

CS201: Recitation 1

GNU Compiler, Server, Makefile

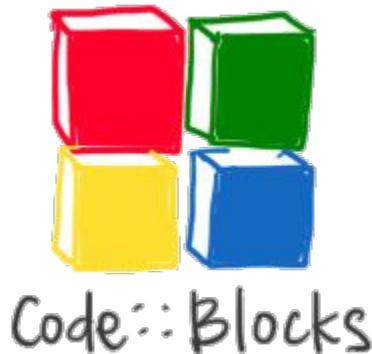
(not writing any code)

Content

- 1) Downloading an IDE with GNU compiler
- 2) Running your first simple C++ code
- 3) Running your first simple C++ on the server
- 4) Headers and running projects with multiple files
- 5) Makefile, making compiling your project a bit easier
- 6) Debugging your code in IDE

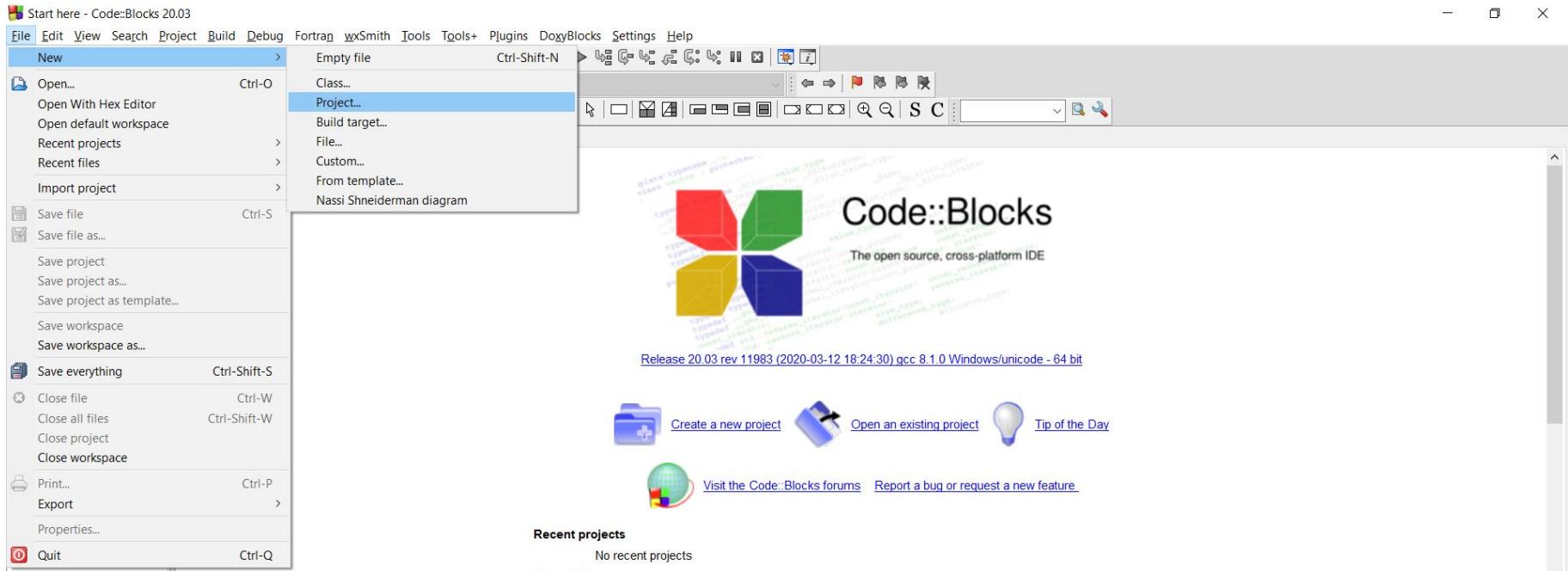
1. Download an IDE

- 1) One good option is CodeBlocks:
<https://www.codeblocks.org/downloads/binaries/>
- 2) For windows make sure you download 'mingw' option



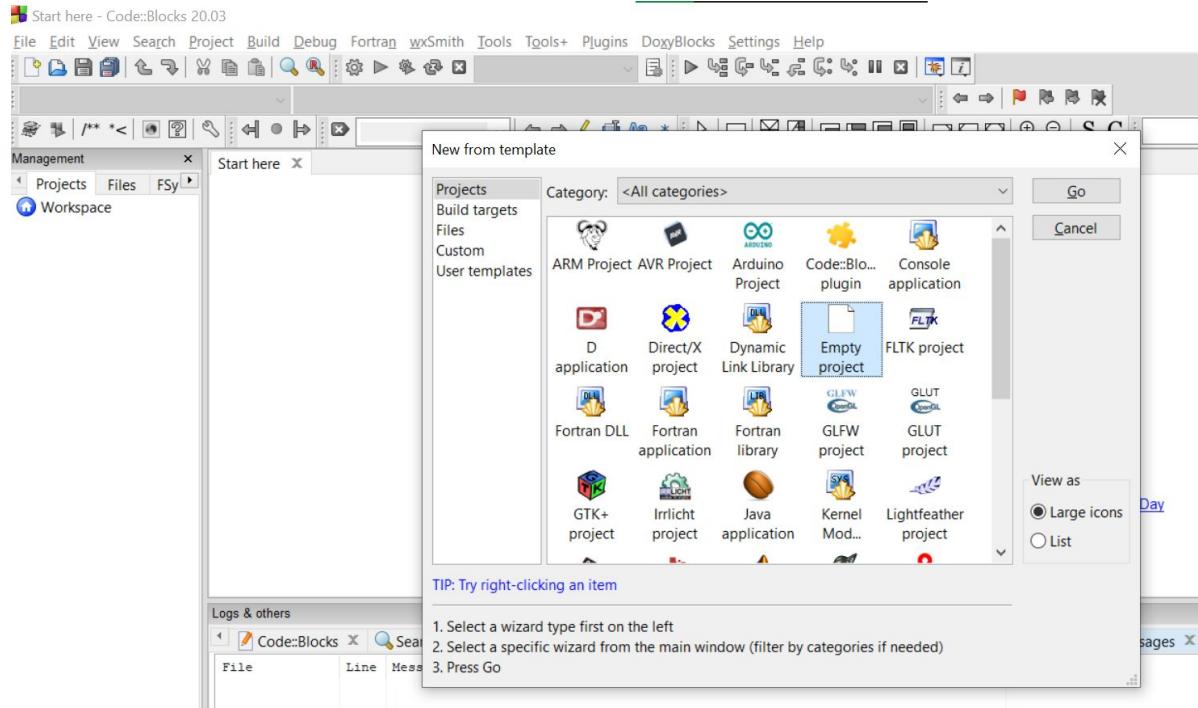
2. Create and run a simple C++ code as a part of a project

In upper left corner click: File > New > Project



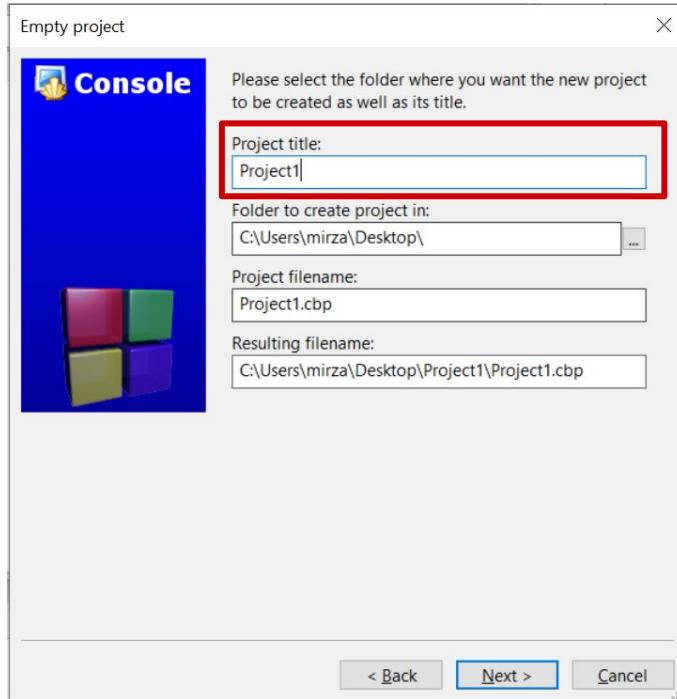
2. Create and run a simple C++ code as a part of a project

