University of Nottingham
UK | CHINA | MALAYSIA

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

EEEE1042/1032 C Programming: Scheduled classes

EEEE1042: for EE students, EEEE1032: for Mecha students.

Week	Dates	Lecture	EEEE1042 Practical	EEEE1032 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%)			

EEEE1042/1032 Course Information

University of Nottingham
UK | CHINA | MALAYSIA

Targeted students

This course is targeted at first year electrical (EEEE1042) and mechatronics (EEEE1032) 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.

EEEE1042 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:

Introduction to C

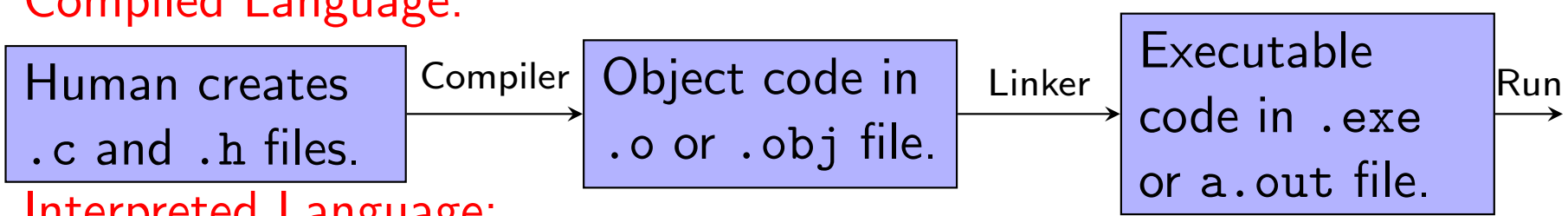
- C is a **compiled**, **low-level**, **procedural** computer programming language developed in 1972 by Dennis Ritchie at Bell Labs and it was used to write the UNIX operating system.

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

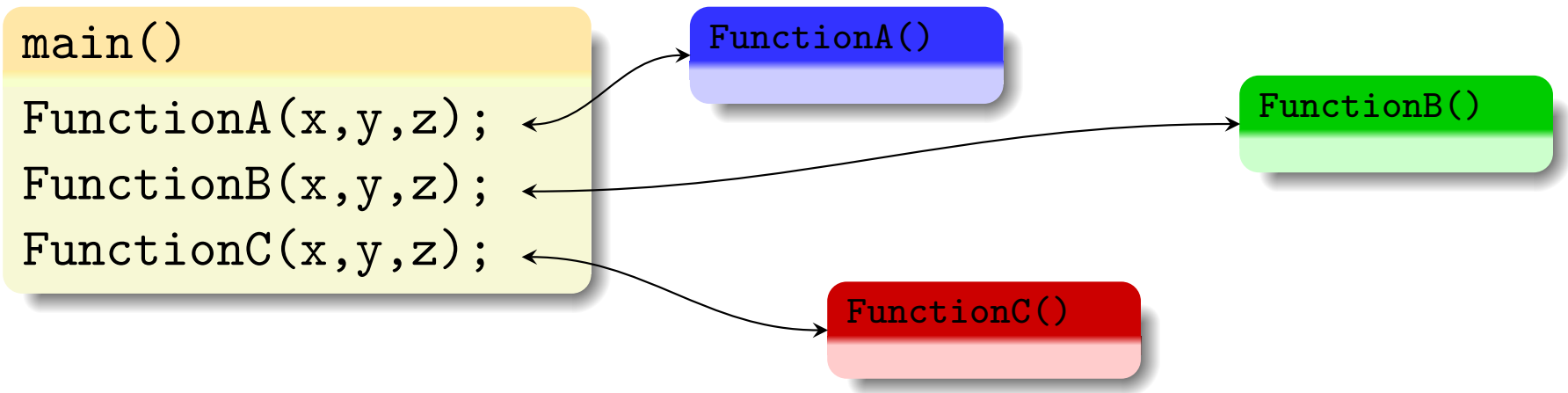
Introduction to C

- 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**.



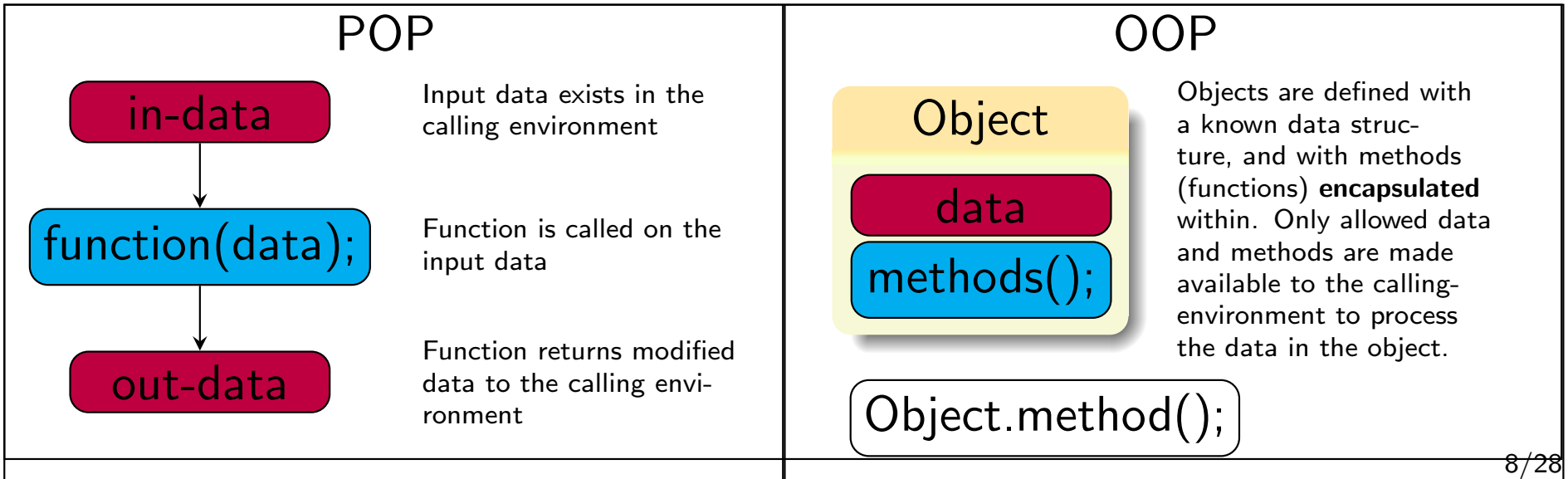
Introduction to C++

- 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 Programing**, a programming paradigm that defines/uses the object as opposed to **procedural oriented programming** (POP) that defines/uses functions

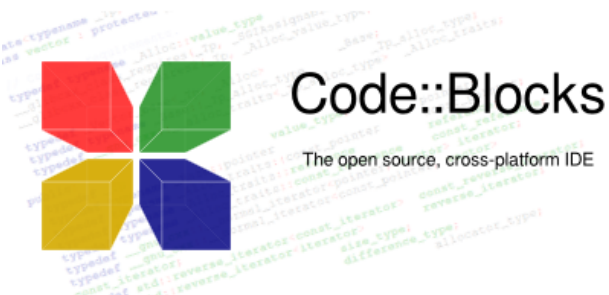


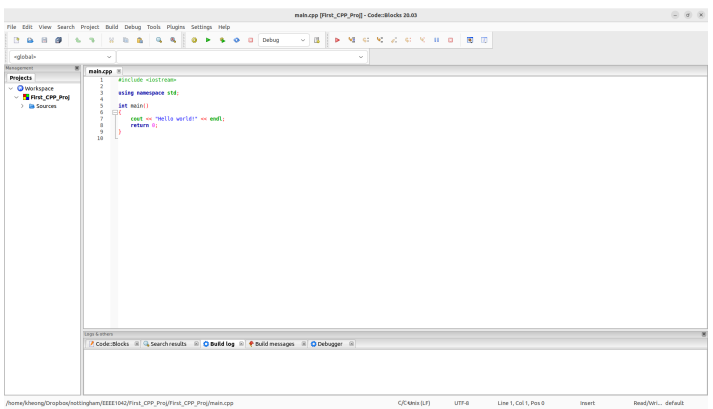
OS, IDE and compiler

- 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

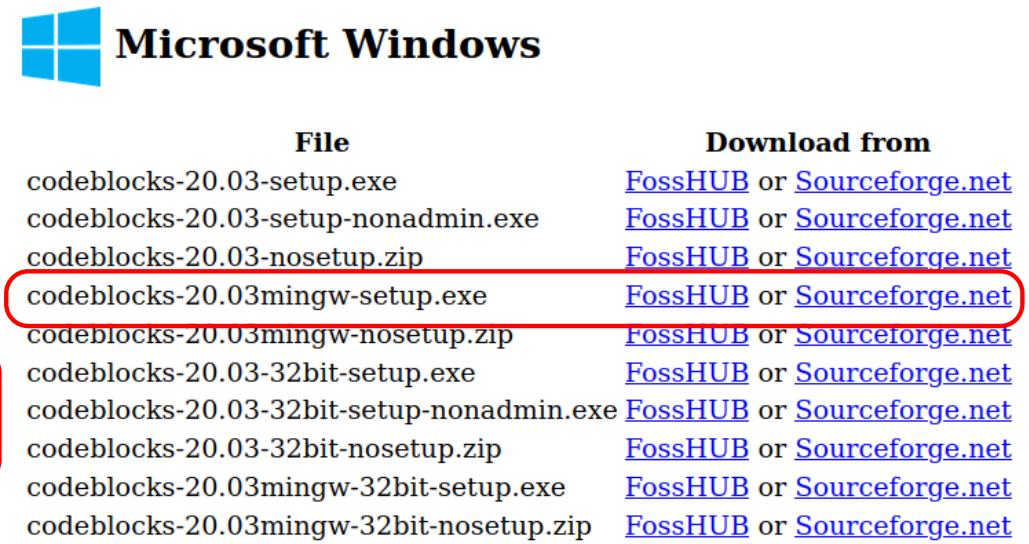
File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	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

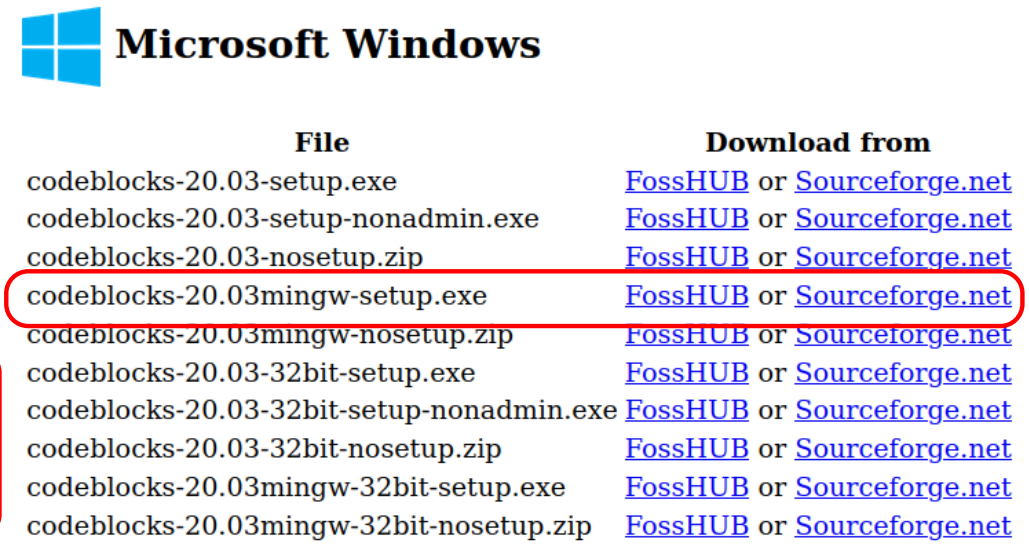
File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	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	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

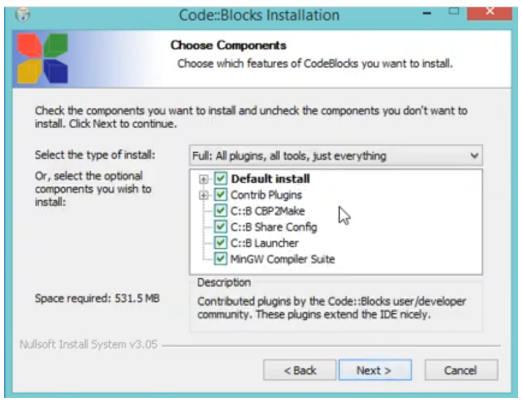
Downloading and Installing Code::Blocks.

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

Downloading and Installing Code::Blocks.

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

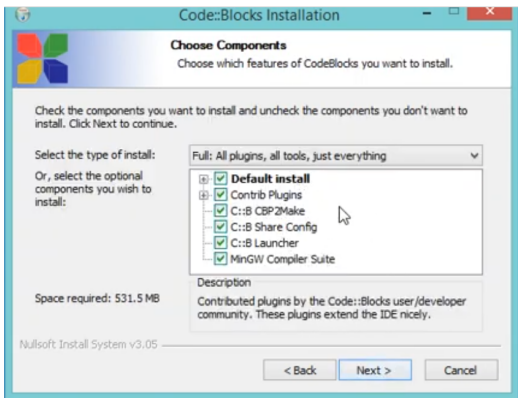
4. You can choose which components to include in the install. Be sure you include MinGW to have the compiler



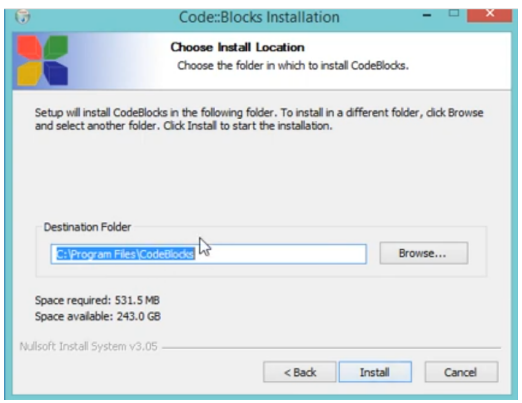
Downloading and Installing Code::Blocks.

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.

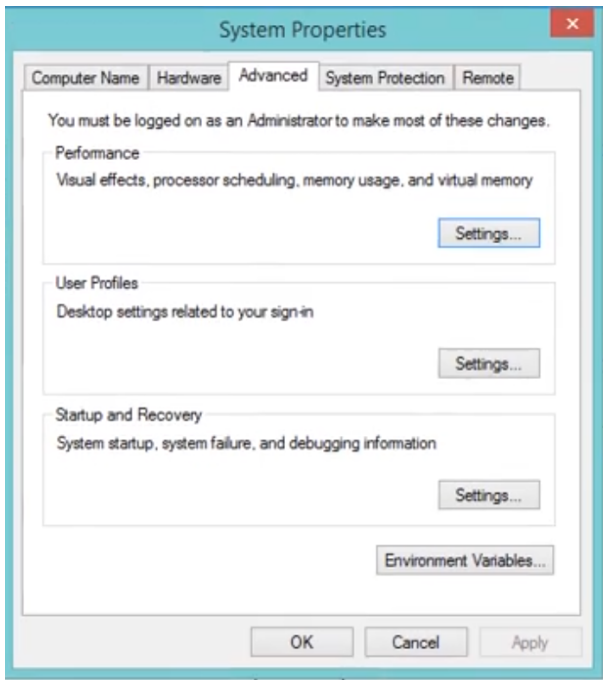


Set MinGW to the system path.

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`

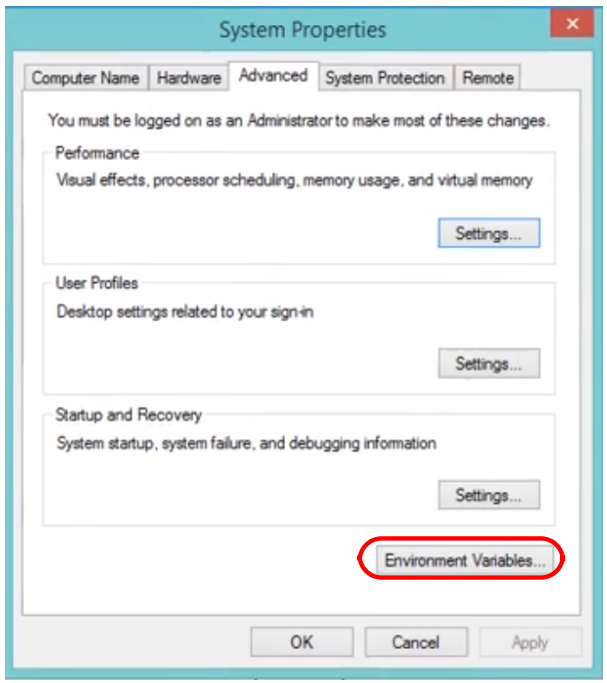
Set MinGW to the system path.

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.

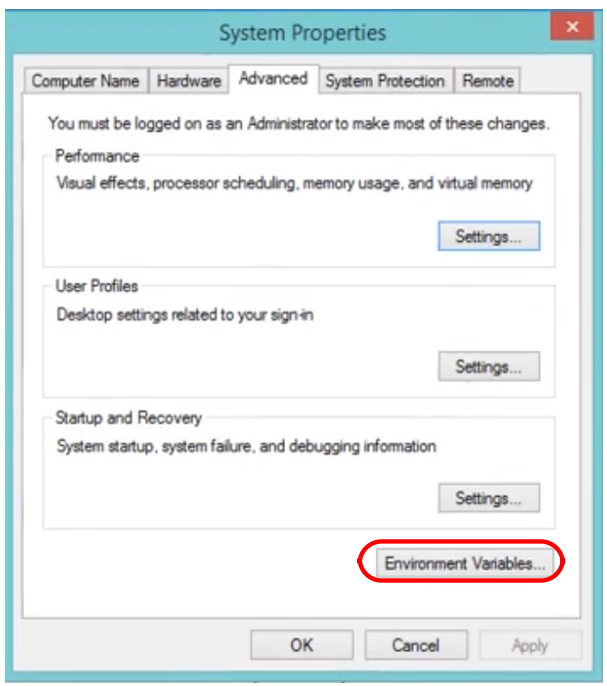
6. After installation complete, add MinGW to the system path. This enables Code::Blocks to find and use it. If you used the default install location, this will be at `C:\Program Files\CodeBlocks\MinGW\bin`
7. Add this to the path in the system properties

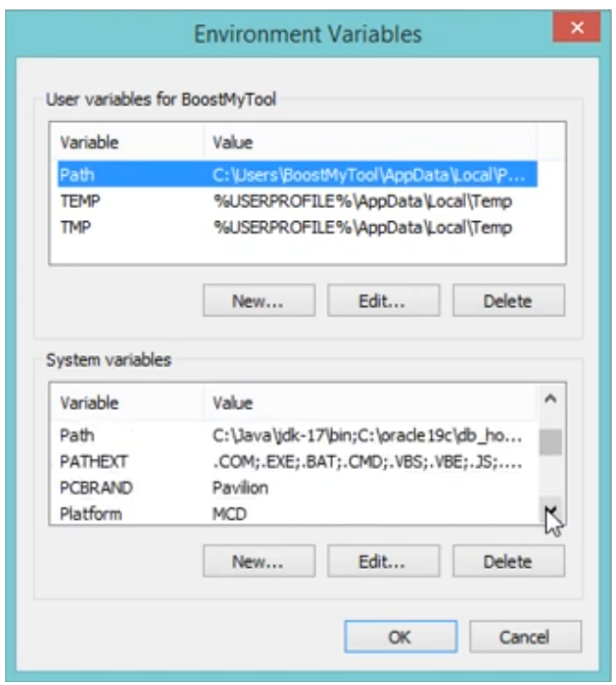


Click Environment variables.

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 varible.

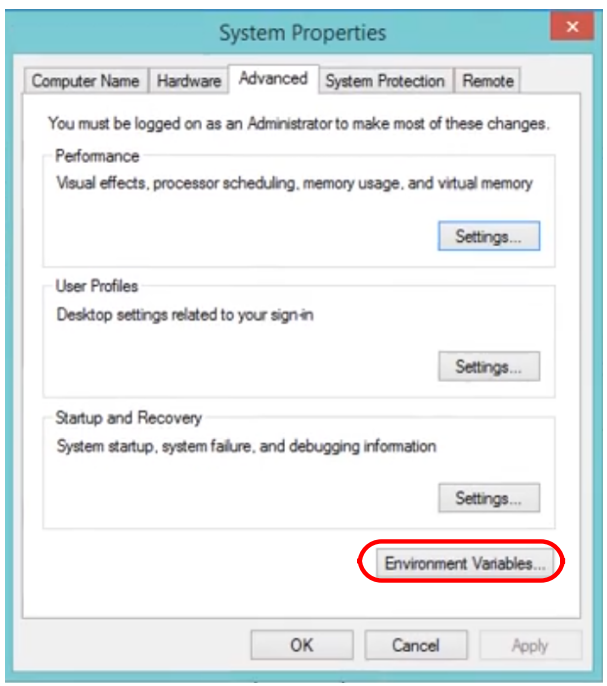




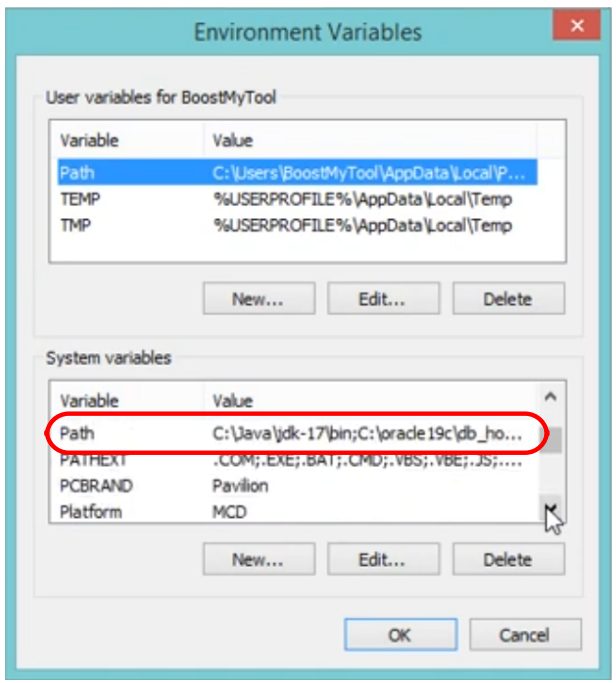
Click Environment variables.

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 varible.



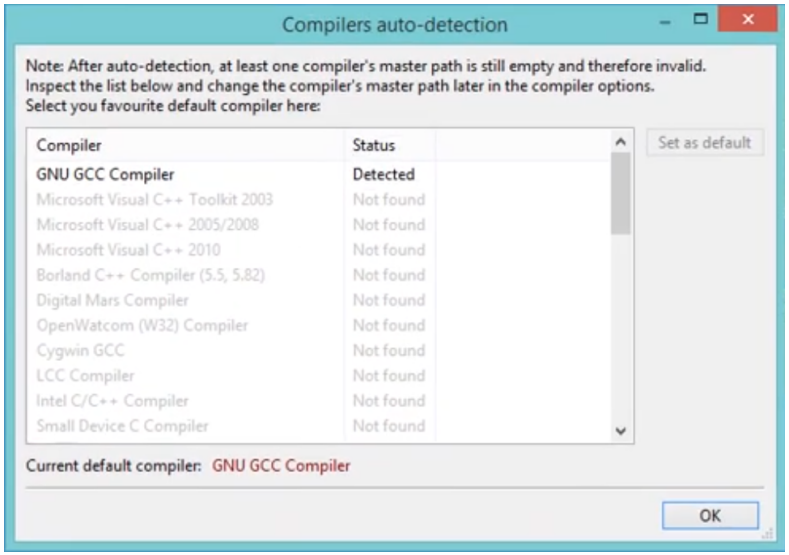
Click Environment variables.



Add MinGW to the system path here.

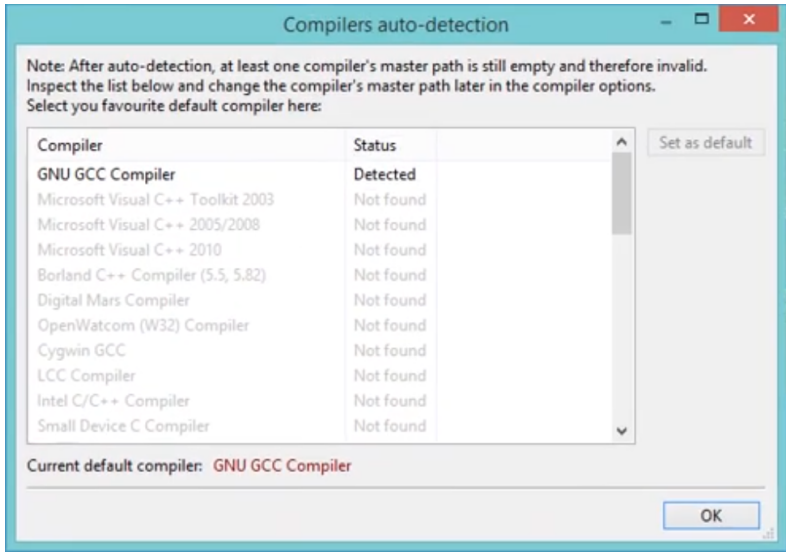
Run Code::Blocks for the first time.

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

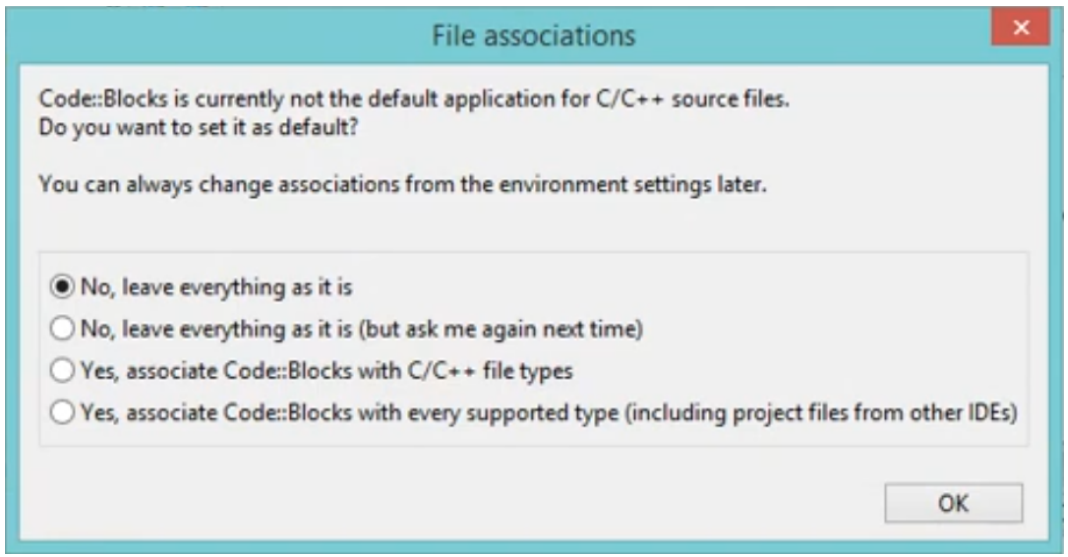


Run Code::Blocks for the first time.

