, and Workspace. The central area is titled "Global compiler settings" with a sub-section "Selected compiler: GNU GCC Compiler". It contains tabs for Compiler settings, Linker settings, Search directories, Toolchain executables (which is highlighted with a red box), Custom variables, and Built-in. Under "Toolchain executables", the "Compiler's installation directory" is set to "C:\Program Files\CodeBlocks\MinGW". A note below says: "NOTE: All programs must exist either in the "bin" sub-directory of this path, or in any of the "Additional Program Files" listed below." Below this are sections for "Program Files" and "Additional Paths" for C compiler (gcc.exe) and C++ compiler (g++.exe). Other toolchain executables listed include ar.exe, GDB/CDB debugger : Default, windres.exe, and mingw32-make.exe. A red callout box points to the "Toolchain executables" tab with the text "Go to Toolchain executables".

Run Code::Blocks for the first time.



The screenshot shows the Code::Blocks 20.03 interface. The main window title is "Start here - Code::Blocks 20.03". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DocsBlocks, Settings, Help. A toolbar with various icons is visible. On the left, there's a "Management" panel with "Projects", "Files", "FSymbols", and "Workspace" tabs, and a "Workspace" icon. The central area is titled "Global compiler settings" with a sub-section "Selected compiler: GNU GCC Compiler". It contains buttons for "Set as default", "Copy", "Rename", "Delete", and "Reset defaults". Below this are tabs for "Compiler settings", "Linker settings", "Search directories", "Toolchain executables" (which is selected), "Custom variables", and "Build". A note says "NOTE: All programs must exist either in the "bin" sub-directory of this path, or in any of the "Additional Program Files" or "Additional Paths" listed below." A red box highlights the "Toolchain executables" tab and the "Compiler's installation directory" input field, which contains "C:\Program Files\CodeBlocks\MinGW". A callout bubble with a gear icon points to this field with the text "Make sure the path to MinGW is entered correctly here." At the bottom, there are sections for "Batch builds", "Debugger: GDB/CDB debugger : Default", "Resource compiler: windres.exe", and "Make program: mingw32-make.exe".

Start here - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DocsBlocks Settings Help

Compiler settings

Global compiler settings

Selected compiler: GNU GCC Compiler

Set as default Copy Rename Delete Reset defaults

Compiler's installation directory: C:\Program Files\CodeBlocks\MinGW

NOTE: All programs must exist either in the "bin" sub-directory of this path, or in any of the "Additional Program Files" or "Additional Paths" listed below.

Make sure the path to MinGW is entered correctly here.

Toolchain executables

Compiler settings Linker settings Search directories Custom variables Build

Compiler's installation directory: C:\Program Files\CodeBlocks\MinGW

Auto-detect

Program Files Additional Paths

C compiler: gcc.exe

C++ compiler:

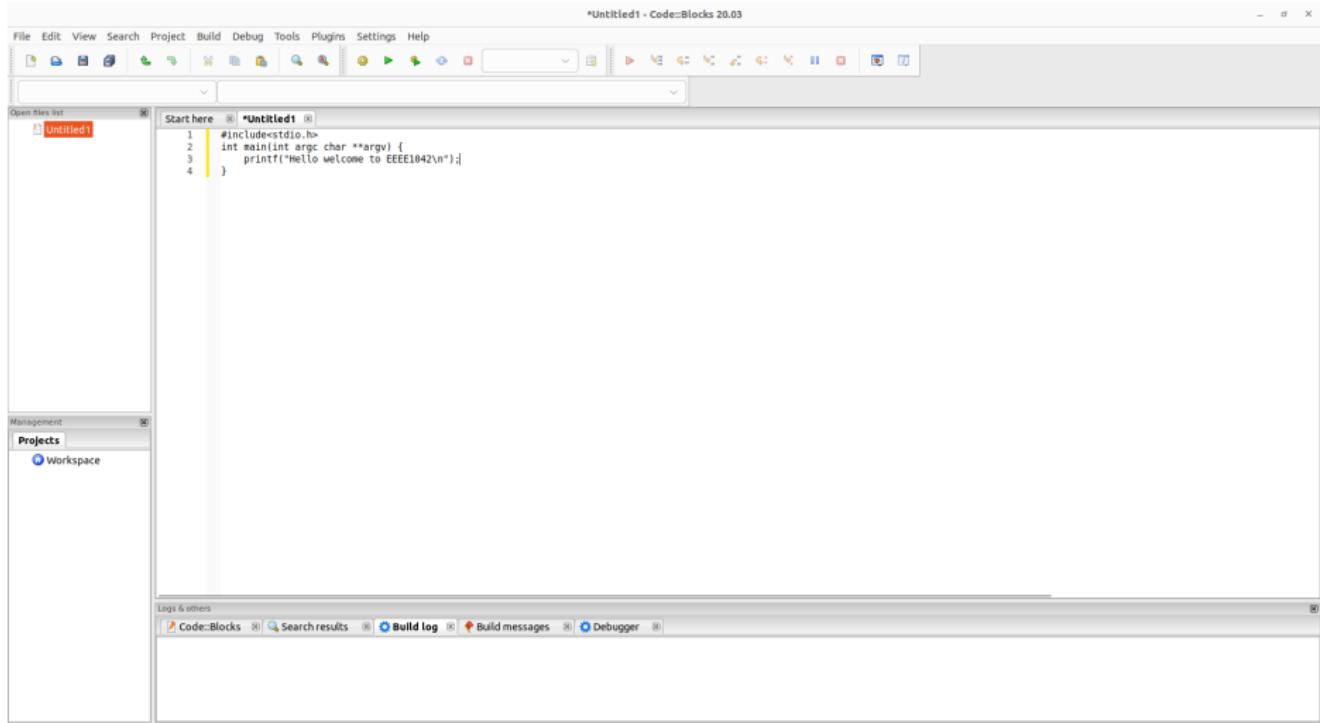
Batch builds

Debugger: GDB/CDB debugger : Default

Resource compiler: windres.exe

Make program: mingw32-make.exe

Run Code::Blocks for the first time: generate C-program without project.

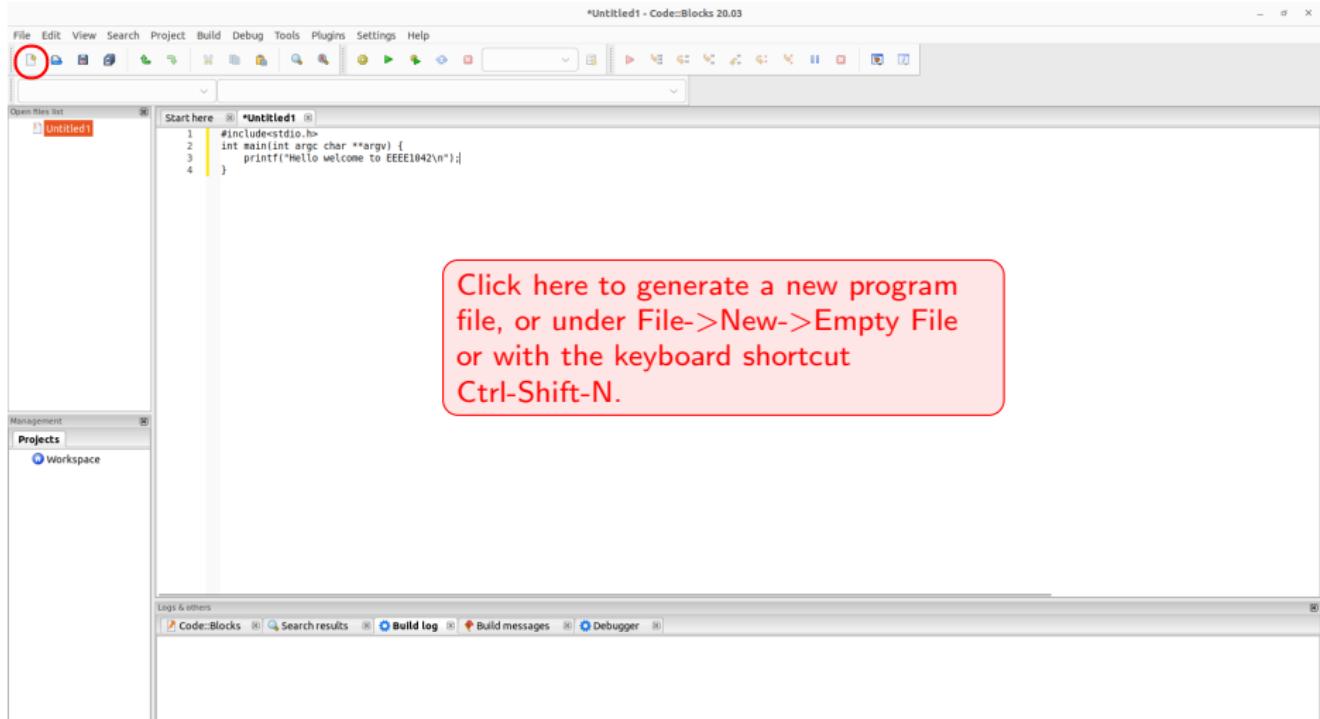


The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The title bar says "*Untitled1 - Code::Blocks 20.03". The main window displays a code editor with the following C code:

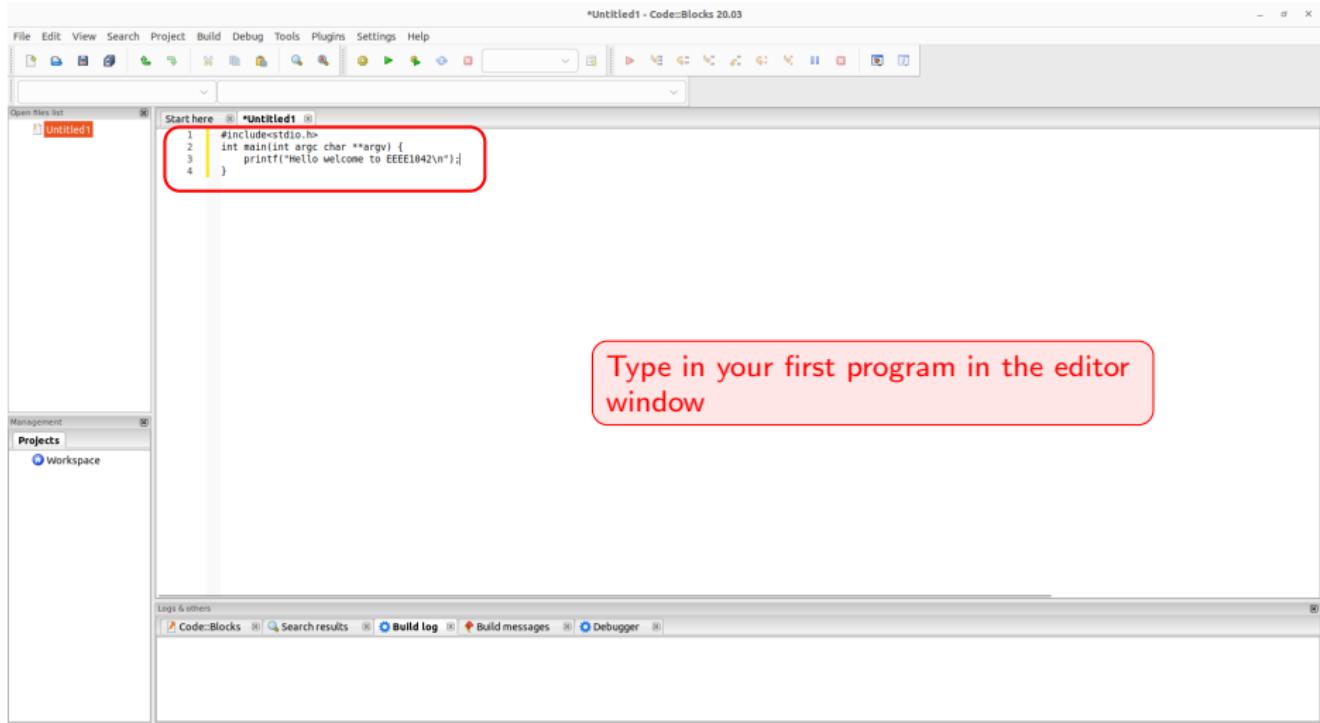
```
#include<stdio.h>
int main(int argc char **argv) {
    printf("Hello welcome to EEEE1042\n");
}
```

The left sidebar has an "Open files list" showing "Untitled1" and a "Management" section with "Projects" selected, showing "Workspace". The bottom status bar indicates Unix (LF), UTF-8, Line 3, Col 43, Pos 93, Insert, Modif..., Read/Wri..., default.

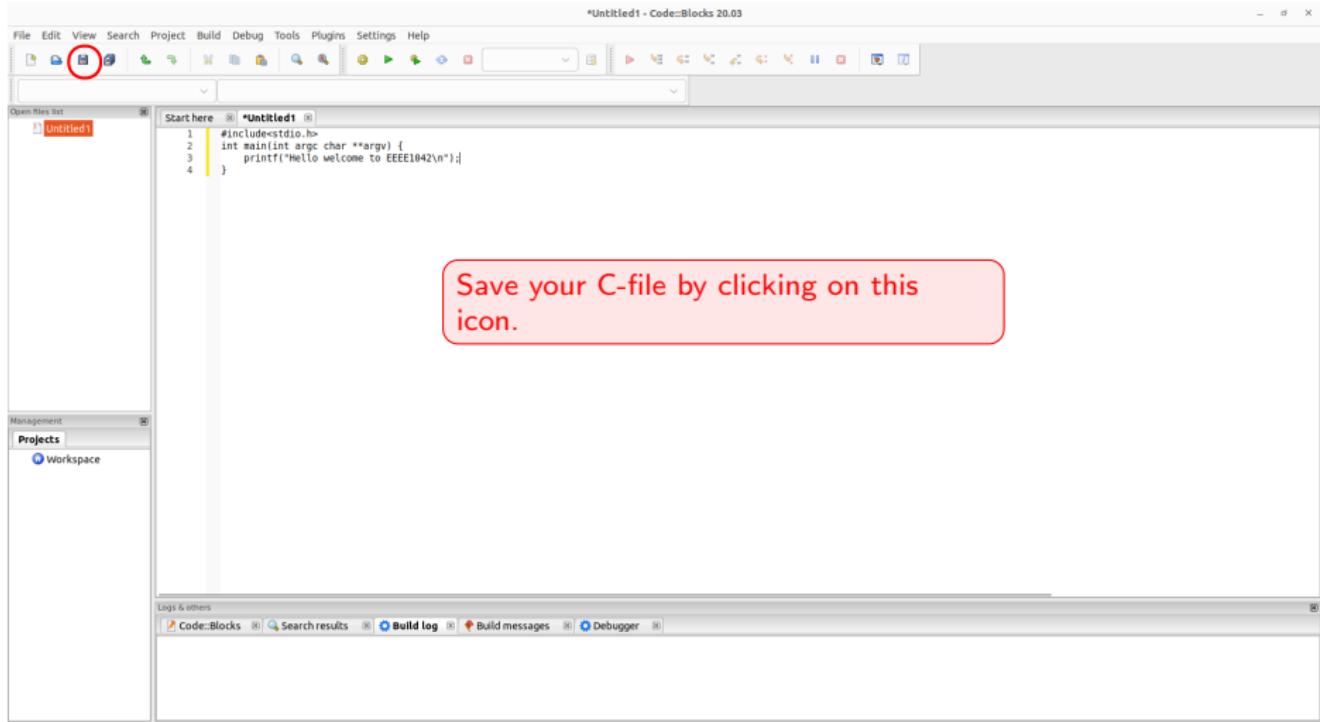
Run Code::Blocks for the first time: generate C-program without project.



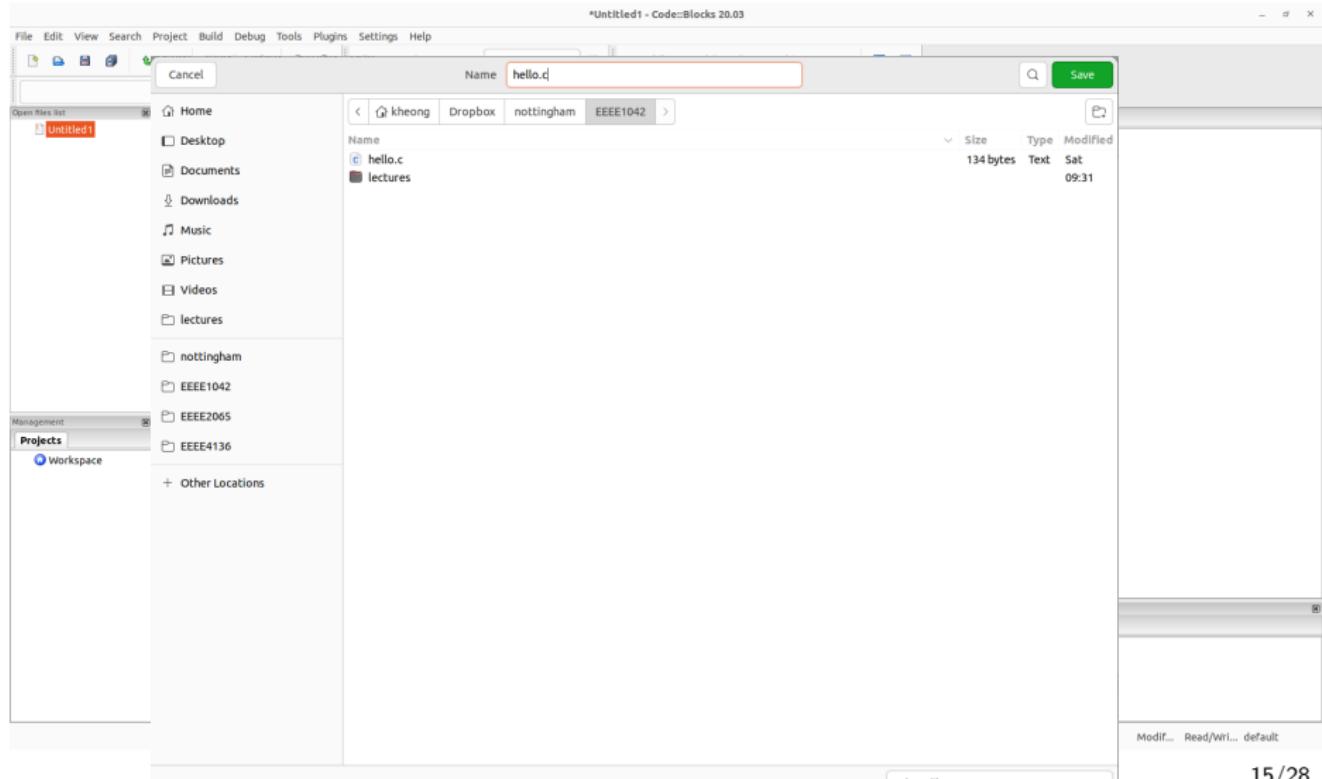
Run Code::Blocks for the first time: generate C-program without project.