Choose Empty Project

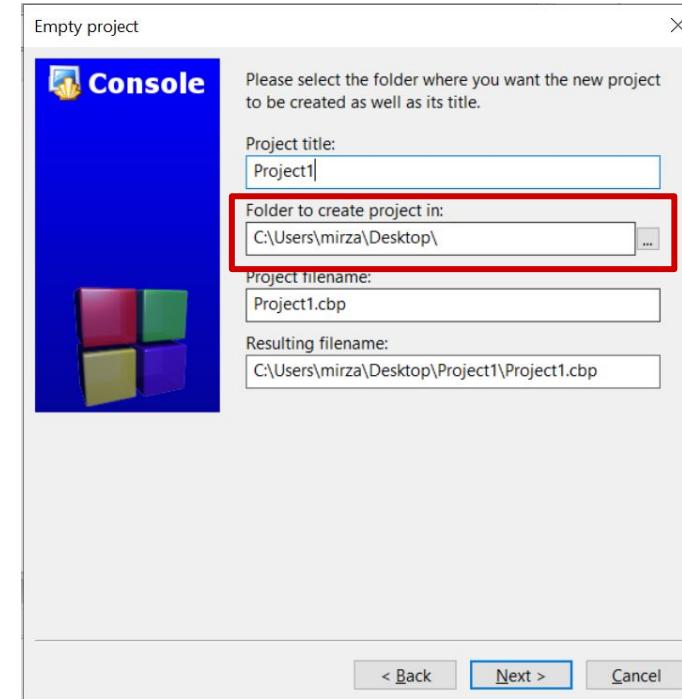


2. Create and run a simple C++ code as a part of a project

Give your project a name

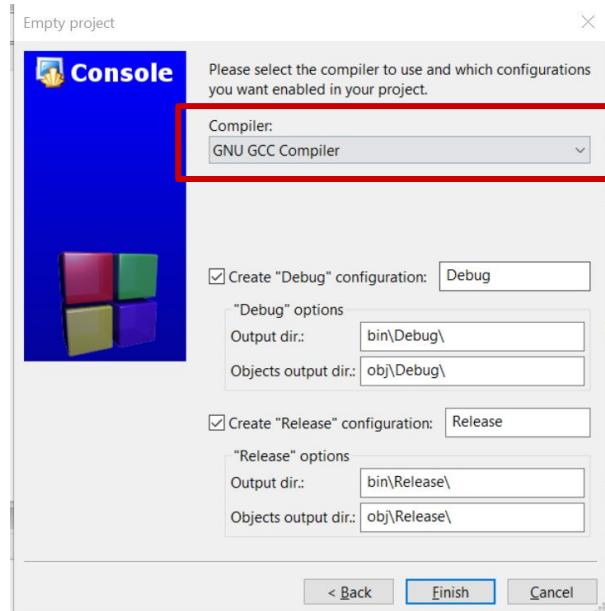


And choose a project folder



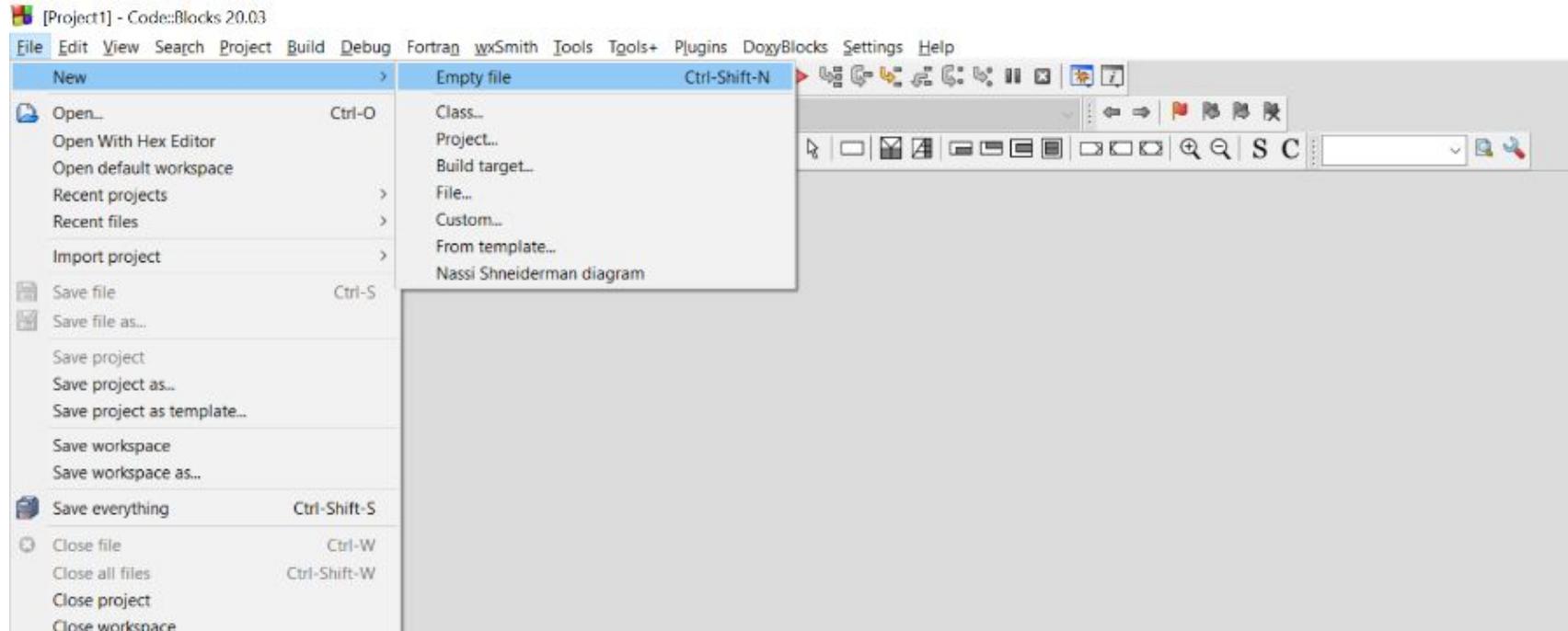
2. Create and run a simple C++ code as a part of a project

VERY IMPORTANT: For this and all your future CodeBlocks projects in CS201 course please make absolutely sure that you choose GNU GCC Compiler. Press Finish.