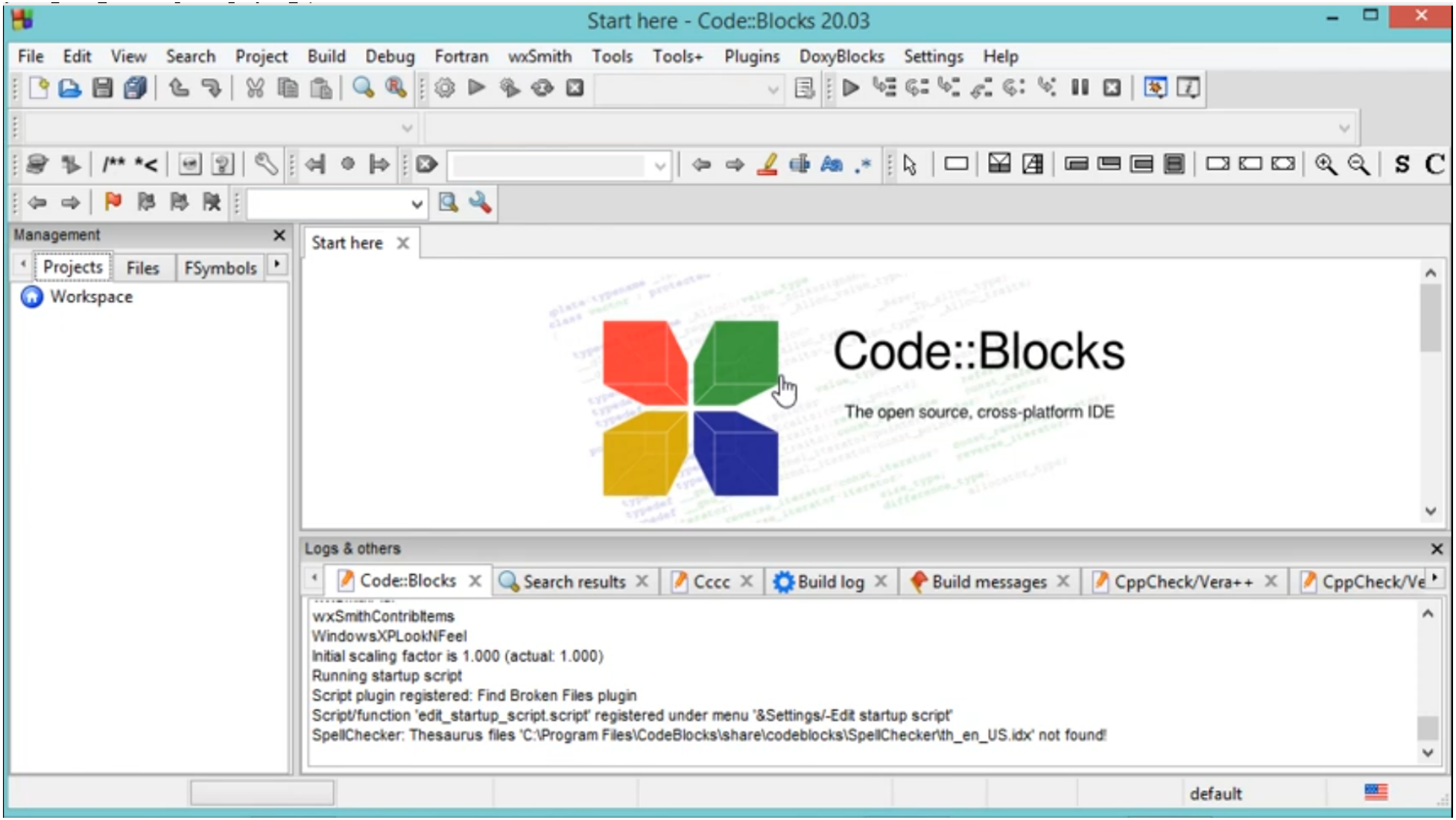
1. Executing Code::Blocks for the first time, it auto-detects the MinGW GCC compiler. You can set it as the default.



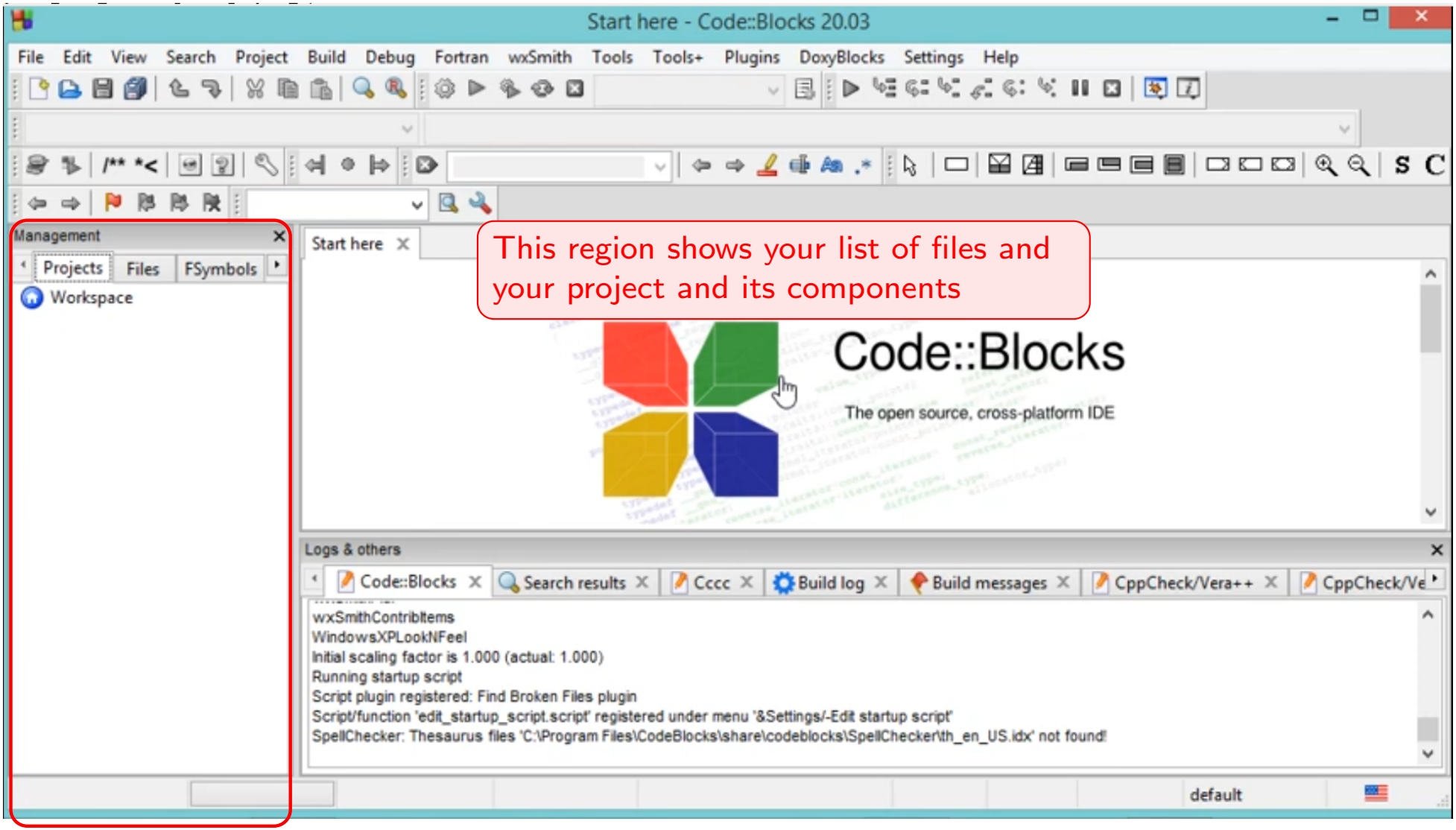
2. You can choose to associate Code::Blocks with .c and .cpp files so that it is the chosen application to run whenever you click them.



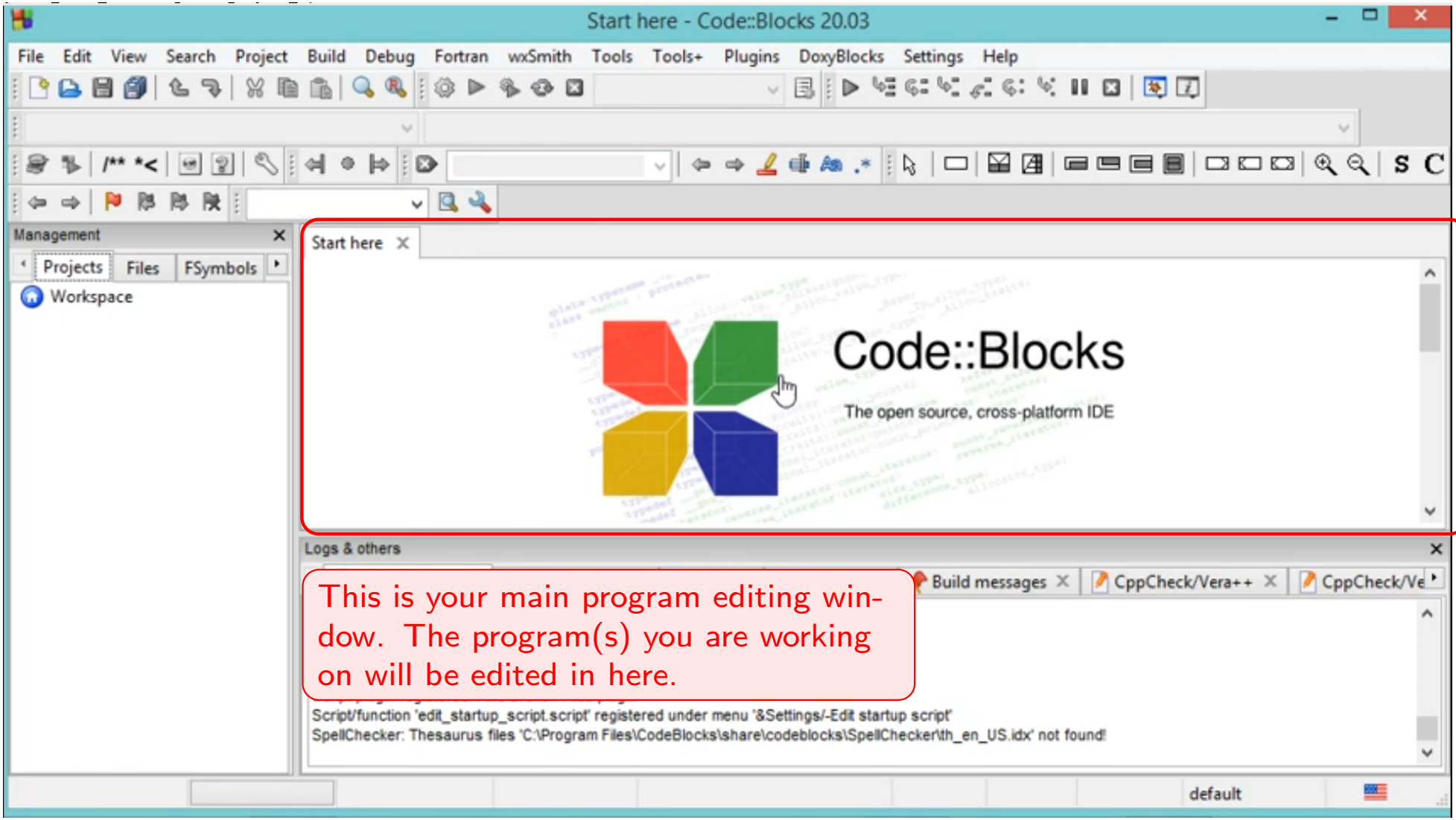
Run Code::Blocks for the first time.



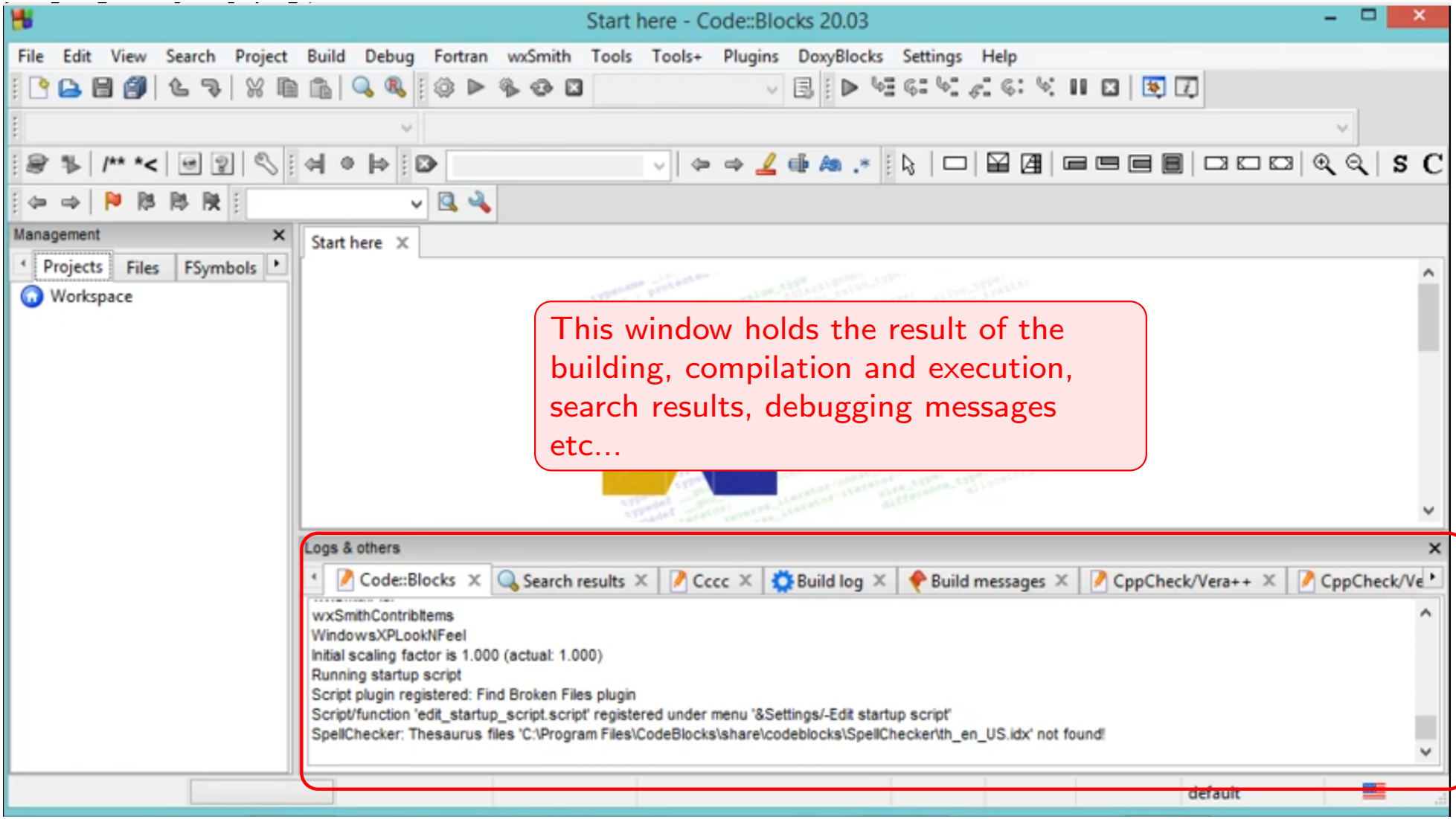
Run Code::Blocks for the first time.



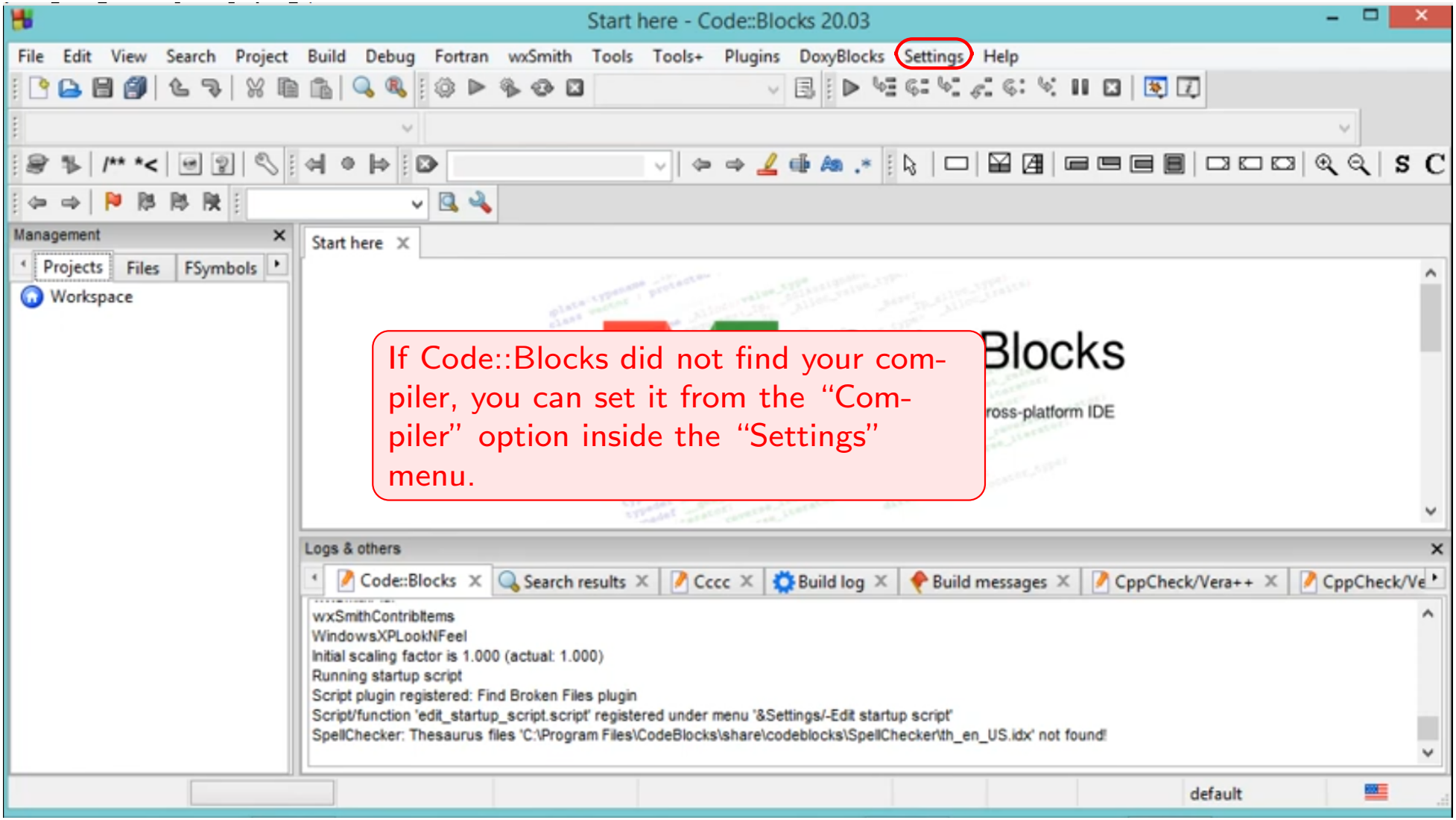
Run Code::Blocks for the first time.



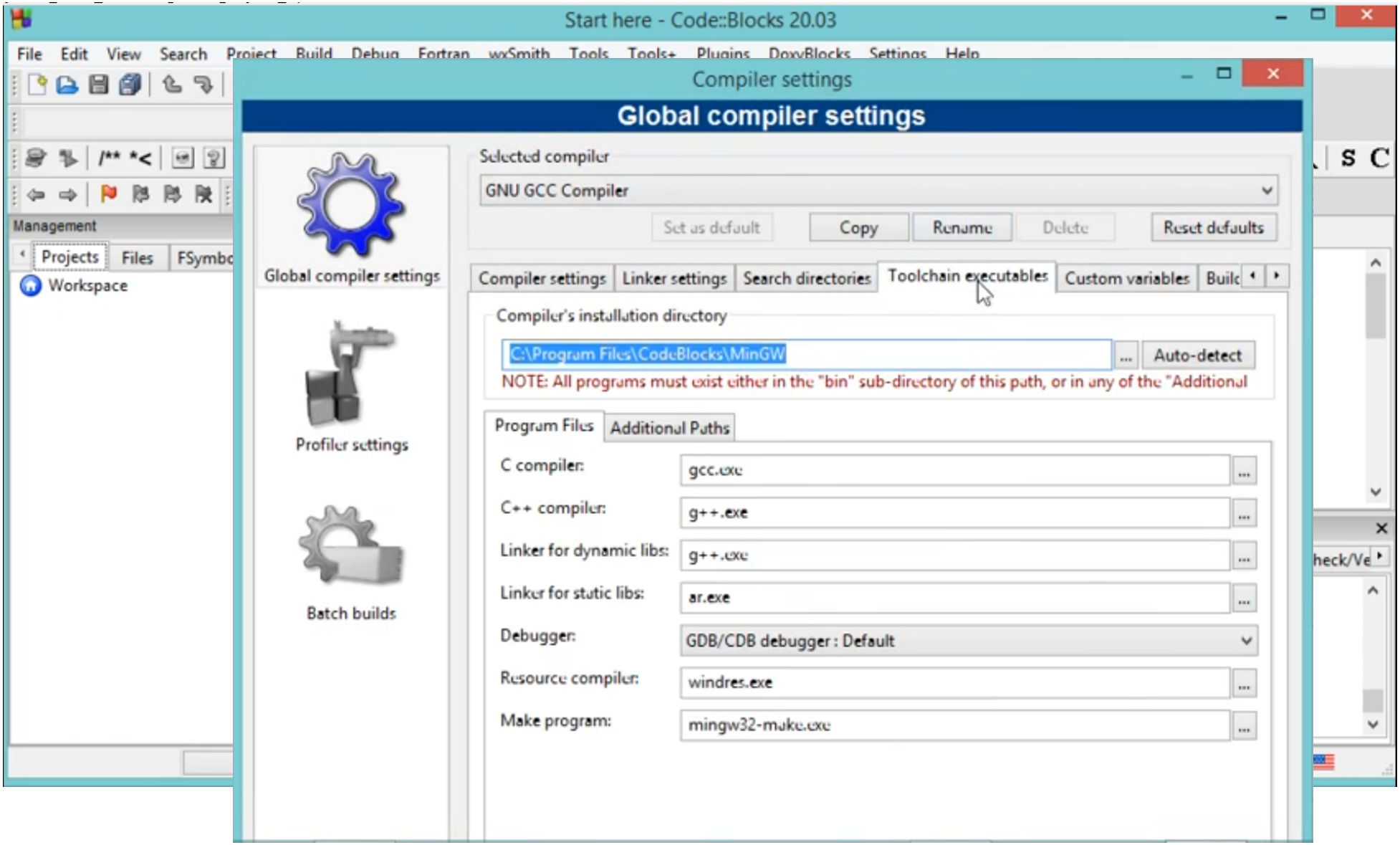
Run Code::Blocks for the first time.



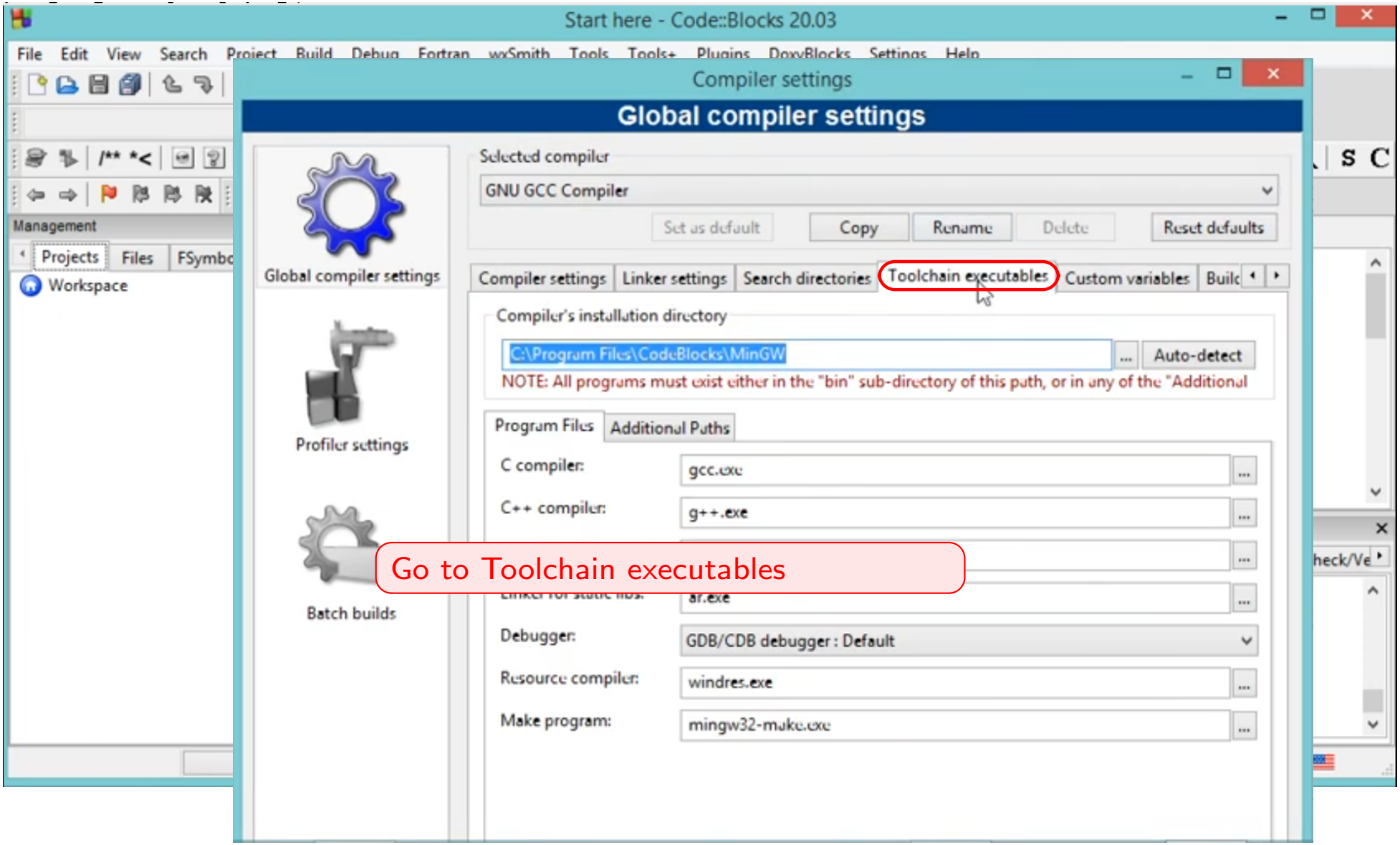
Run Code::Blocks for the first time.



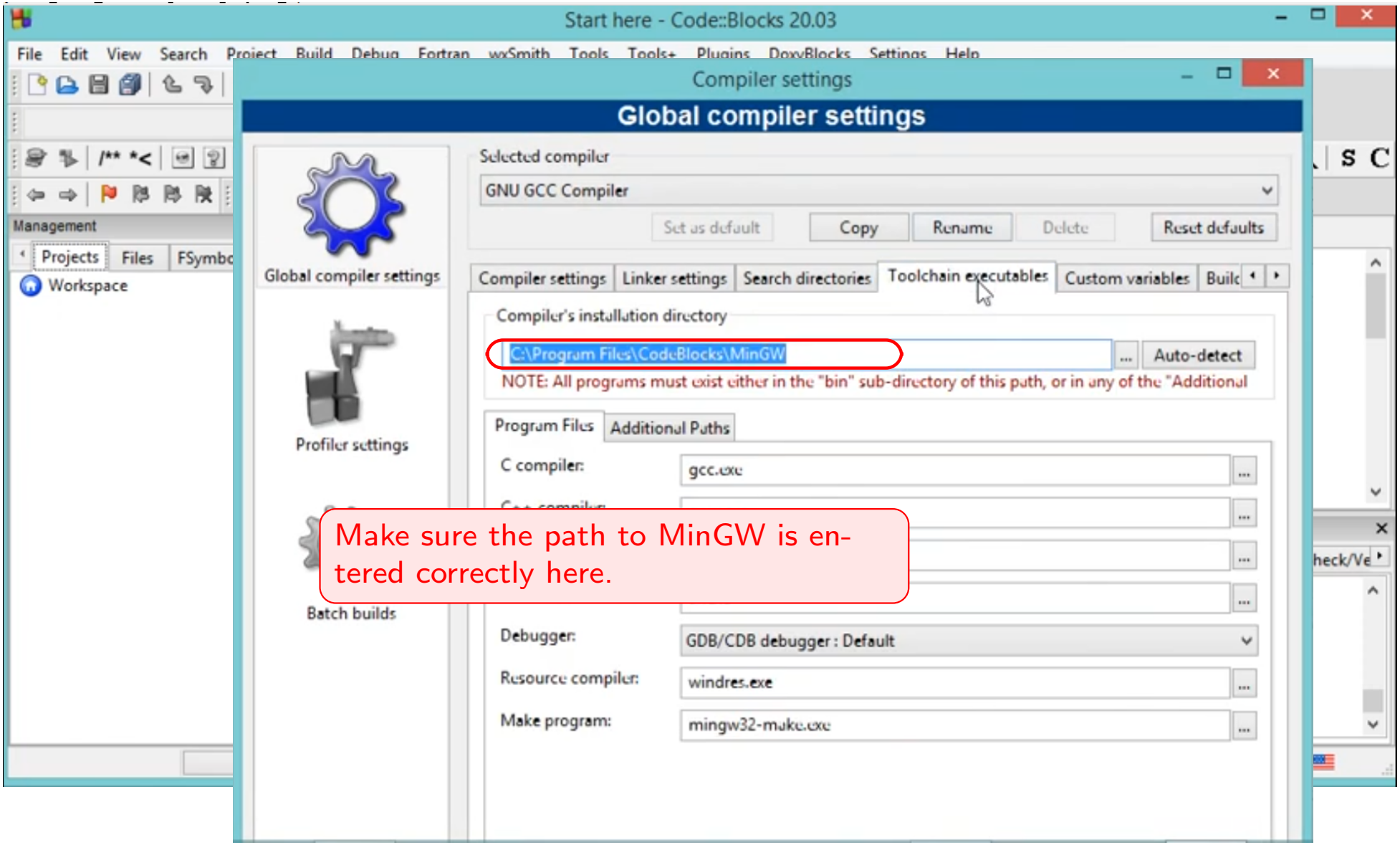
Run Code::Blocks for the first time.