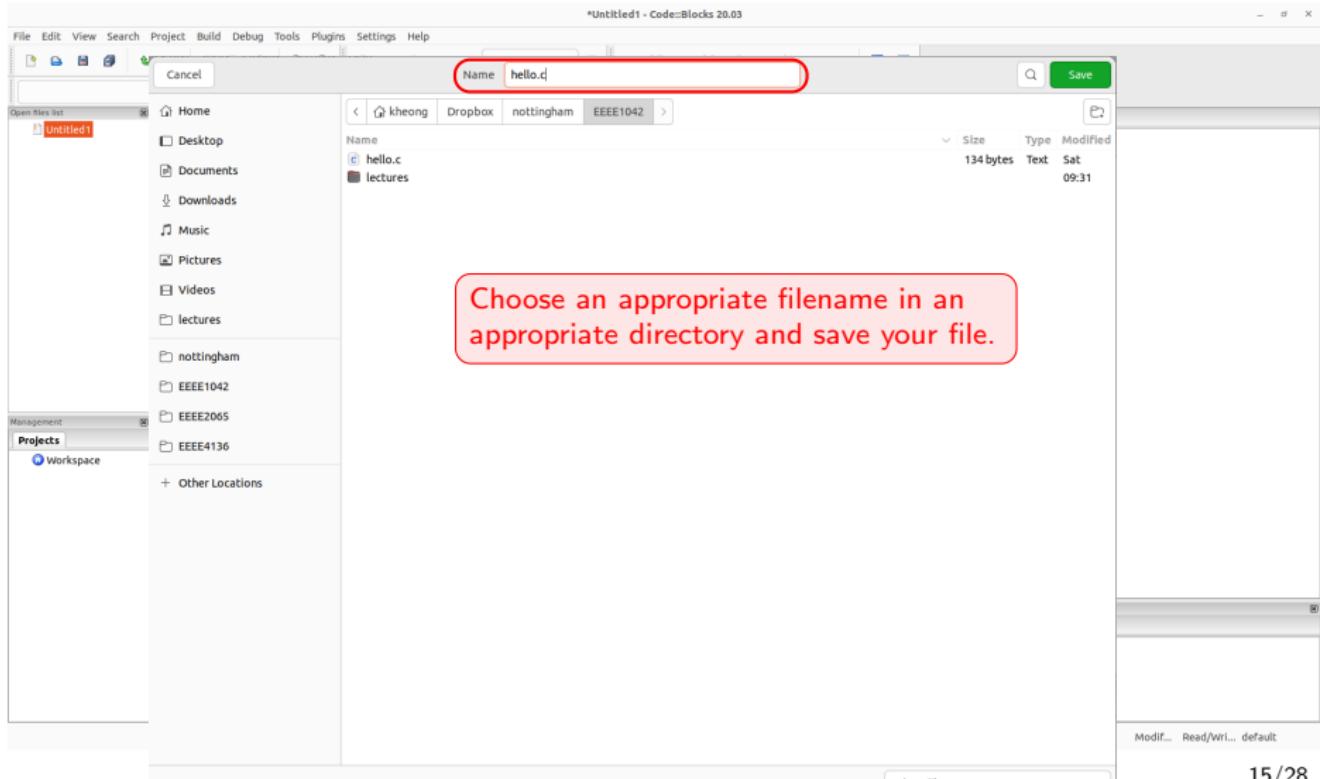
Run Code::Blocks for the first time: generate C-program without project.



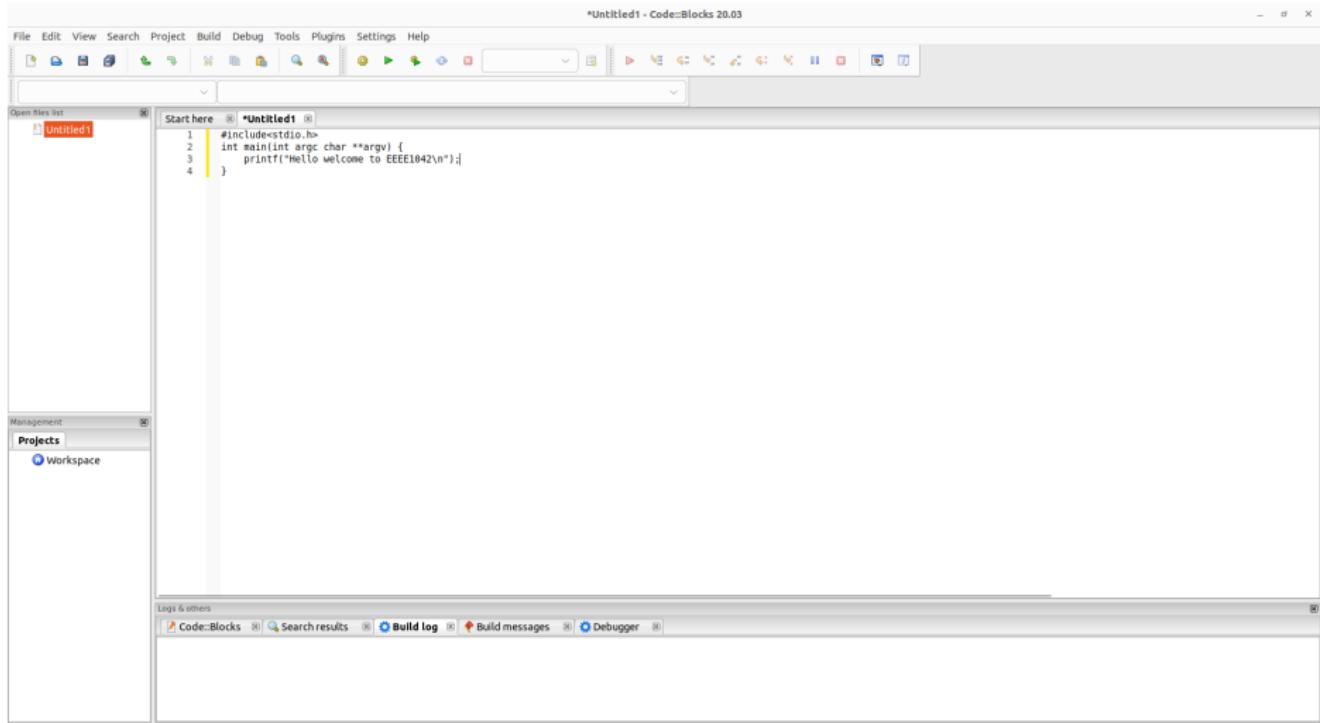
Run Code::Blocks for the first time: generate C-program without project.



Run Code::Blocks for the first time: generate C-program without project.



Run Code::Blocks for the first time: compile and execute your program without project.

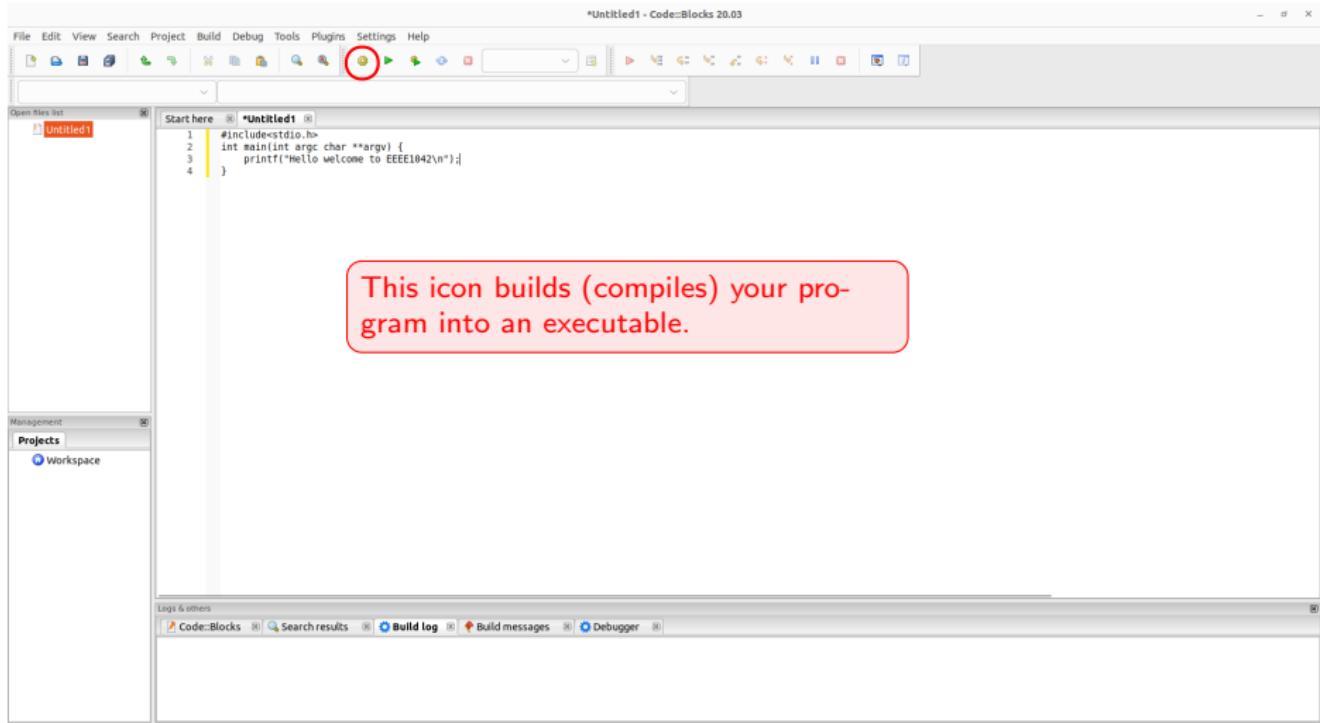


The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The title bar says "Untitled1 - Code::Blocks 20.03". The main window displays a code editor with the following C code:

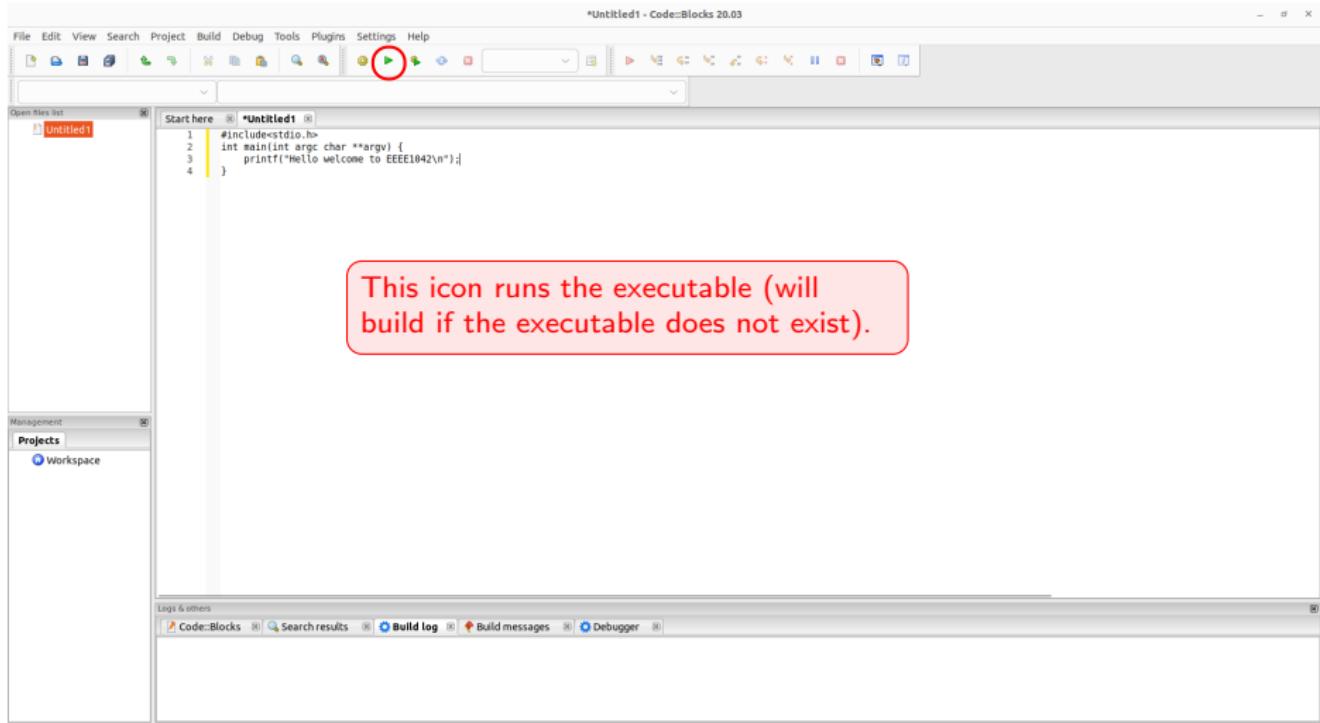
```
#include<stdio.h>
int main(int argc char **argv) {
    printf("Hello welcome to EEEE1042\n");
}
```

The left sidebar has an "Open files list" showing "Untitled1" and a "Management" section with "Projects" selected, showing "Workspace". The bottom status bar indicates "Unix (LF)", "UTF-8", "Line 3, Col 43, Pos 93", and "Insert Modif... Read/Wri... default".

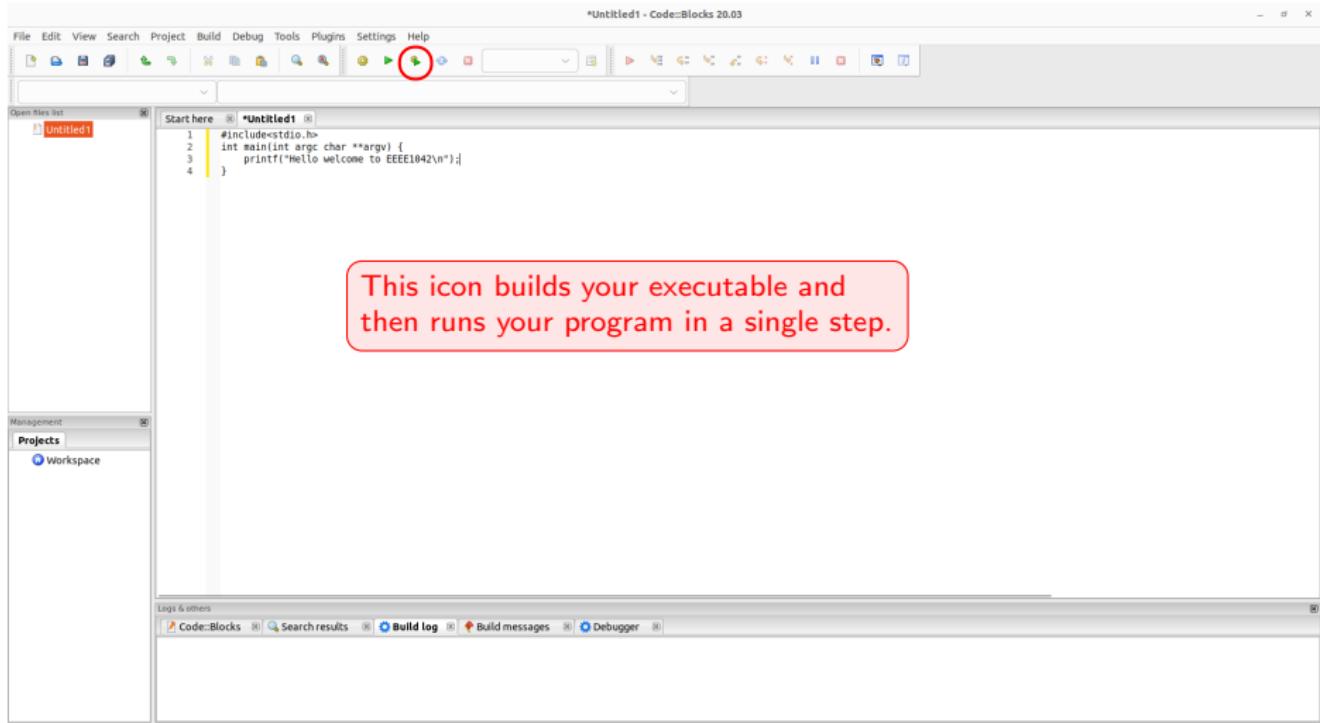
Run Code::Blocks for the first time: compile and execute your program without project.