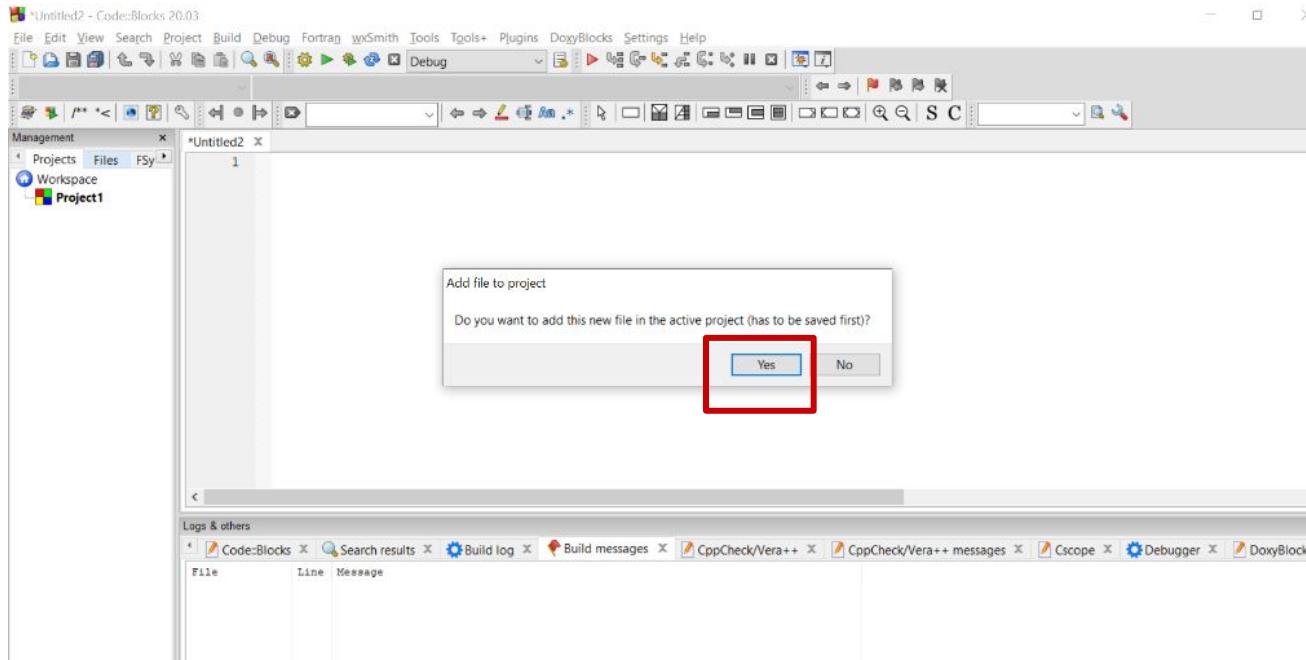
2. Create and run a simple C++ code as a part of a project

Add a C++ file your project: File>New>Empty file



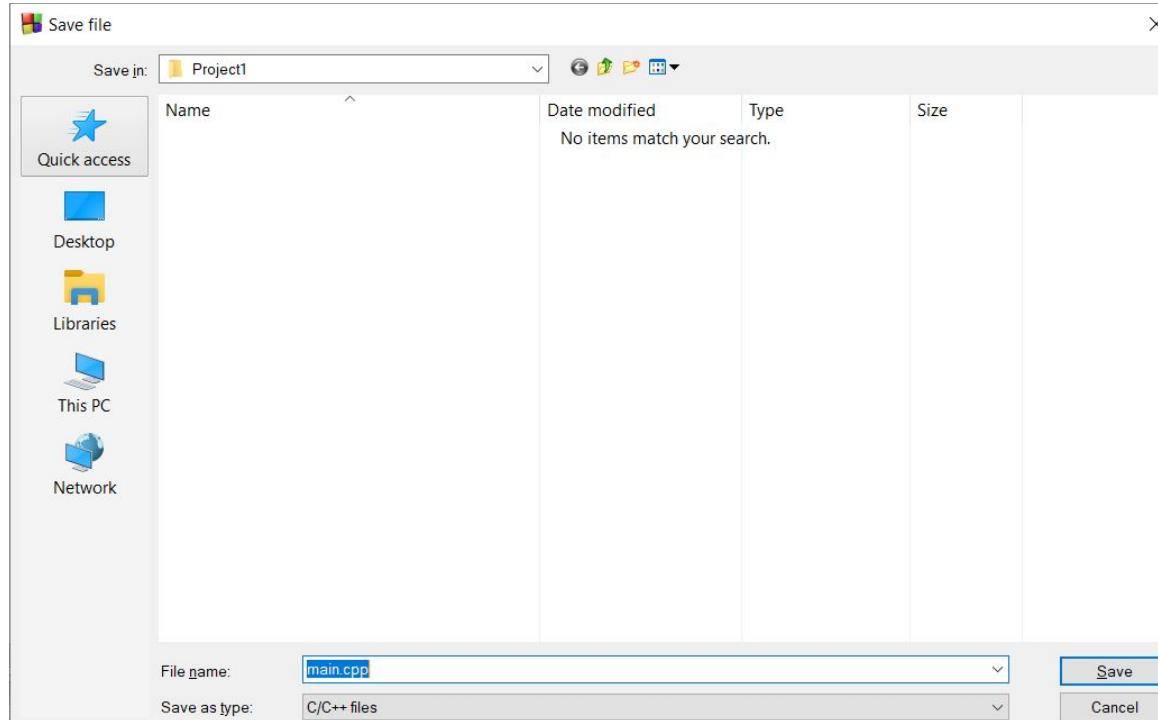
2. Create and run a simple C++ code as a part of a project

Press yes



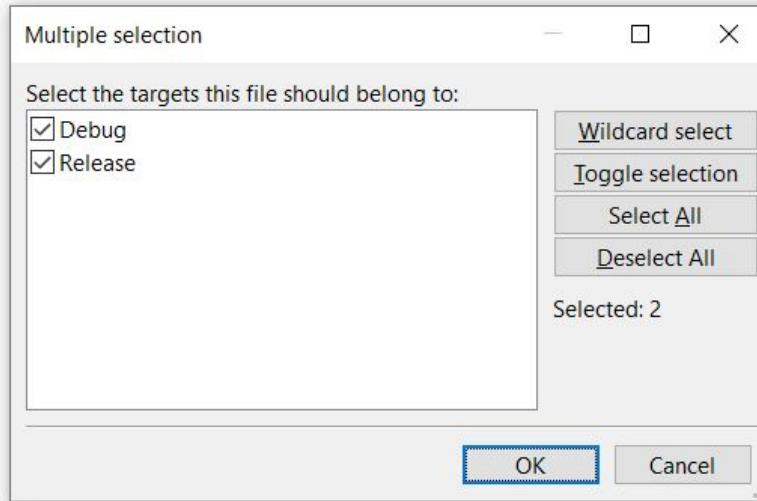
2. Create and run a simple C++ code as a part of a project

Name it 'main.cpp' (.cpp is standard c++ file extension)