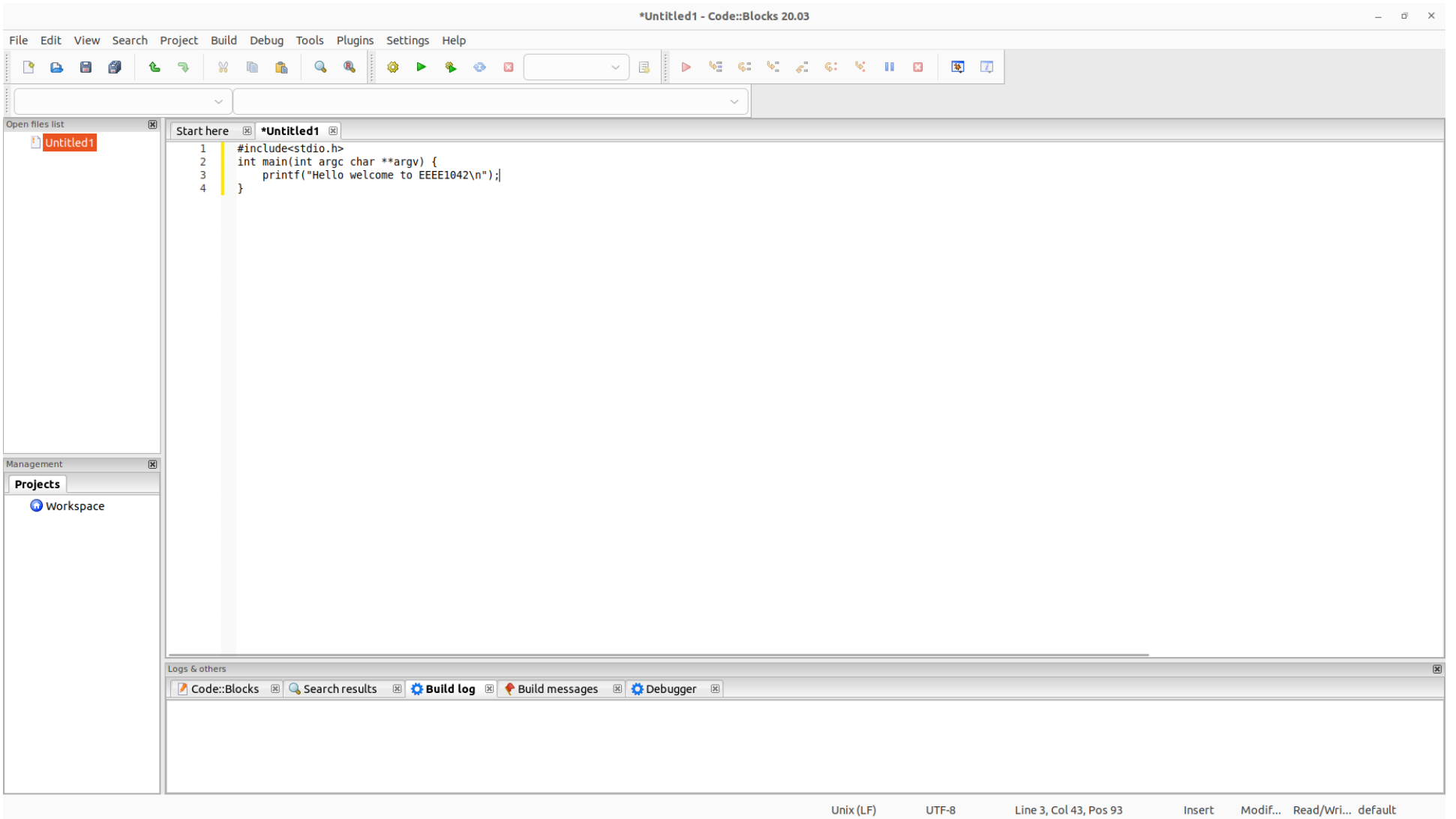
Run Code::Blocks for the first time.



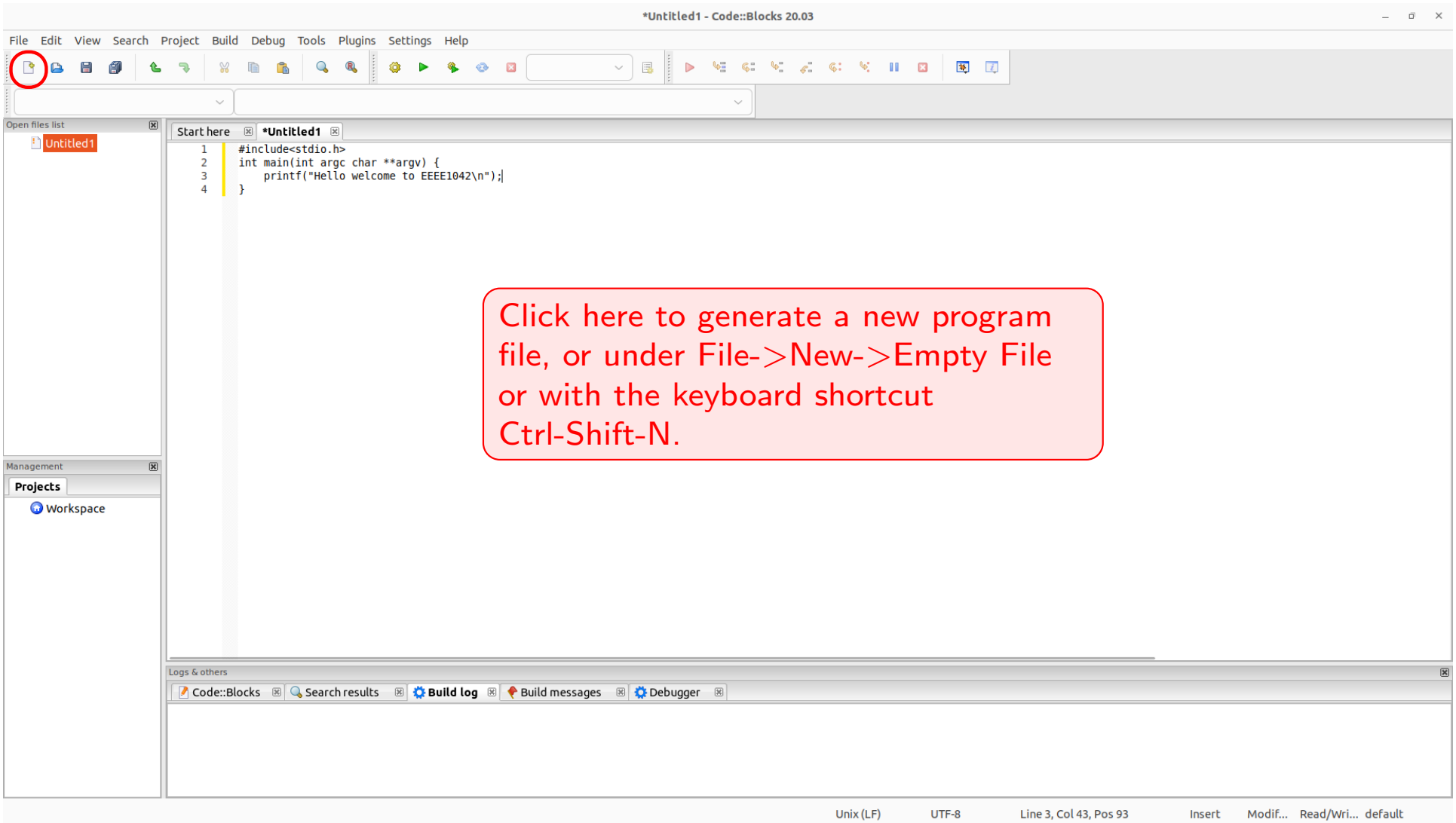
Run Code::Blocks for the first time.



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

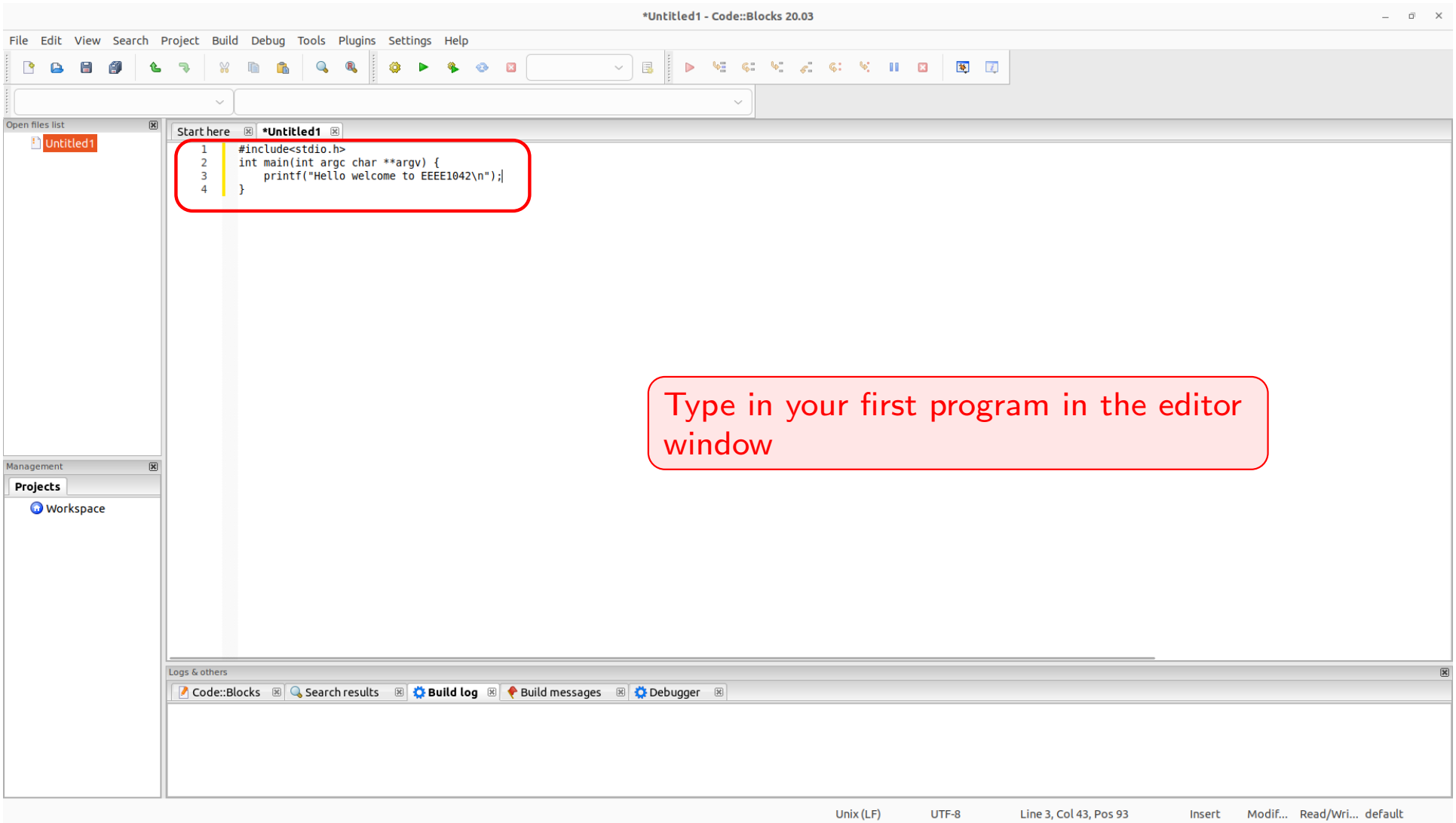


Run Code::Blocks for the first time: generate C-program **without** project.



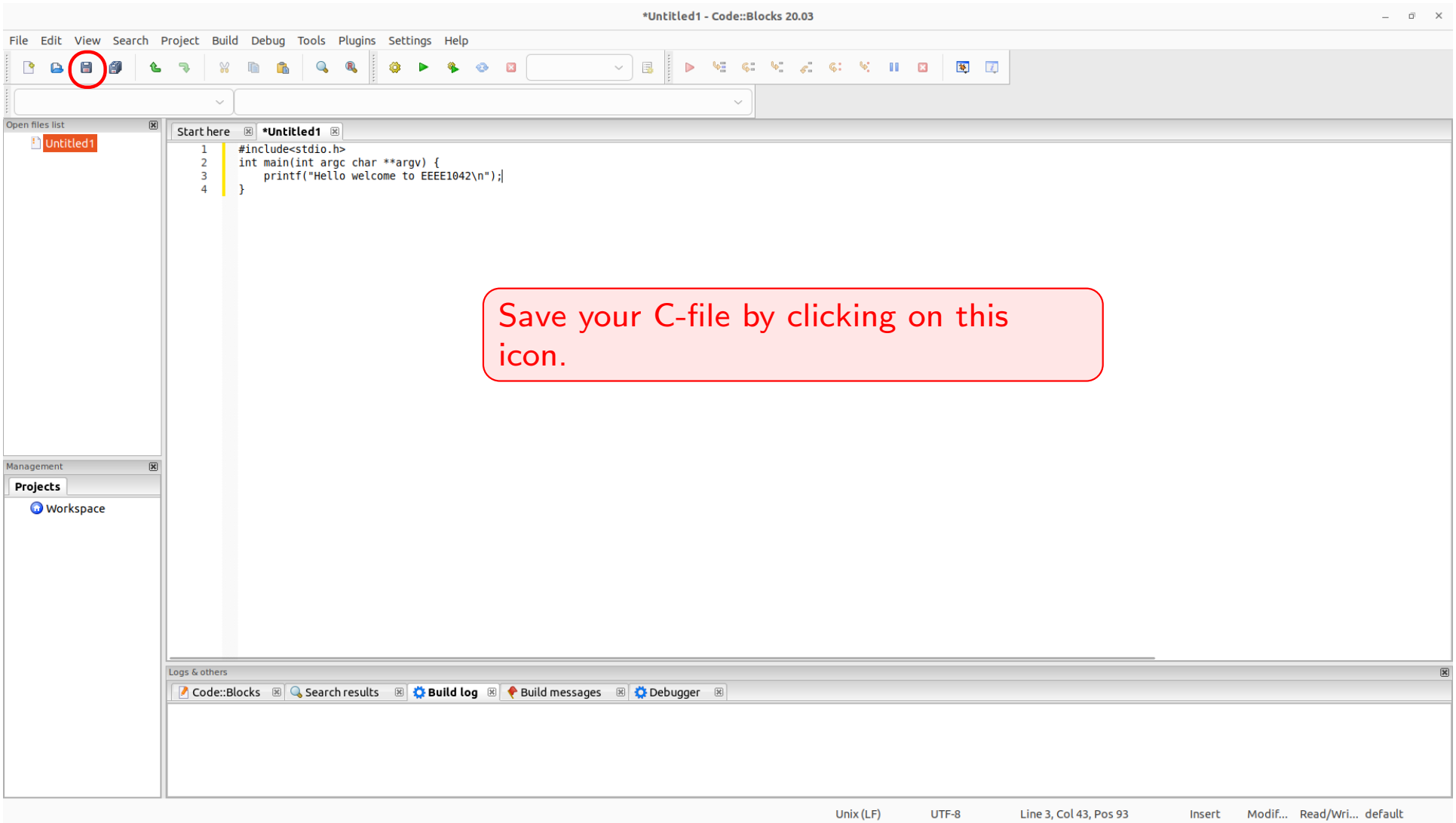
Click here to generate a new program file, or under File->New->Empty File or with the keyboard shortcut Ctrl-Shift-N.

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



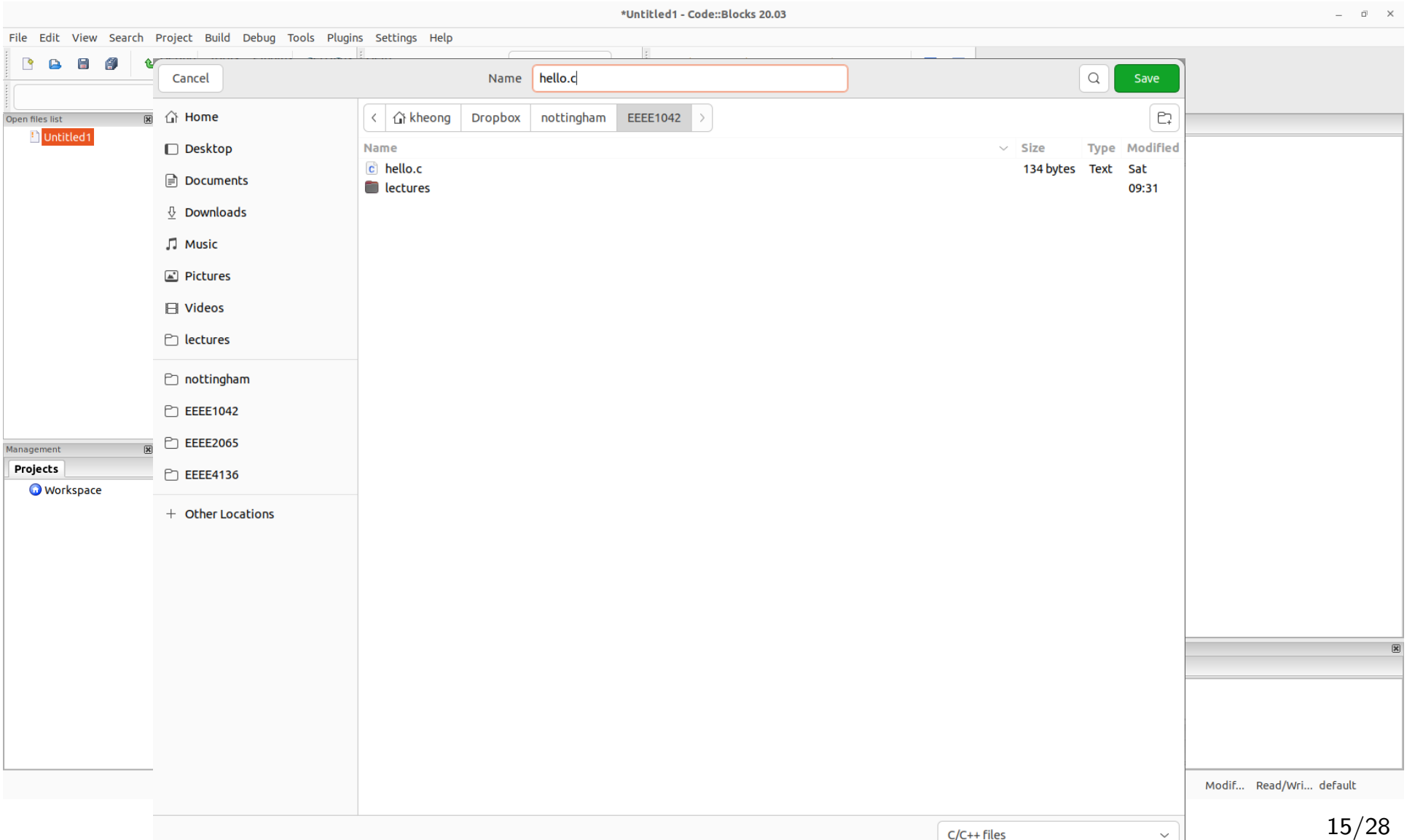
Type in your first program in the editor window

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

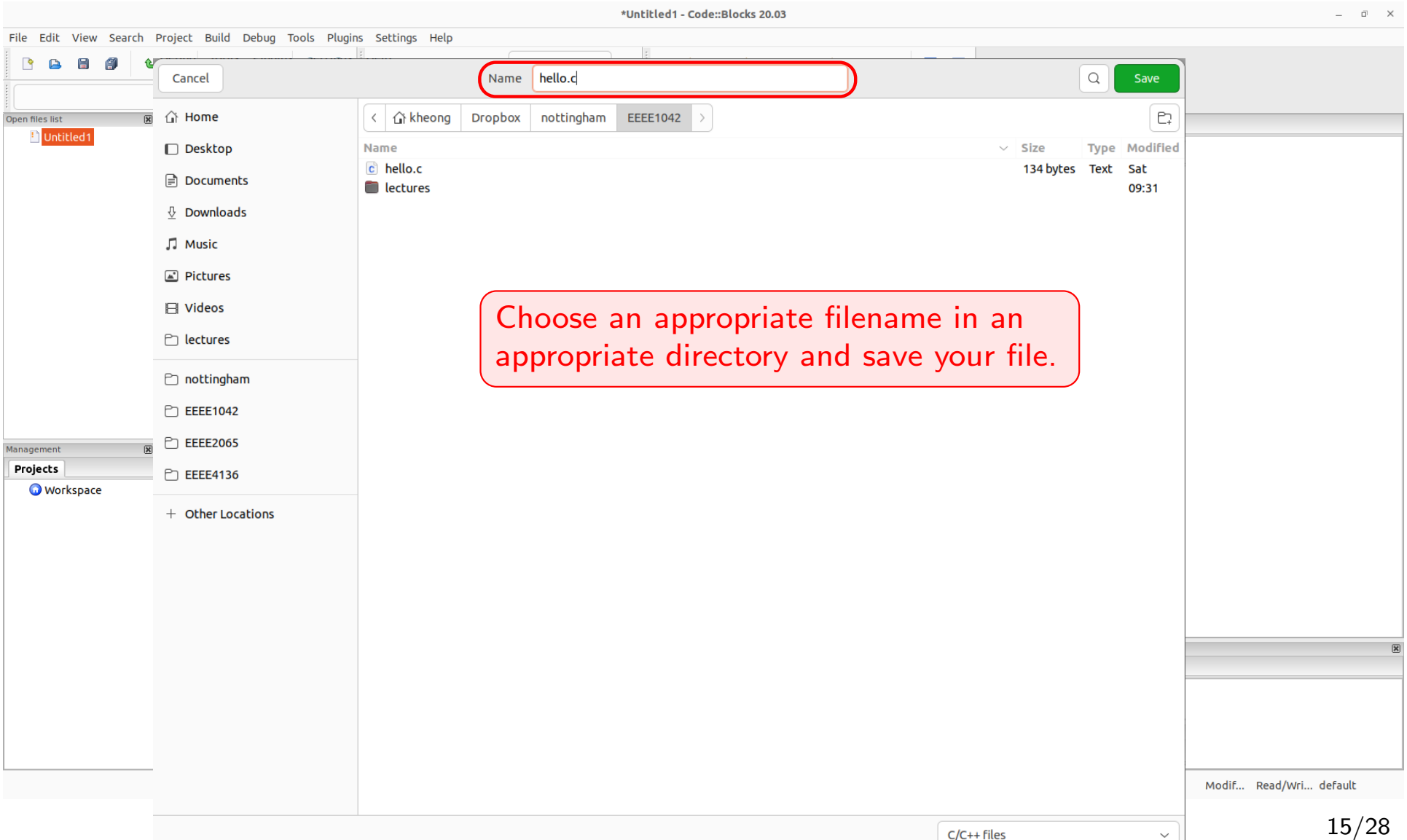


Save your C-file by clicking on this icon.

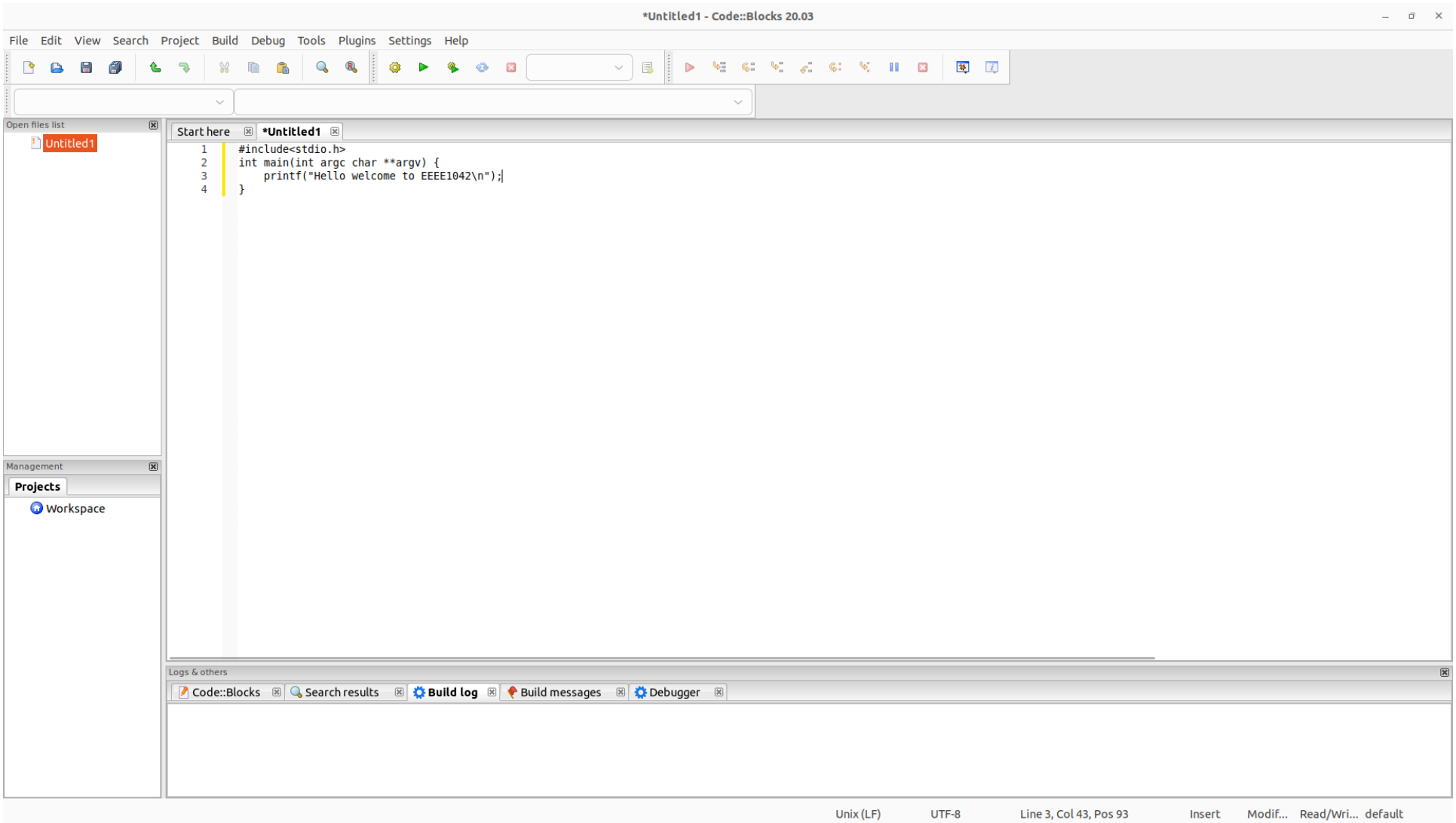
Run Code::Blocks for the first time: generate C-program **without** project.