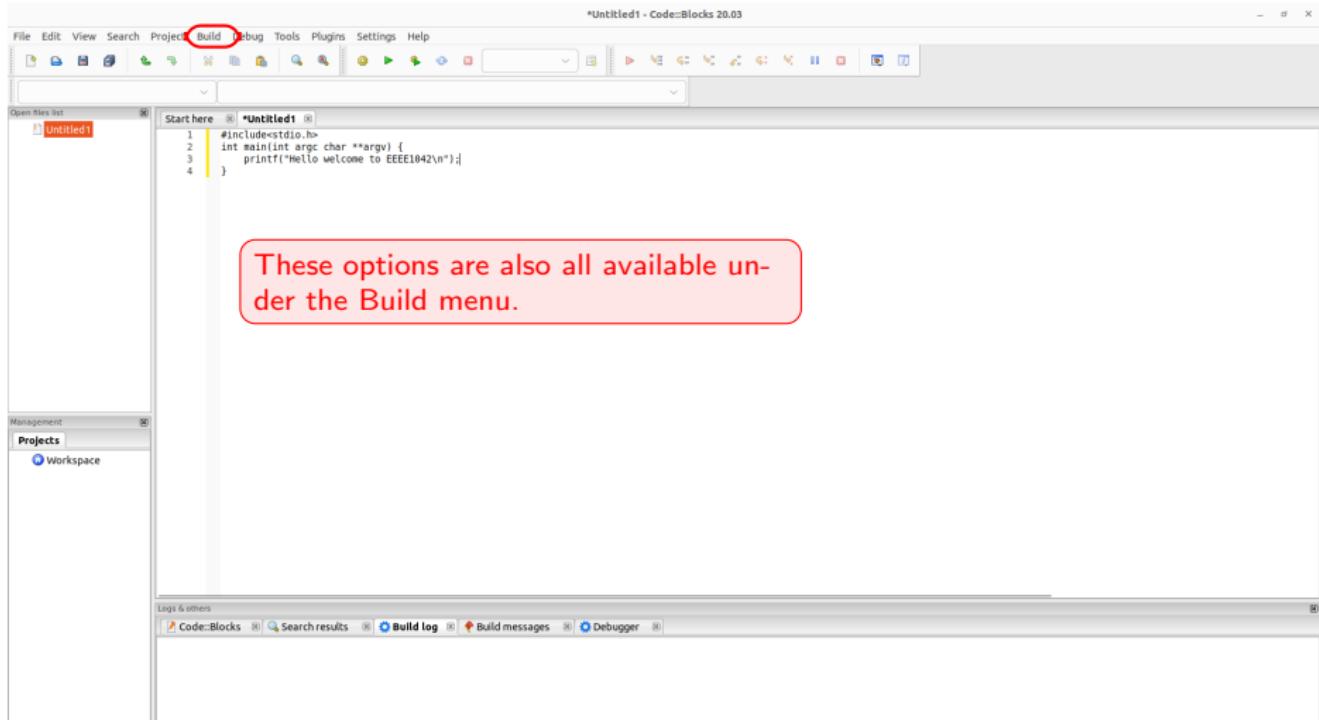
Run Code::Blocks for the first time: compile and execute your program without project.



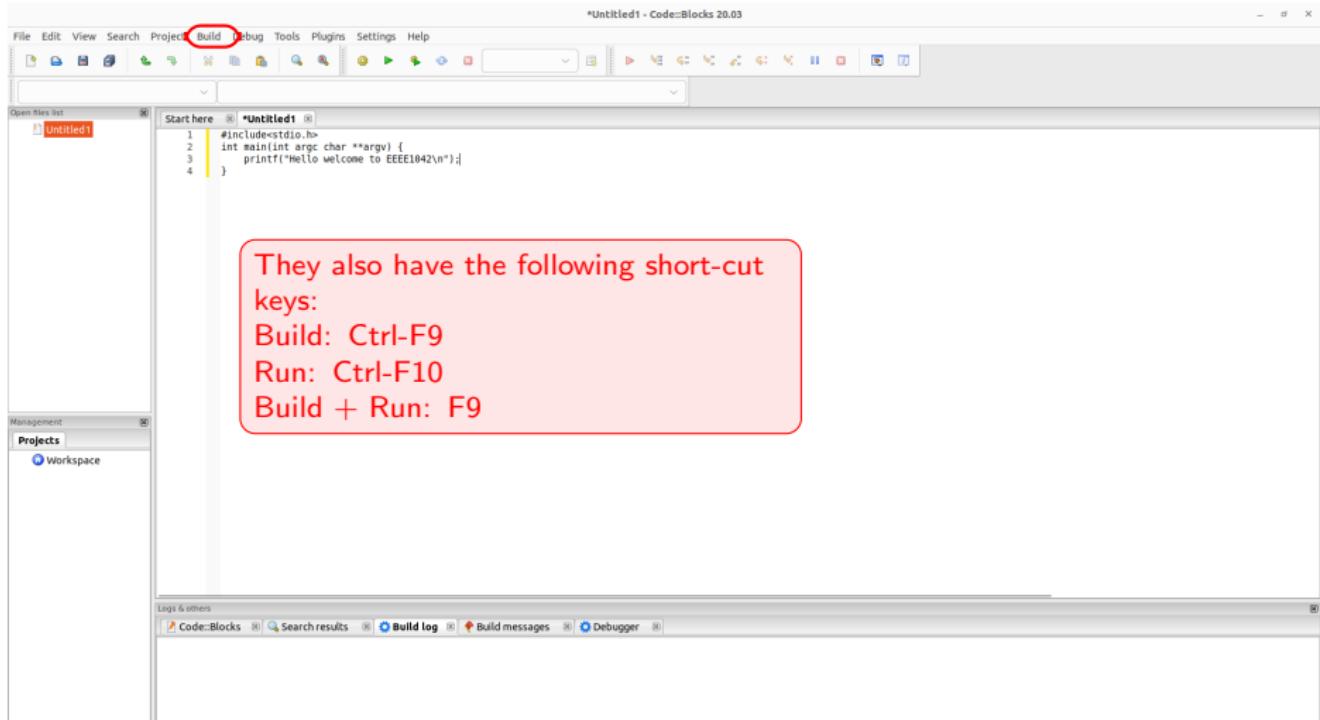
Run Code::Blocks for the first time: compile and execute your program without project.



Run Code::Blocks for the first time: compile and execute your program without project.



Run Code::Blocks for the first time: compile and execute your program without project.



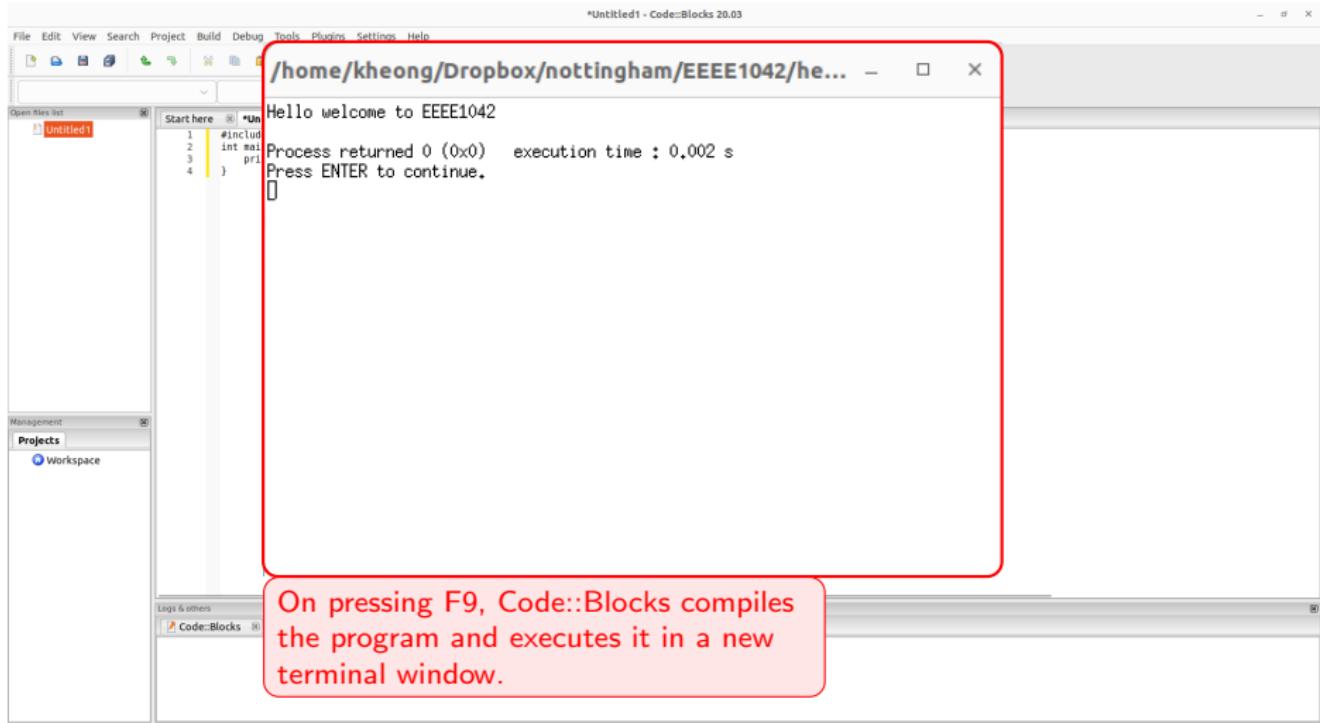
They also have the following short-cut keys:

Build: Ctrl-F9

Run: Ctrl-F10

Build + Run: F9

Run Code::Blocks for the first time: compile and execute your program without project.



The screenshot shows the Code::Blocks IDE interface. On the left, there's a file browser titled 'Open files list' with 'Untitled1' selected. Below it is a 'Management' panel with 'Projects' selected, showing 'Workspace'. In the center, there's a code editor window with the following content:

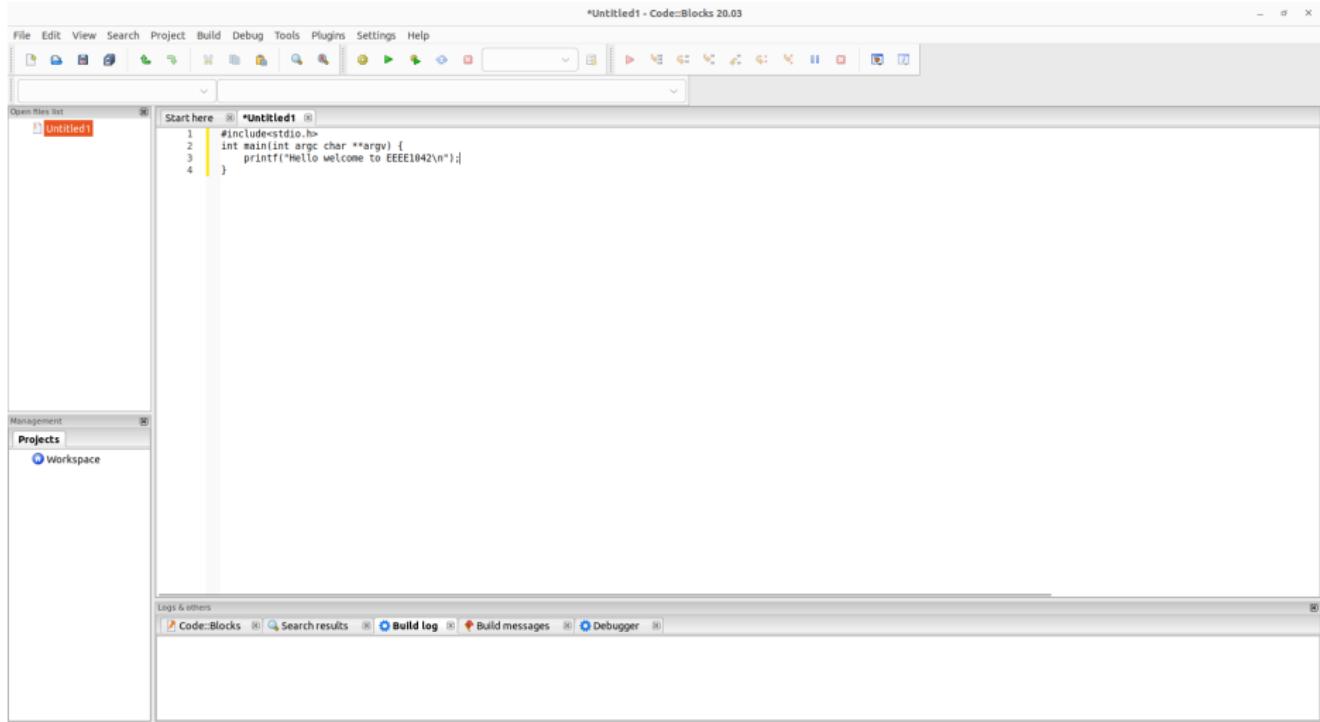
```
Start here #include <stdio.h>
int main()
{
    printf("Hello welcome to EEEE1042\n");
    return 0;
}
```