2. Create and run a simple C++ code as a part of a project

Make sure debug and release options are checked and press ok



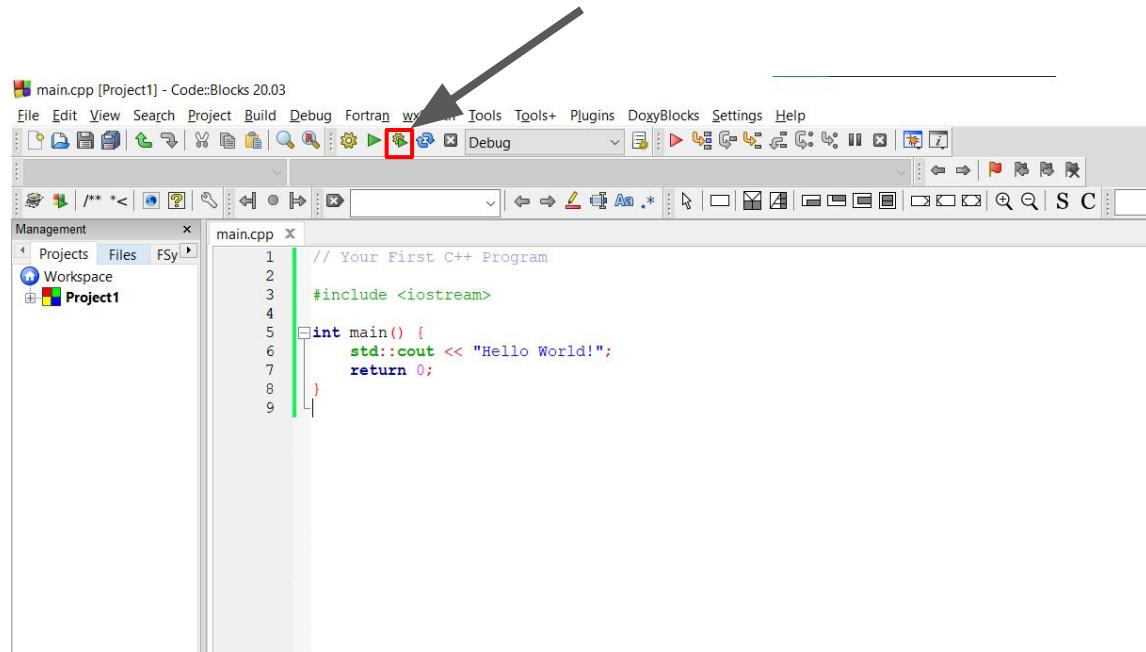
2. Create and run a simple C++ code as a part of a project