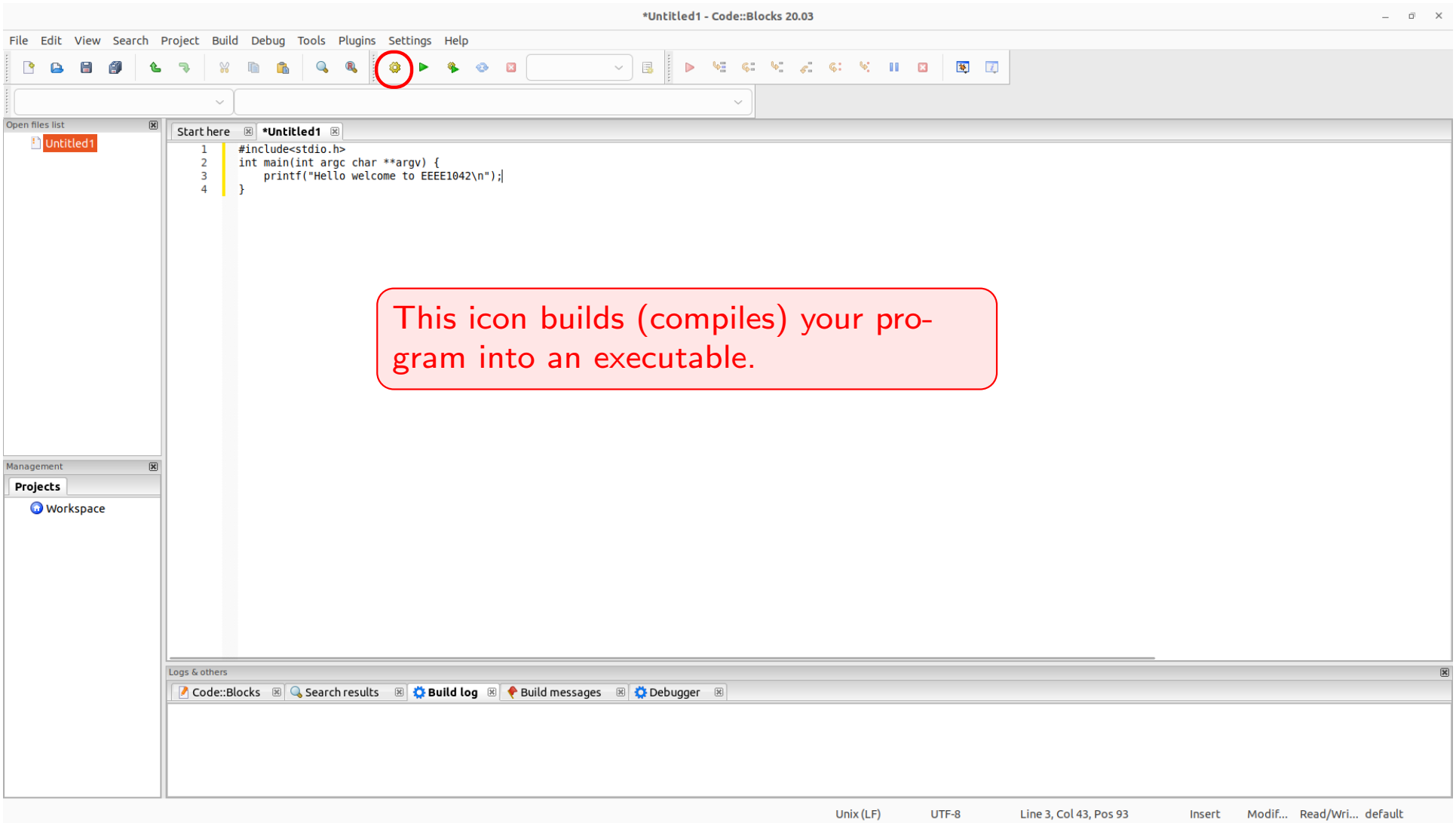
Run Code::Blocks for the first time: generate C-program **without** project.



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

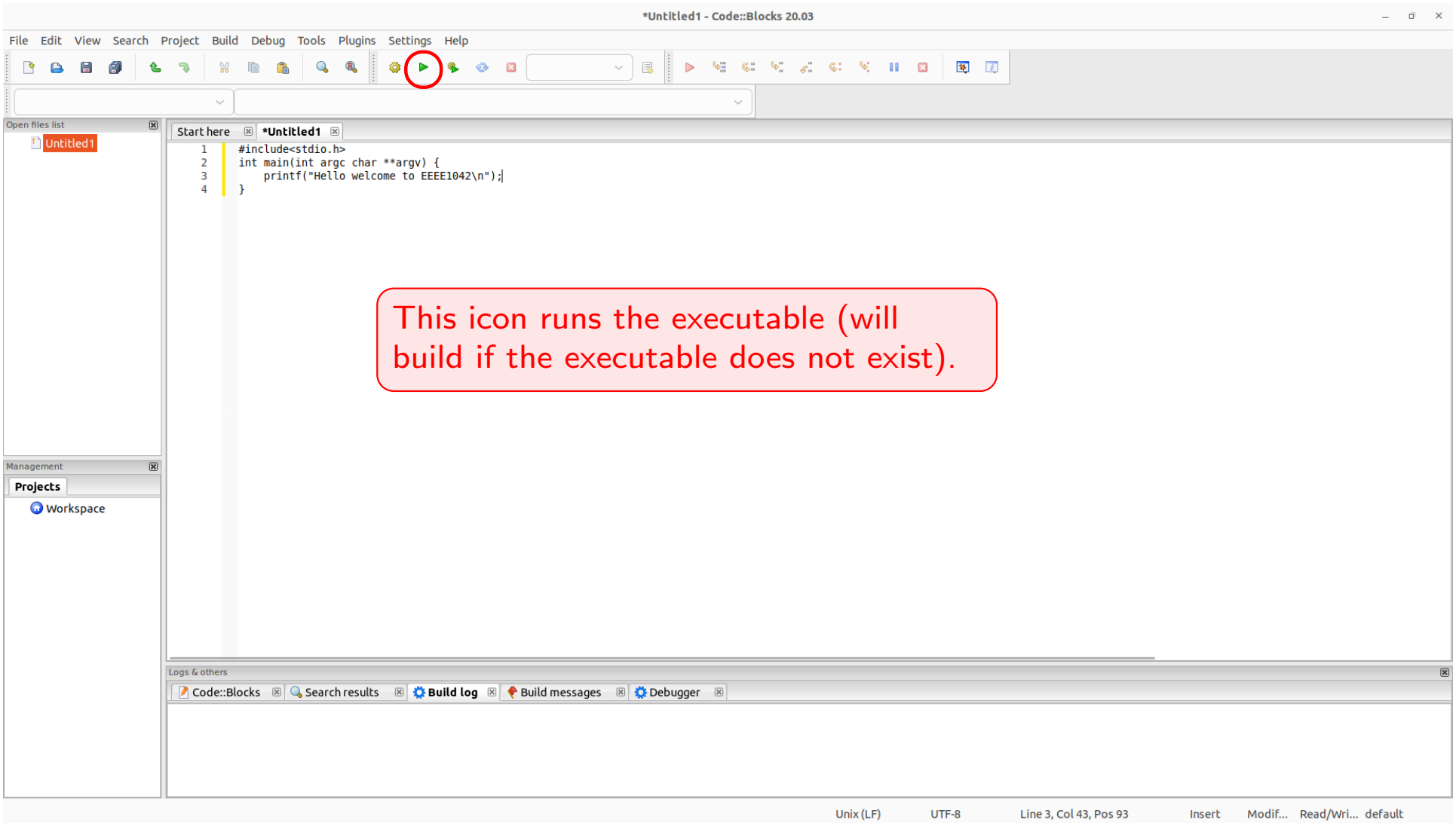


Run Code::Blocks for the first time: compile and execute your program without project.



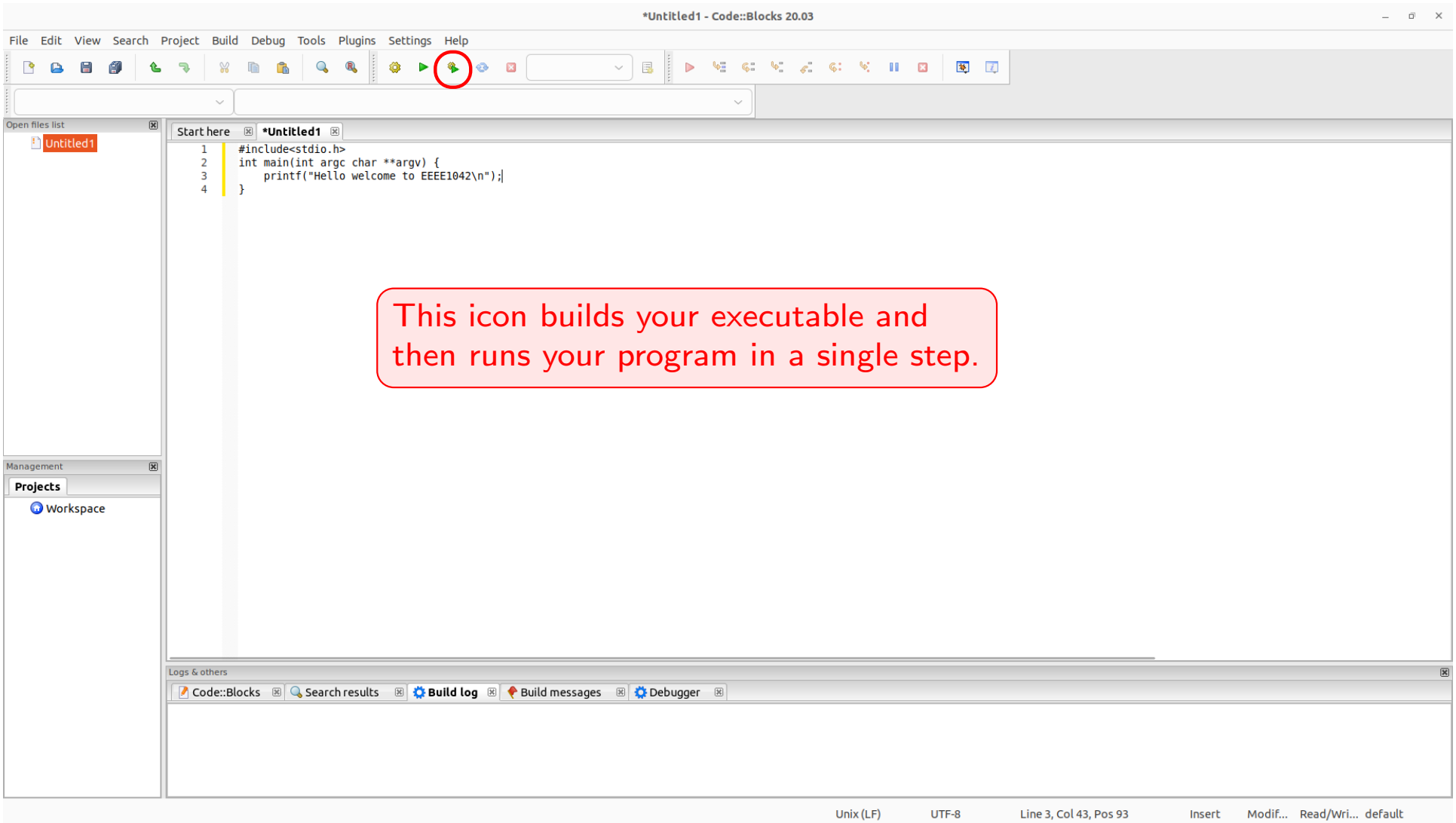
This icon builds (compiles) your program into an executable.

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



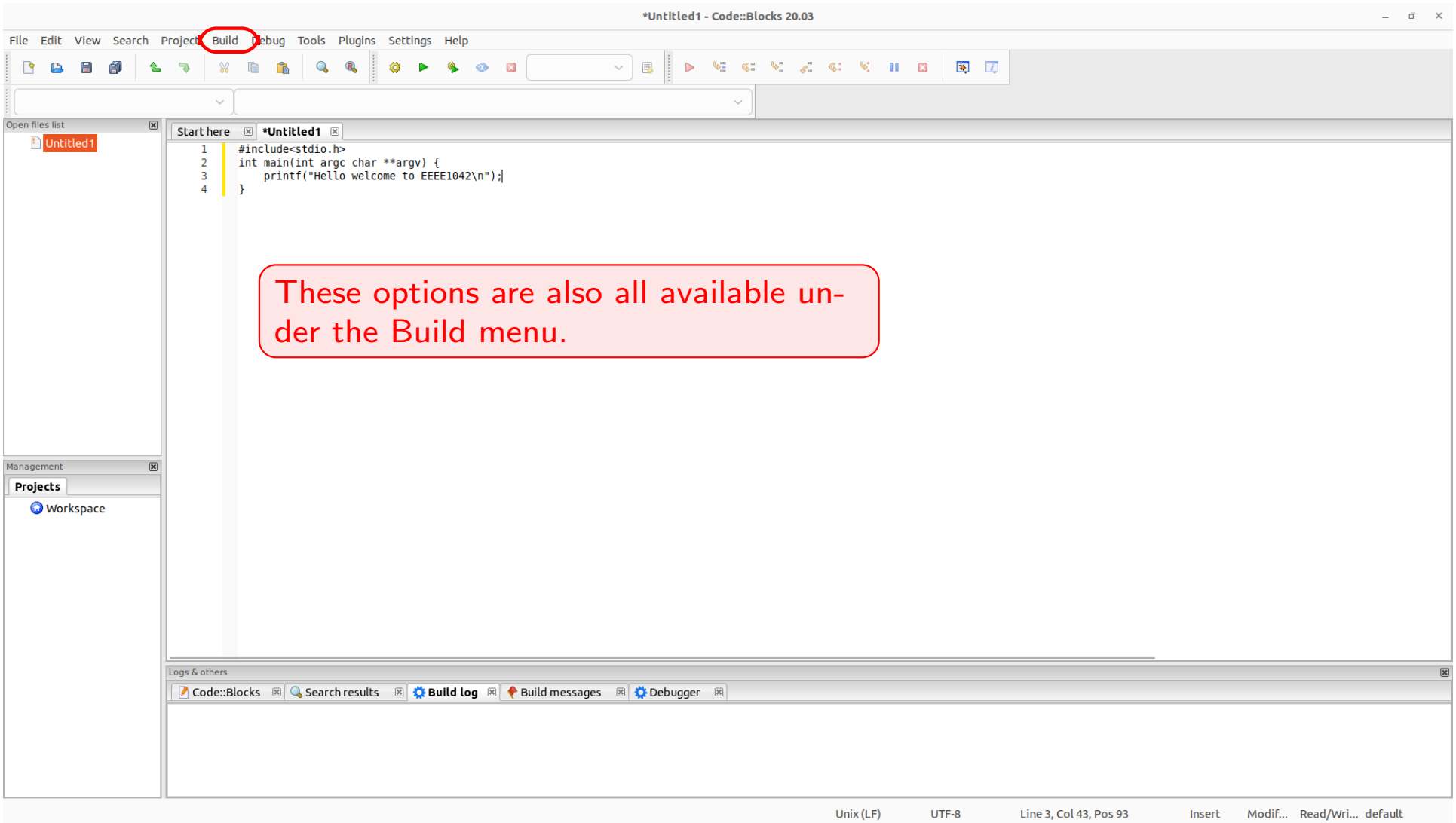
This icon runs the executable (will build if the executable does not exist).

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

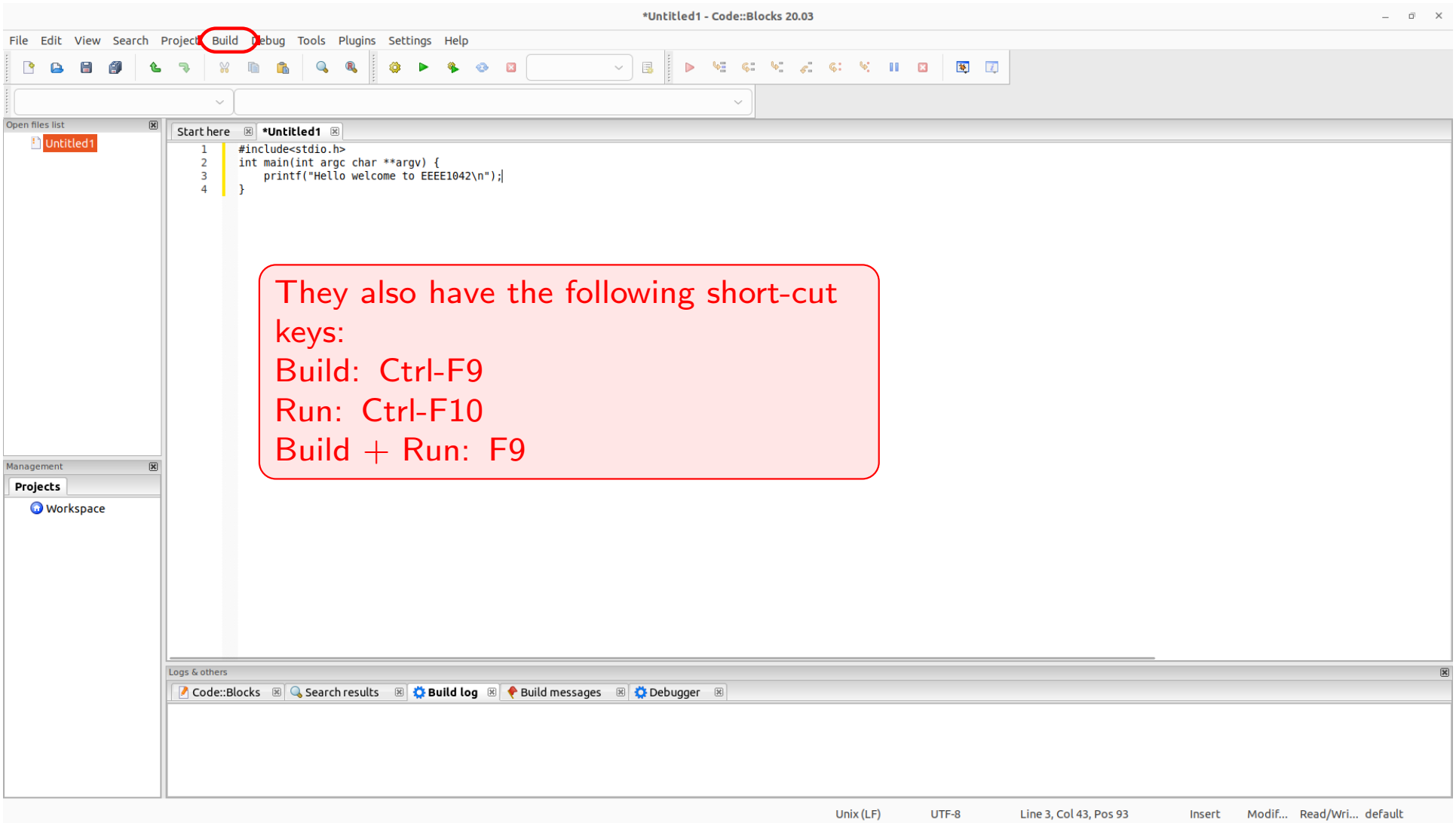


This icon builds your executable and then runs your program in a single step.

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

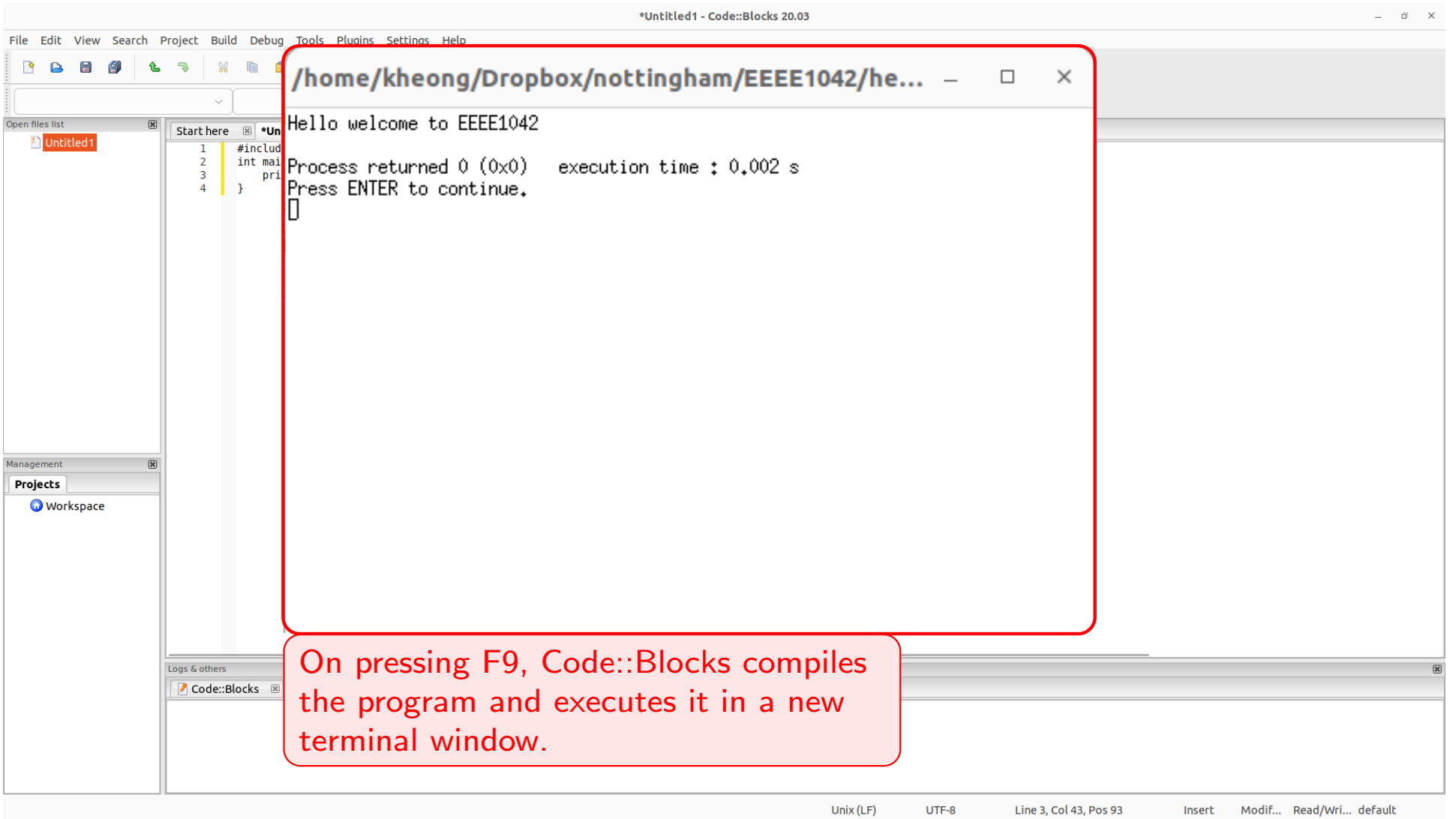


Run Code::Blocks for the first time: compile and execute your program without project.



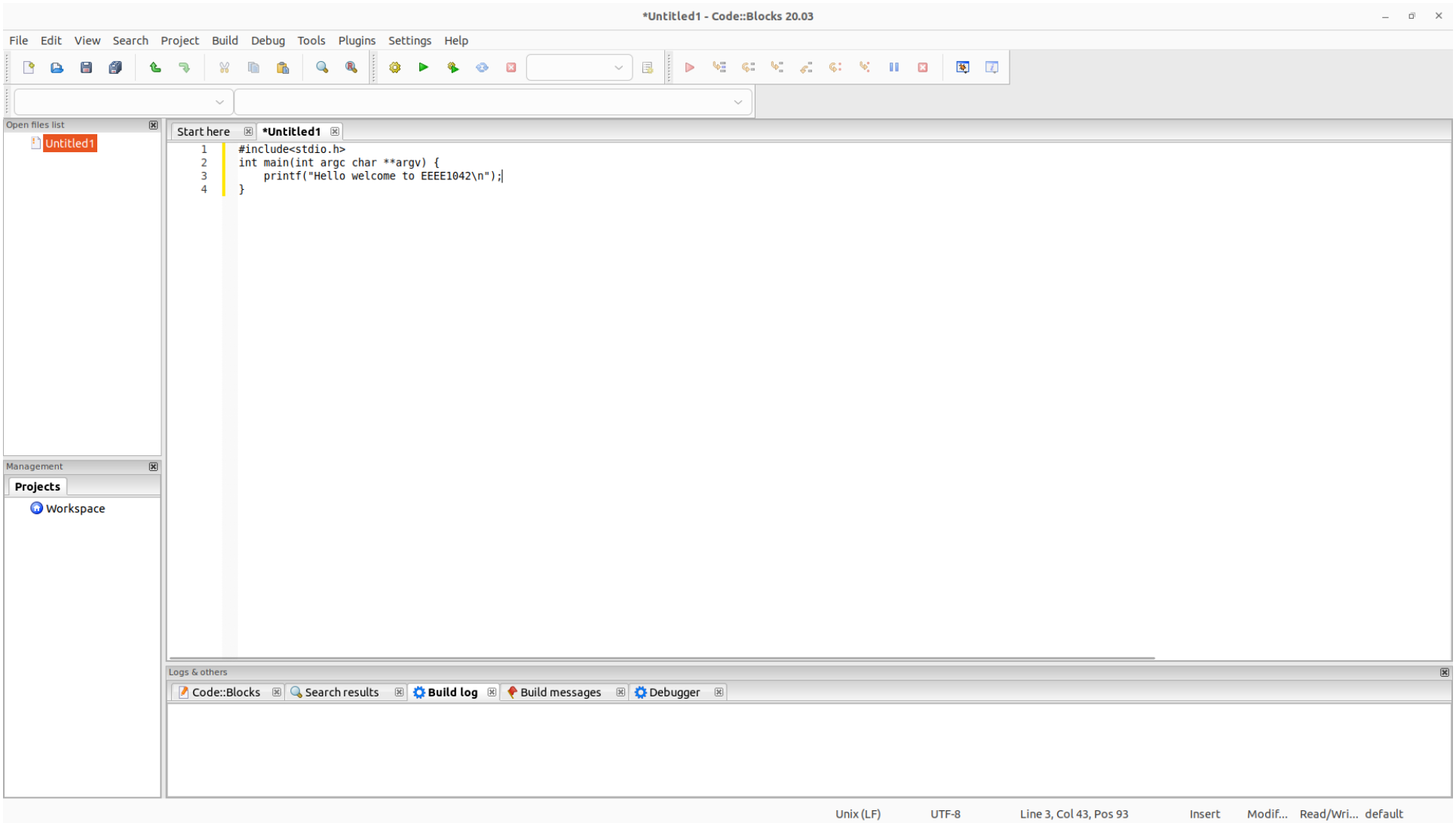
They also have the following short-cut keys:
Build: Ctrl-F9
Run: Ctrl-F10
Build + Run: F9

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

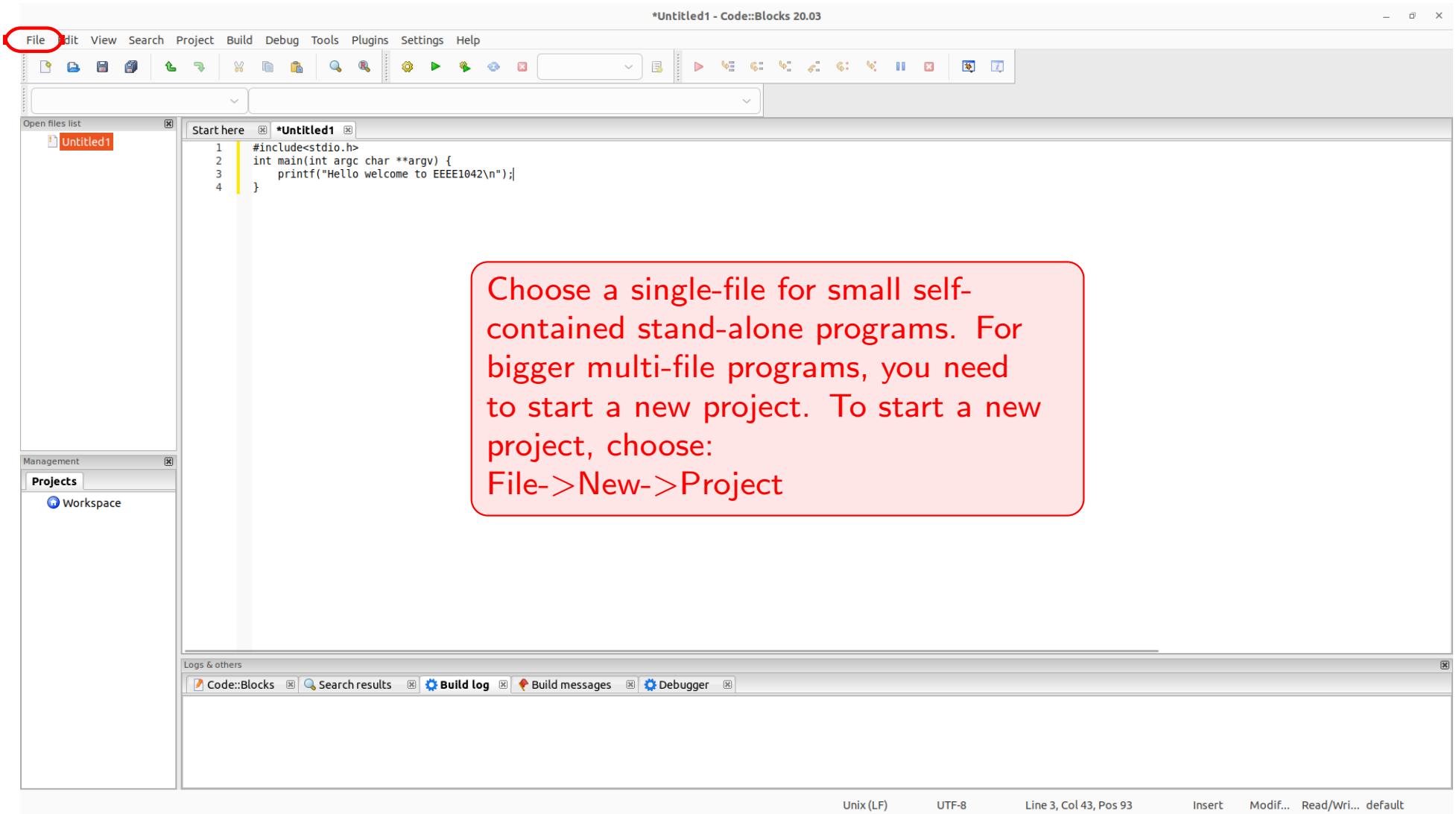


On pressing F9, Code::Blocks compiles the program and executes it in a new terminal window.