Below the code editor is a terminal window with a red border. It displays the following output:

```
Hello welcome to EEEE1042
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

A red callout box at the bottom left of the terminal window contains the text: "On pressing F9, Code::Blocks compiles the program and executes it in a new terminal window."

Run Code::Blocks for the first time: generate C-program with project.

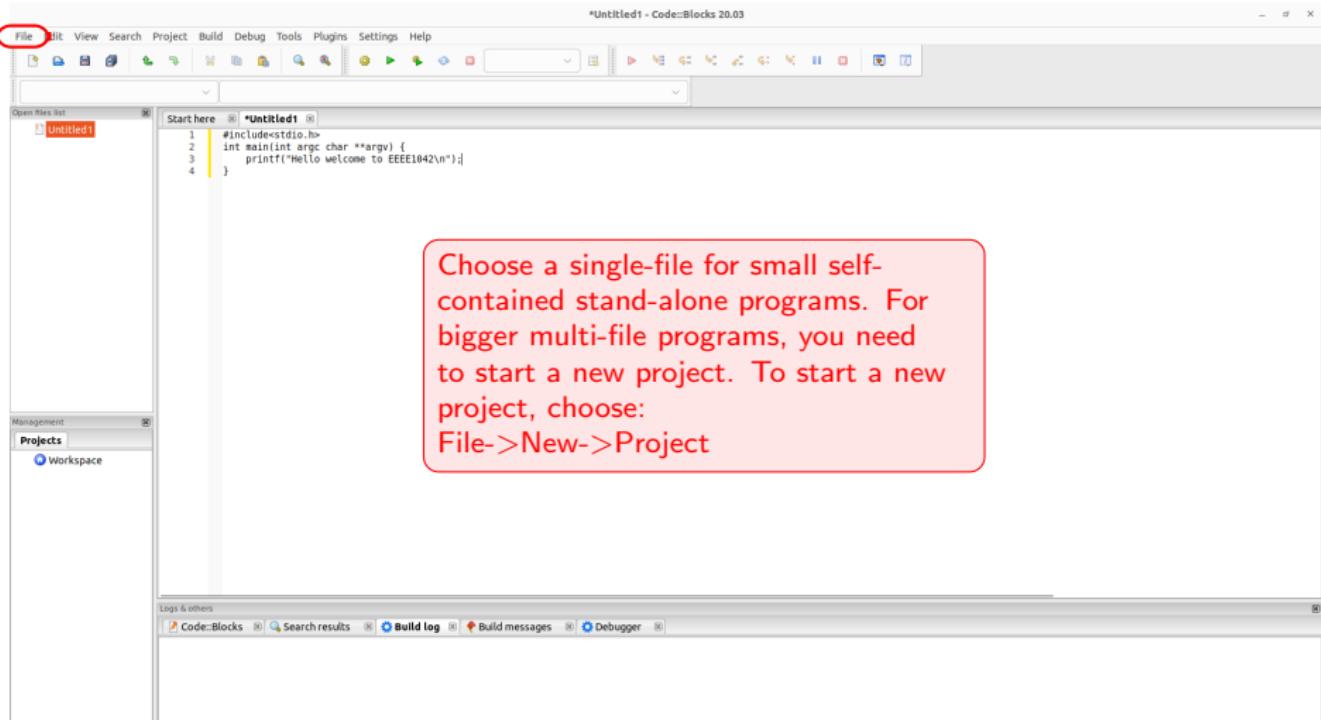


The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The title bar says "Untitled1 - Code::Blocks 20.03". The main window displays a code editor with the following C code:

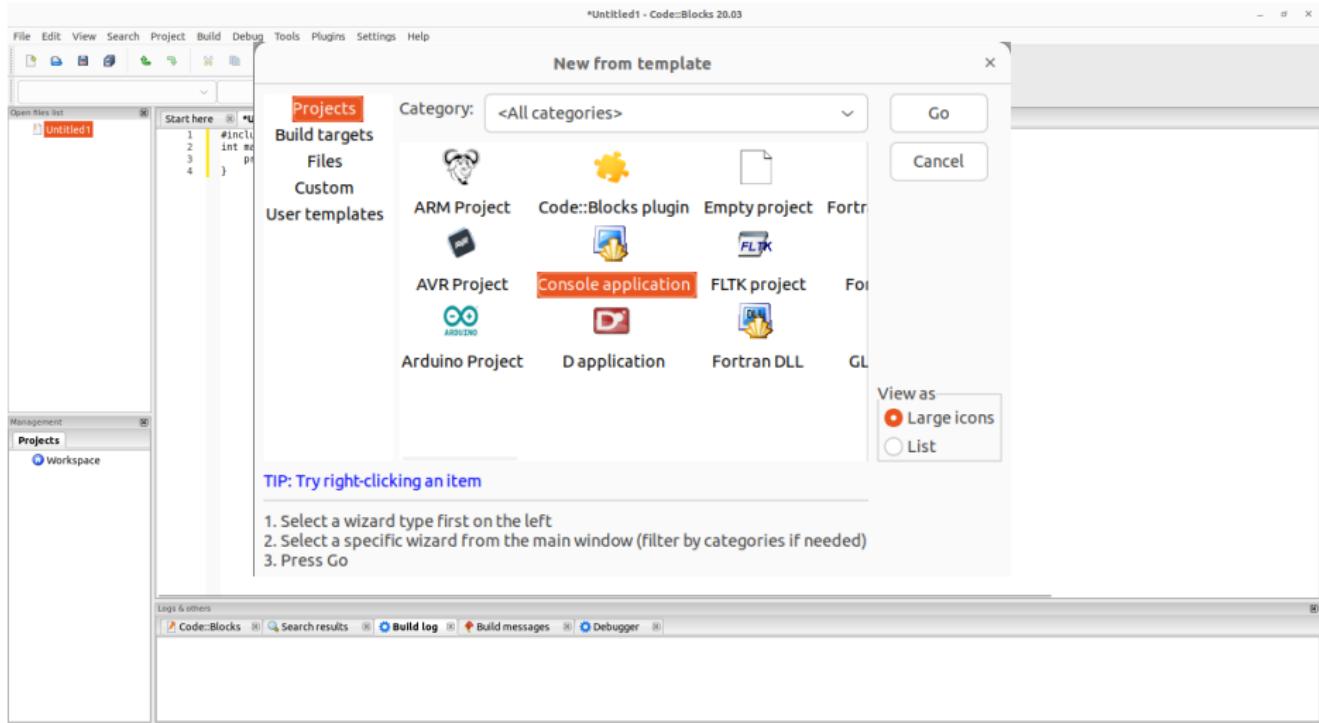
```
#include<stdio.h>
int main(int argc char **argv) {
    printf("Hello welcome to EEEE1042\n");
}
```

The left sidebar has an "Open files list" showing "Untitled1" and a "Management" section with "Projects" selected, showing "Workspace". The bottom status bar indicates "Unix (LF)", "UTF-8", "Line 3, Col 43, Pos 93", and "Insert Modif... Read/Wri... default".

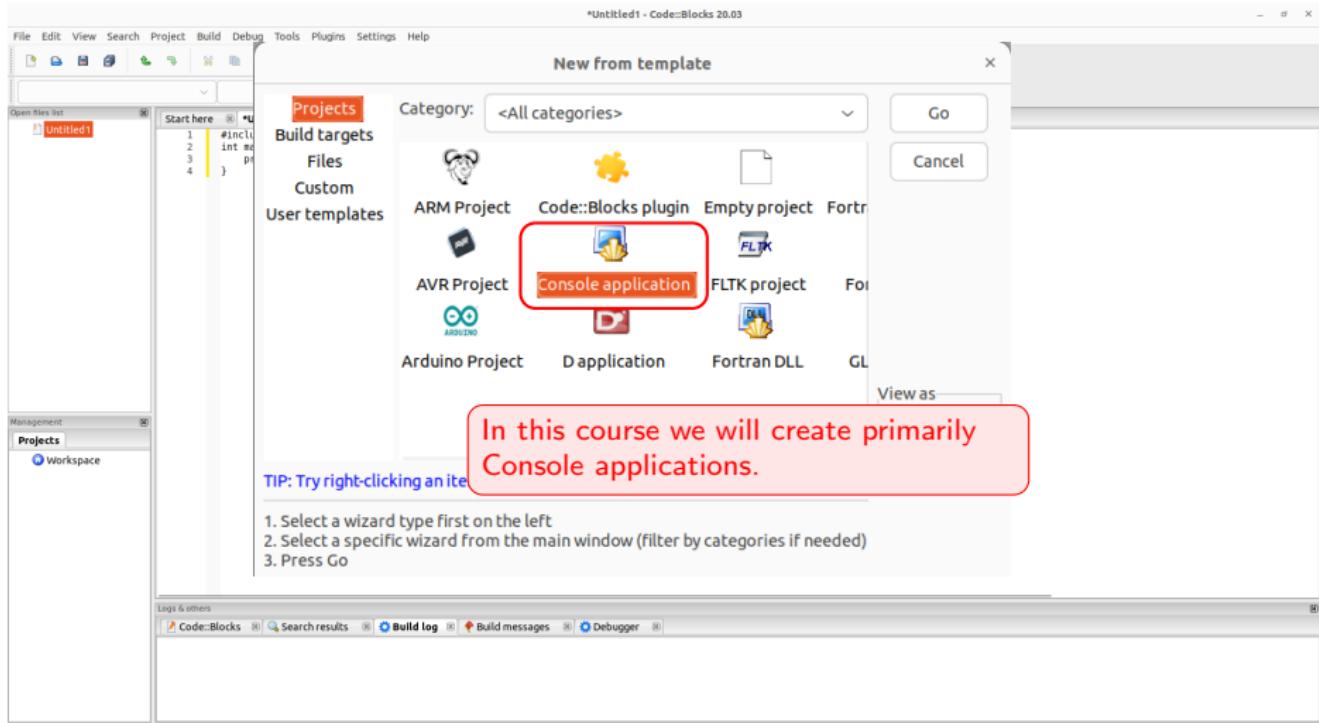
Run Code::Blocks for the first time: generate C-program with project.



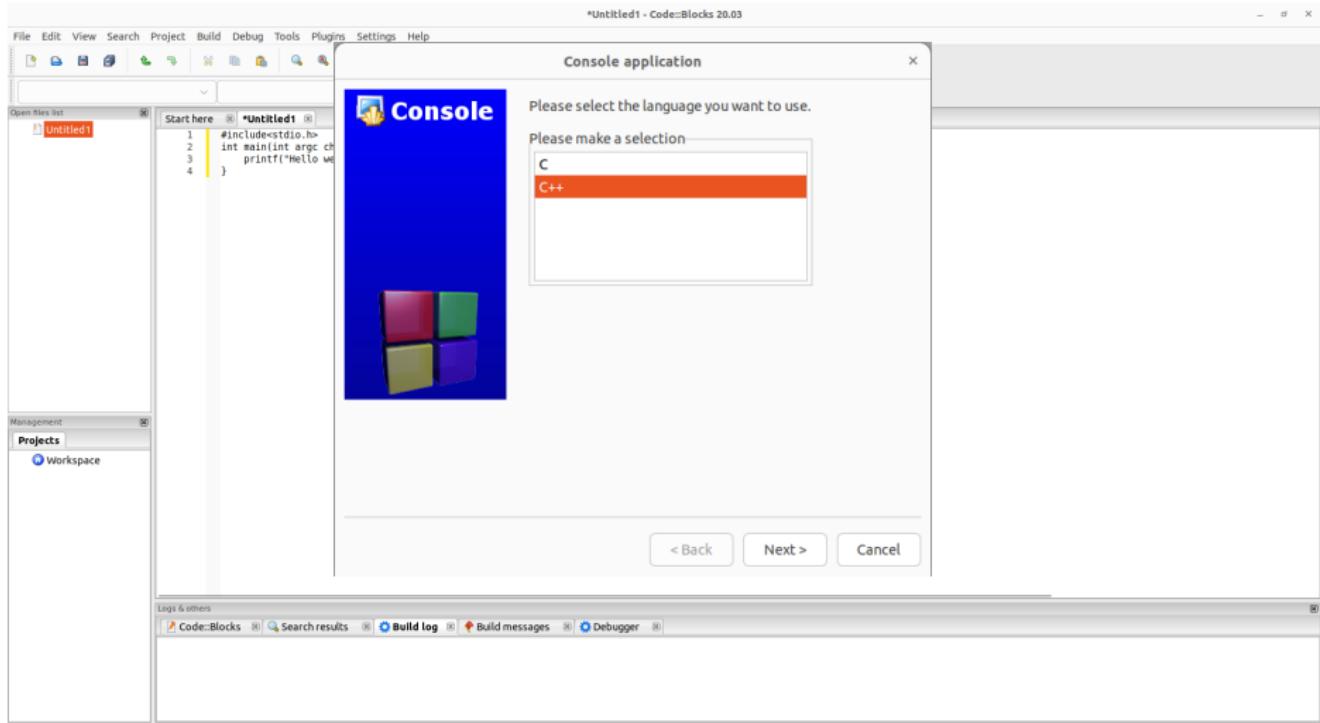
Run Code::Blocks for the first time: generate C-program with project.