Copy and paste this code and then press 'build and run' icon

```
// Your First C++ Program

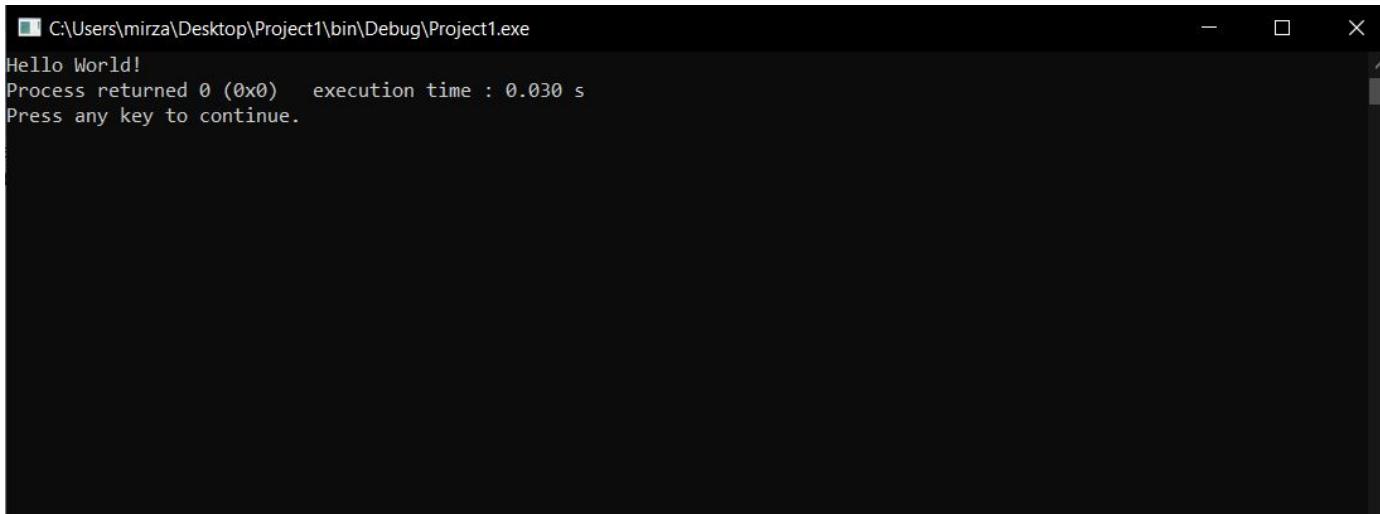
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```



2. Create and run a simple C++ code as a part of a project

If you did everything right, a window will pop up and display something like this.



A screenshot of a terminal window titled "C:\Users\mirza\Desktop\Project1\bin\Debug\Project1.exe". The window contains the following text:
Hello World!
Process returned 0 (0x0) execution time : 0.030 s
Press any key to continue.

Congratulations, you ran your first C++ code!

3. Running your code on a remote server

- Unlike your previous coding classes, assignments in CS201 are evaluated not locally but on a remote server
- This means that during grading we will upload your code to a remote server and run it there, and give you a grade based on the output we get
- Therefore, to make sure that there are no unexpected errors and problems when your code is ran on the server, you too have to connect to the server and run your code there before you submit it
- If after submission we cannot run your code on the server, you are likely to lose a good portion of your grade even if your code works perfectly on your computer.

3. Running your code on a remote server

- Luckily, uploading your code to the server and running it there is very simple
- It is a two-step process 1) [Uploading your code](#) and 2) [Running your code](#). For each of these processes we use a separate software.

Software for uploading the code: [FileZilla](#)

<https://filezilla-project.org/download.php?type=client>

Software for running the code: [Your Command Prompt](#)

Download FileZilla and let's move to the next step. Command prompt is already available on your pc.

3. Running your code on a remote server

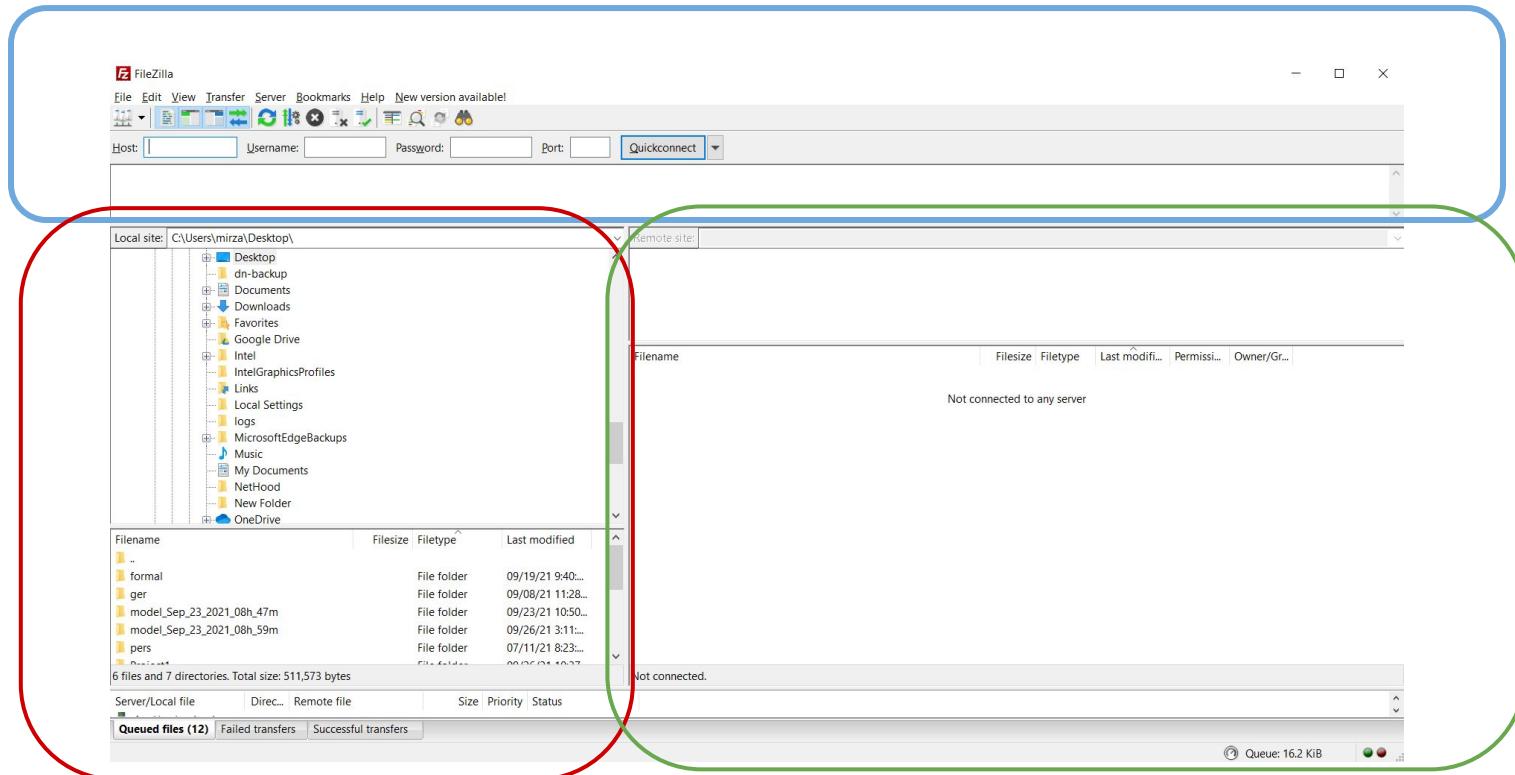
- The server we are going to use is located at Bilkent and is called **Dijkstra**. Think of it as a computer somewhere in one of our buildings which is connected to the bilkent network.
- To connect to this server, you too, must be connected to the Bilkent network. Which means you either need to be using **campus internet** connection, **or** connect to your **Bilkent VPN** if you are not inside the campus.
- Use this link if you don't already have a Bilkent VPN
<http://web3.bilkent.edu.tr/vpn/>
- Again, you don't need VPN if you are connected to Bilkent network.

3. Running your code on a remote server

- Before this recitation or shortly after, each student taking this course will receive a unique server ID and password which they will use to connect to the server.
- With that being said. Lets go and upload our code to the server using FileZilla, our ID and password.

3. Running your code on a remote server

Open FileZilla. It has 3 sections. The one marked with blue is for establishing connection, the red section is the contents of your computer and green section is the contents of the server. Because we are not connected yet, the blue and green sections are empty



3. Running your code on a remote server

Establish a connection. On the upper corner you will see four slots.

Host: dijkstra.ug.bcc.bilkent.edu.tr

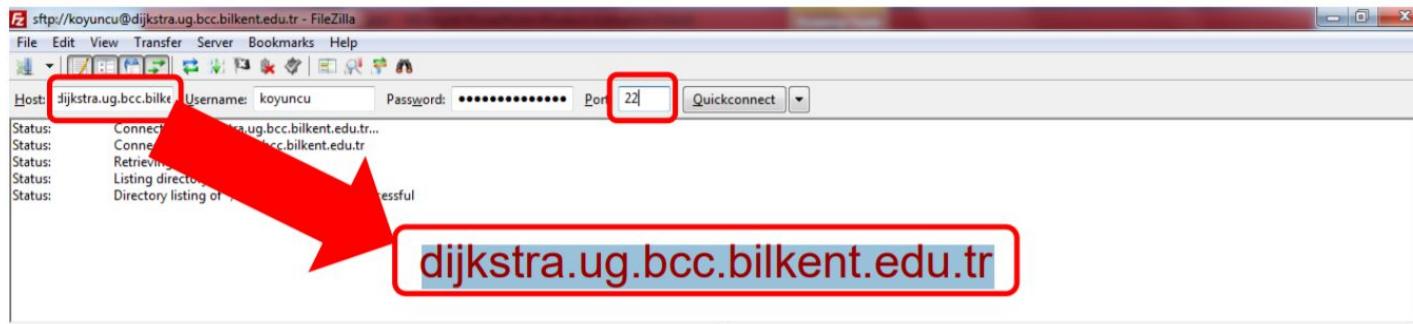