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

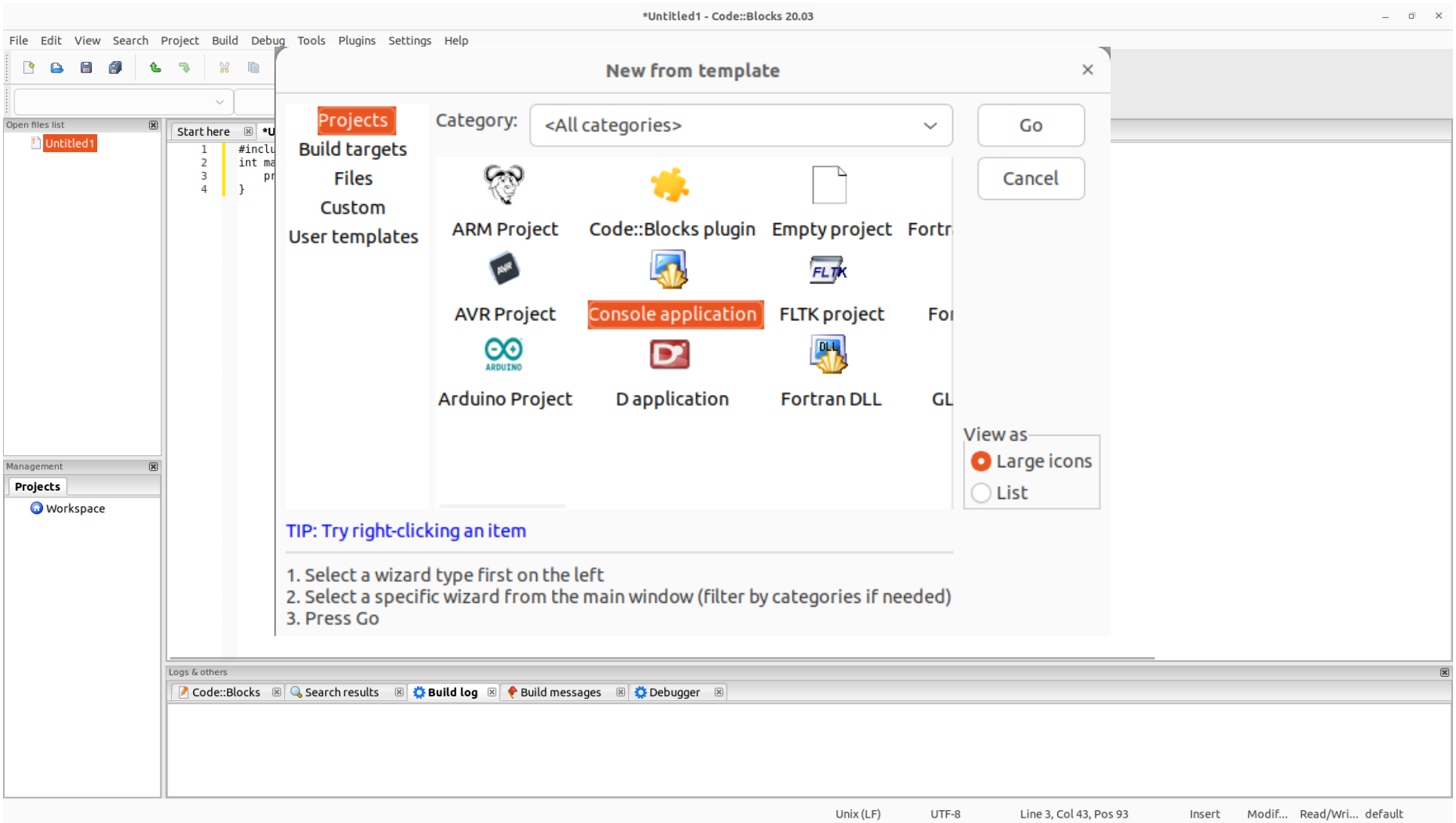


Run Code::Blocks for the first time: generate C-program **with** project.

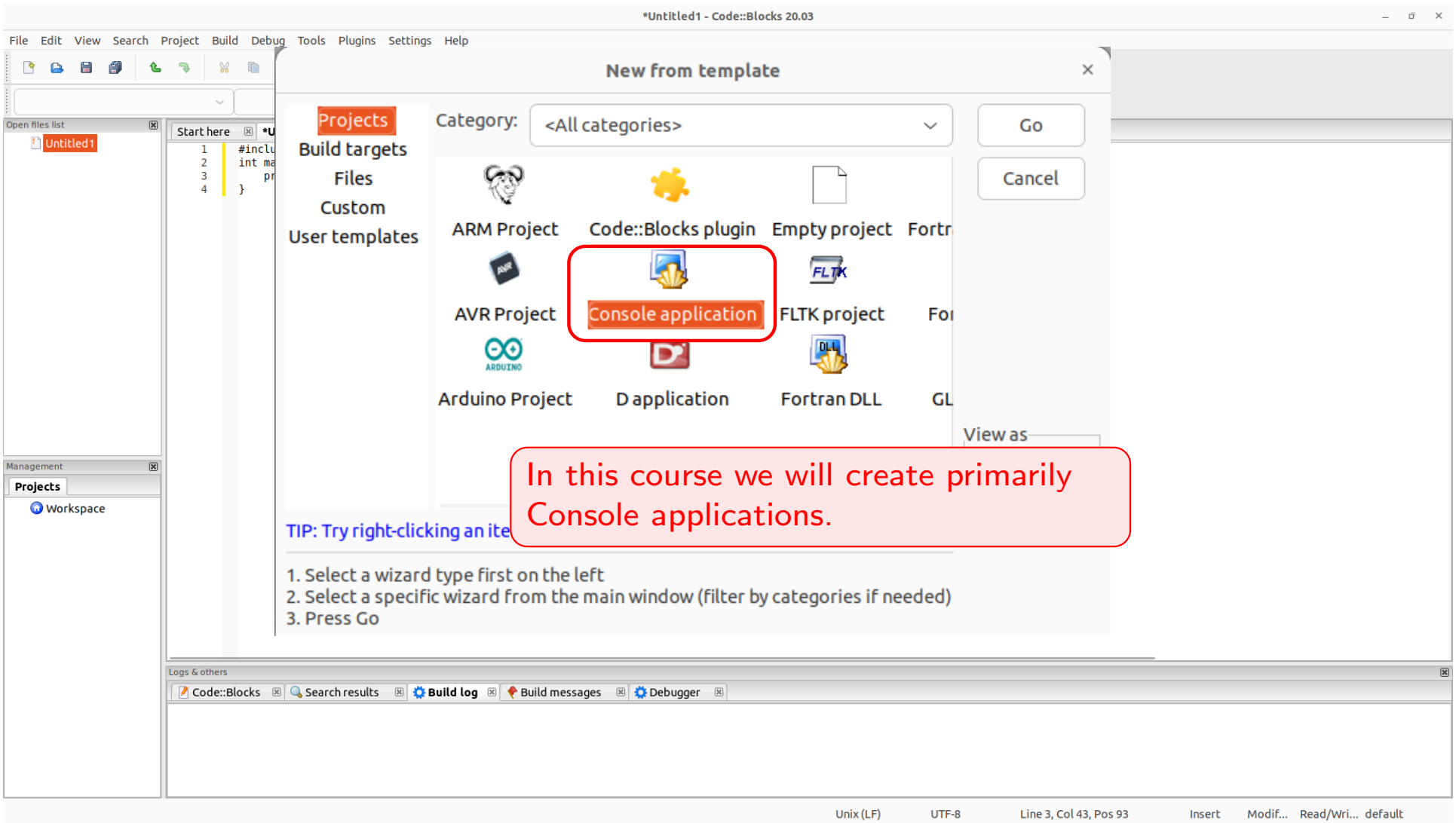


Choose a single-file for small self-contained stand-alone programs. For bigger multi-file programs, you need to start a new project. To start a new project, choose:
File->New->Project

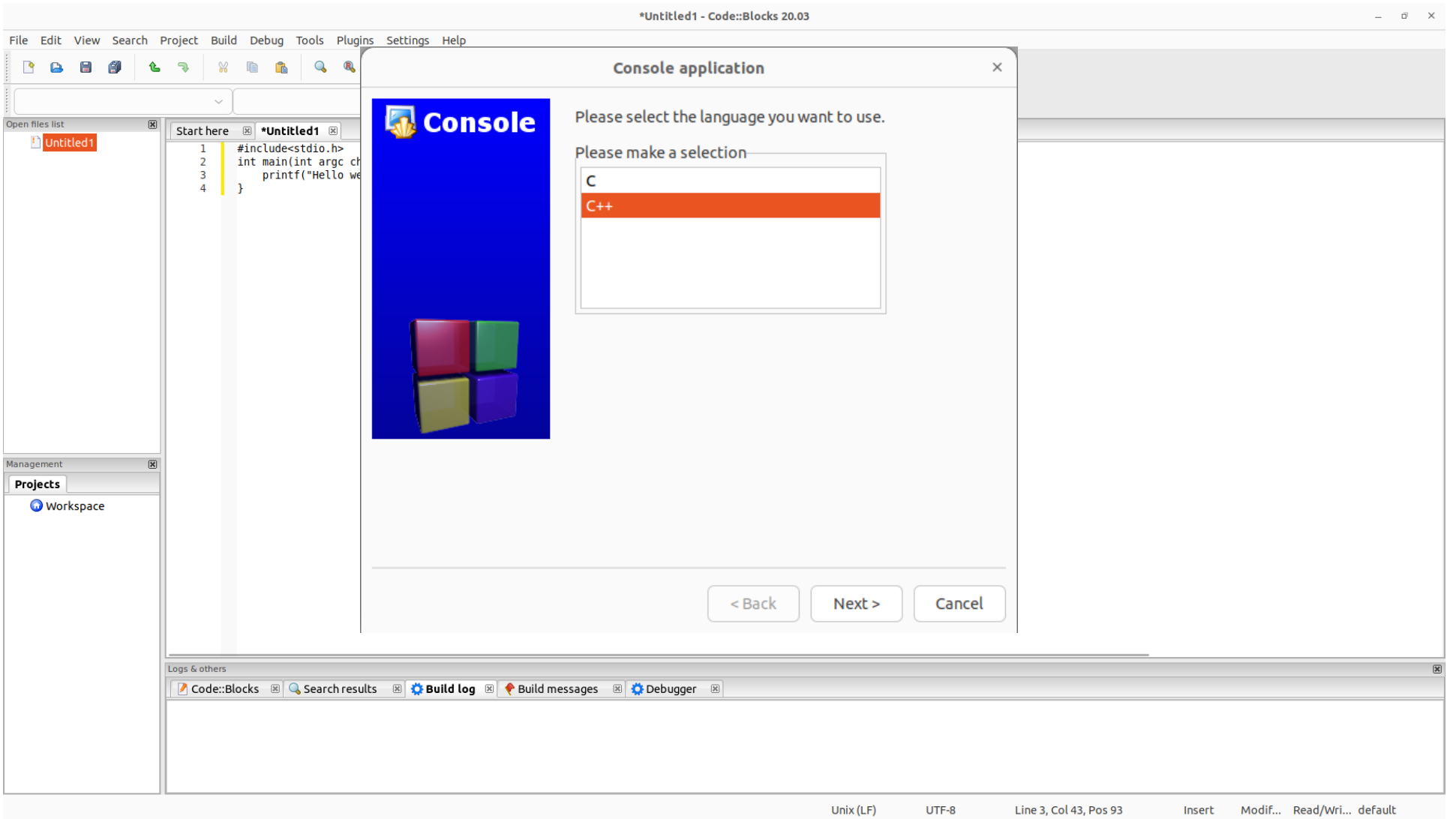
Run Code::Blocks for the first time: generate C-program **with** project.



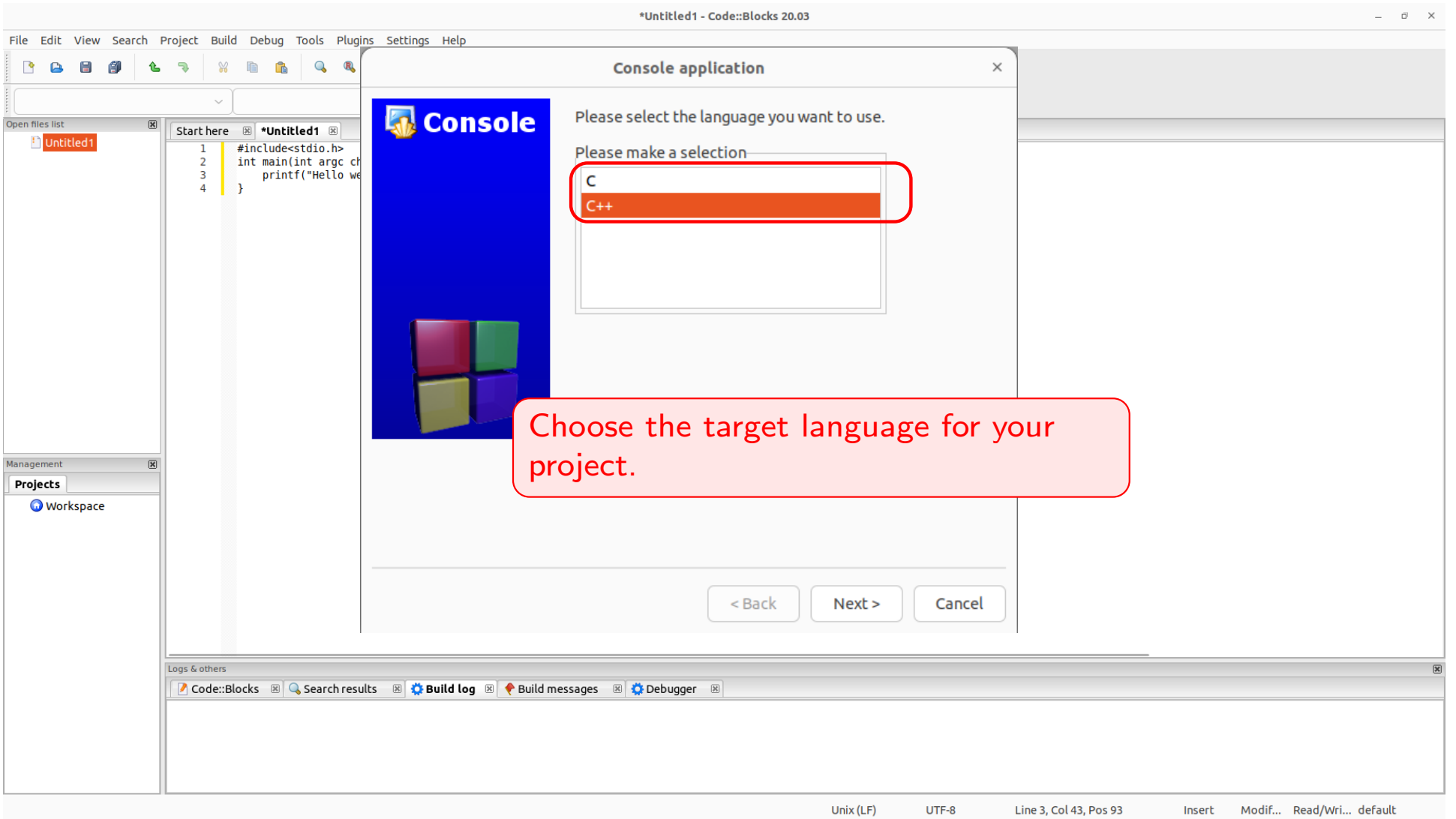
Run Code::Blocks for the first time: generate C-program **with** project.



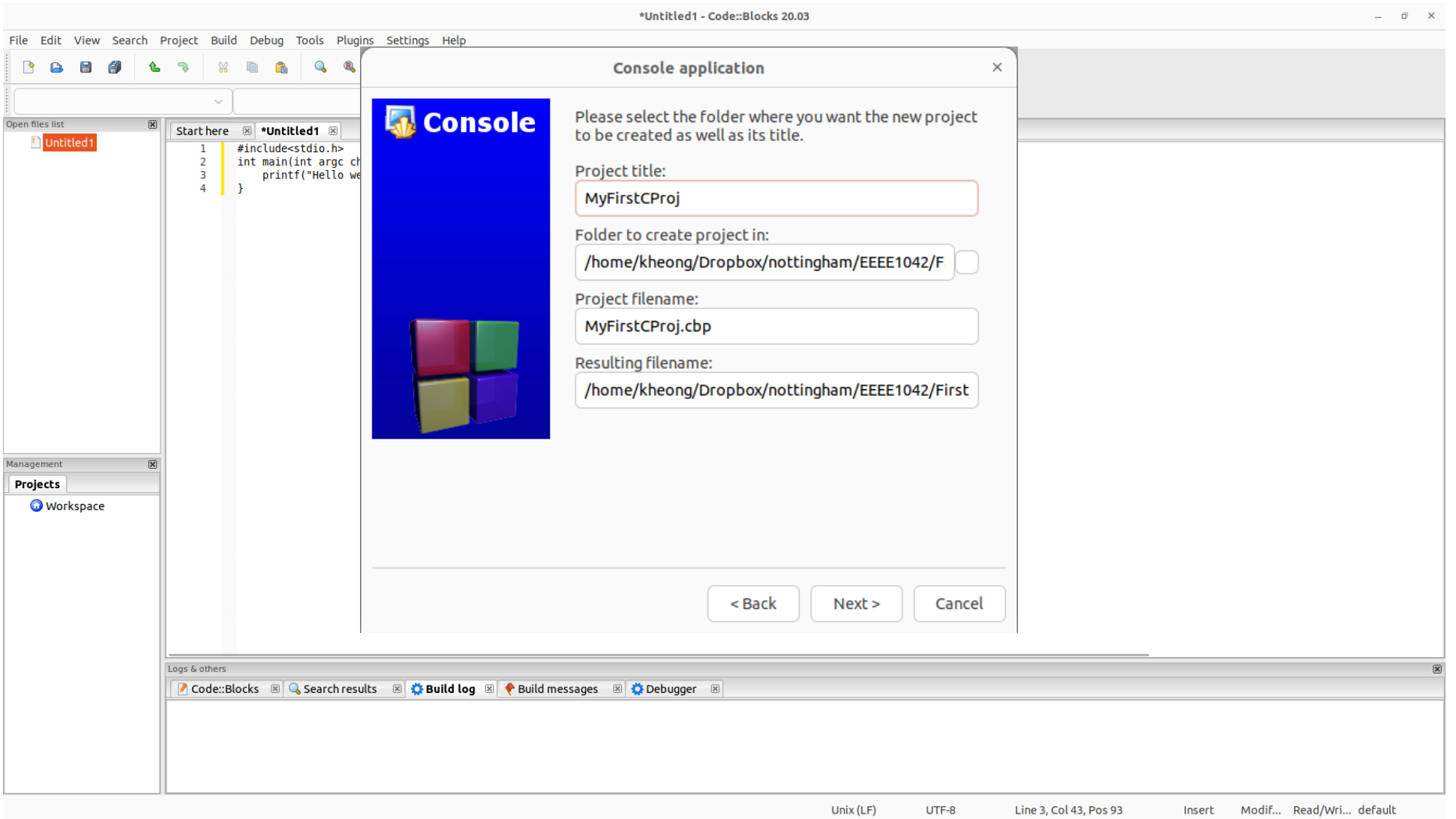
Run Code::Blocks for the first time: generate C-program **with** project.



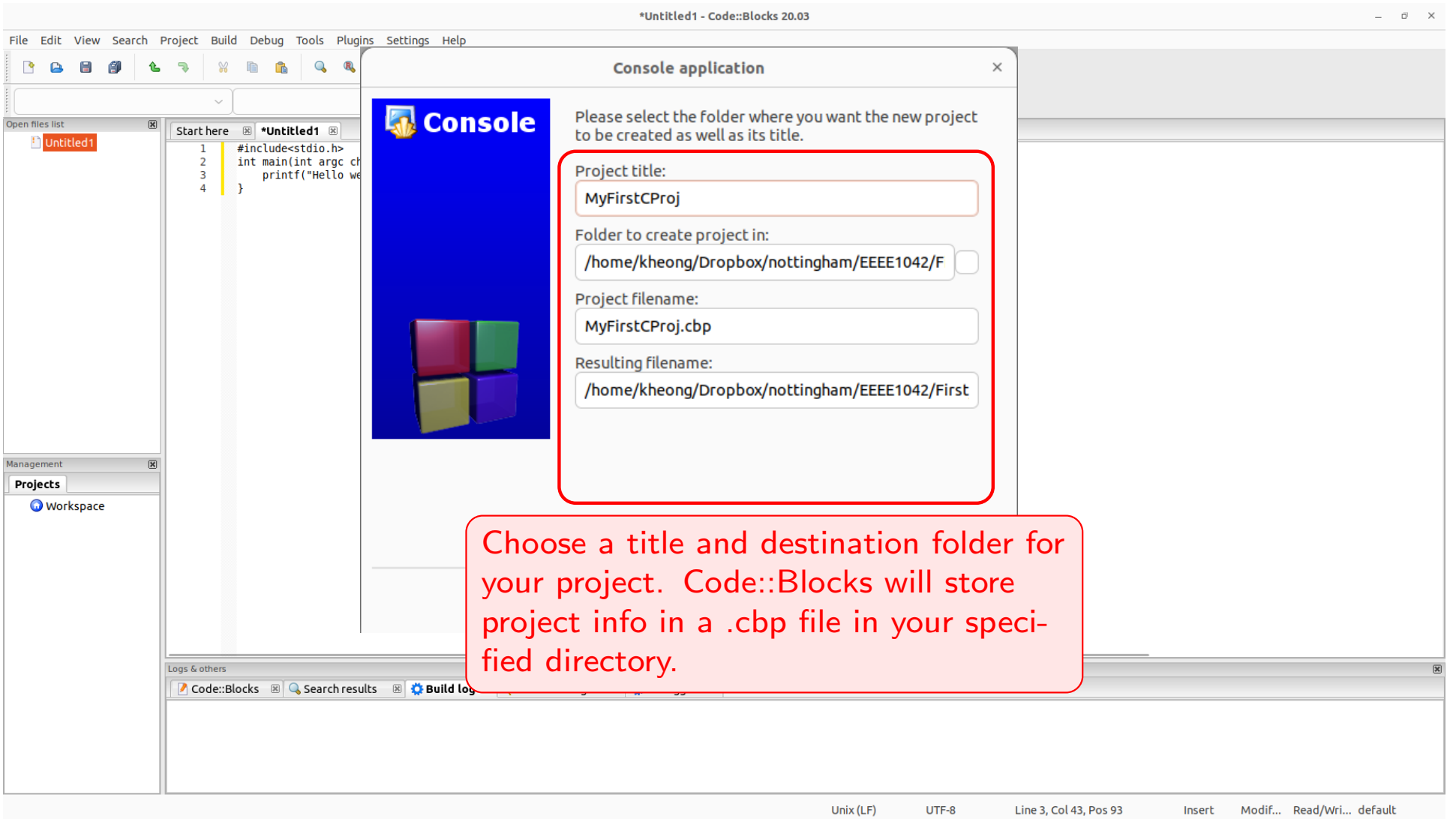
Run Code::Blocks for the first time: generate C-program **with** project.



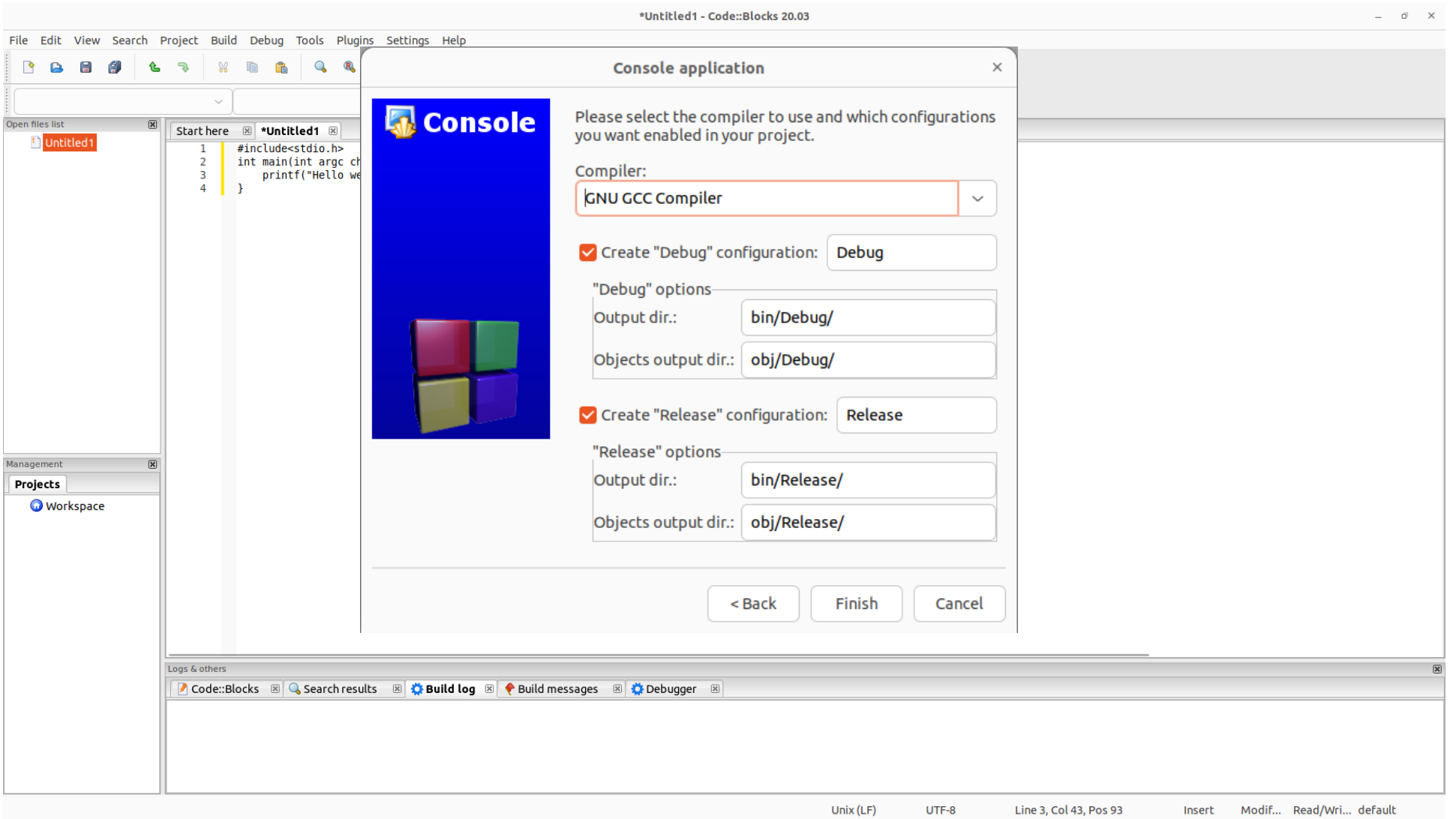
Run Code::Blocks for the first time: generate C-program **with** project.