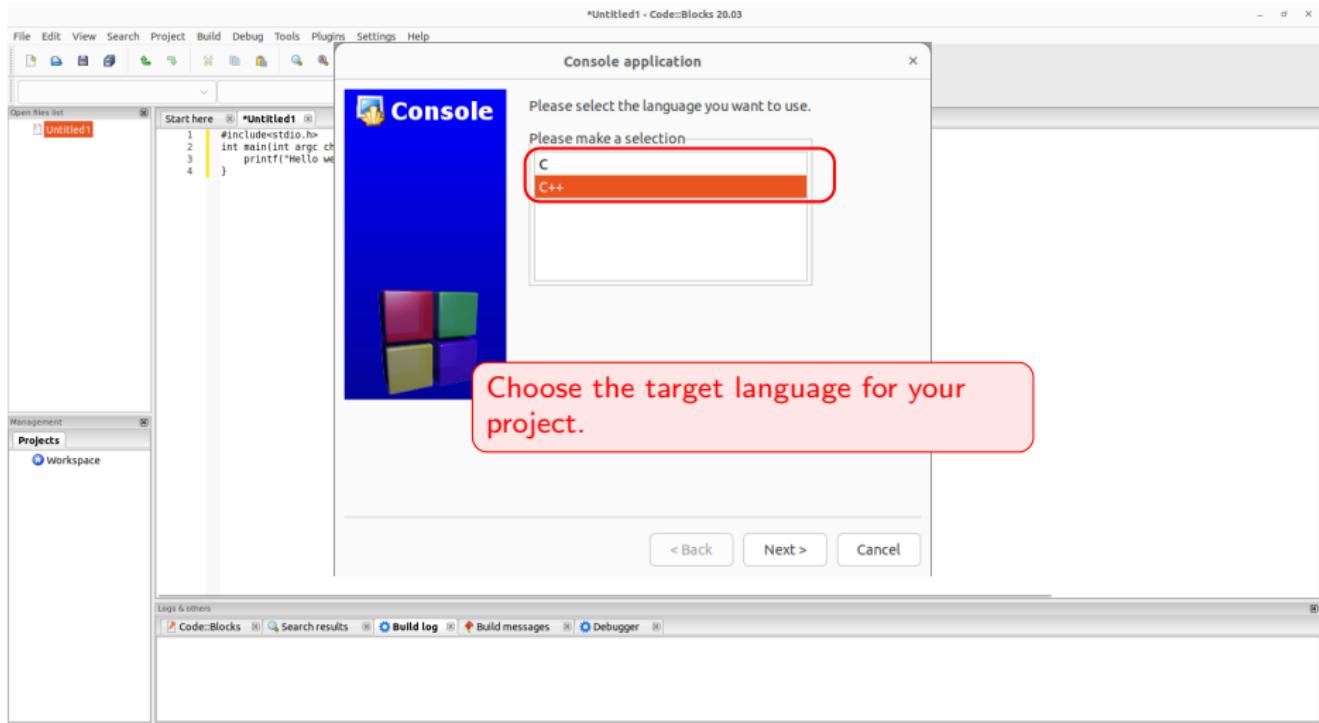
Run Code::Blocks for the first time: generate C-program with project.



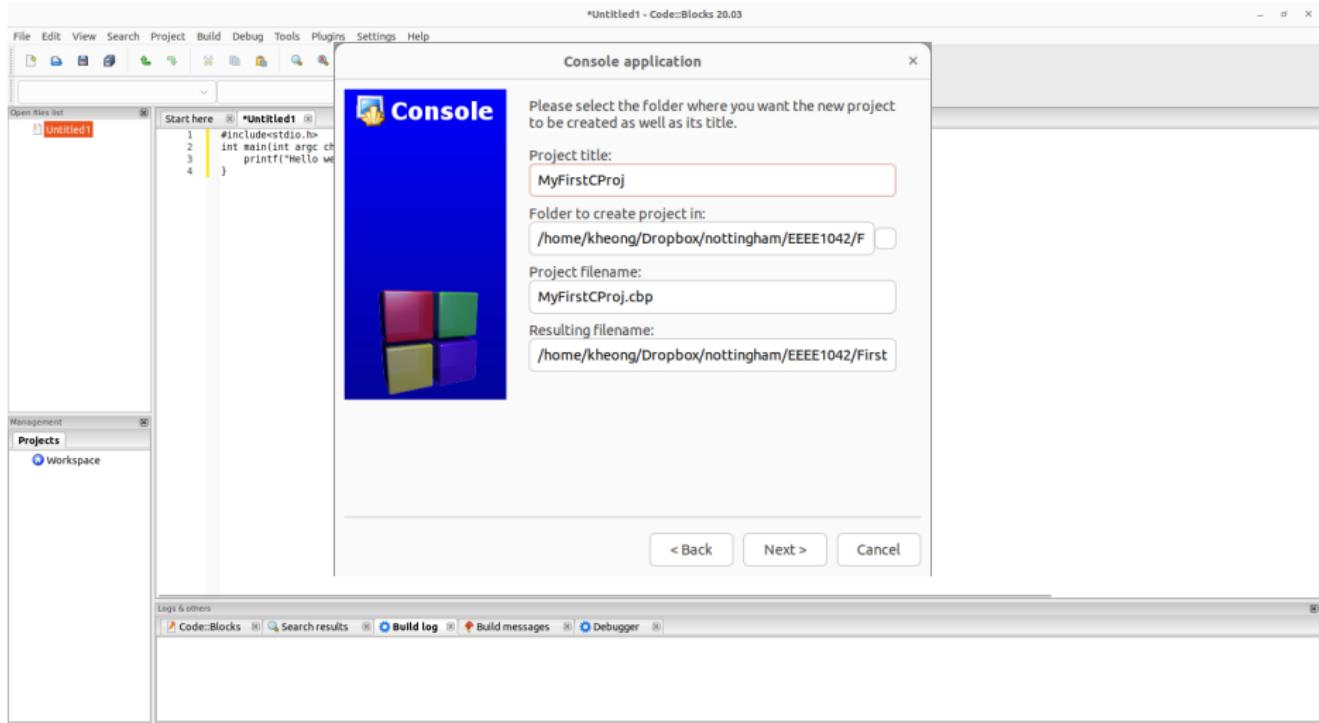
Run Code::Blocks for the first time: generate C-program with project.



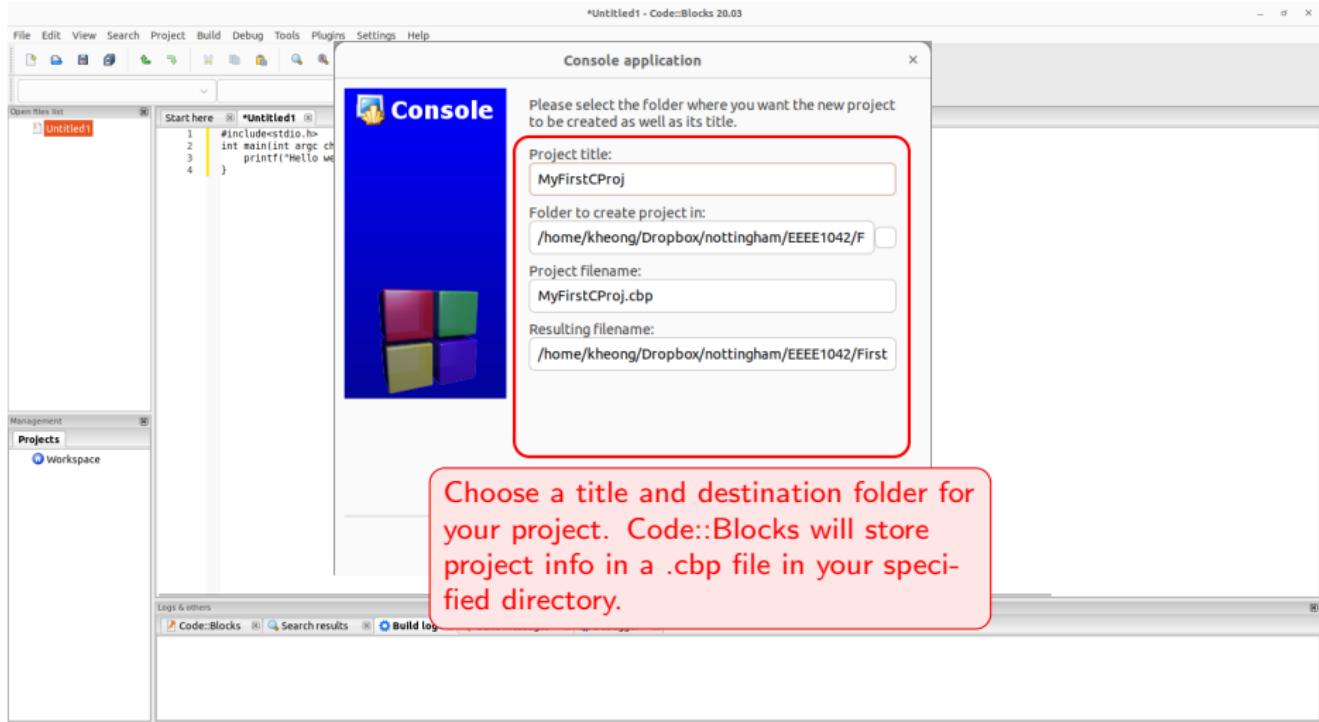
Run Code::Blocks for the first time: generate C-program with project.



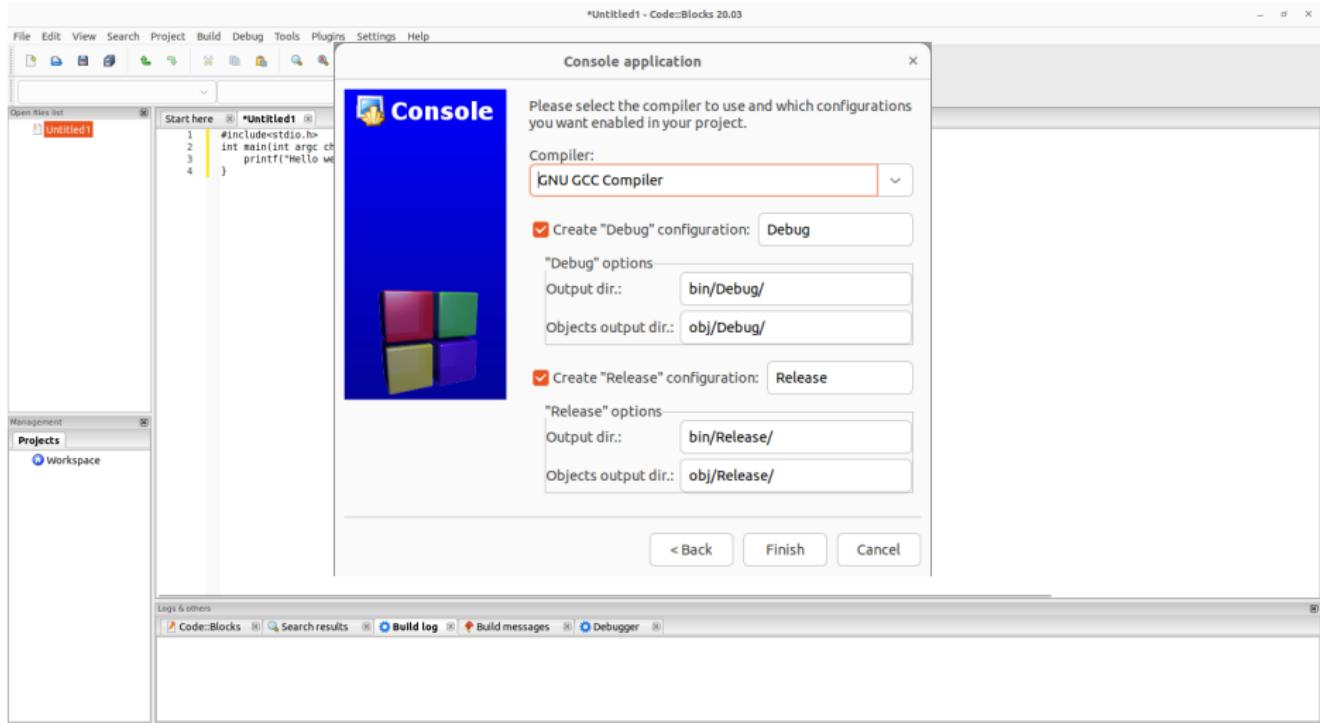
Run Code::Blocks for the first time: generate C-program with project.



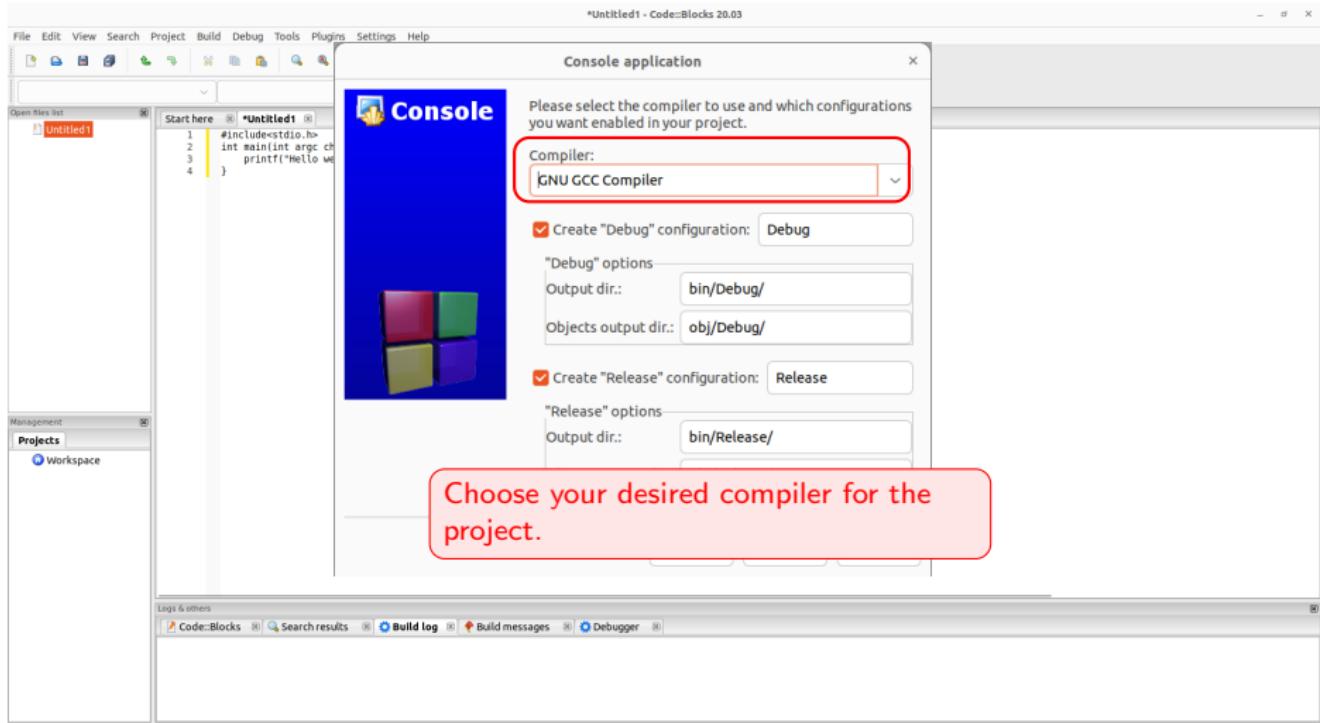
Run Code::Blocks for the first time: generate C-program with project.



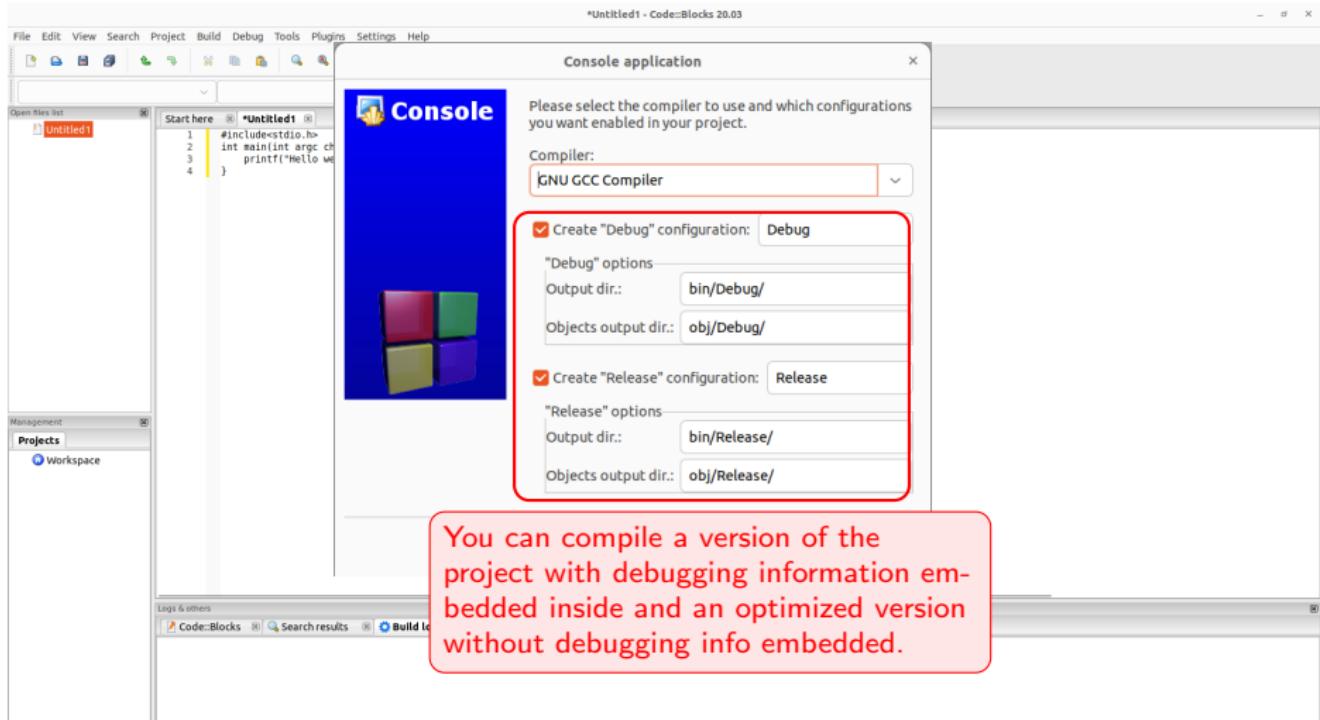
Run Code::Blocks for the first time: generate C-program with project.



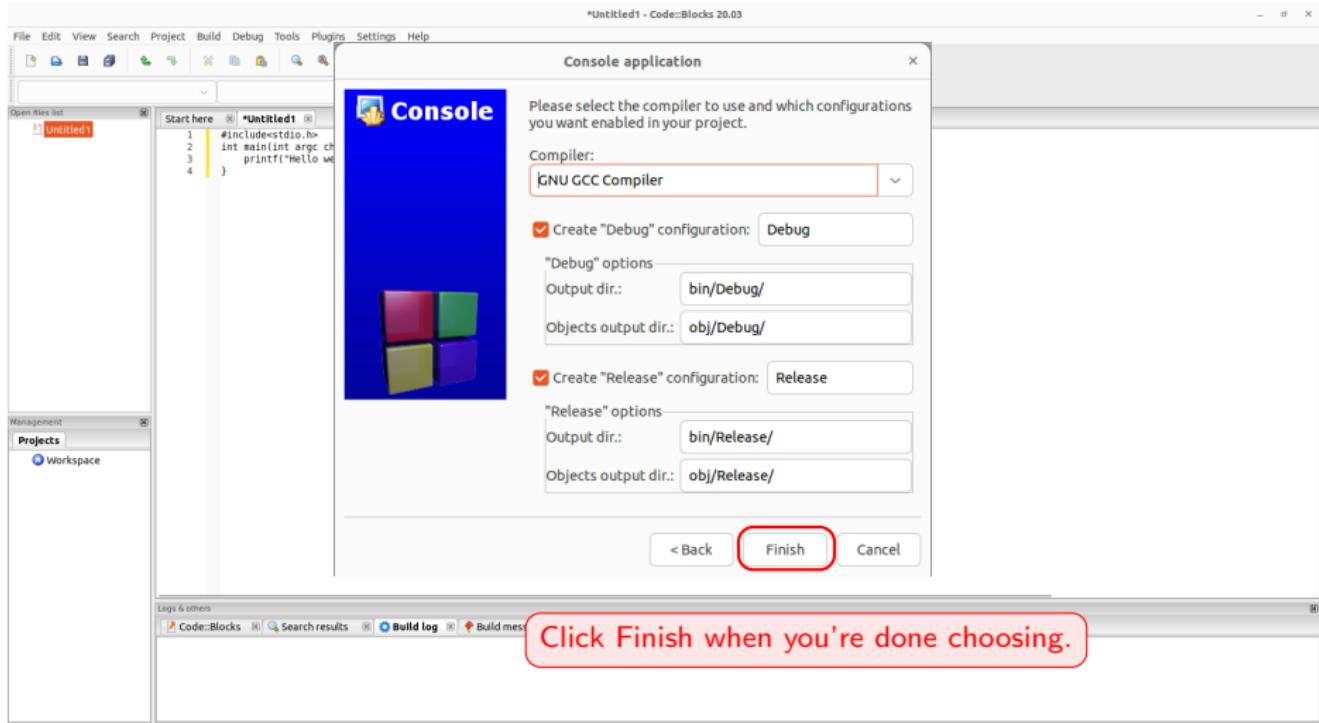
Run Code::Blocks for the first time: generate C-program with project.



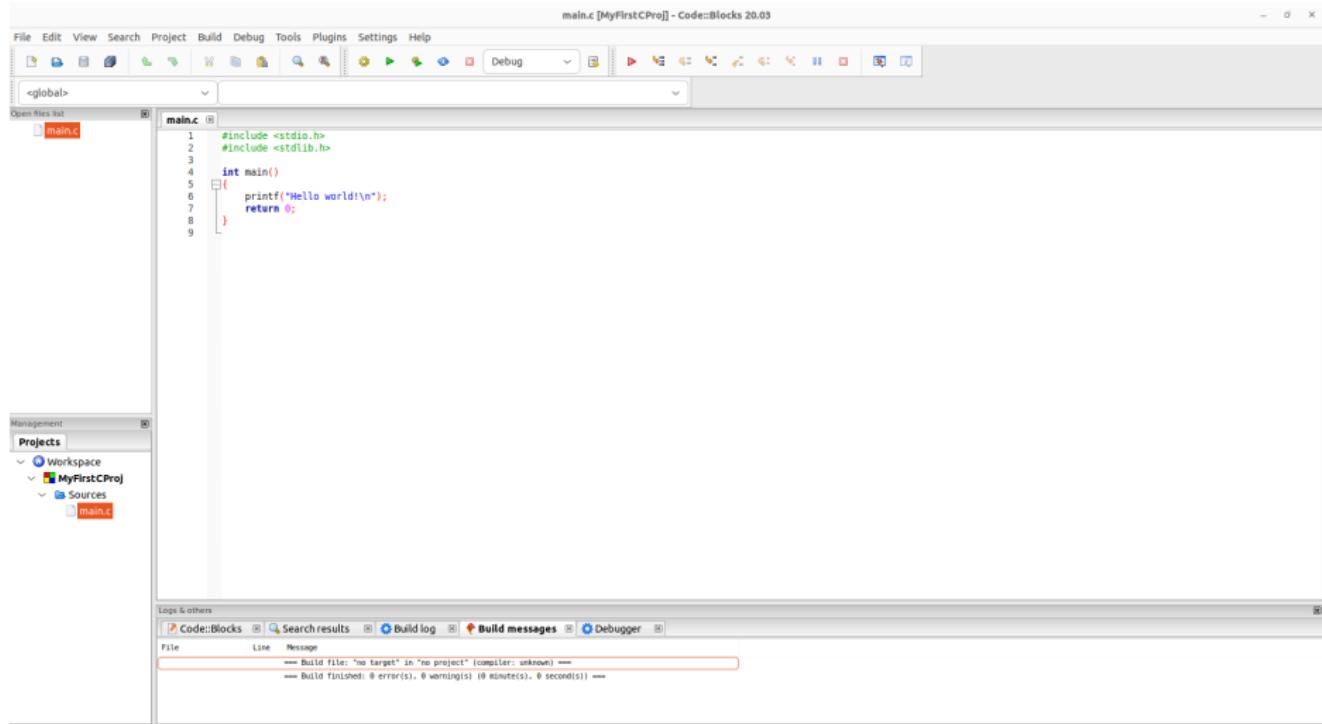
Run Code::Blocks for the first time: generate C-program with project.



Run Code::Blocks for the first time: generate C-program with project.



Run Code::Blocks for the first time: generate C-program with project.



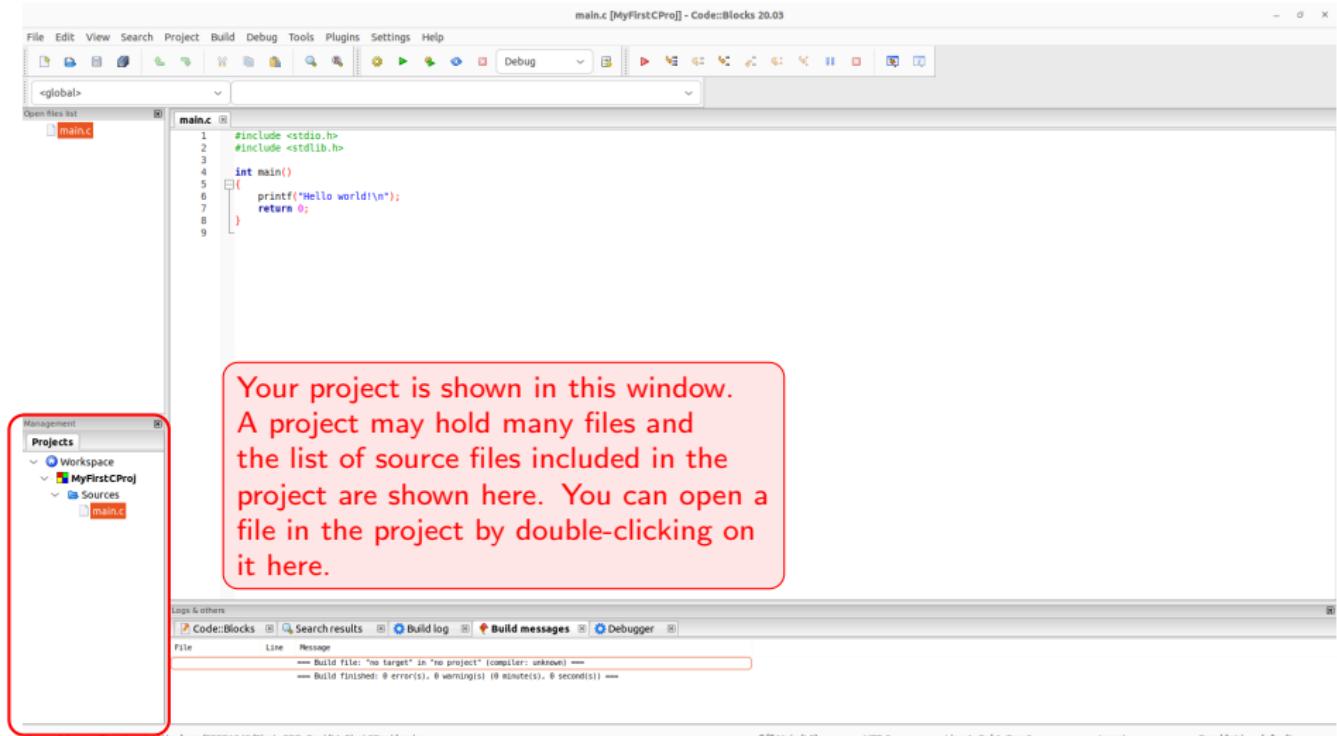
The screenshot shows the Code::Blocks IDE interface. The main window displays a C source code file named `main.c` with the following content:

```
<global>
main.c [R]
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

The left sidebar shows the project management, with a workspace named "MyFirstCProj" containing a "Sources" folder which includes the `main.c` file. The bottom panel shows the "Build messages" tab with the following output:

File	Line	Message
		==> Build file: "no target" in "no project" (compiler: unknown) ==>
		==> Build finished: 0 errors(s), 0 warning(s) (0 minute(s), 0 second(s)) ==>

Run Code::Blocks for the first time: generate C-program with project.



The screenshot shows the Code::Blocks IDE interface. At the top, the title bar reads "main.c [MyFirstCProj] - Code::Blocks 20.03". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. Below the menu is a toolbar with various icons. The main workspace contains a code editor with the following C code:

```
<global>
main.c (R)
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

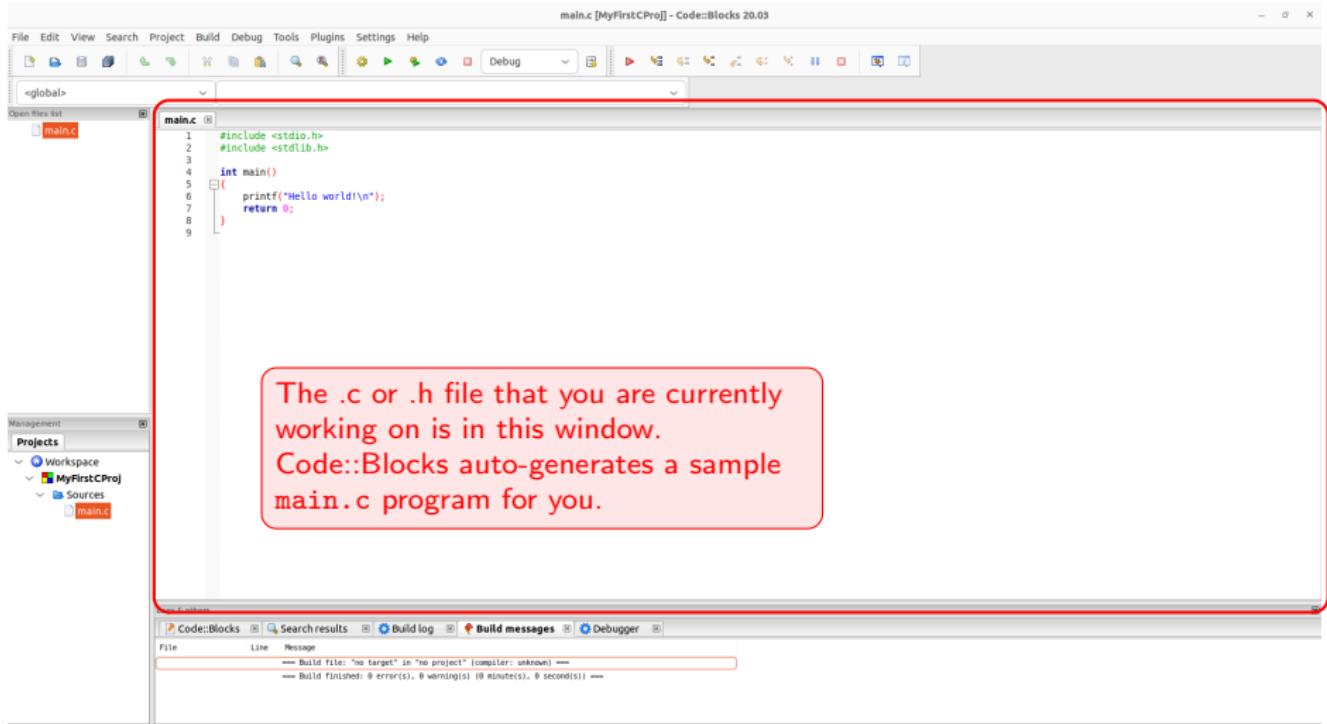
To the left of the code editor is the "Open files list" which shows "main.c". On the far left is the "Management" panel under "Projects", showing a workspace named "MyFirstCProj" with a source folder "Sources" containing "main.c". A red box highlights this "Management" panel. At the bottom of the interface is a "Logs & others" window showing build messages:

File	Line	Message
		==> Build file: "no target" in "no project" (compiler: unknown) ==
		==> Build finished: 0 errors(s), 0 warning(s) (0 minute(s), 0 second(s)) ==

A red box also highlights the "Logs & others" window. The status bar at the bottom displays the path "/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c", the compiler "C/C4Linux (LF)", encoding "UTF-8", position "Line 1, Col 1, Pos 0", and other settings like "Insert" and "Read/Wri... default".

Your project is shown in this window.
A project may hold many files and
the list of source files included in the
project are shown here. You can open a
file in the project by double-clicking on
it here.

Run Code::Blocks for the first time: generate C-program with project.



The screenshot shows the Code::Blocks IDE interface. The main window displays a code editor with the file "main.c" open. The code contains a simple "Hello world!" program:

```
<global>
main.c [R]
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

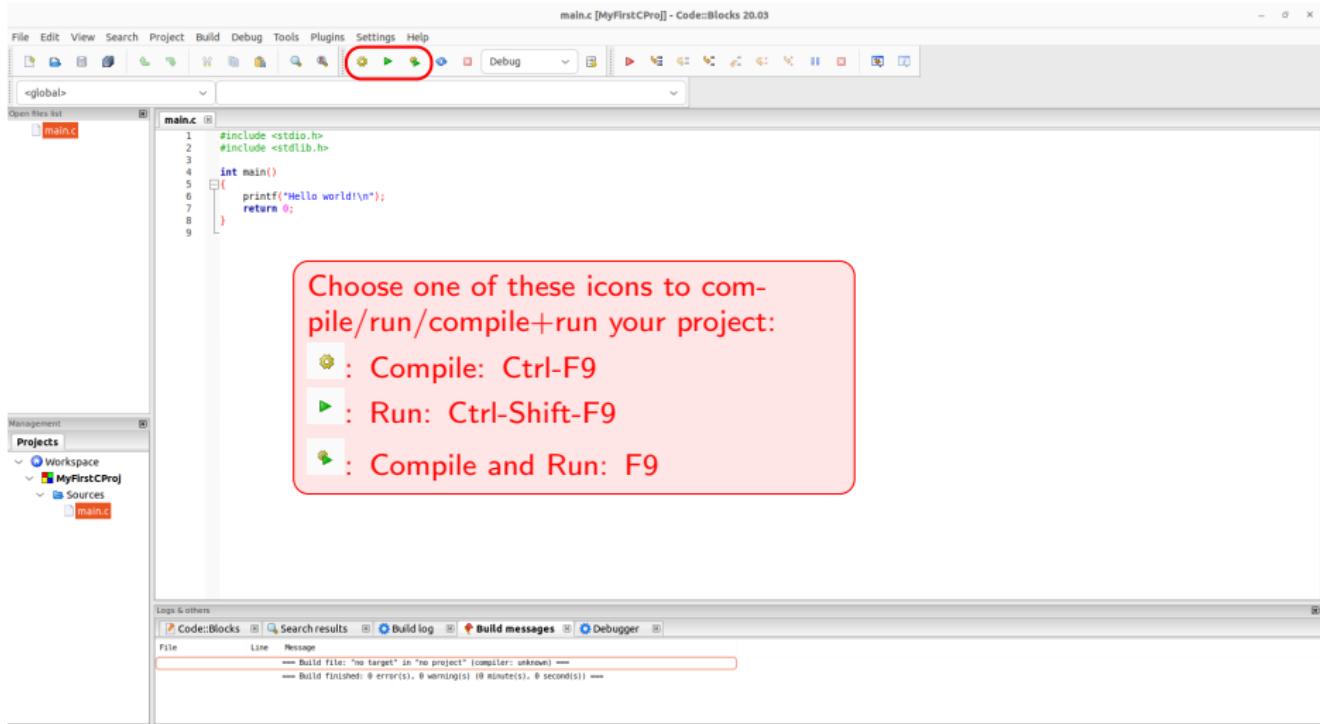
A red box highlights the code editor area. A callout bubble in the center of this box contains the following text:

The .c or .h file that you are currently working on is in this window.
Code::Blocks auto-generates a sample main.c program for you.

At the bottom of the interface, the "Build messages" tab is active, showing the following output:

```
Code::Blocks [ ] Search results [ ] Build log [ ] Build messages [ ] Debugger [ ]
File Line Message
--- --- ---
==> Build file: "no target" in "no project" (compiler: unknown) ===
==> Build finished: 0 errors(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

Run Code::Blocks for the first time: generate C-program with project.



Choose one of these icons to compile/run/compile+run your project:

-  : Compile: Ctrl-F9
-  : Run: Ctrl-Shift-F9
-  : Compile and Run: F9

Run Code::Blocks for the first time: generate C-program with project.

main.c [MyFirstCProj] - Code::Blocks 20.03

File Edit View Search Project Build Debug Tools Plugins Settings Help

Debug

<global> main() : int

Open files list main.c

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

Management Projects

Workspace MyFirstCProj Sources main.c

Logs & others

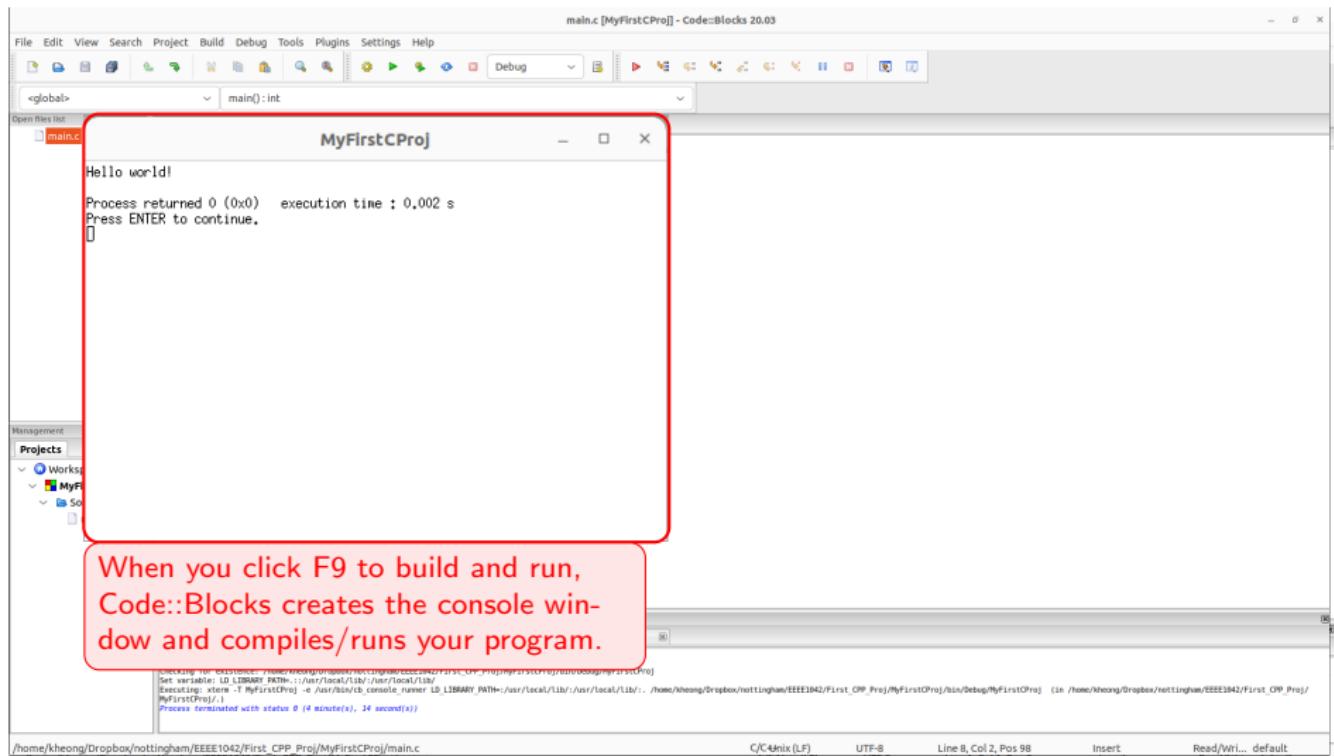
Code::Blocks Search results Build log Build messages Debugger

```
***** Run: Debug in MyFirstCProj (compiler: GNU GCC Compiler) *****
Code::Blocks executing: /usr/bin/g++ -o /home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj
Set working AD LIBRARY PATH=:/usr/local/lib:/usr/local/lib64:/usr/lib:/usr/lib64:/lib:/lib64:/usr/local/lib:/lib:/usr/local/lib64:/lib64:/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj
Executing: xterm -T MyFirstCProj -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64:/lib:/lib64:/usr/local/lib:/lib:/usr/local/lib64:/lib64:/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj .. (in /home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj)
MyFirstCProj
Process terminated with status 0 (4 minutes(s), 14 second(s))

/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c
/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c
```

C/C++(LF)	UTF-8	Line 8, Col 2, Pos 98	Insert	Read/Wr... default
C/C++(LF)	UTF-8	Line 1, Col 1, Pos 0	Insert	Read/Wr... default

Run Code::Blocks for the first time: generate C-program with project.



The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The main window displays the code in 'main.c' and the output of the program's execution. A red box highlights the terminal window where the output is shown. The terminal output is:

```
Hello world!  
Process returned 0 (0x0)   execution time : 0.002 s  
Press ENTER to continue.
```

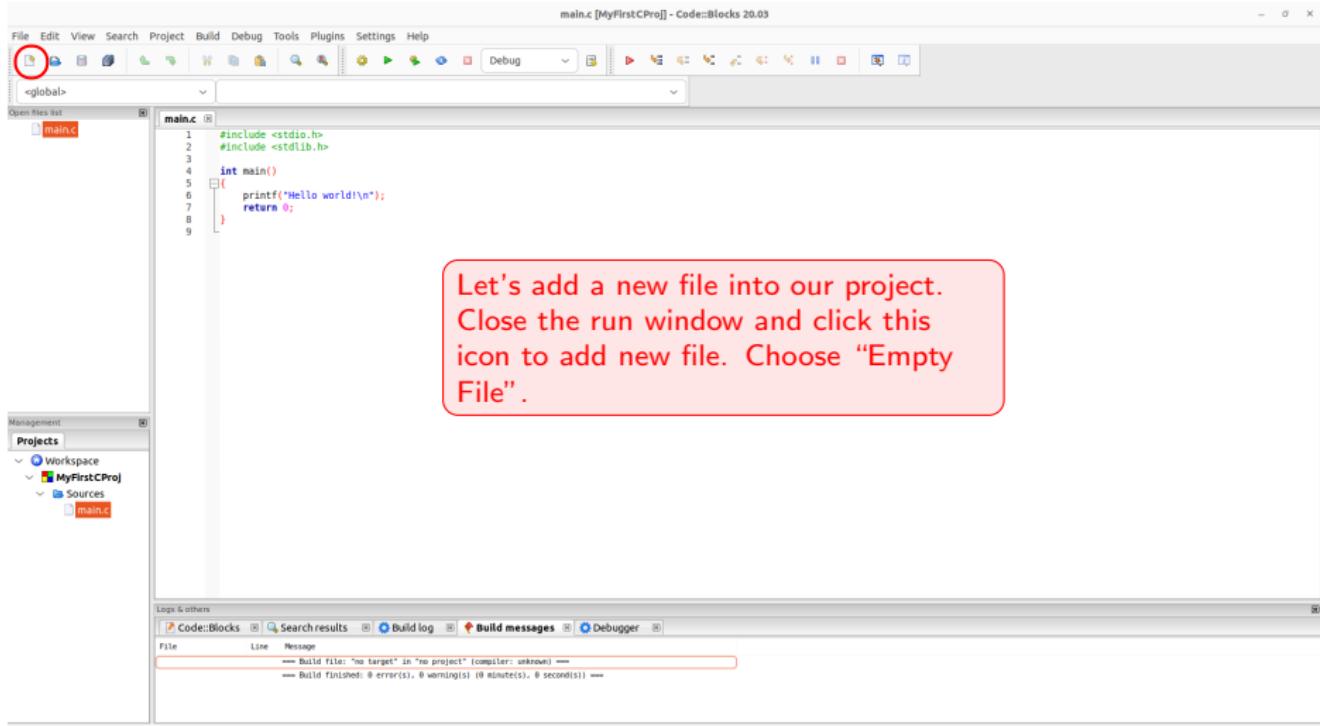
The 'Management Projects' panel shows a workspace named 'MyF' containing a project named 'MyFirstCProj'. A red callout box points to this panel with the text: "When you click F9 to build and run, Code::Blocks creates the console window and compiles/runs your program."

At the bottom of the terminal window, there is a scrollback area showing command-line details:

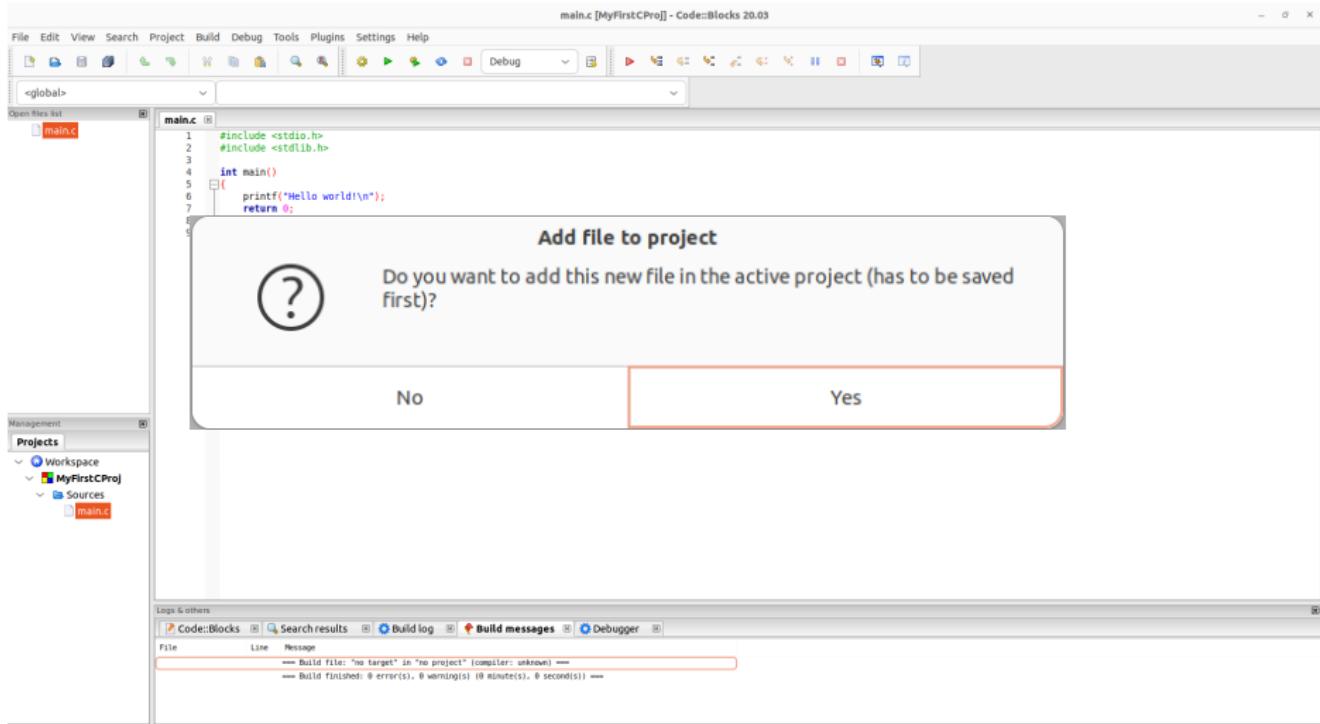
```
Set variable LD_LIBRARY_PATH=::/usr/local/lib:/usr/local/lib64  
Executing: xterm -T MyFirstCProj -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64:./home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/bin/debug/MyFirstCProj  
MyFirstCProj  
Process terminated with status 0 (4 minutes(s), 14 second(s))
```

The status bar at the bottom right shows file paths and file statistics: C:\44m\ (LF), UTF-8, Line 8, Col 2, Pos 98; C:\44m\ (LF), UTF-8, Line 1, Col 1, Pos 0; Insert, Insert; Read/Wr..., default; Read/Wr..., default.

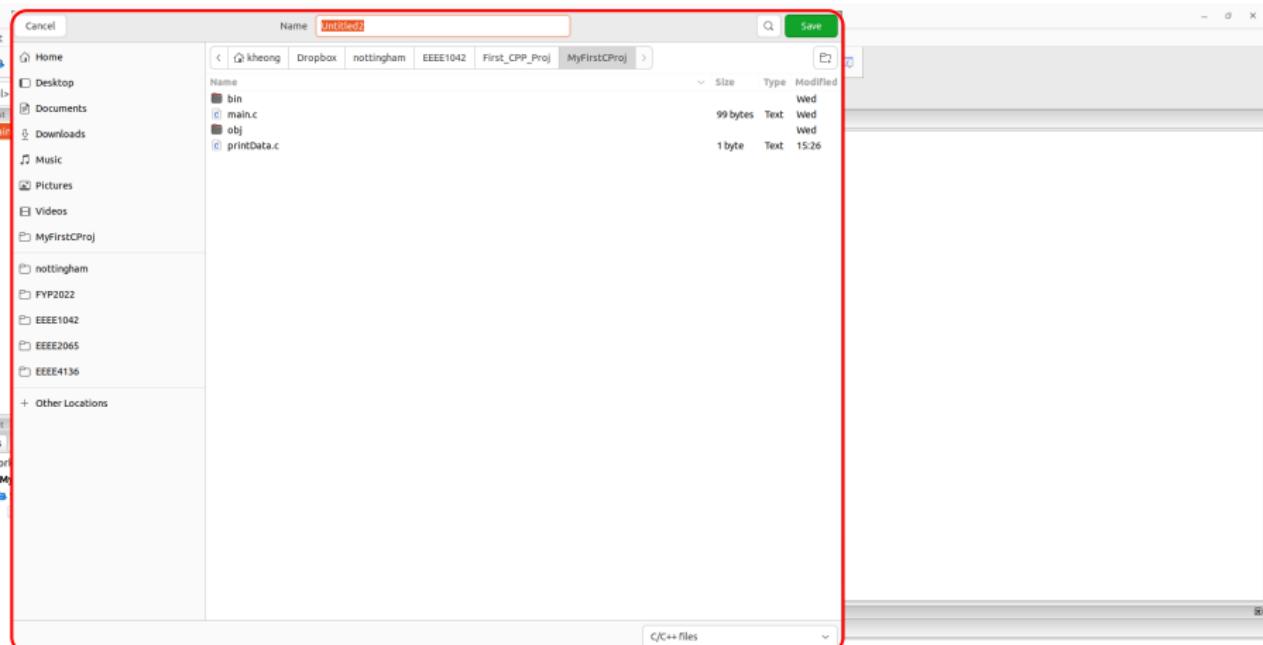
Run Code::Blocks for the first time: generate C-program with project.



Run Code::Blocks for the first time: generate C-program with project.

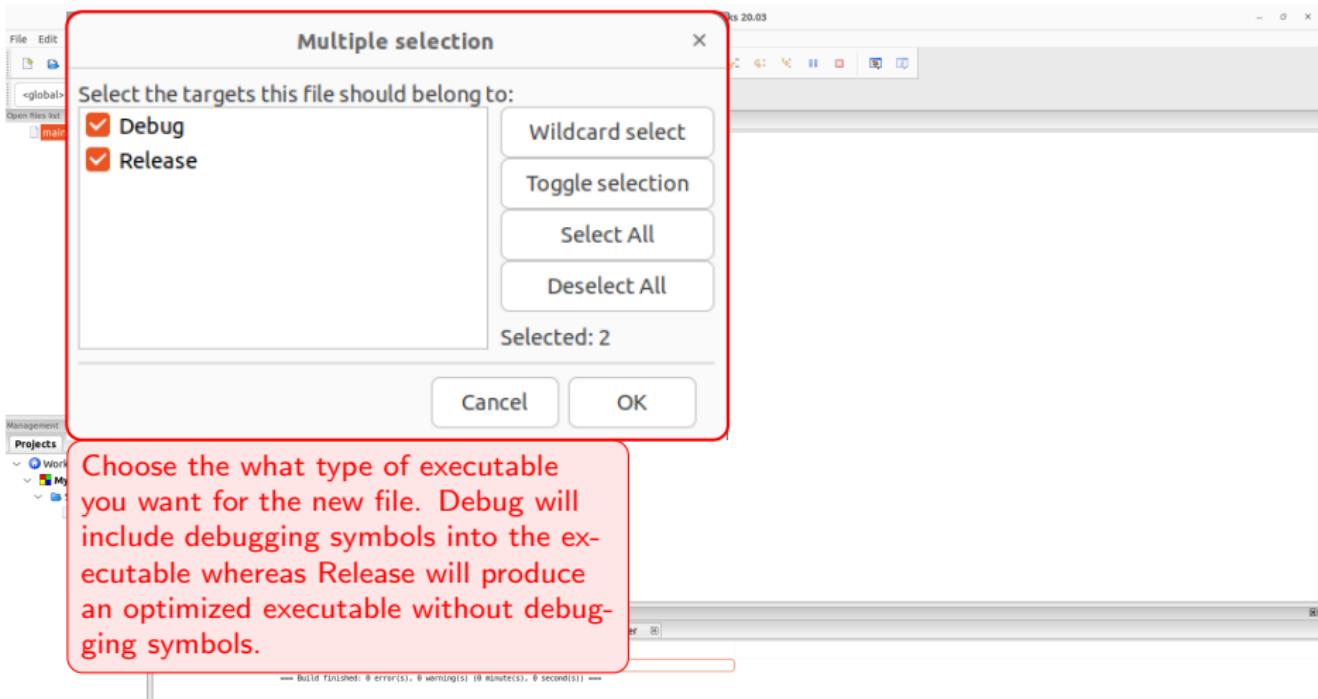


Run Code::Blocks for the first time: generate C-program with project.

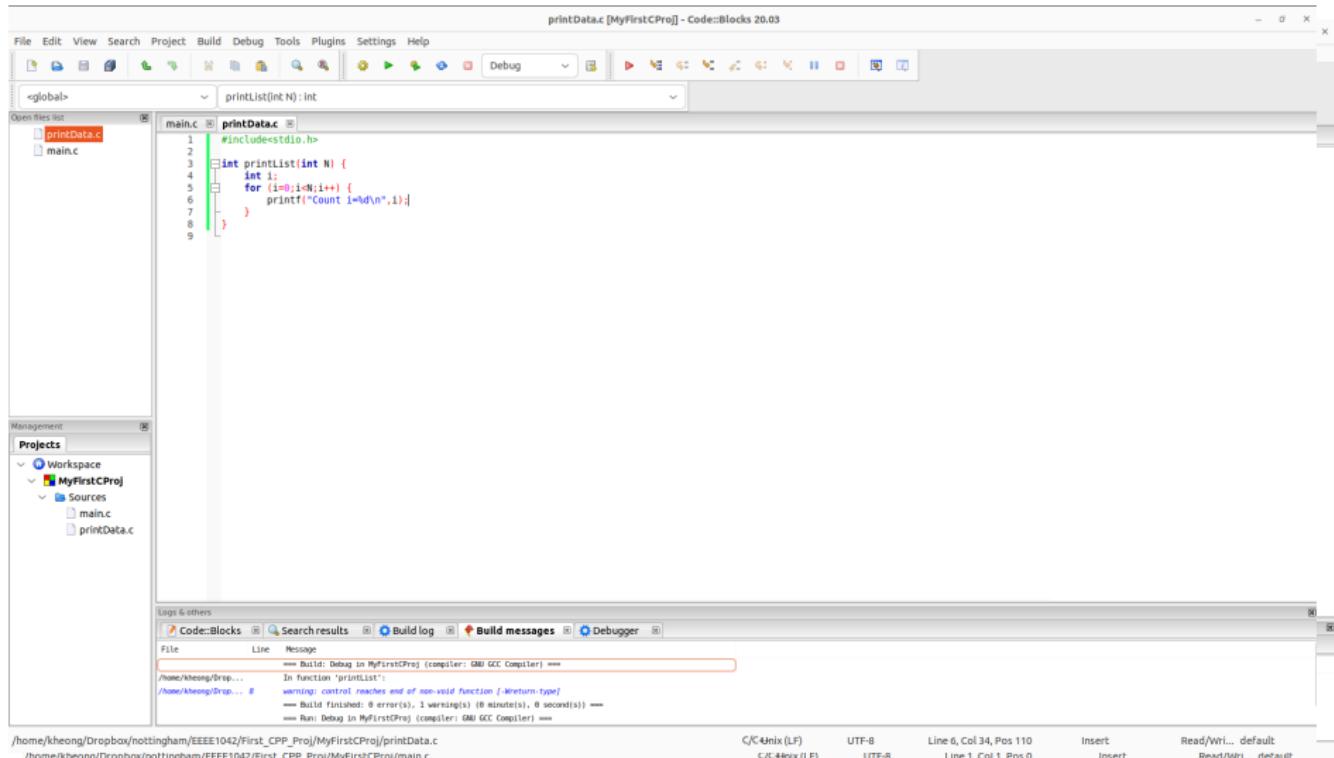


Choose the name of the new file you want to add to your project.

Run Code::Blocks for the first time: generate C-program with project.



Run Code::Blocks for the first time: generate C-program with project.



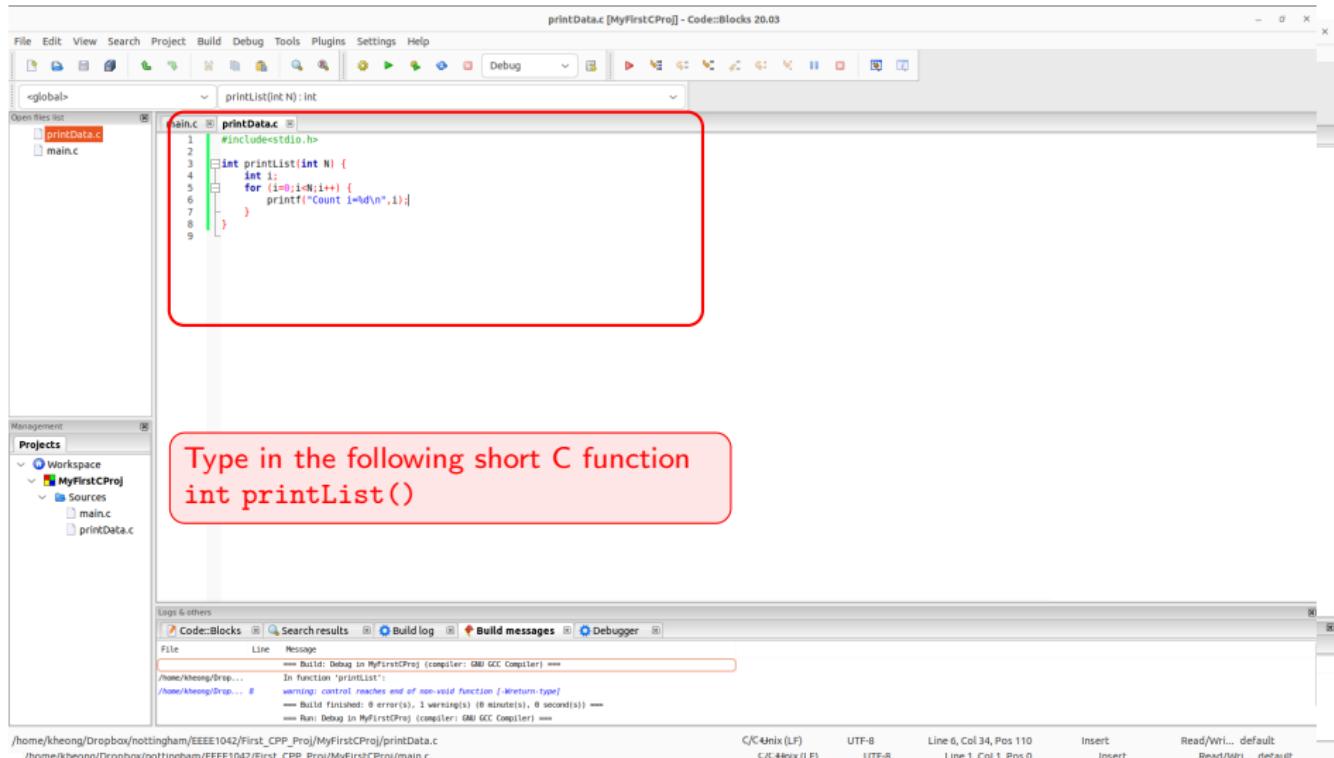
The screenshot shows the Code::Blocks IDE interface. The title bar says "printData.c [MyFirstCProj] - Code::Blocks 20.03". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The code editor window displays two files: "main.c" and "printData.c". "main.c" contains a simple main function. "printData.c" contains a function "printList" that prints a sequence of numbers from 1 to N. The Projects panel shows a workspace named "MyFirstCProj" containing sources "main.c" and "printData.c". The Logs & others panel shows the build messages. The build log output is as follows:

```
== Build: Debug in MyFirstCProj (compiler: GNU GCC Compiler) ==
/home/kheong/Drop... In function `printList':
/home/kheong/Drop... warning: control reaches end of non-void function [-Wreturn-type]
== Build finished: 0 errors(s), 1 warning(s) (0 minutes, 0 second(s)) ==
== Run: Debug in MyFirstCProj (compiler: GNU GCC Compiler) ==

```

The command line at the bottom shows the paths to the source files: "/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/printData.c" and "/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c". The status bar at the bottom right shows file information: Line 6, Col 34, Pos 110 for printData.c and Line 1, Col 1, Pos 0 for main.c, along with Insert and Read/Wr... default buttons.

Run Code::Blocks for the first time: generate C-program with project.



File Edit View Search Project Build Debug Tools Plugins Settings Help

printData.c [MyFirstCProj] - Code::Blocks 20.03

printList(int N): int

```
main.c printData.c
1 #include<stdio.h>
2
3 int printList(int N) {
4     int i;
5     for (i=0;i<N;i++) {
6         printf("Count = %d\n",i);
7     }
8 }
```

Open files list

Projects

Management

Logs & others

Type in the following short C function

int printList()

```
File Line Message
=====  
/home/kheong/Drop... In function `printList':  
/home/kheong/Drop... warning: control reaches end of non-void function [-Wreturn-type]  
=====  
Build finished: 0 errors(s), 1 warning(s) (0 minutes, 0 second(s))  
=====  
Run: Debug in MyFirstCProj (compiler: GNU GCC Compiler)
```

/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/printData.c
/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c

C/C++/Unix (LF) UTF-8 Insert Read/Wr... default
C/C++/Mix (LF) UTF-8 Line 6, Col 34, Pos 110 Line 1, Col 1, Pos 0 Insert Read/Wr... default
Read/Wr... default



Run Code::Blocks for the first time: generate C-program with project.

File Edit View Search Project Build Debug Tools Plugins Settings Help

<global> printList(int N) : int

Open files list

printData.c main.c

Management

Projects

Workspace MyFirstCProj Sources main.c printData.c

Logs & others

Code:Blocks Search results Build log Build messages Debugger

File Line Message

==== Build: Debug In MyFirstCProj (compiler: GNU GCC Compiler) ====
/home/kheong/Dropbox/... In function 'printList':
/home/kheong/Dropbox/... # warning: control reaches end of non-void function [-Wreturn-type]
==== Build finished: 0 error(s), 1 warning(s) (0 minutes, 0 second(s)) ====
Run: Debug In MyFirstCProj (compiler: GNU GCC Compiler) ===

/home/kheong/Dropbox/Nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/printData.c

C/C++ source (LF) UTF-8 Line 6, Col 34, Pos 110 Insert Read/Wr... default
C/C++ Header (LF) UTF-8 Line 1, Col 1, Pos 0 Insert Read/Wr... default

Run Code::Blocks for the first time: generate C-program with project.

main.c [MyFirstCProj] - Code::Blocks 20.05

File Edit View Search Project Build Debug Tools Plugins Settings Help

<global> main() : int

main.c printData.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     printlist();
8     return 0;
9 }
10

```

Open files list

Projects

MyFirstCProj

Sources

main.c

printData.c

Logs & others

Code::Blocks Search results Build log Build messages Debugger

```

----- Run Dialog in MyFirstCProj (compiler: GM GCC Compiler) -----
Checking for existence: /home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj
Set variable: LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64:/usr/local/lib/x86_64-linux-gnu:/lib:/lib64
Executing: xtrem -T MyFirstCProj -e /usr/bin/tb_console_runner LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64:/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj /MyFirstCProj...