Username: yourusername

Password: yourpassword

Port: 22

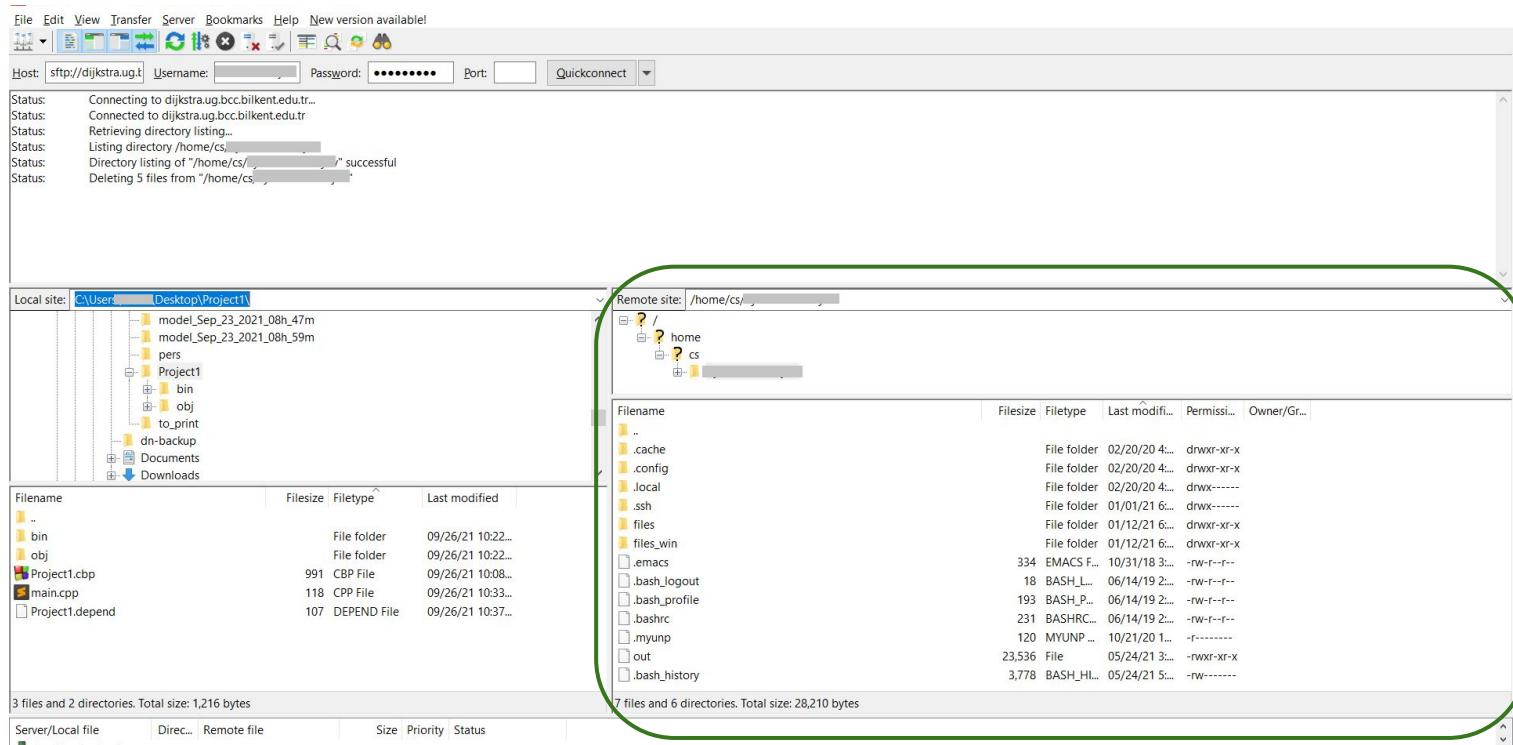
Here host (dijkstra.ug.bcc.bilkent.edu.tr) is the address of our server on the Bilkent network. And username and password are the ID details that each individual student received.

Press **Quickconnect** to connect to the server.



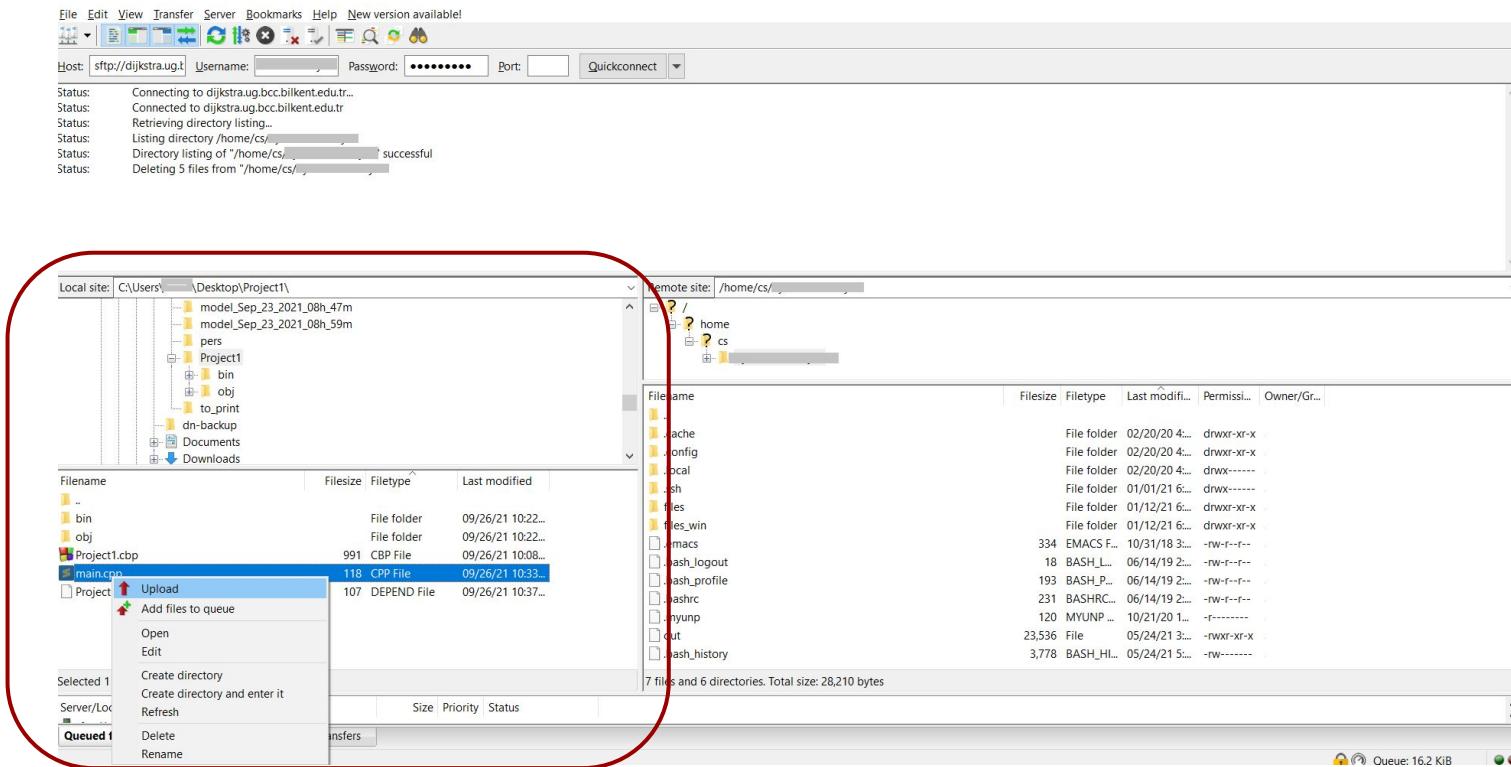
3. Running your code on a remote server

Once connected it will look something like this. On the right you will see the contents of your server folder. For now it only contains some random files that you don't need to worry about. Let's upload our code.



3. Running your code on a remote server

On the left locate the main.cpp file we previously created, right click and press upload. Once you do that the same file will appear in the window on the right. This means you successfully uploaded your code to the server. You are now ready to compile and run it.

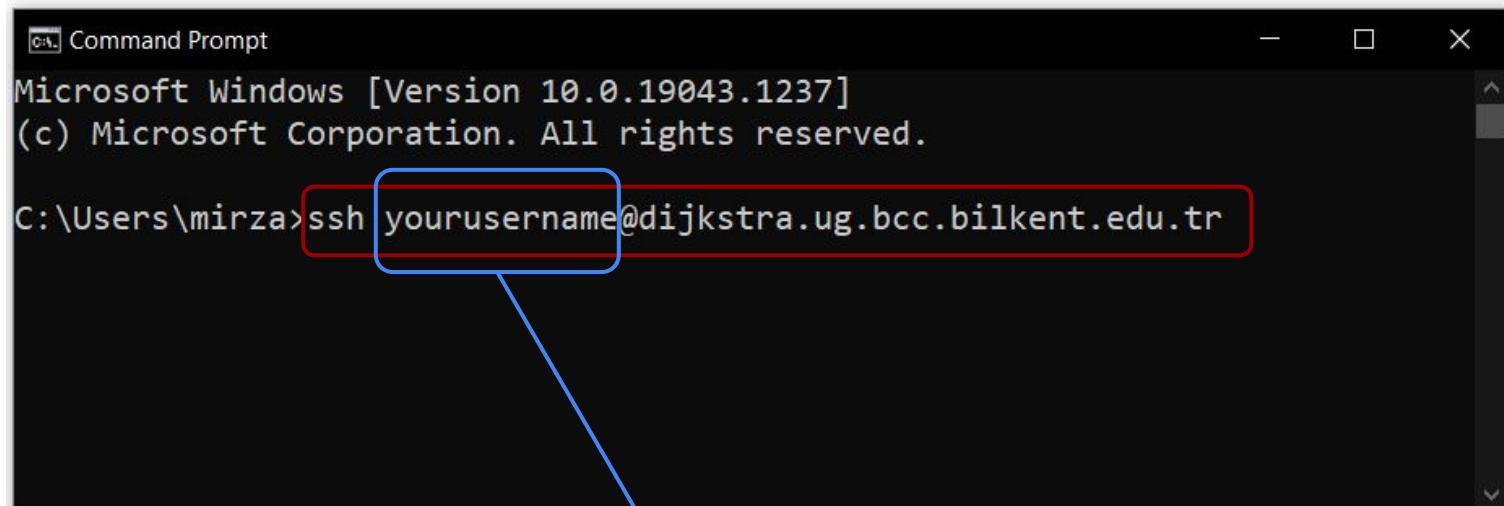


3. Running your code on a remote server

- Next we will run our code on the server
- Some of you may already know that it is possible to run programs from your command prompt just as you do from IDE by pressing run button. Some of you might have already used command prompt to run codes.
- Here we will use our command prompt to connect to the remove server and run codes on the server.

3. Running your code on a remote server

Open your command prompt and type the following command:



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

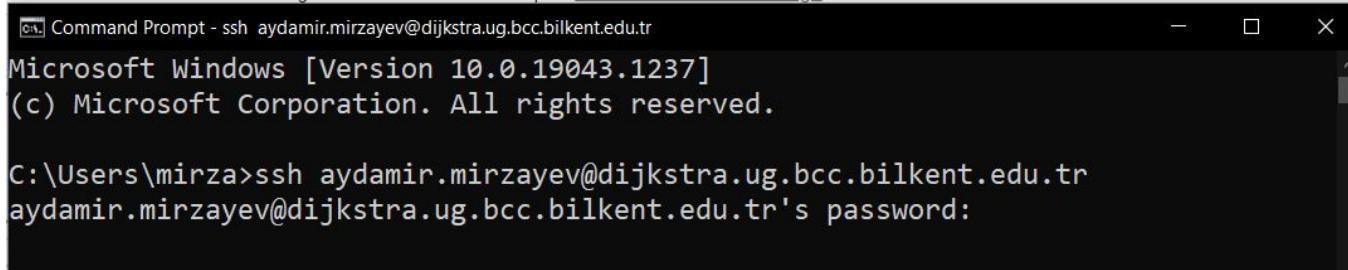
C:\Users\mirza>ssh yourusername@dijkstra.ug.bcc.bilkent.edu.tr

The text "yourusername" in the command line is highlighted with a red rectangular box, and a blue arrow points from below the word "yourusername" to the explanatory text below.

Naturally, you need to replace '[yourusername](#)' part with the username that you have been provided for the server. Type the command and press 'Enter'.

3. Running your code on a remote server

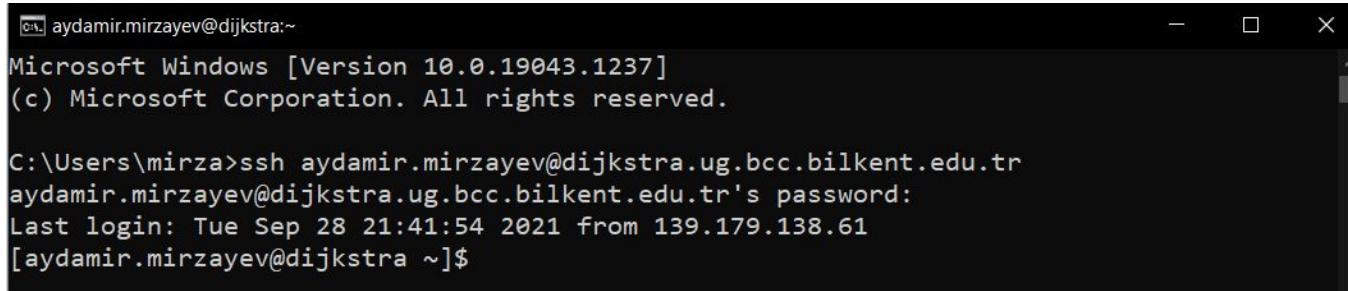
You will be asked to enter a password. As you enter your password you will not see it being typed on the prompt, it is normal, just type the password and press enter.



A screenshot of a Windows Command Prompt window titled "Command Prompt - ssh aydamir.mirzayev@dijkstra.ug.bcc.bilkent.edu.tr". The window shows the following text:
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mirza>ssh aydamir.mirzayev@dijkstra.ug.bcc.bilkent.edu.tr
aydamir.mirzayev@dijkstra.ug.bcc.bilkent.edu.tr's password:

If you entered it correctly then you will see something like below. Now your command prompt is connected to the server.

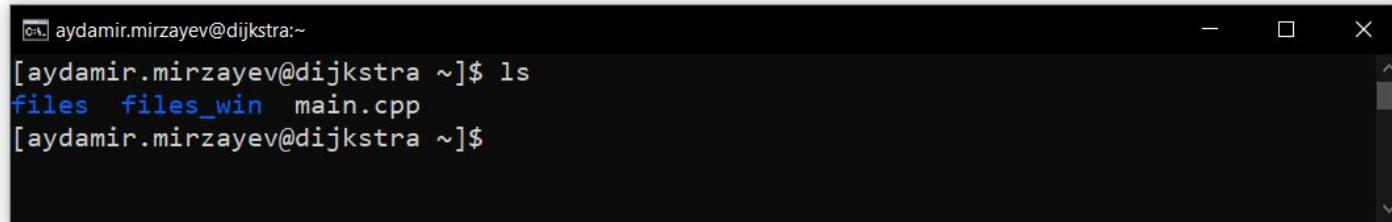


A screenshot of a Windows Command Prompt window titled "aydamir.mirzayev@dijkstra:~". The window shows the following text:
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mirza>ssh aydamir.mirzayev@dijkstra.ug.bcc.bilkent.edu.tr
aydamir.mirzayev@dijkstra.ug.bcc.bilkent.edu.tr's password:
Last login: Tue Sep 28 21:41:54 2021 from 139.179.138.61
[aydamir.mirzayev@dijkstra ~]\$

3. Running your code on a remote server

For example if you now type 'ls' you will see same files listed in the command prompt.