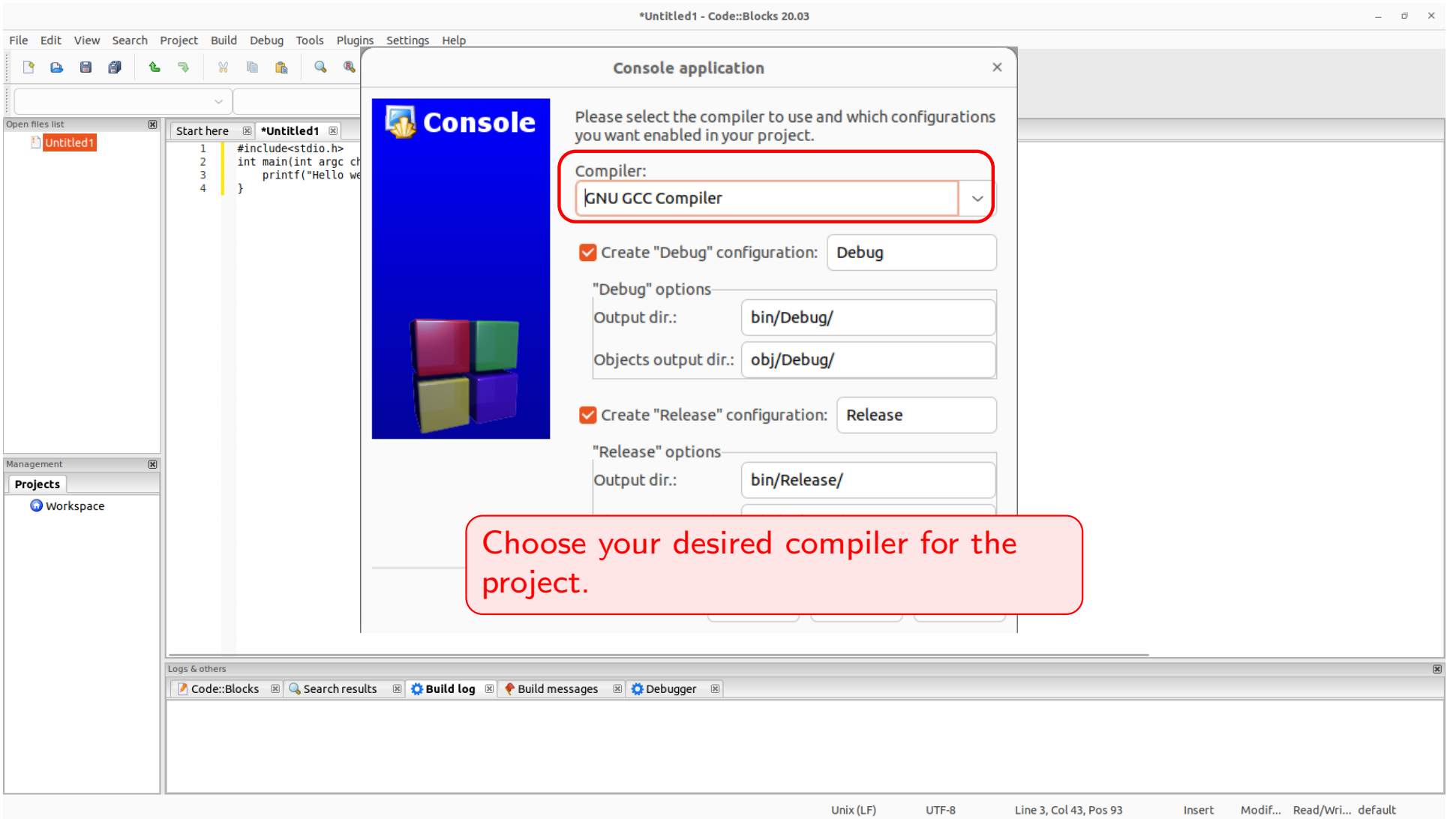
Run Code::Blocks for the first time: generate C-program **with** project.



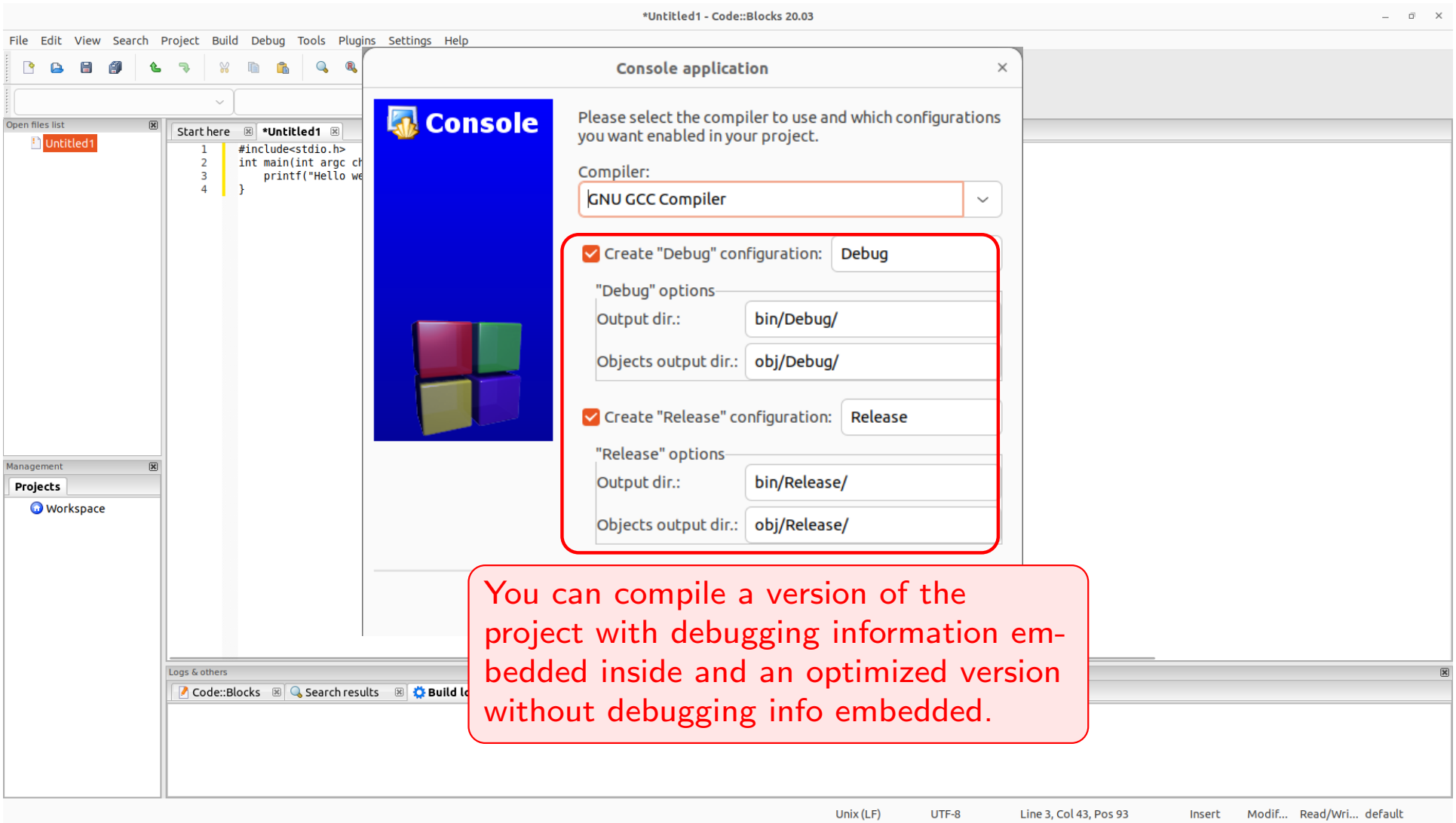
Run Code::Blocks for the first time: generate C-program **with** project.



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

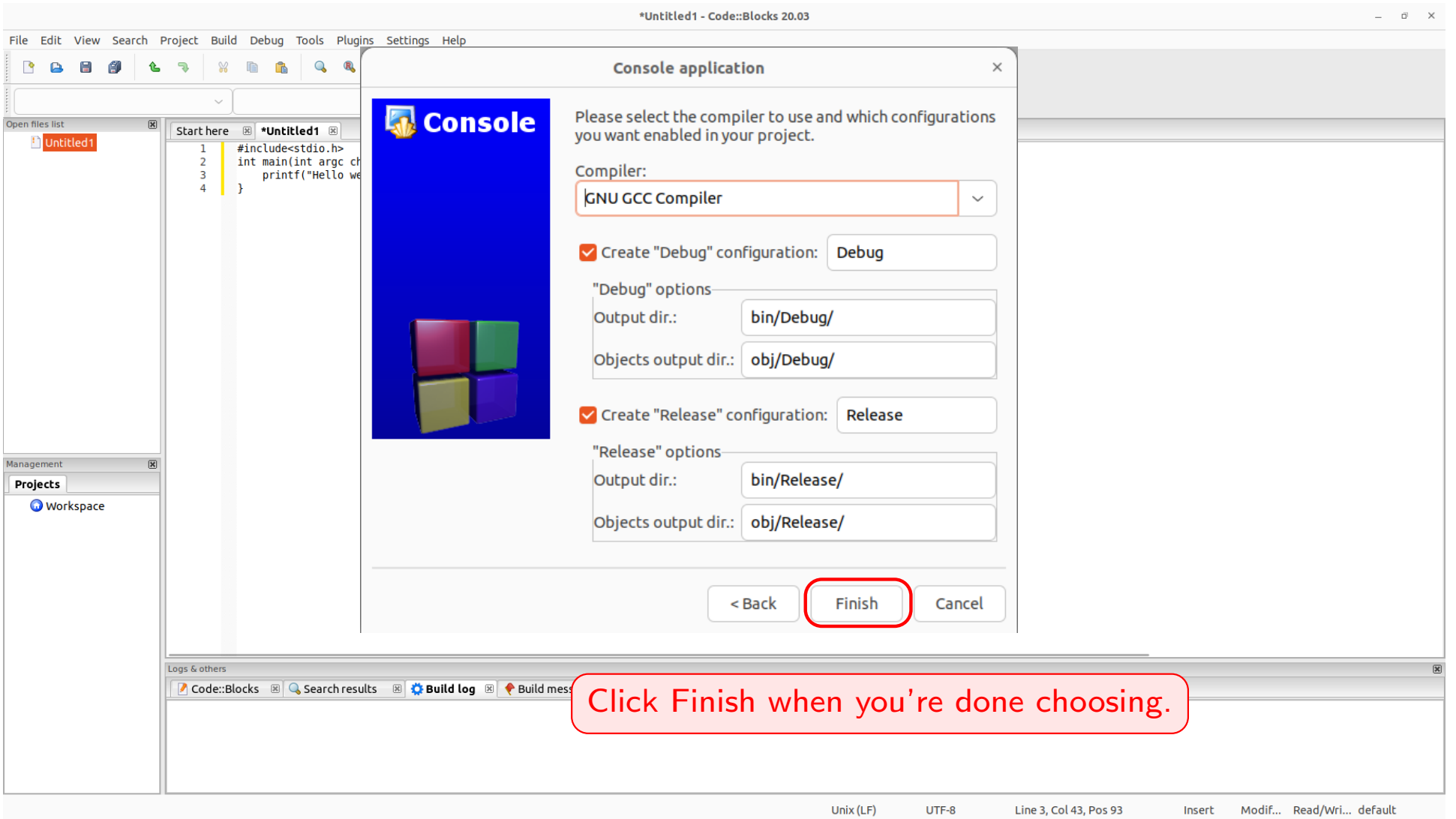


Run Code::Blocks for the first time: generate C-program **with** project.

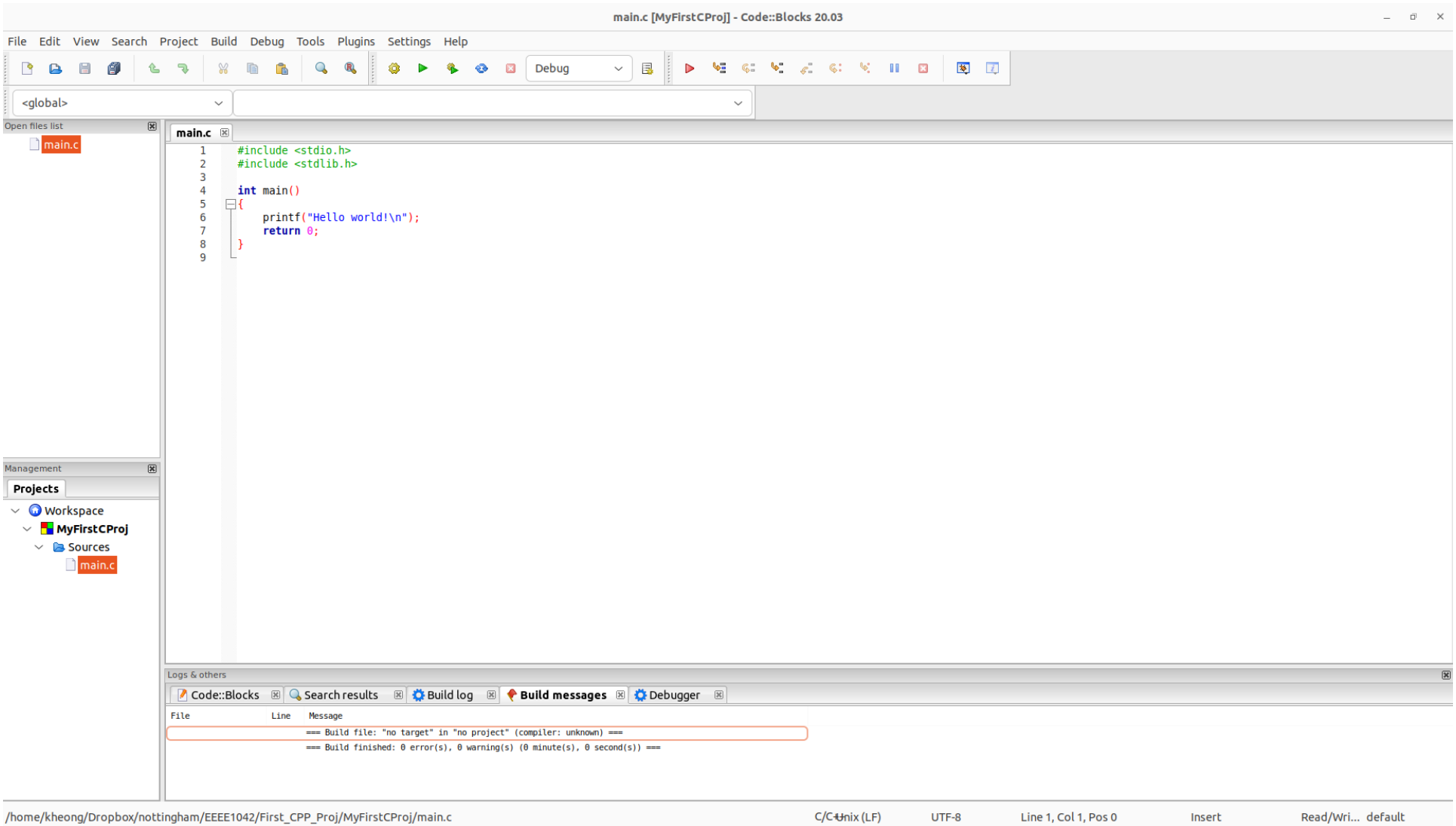


You can compile a version of the project with debugging information embedded inside and an optimized version without debugging info embedded.

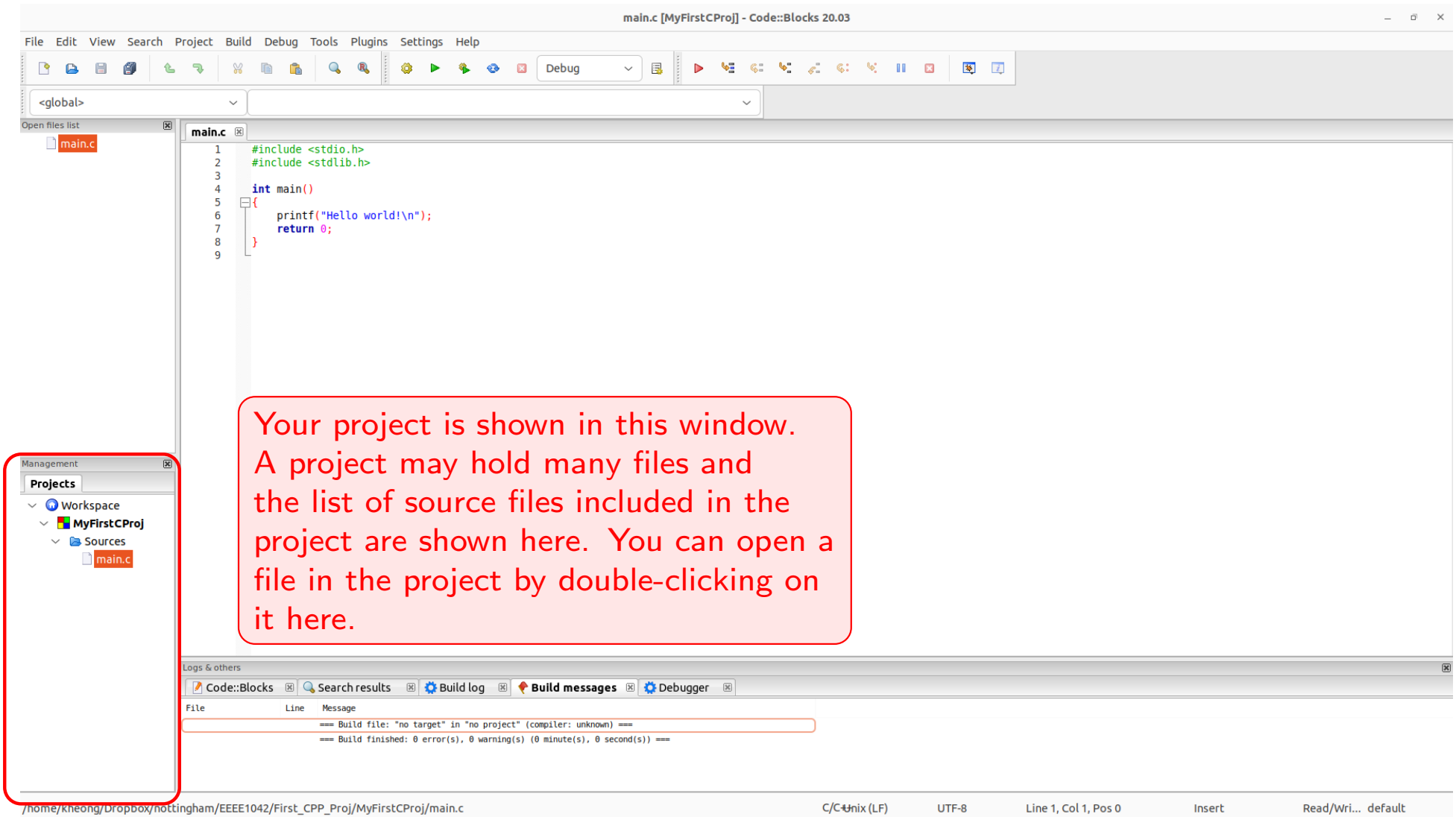
Run Code::Blocks for the first time: generate C-program **with** project.



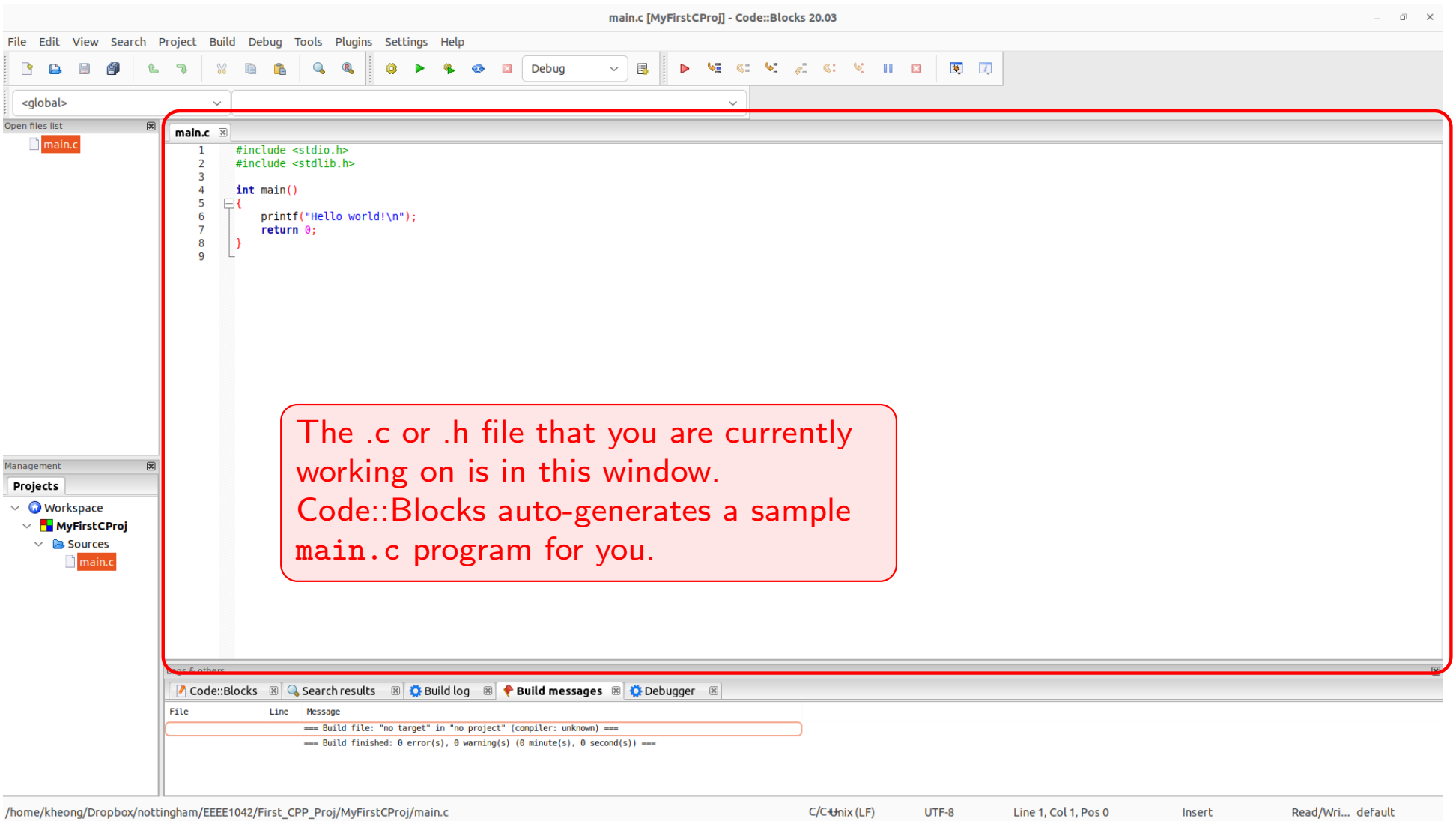
Run Code::Blocks for the first time: generate C-program **with** project.



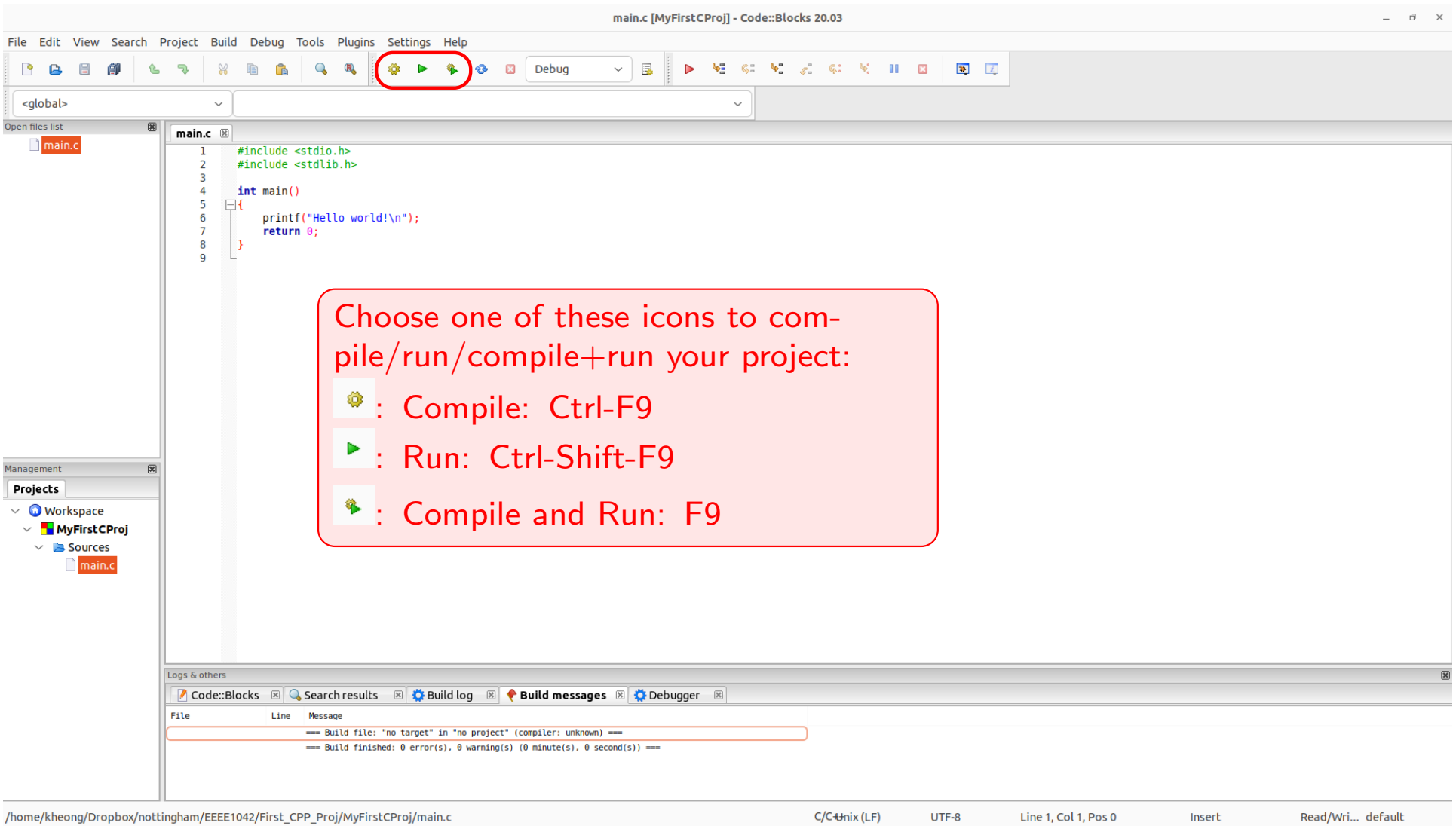
Run Code::Blocks for the first time: generate C-program **with** project.