```

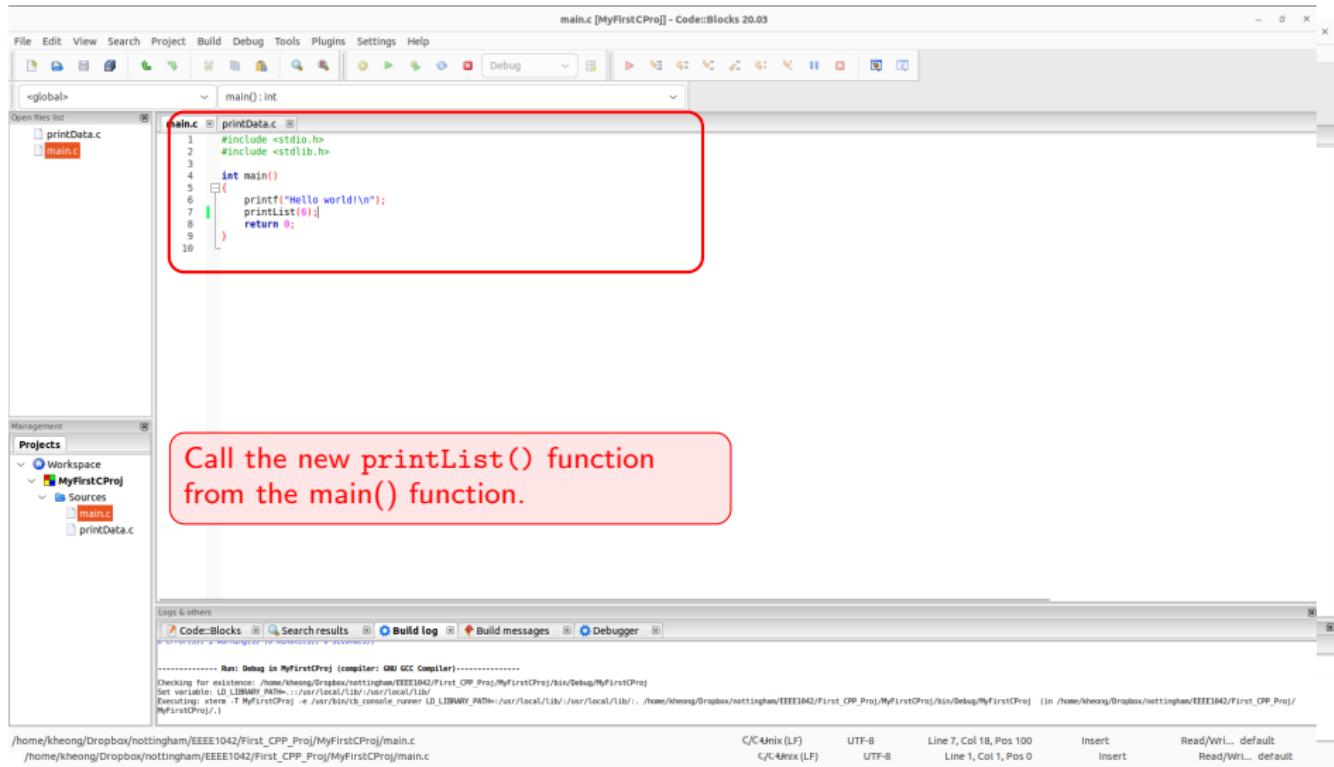
```

/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c
/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c

```

C/C++/Unix (LF)	UTF-8	Line 7, Col 18, Pos 100	Insert	Read/Wr... default
C/C++/Unix (LF)	UTF-8	Line 1, Col 1, Pos 0	Insert	Read/Wr... default

Run Code::Blocks for the first time: generate C-program with project.



The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The title bar says "main.c [MyFirstCProj] - Code::Blocks 20.05". The toolbar has various icons for file operations like Open, Save, Build, Run, and Debug. The code editor window displays two files: "main.c" and "printData.c". The "main.c" file contains the following code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!\n");
    printList();
    return 0;
}
```

A red box highlights the line "printList();". The Projects panel shows a workspace named "MyFirstCProj" with a source folder containing "main.c" and "printData.c". The status bar at the bottom shows the command line path and some build log information.

Call the new `printList()` function from the `main()` function.

21/28

Run Code::Blocks for the first time: generate C-program with project.

main.c [MyFirstCProj] - Code::Blocks 20.05

File Edit View Search Project Build Debug Tools Plugins Settings Help

main() : int

main.c printData.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     printList();
8     return 0;
9 }
```

Click the build/run icon, or press F9 to build and run your project which now has 2 files in it. The main() function is now calling the printList() function in the printData.c file.

Management Projects

Workspace MyFirstCProj Sources main.c printData.c

Logs & others

Code::Blocks Search results Build log Build messages Debugger

```

----- Run Dialog in MyFirstCProj (compiler: GNU GCC Compiler) -----
Checking for existence: /home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj
Set variable: LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64
Executing: xtrem -T MyFirstCProj -e /usr/bin/tb_console_runner LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64:/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/bin/Debug/MyFirstCProj /MyFirstCProj

```

```

/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c
/home/kheong/Dropbox/nottingham/EEEE1042/First_CPP_Proj/MyFirstCProj/main.c
```

C/C++(Unix (LF))	UTF-8	Line 7, Col 18, Pos 100	Insert	Read/Wr... default
C/C++(Win (LF))	UTF-8	Line 1, Col 1, Pos 0	Insert	Read/Wr... default

Run Code::Blocks for the first time: generate C-program with project.

The screenshot shows the Code::Blocks IDE interface. The top menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The title bar indicates "main.c [MyFirstCProj] - Code::Blocks 20.03". The left sidebar has an "Open files list" with "printData.c" and "main.c" selected. The "Management" section shows a "Projects" tree with "MyFirstCProj" expanded, revealing "Sources" containing "main.c" and "printData.c". The main workspace contains two code editors: "main.c" and "printData.c". The "main.c" editor shows the following code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!\n");
    printList();
    return 0;
}
```

The "printData.c" editor shows the implementation of the `printList()` function:

```
int printList()
{
    int i=0;
    Count i=1;
    Count i=2;
    Count i=3;
    Count i=4;
    Count i=5
}
```

A red box highlights the output window titled "MyFirstCProj" which displays the program's output:

```
Hello world!
Count i=0
Count i=1
Count i=2
Count i=3
Count i=4
Count i=5

Process returned 0 (0x0)   execution time : 0.290 s
Press ENTER to continue.
```

Now we can see that the `main()` program has called the subfunction `printList()` that resides within the `printData.c` file that is part of the same project.

Maintainable, Modular, Re-useable Code

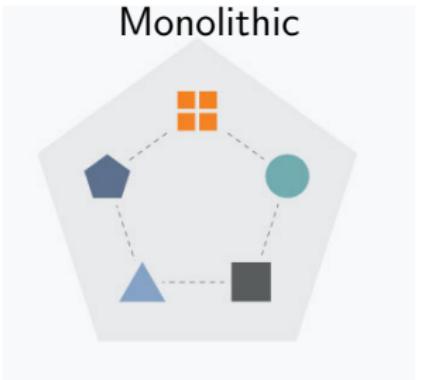
- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.

Maintainable, Modular, Re-useable Code

- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.

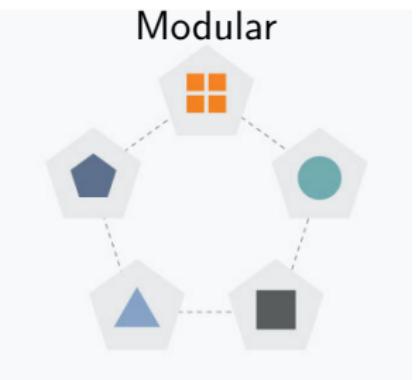
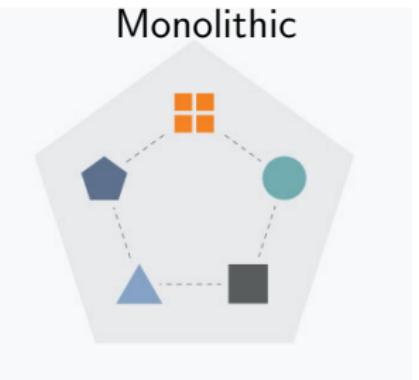
Maintainable, Modular, Re-useable Code

- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.
- Monolithic: One single big monolithic chunk of code → **unmaintainable, not reusable**.



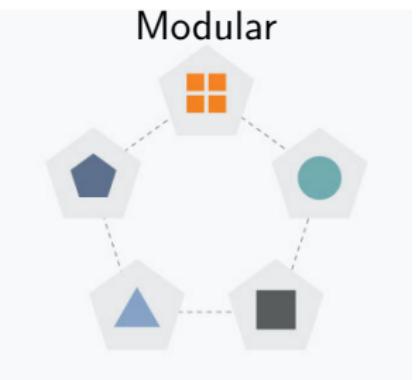
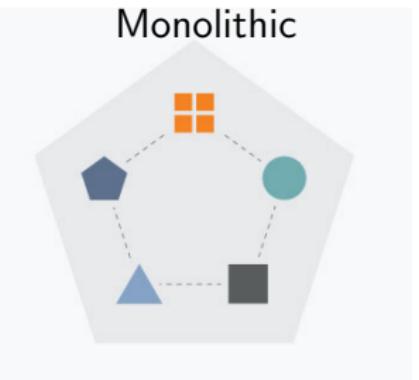
Maintainable, Modular, Re-useable Code

- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.
- Monolithic: One single big monolithic chunk of code → **unmaintainable, not reuseable**.
- Modular: Break monolithic code up into subparts→ functions → **maintainable and reusable**.



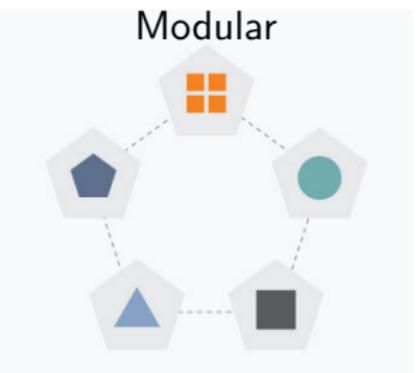
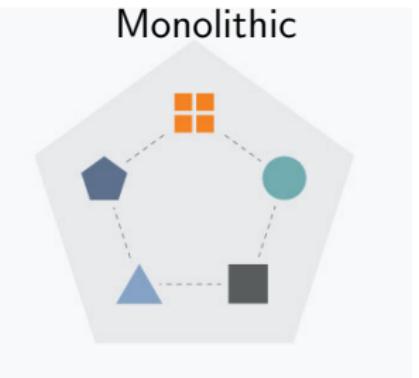
Maintainable, Modular, Re-useable Code

- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.
- Monolithic: One single big monolithic chunk of code → **unmaintainable, not reuseable**.
- Modular: Break monolithic code up into subparts→ functions → **maintainable and reusable**.
- Modularity is **implemented by putting code into functions**.



Maintainable, Modular, Re-useable Code

- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.
- Monolithic: One single big monolithic chunk of code → **unmaintainable, not reuseable**.
- Modular: Break monolithic code up into subparts→ functions → **maintainable and reusable**.
- Modularity is **implemented by putting code into functions**.
- Each function can be re-used over and again, when handled/documentated properly.



Commenting code → greater reuseability.

Commenting code allows the next person (usually ourselves) to understand what is being done.

Commenting code → greater reuseability.

Commenting code allows the next person (usually ourselves) to understand what is being done.

- Writing code for the first time we see things differently than when re-reading it on subsequent times.

Commenting code → greater reuseability.

Commenting code allows the next person (usually ourselves) to understand what is being done.

- Writing code for the first time we see things differently than when re-reading it on subsequent times.
- Often happens: when we revisit code 2 weeks later, we cannot understand what we first wrote it.

Commenting code → greater reuseability.

Commenting code allows the next person (usually ourselves) to understand what is being done.

- Writing code for the first time we see things differently than when re-reading it on subsequent times.
- Often happens: when we revisit code 2 weeks later, we cannot understand what we first wrote it.
- **Goal: Write code that is as easy to understand as possible.**

Commenting code → greater reuseability.

Commenting code allows the next person (usually ourselves) to understand what is being done.

- Writing code for the first time we see things differently than when re-reading it on subsequent times.
- Often happens: when we revisit code 2 weeks later, we cannot understand what we first wrote it.
- **Goal: Write code that is as easy to understand as possible.**



Implementing comments.

In C

Implementing comments.

In C

- Comments are started using /*

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.
- Use // at the beginning of every line you want to comment.

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.
- Use // at the beginning of every line you want to comment.

Nowadays, compilers will accept either format as a comment.

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.
- Use // at the beginning of every line you want to comment.

Nowadays, compilers will accept either format as a comment.

Although it is not considered professional coding practice, you can temporarily remove lines from the code by commenting them out.

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.
- Use // at the beginning of every line you want to comment.

Nowadays, compilers will accept either format as a comment.

Although it is not considered professional coding practice, you can temporarily remove lines from the code by commenting them out.

Try not leave your final program with chunks of code commented out

Implementing comments.

In C

- Comments are started using /*
- Comments are ended using */
- Everything between /* and */ are comments.

In C++

- Comments start using // until the end of the line.
- Use // at the beginning of every line you want to comment.

Nowadays, compilers will accept either format as a comment.

Although it is not considered professional coding practice, you can temporarily remove lines from the code by commenting them out.

Try not leave your final program with chunks of code commented out
→ sloppy programming.

Use-case examples in C++.

- ① Keep comments concise and to the point

Use-case examples in C++.

- ① Keep comments concise and to the point
- ② Use comments plentifully to document your functions

Use-case examples in C++.

- ① Keep comments concise and to the point
- ② Use comments plentifully to document your functions

```
#include<stdio.h>
///////////////////////////////
// This is the main program where C/C++ will begin its execution
// The main program has an output that returns an int
// The main function takes two inputs: argc (an integer) and argv (an
array of strings)
// argc is the number of inputs
// argv is a list of each input as a string
// The return value is a code indicating the success or failure of the
C/C++ program.
int main(int argc, char **argv) {
    return(0);
}
```

Use-case examples in C++.

- ① Keep comments concise and to the point
- ② Use comments plentifully to document your functions

```
#include<stdio.h>
///////////////////////////////
// This is the main program where C/C++ will begin its execution
// The main program has an output that returns an int
// The main function takes two inputs: argc (an integer) and argv (an
array of strings)
// argc is the number of inputs
// argv is a list of each input as a string
// The return value is a code indicating the success or failure of the
C/C++ program.
int main(int argc, char **argv) {
    return(0);
}
```

Always document: the purpose, the inputs and the outputs of every function you write.

Best C++ commenting practices.

- ① Use comments to indicate what a chunk of code is supposed to do
- ② Use comments to document purpose, inputs and outputs of all functions
- ③ Use comments to temporarily remove a chunk of code
- ④ Use comments to delimit sections of your program (the quintessential section of your program being a function)

Outline EEEE1042 C Lecture 1:

1 Operating System (OS) and Integrated Development Environment (IDE)

- Introduction
- OS/IDE/compiler
- Installing Ubuntu
- Installing Code::Blocks.
- Running Code::Blocks
 - Running without project
 - Running with project
- Modular programming
- Commenting your code
 - Purpose Of Comments
 - Implementing comments
 - Use-cases examples
 - Best practices

2 Key Takeaways

Key takeaways

- ① Overview of how compilers work
- ② Introduced and installed the main coding environment we will be using in this course:
Code::Blocks
- ③ Differences between program files and projects
- ④ Importance of Modular programming
- ⑤ Importance of commenting your code