```
aydamir.mirzayev@dijkstra:~  
[aydamir.mirzayev@dijkstra ~]$ ls  
files files_win main.cpp  
[aydamir.mirzayev@dijkstra ~]$
```

A screenshot of a terminal window with a black background and white text. The window title bar says 'aydamir.mirzayev@dijkstra:~'. The command 'ls' is entered at the prompt '[aydamir.mirzayev@dijkstra ~]\$'. The output shows three files: 'files', 'files_win', and 'main.cpp'. The window has standard OS X-style controls (minimize, maximize, close) in the top right corner.

Here you can also see the `main.cpp` file that we just uploaded using FileZilla is visible from prompt.
Let's [compile](#) and [run](#) it.

To compile the file, you will use command '[g++ main.cpp -o my_program](#)'

Here:
 '`g++`' is the name of the compiler
 '`main.cpp`' is the name of the file that we want to compile
 '`-o`' is a command for output assignment
 '`my_program`' is a name that we choose to give to our compiled binary file

This command will generate a compiled binary named '`my_program`' that we will use to run the code.



```
aydamir.mirzayev@dijkstra:~  
[aydamir.mirzayev@dijkstra ~]$ ls  
files files_win main.cpp  
[aydamir.mirzayev@dijkstra ~]$ g++ main.cpp -o my_program
```

A screenshot of a terminal window with a black background and white text. The window title bar says 'aydamir.mirzayev@dijkstra:~'. The command 'ls' is entered at the prompt '[aydamir.mirzayev@dijkstra ~]\$', followed by the compilation command 'g++ main.cpp -o my_program'. The window has standard OS X-style controls in the top right corner.

3. Running your code on a remote server

After you execute the compile command you can actually see `my_program` binary that we just created by listing the folder using 'ls'

```
[aydamir.mirzayev@dijkstra:~]
files files_win main.cpp
[aydamir.mirzayev@dijkstra ~]$ g++ main.cpp -o my_program
[aydamir.mirzayev@dijkstra ~]$ ls
files files_win main.cpp my_program
[aydamir.mirzayev@dijkstra ~]$
```

Finally to run the binary, type: `./my_program`