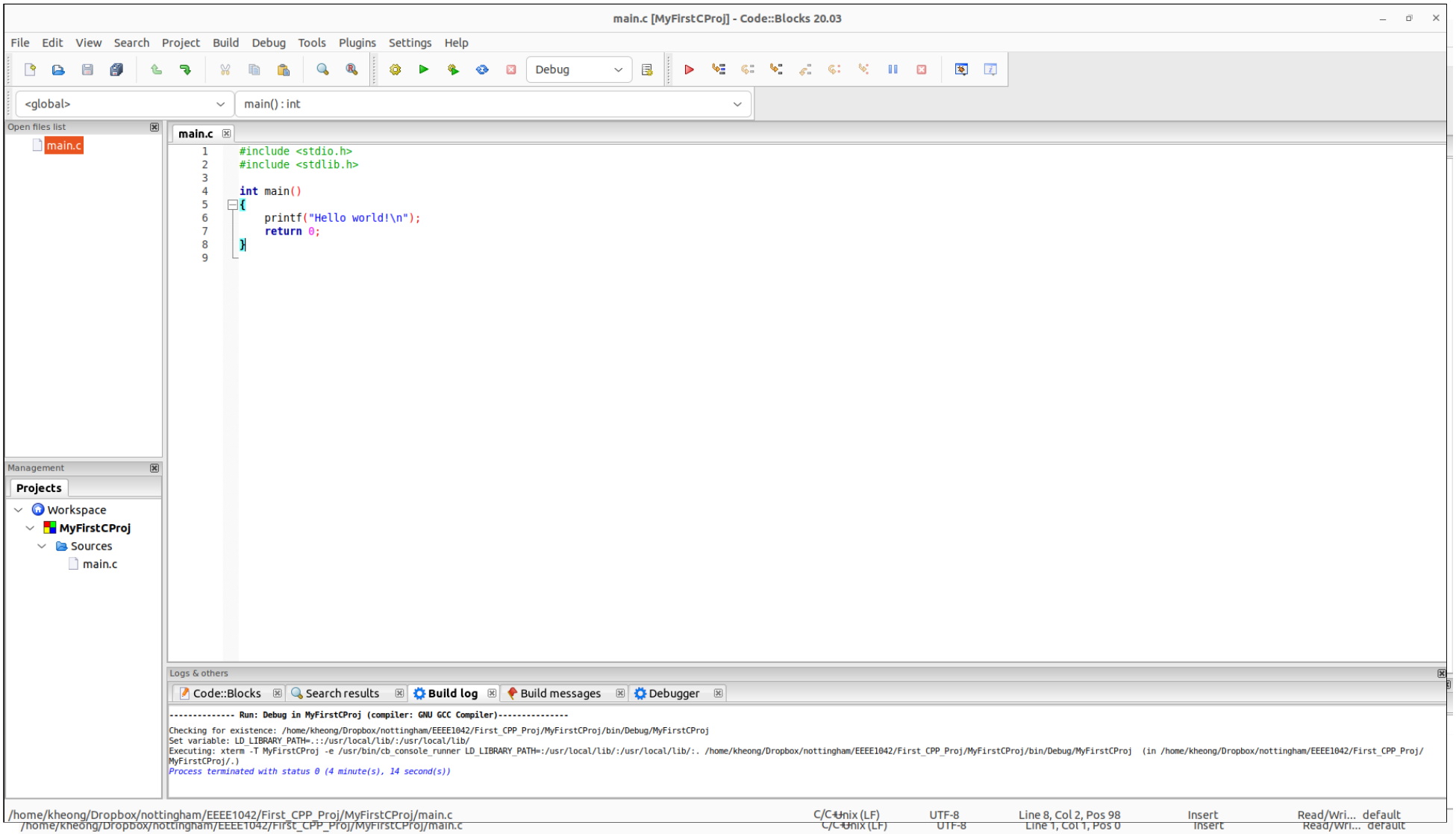
Run Code::Blocks for the first time: generate C-program **with** project.



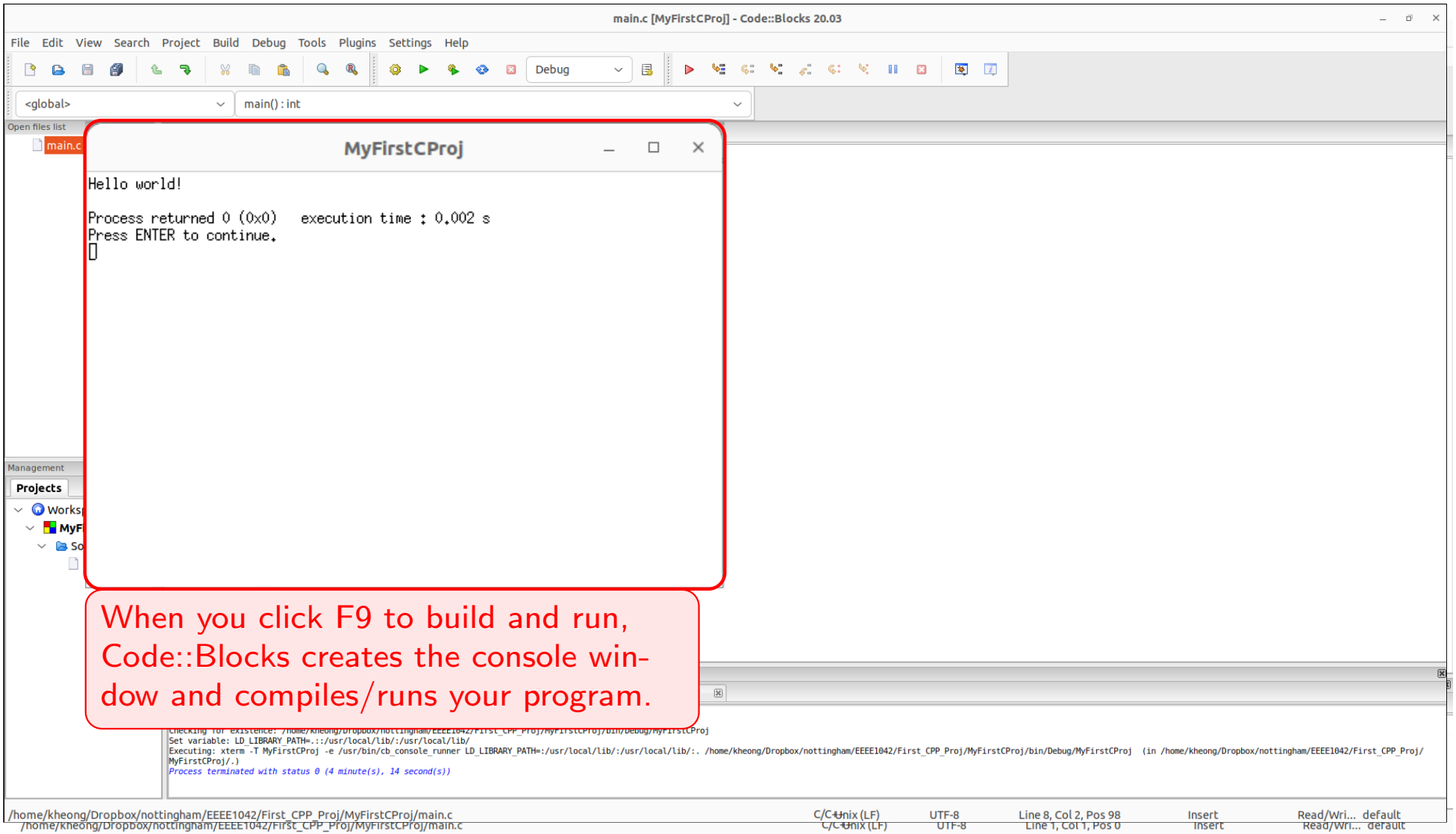
Run Code::Blocks for the first time: generate C-program **with** project.



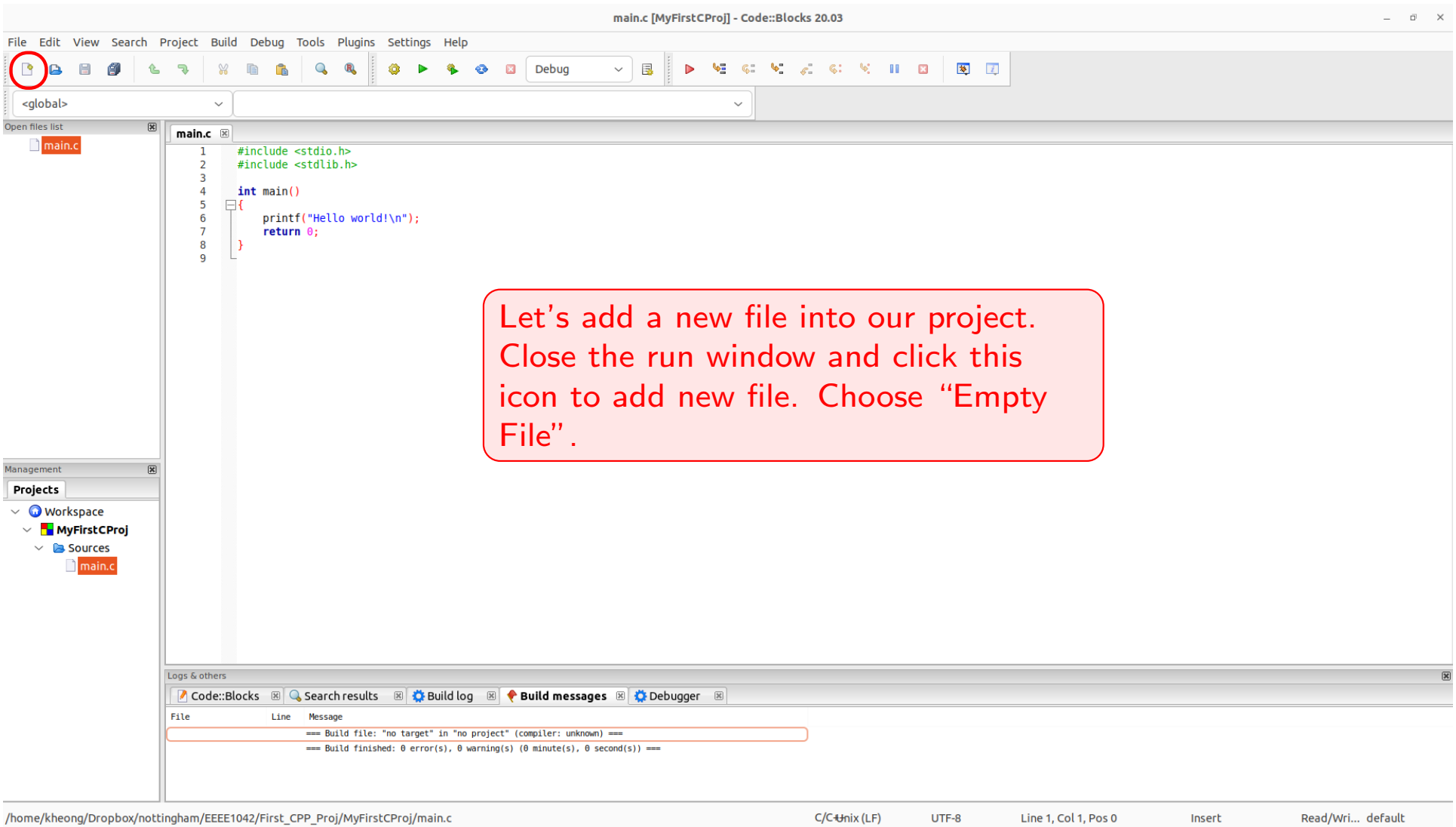
Run Code::Blocks for the first time: generate C-program **with** project.



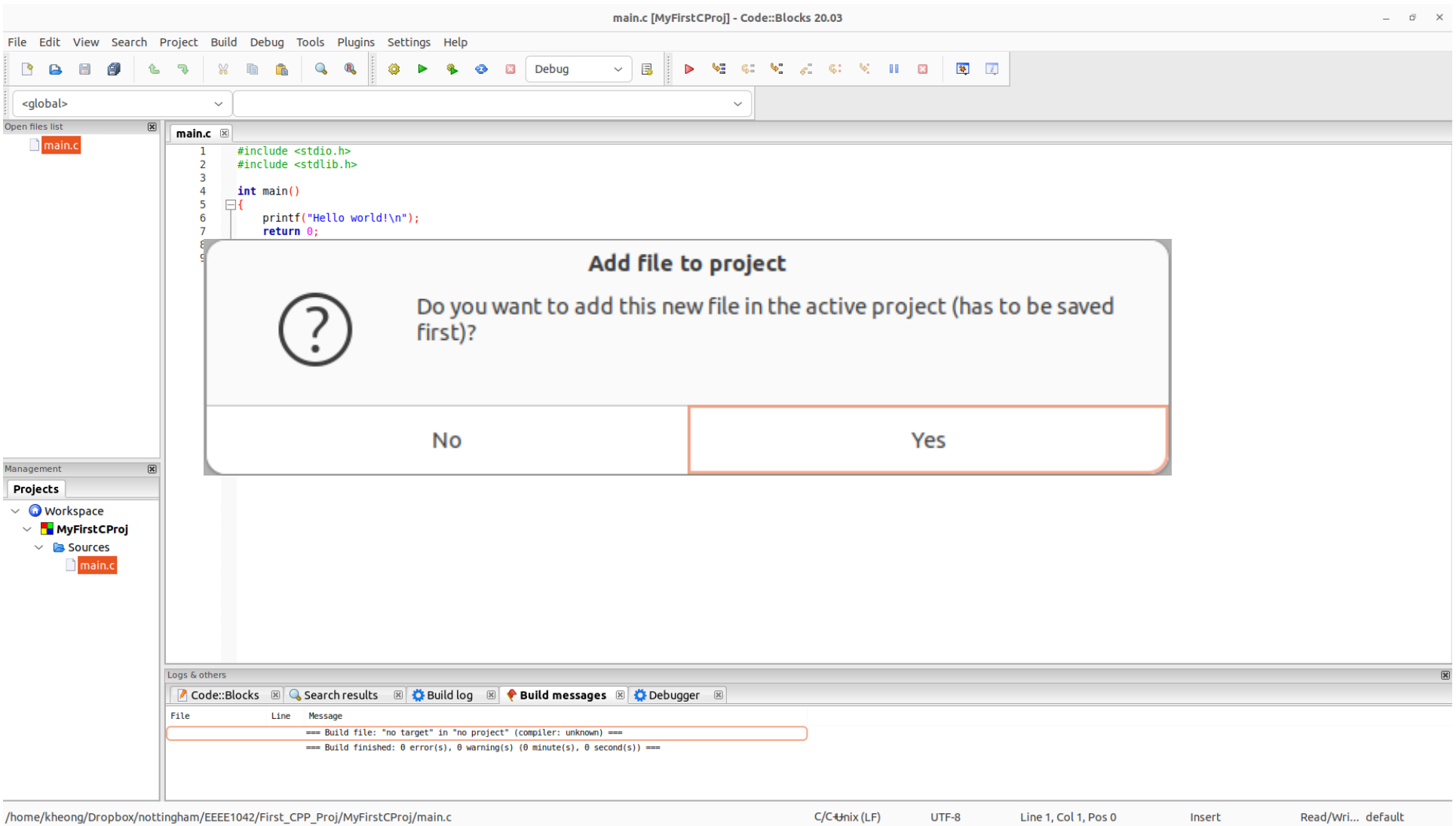
Run Code::Blocks for the first time: generate C-program **with** project.



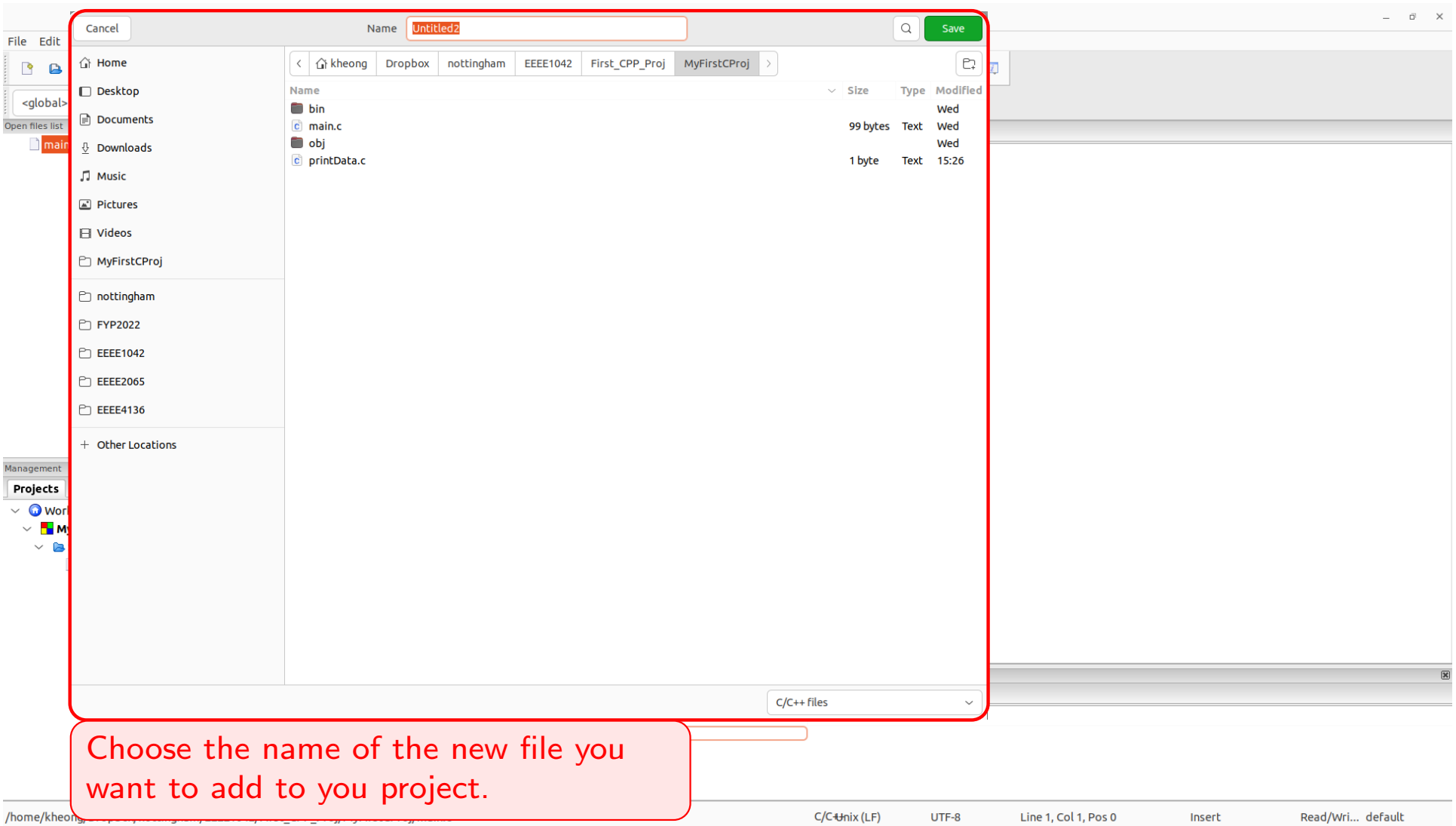
Run Code::Blocks for the first time: generate C-program **with** project.



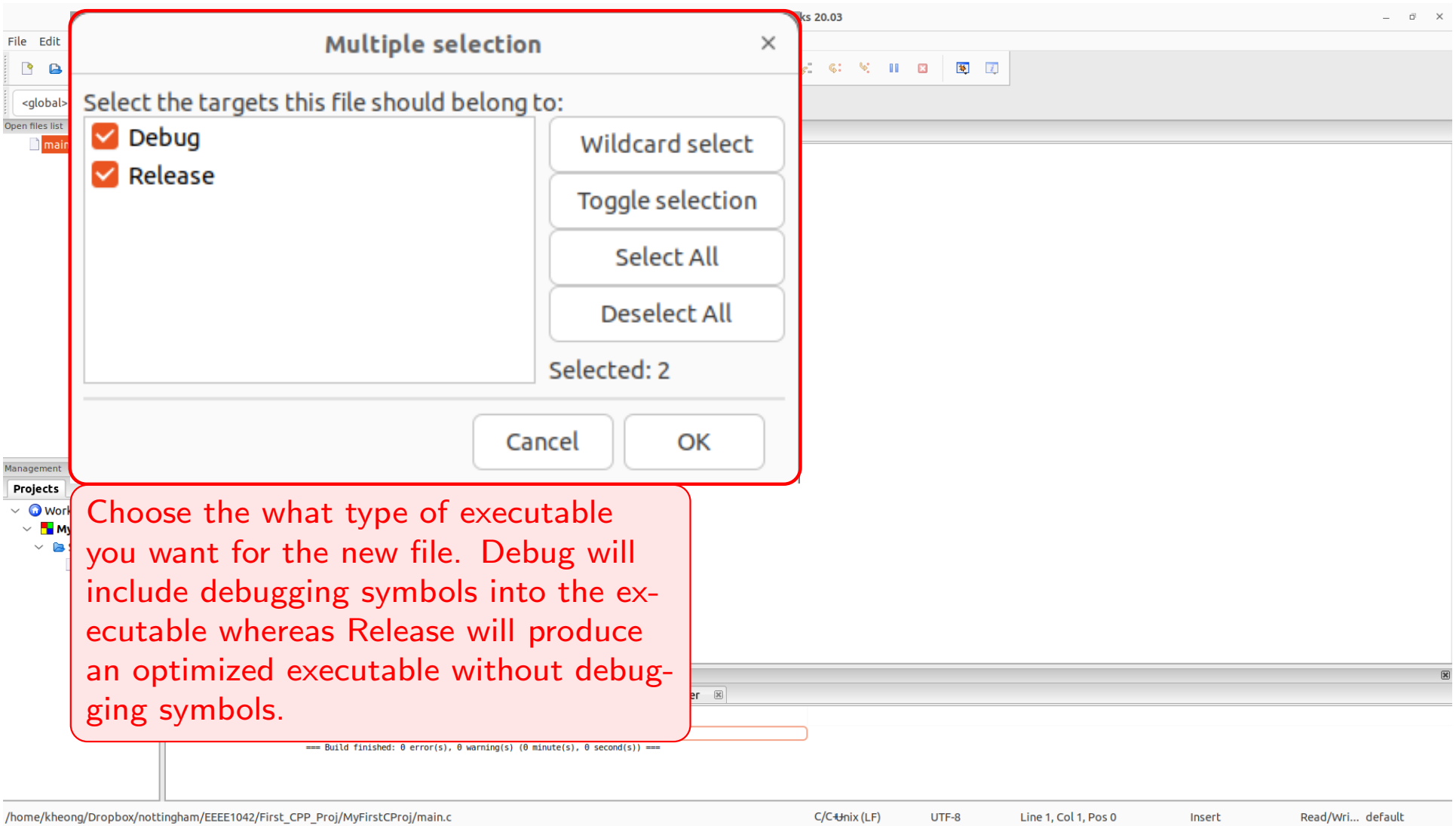
Run Code::Blocks for the first time: generate C-program **with** project.



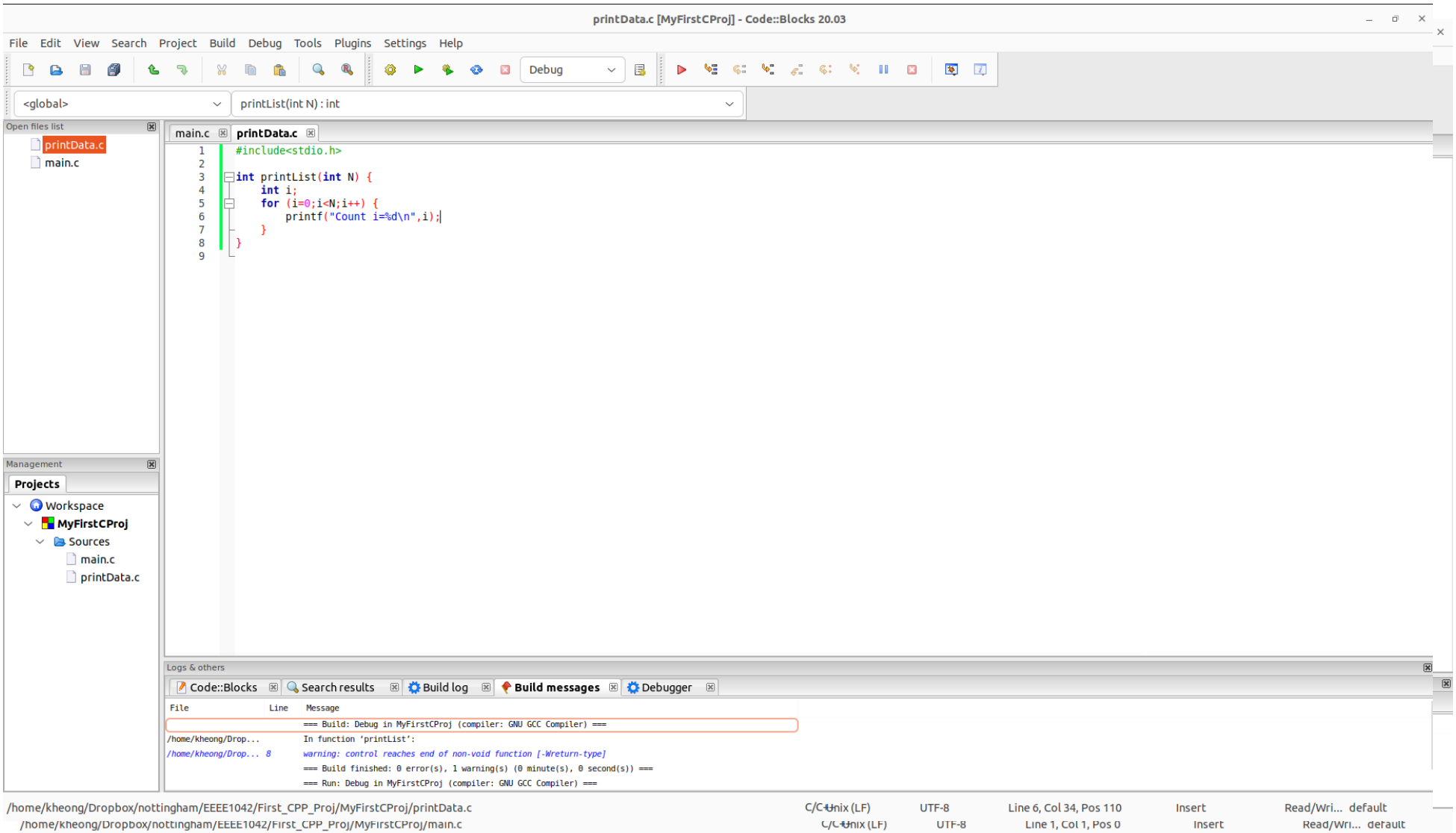
Run Code::Blocks for the first time: generate C-program **with** project.



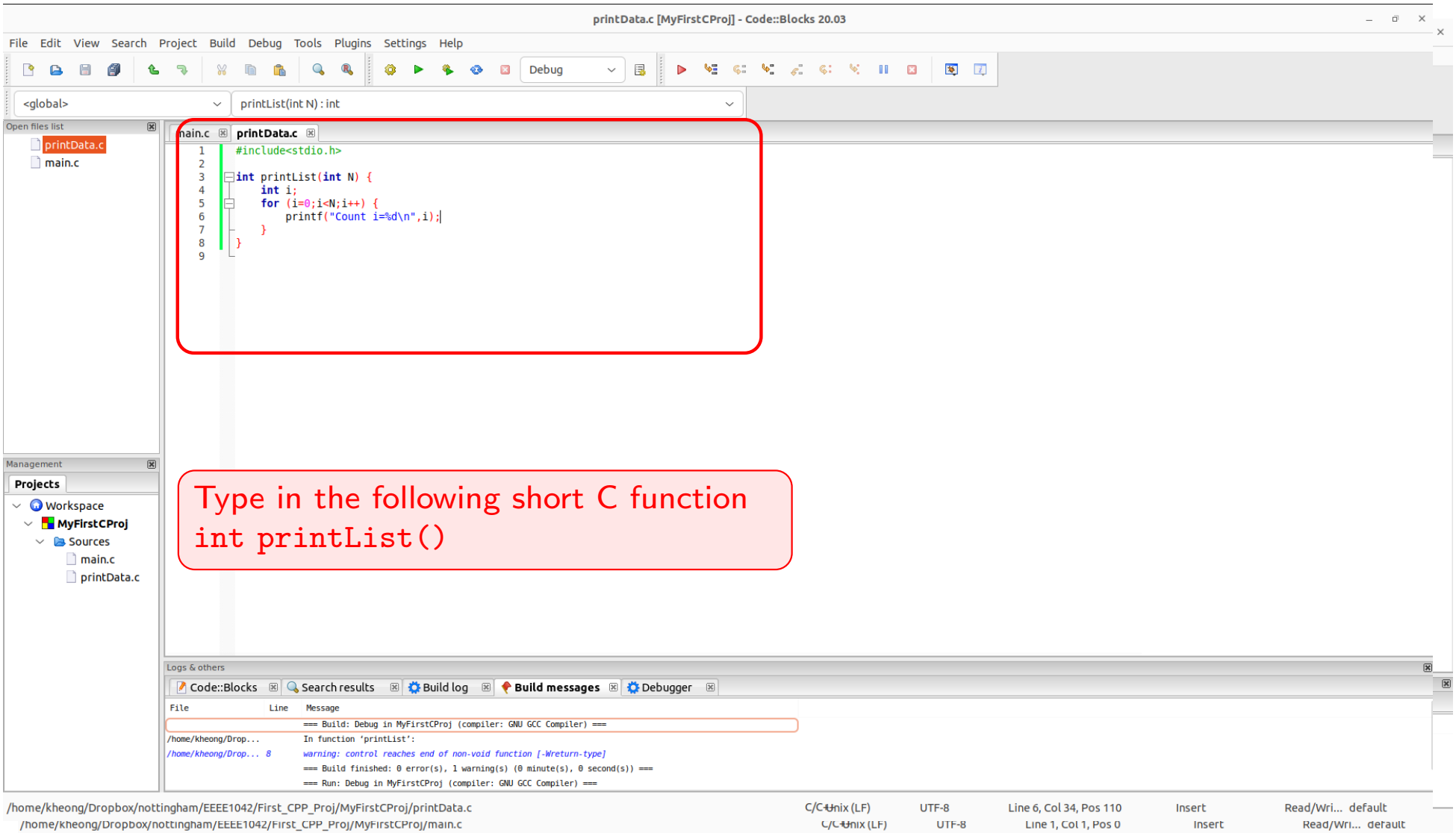
Run Code::Blocks for the first time: generate C-program **with** project.



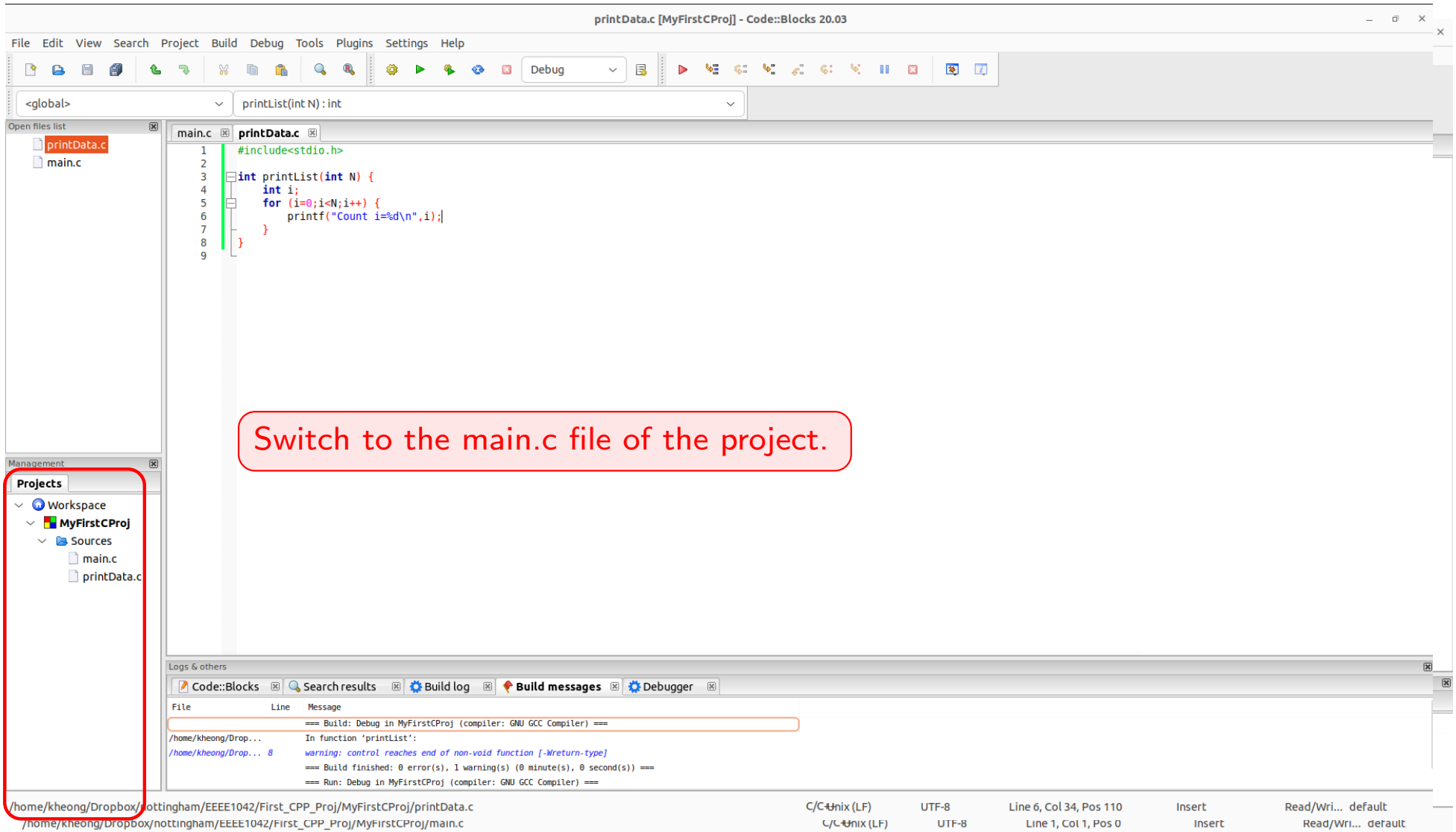
Run Code::Blocks for the first time: generate C-program **with** project.



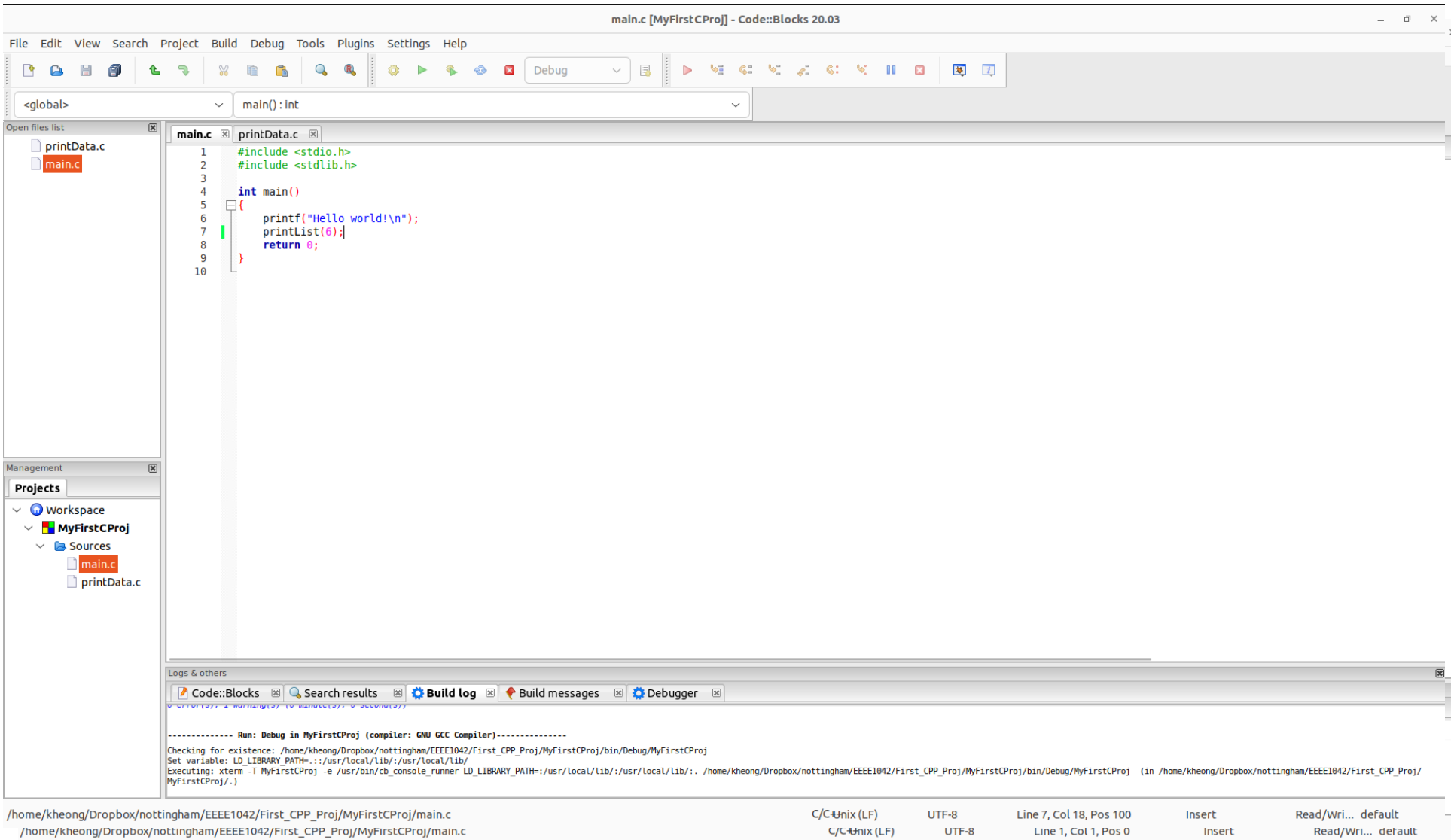
Run Code::Blocks for the first time: generate C-program **with** project.



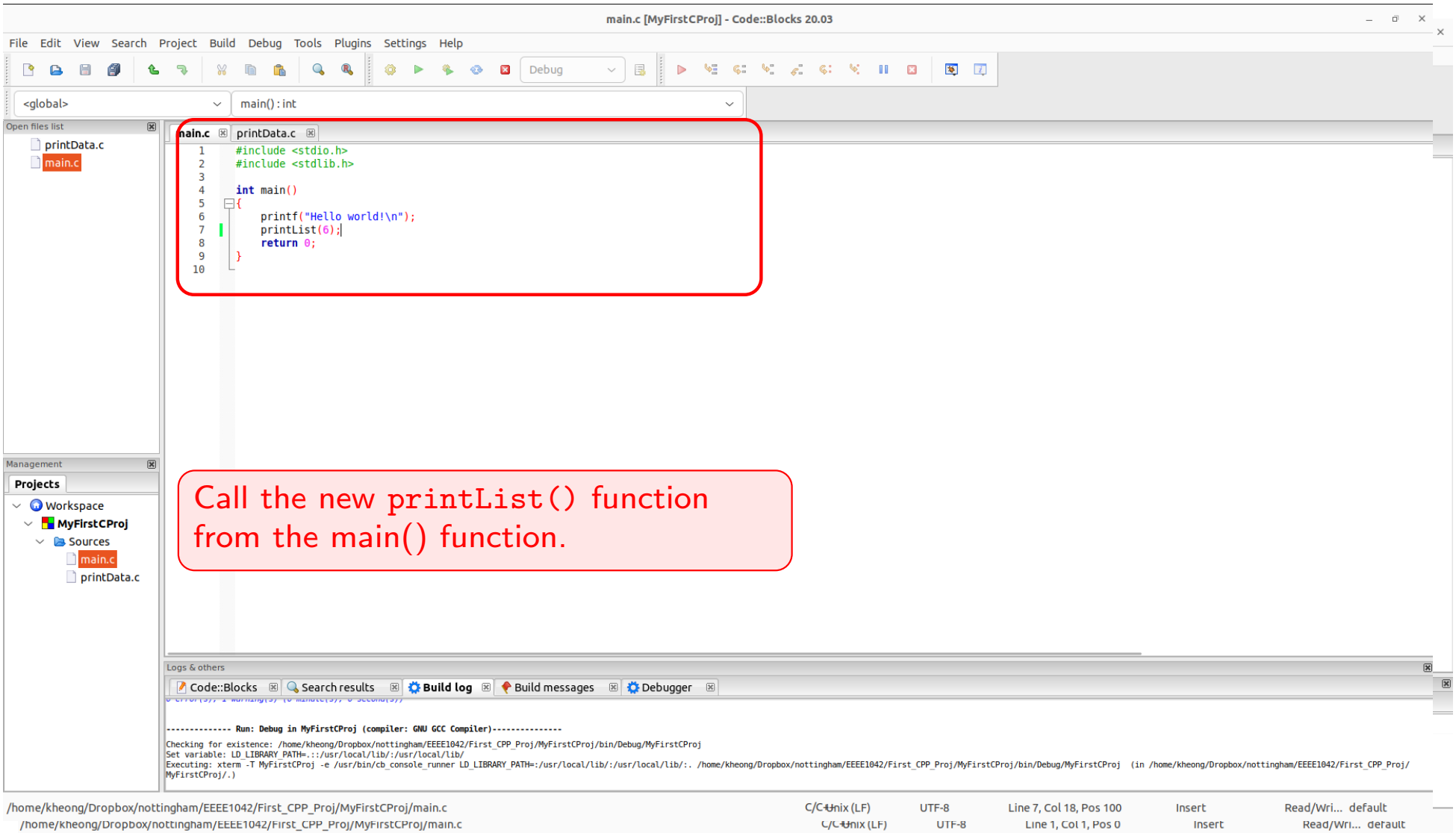
Run Code::Blocks for the first time: generate C-program **with** project.



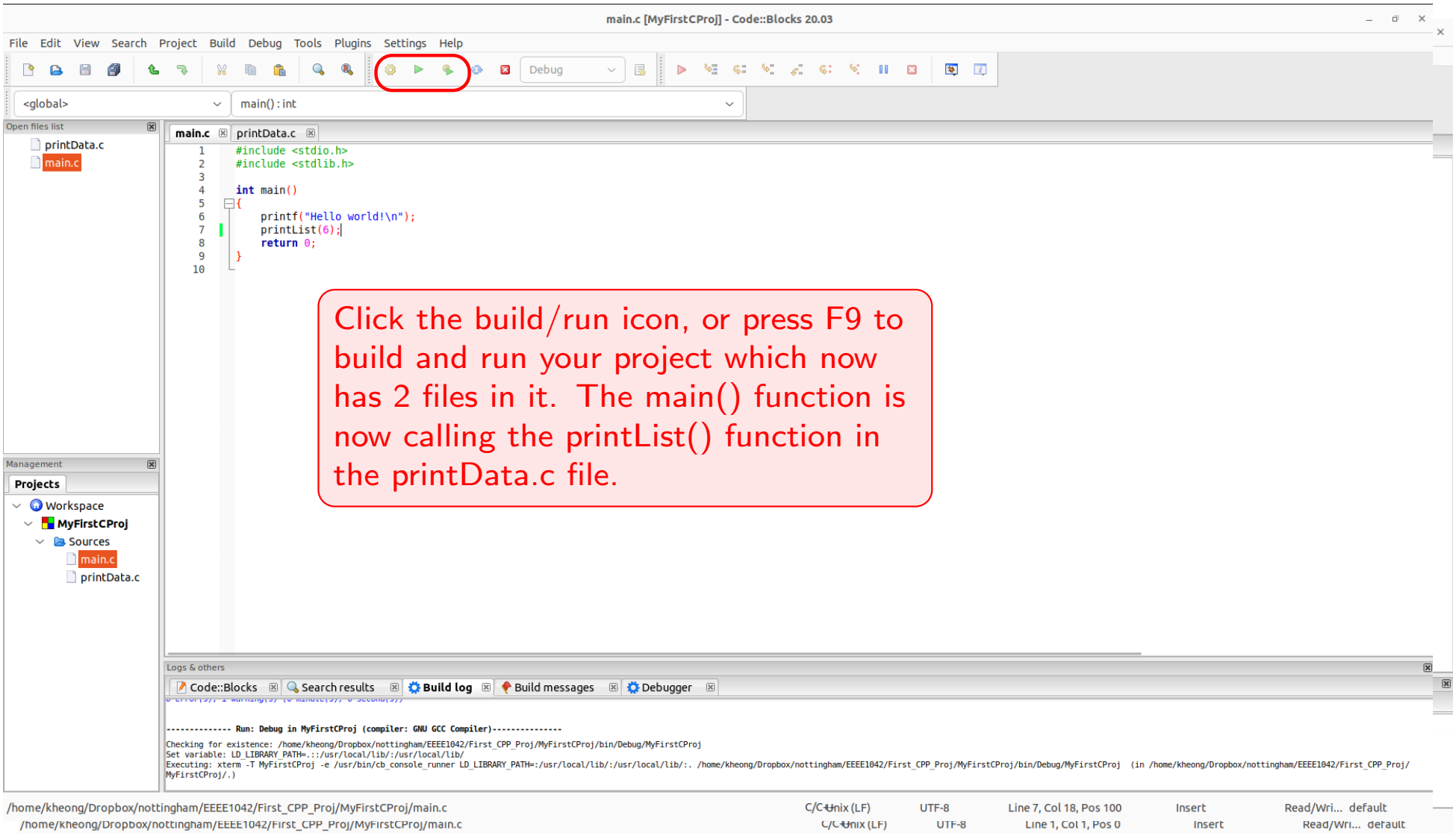
Run Code::Blocks for the first time: generate C-program **with** project.



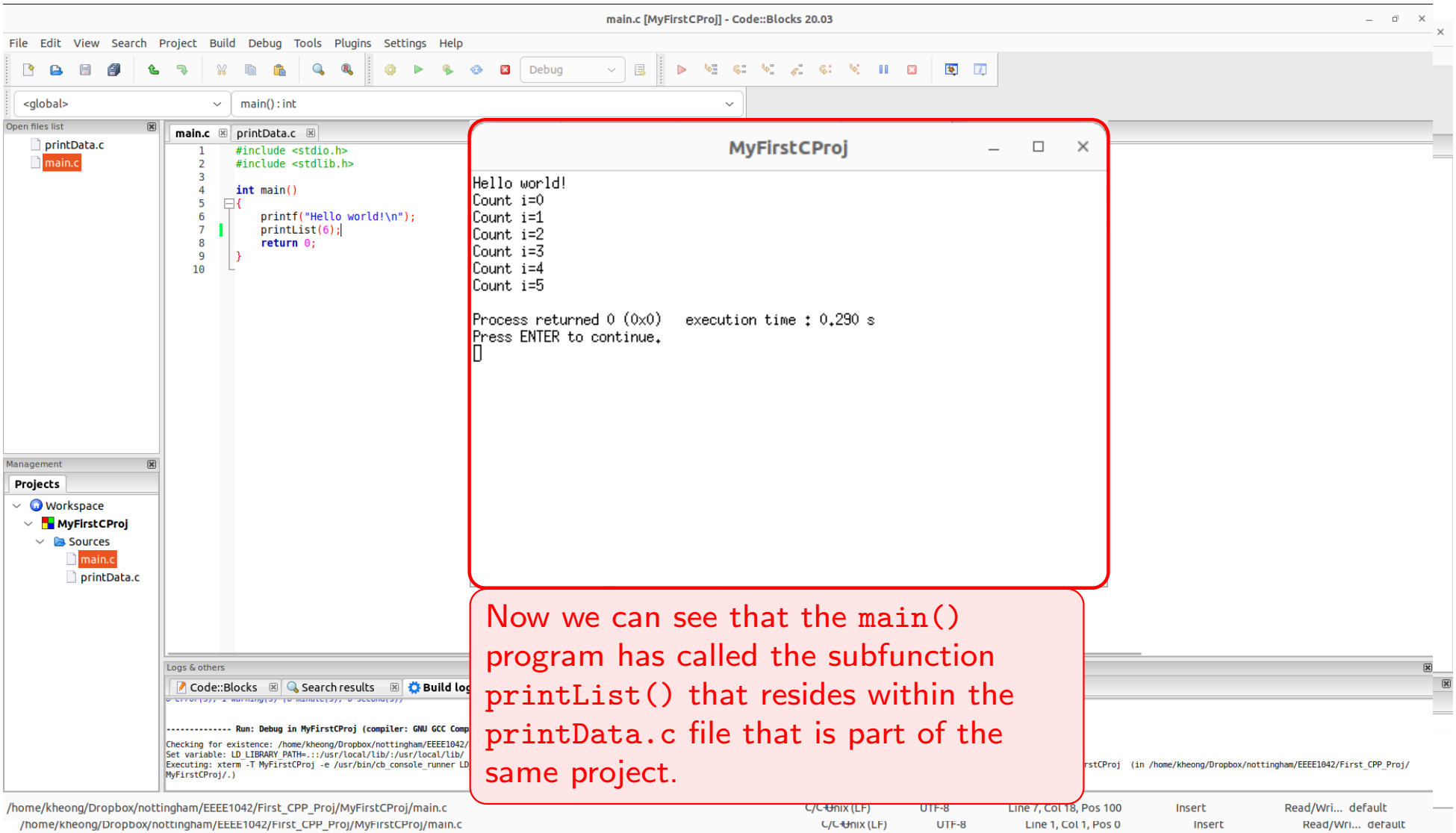
Run Code::Blocks for the first time: generate C-program **with** project.



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



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



Maintainable, Modular, Re-useable Code

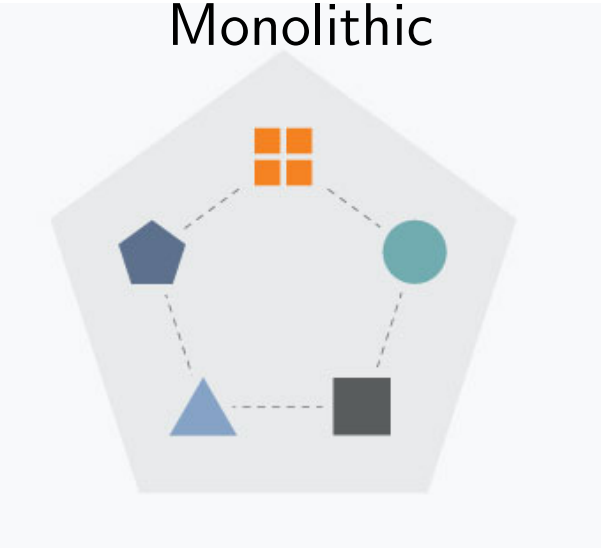
- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.

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

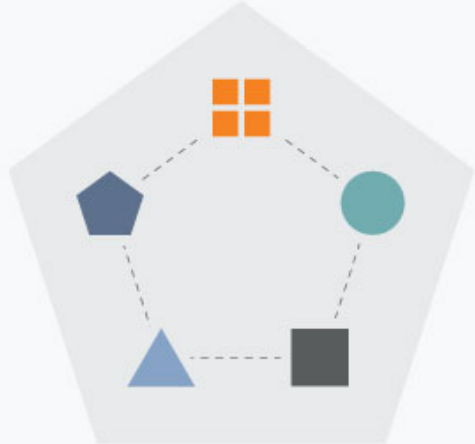
- The **New Project** enables greater **modularity** compared to the **New File** option in Code::Blocks.
- Here we stress the **Importance of modular programming** especially for bigger projects.
- Monolithic: One single big monolithic chunk of code $\rightarrow$ **unmaintainable, not reuseable**.



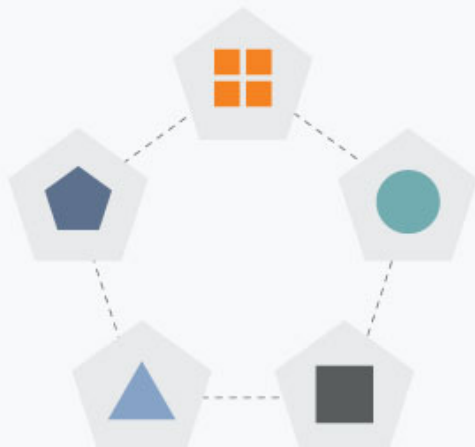
Maintainable, Modular, Re-useable Code

- 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 reuseable**.

Monolithic



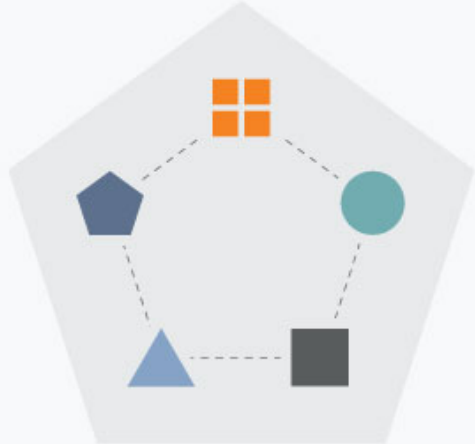
Modular



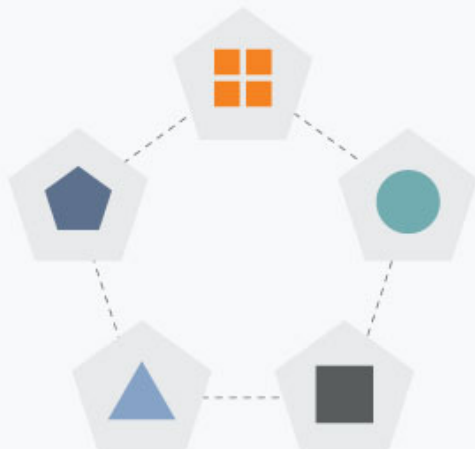
Maintainable, Modular, Re-useable Code

- 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 reuseable**.
- Modularity is **implemented by putting code into functions**.

Monolithic



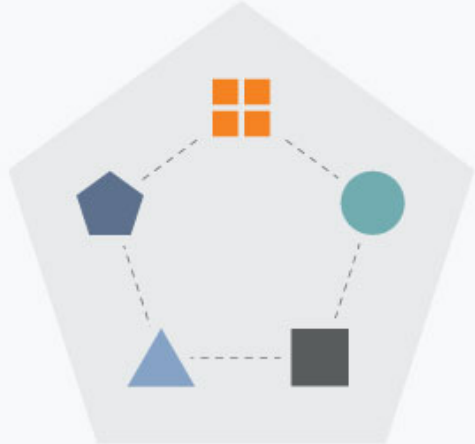
Modular



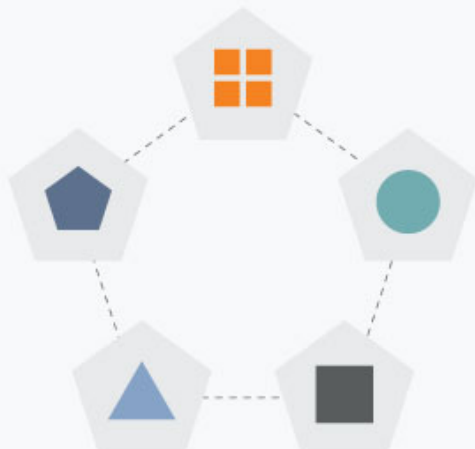
Maintainable, Modular, Re-useable Code

- 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 reuseable**.
- Modularity is **implemented by putting code into functions**.
- Each function can be re-used over and again, when handled/documented properly.

Monolithic



Modular



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 $\rightarrow$ 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 $\rightarrow$ 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:

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