```
[aydamir.mirzayev@dijkstra:~]
files files_win main.cpp
[aydamir.mirzayev@dijkstra ~]$ g++ main.cpp -o my_program
[aydamir.mirzayev@dijkstra ~]$ ls
files files_win main.cpp my_program
[aydamir.mirzayev@dijkstra ~]$ ./my_program
Hello World![aydamir.mirzayev@dijkstra ~]$
```

Hello World! Is printed. Congrats! You ran your code on the server!

3. Running your code on a remote server

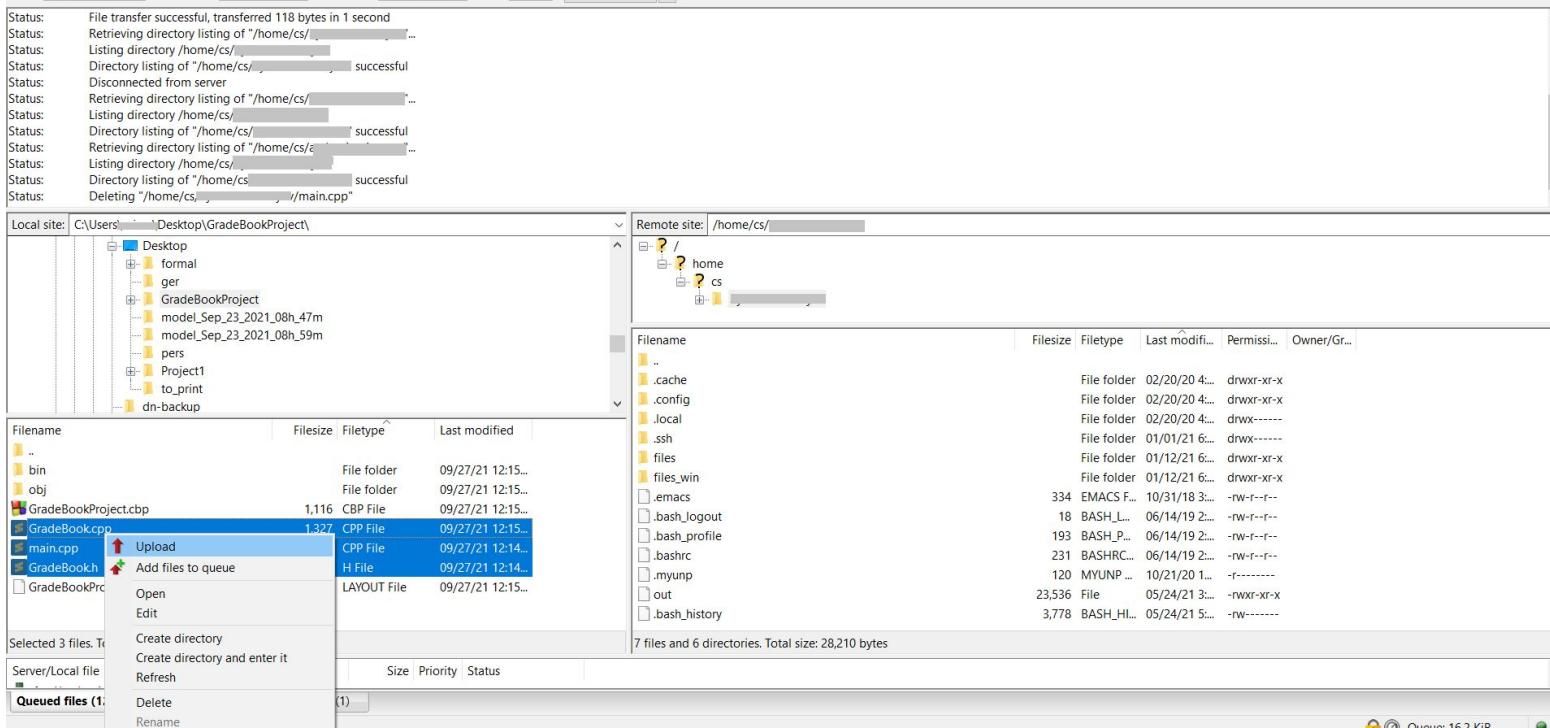
- Okay, now we know how to run a single .cpp file on the server. But how do we run large projects?
- It is not much different, the exact same procedure with a minor difference.
- Project might contain header files. We need to include them. What are they?

4. Running a project with multiple header and cpp files.

- Header files are used to separate declaration and implementation in C++. For the sake of this recitation you just need to know that they are part of the project, they are linked with .cpp files, and they need to be present to be able to compile the project.
- In this recitation we will use a ready project named GradeBook that contains a header file. You don't need to worry about implementation we will provide you with the code.
- The program is very simple, it asks for grades of the student on individual assignments and prints the letter grade that the student is going to receive from the course.
- The program has 3 files. `main.cpp`, `GradeBook.cpp` and `GradeBook.h`

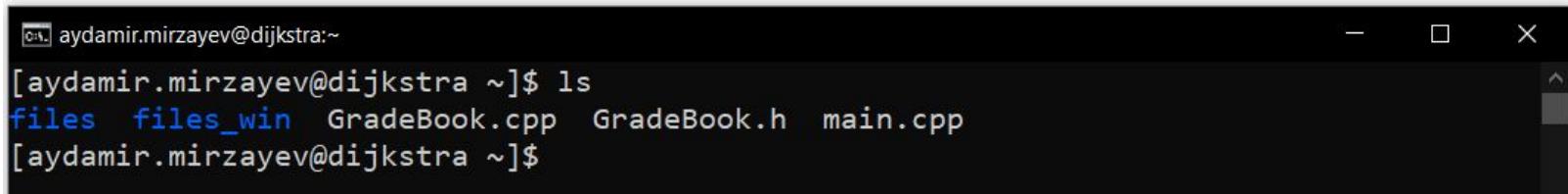
4. Running a project with multiple header and cpp files.

- Locate .h and .cpp files in the project folder and upload them just as we did with single main.cpp file.



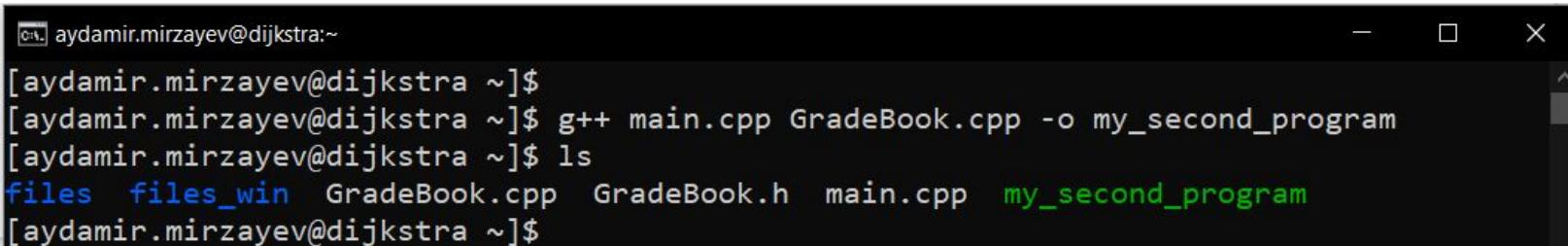
4. Running a project with multiple header and cpp files.

Now, navigate to the command prompt again. Run 'ls' command again, you should be able to see the files that you have uploaded.



```
[aydamir.mirzayev@dijkstra:~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp
[aydamir.mirzayev@dijkstra ~]$
```

Now run: `g++ main.cpp GradeBook.cpp -o my_second_program` to generate the binary of this program. As you might have guessed, when we are compiling a project with multiple cpp files, we simply type the names of all cpp files instead of a single one. You might have also noticed that we don't type the name of .h file. This is because .h files are internally linked with .cpp files and don't need to be included in the command. But they need to be present in the same folder.



```
[aydamir.mirzayev@dijkstra:~]$
[aydamir.mirzayev@dijkstra ~]$ g++ main.cpp GradeBook.cpp -o my_second_program
[aydamir.mirzayev@dijkstra ~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp my_second_program
[aydamir.mirzayev@dijkstra ~]$
```

4. Running a project with multiple header and cpp files.

Now you simply run your new binary.

```
[aydamir.mirzayev@dijkstra:~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp
[aydamir.mirzayev@dijkstra ~]$
[aydamir.mirzayev@dijkstra ~]$ g++ main.cpp GradeBook.cpp -o my_second_program
[aydamir.mirzayev@dijkstra ~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp my_second_program
[aydamir.mirzayev@dijkstra ~]$
[aydamir.mirzayev@dijkstra ~]$ ./my_second_program
Enter the midterm grade: 20
Enter the final grade: 30
Enter the quiz grade: 40
Enter the homework grade: 50
The grades are sdfsd 20, 30, 40, 50
Overall grade: 32
Course created with course sdf sfknsdfsdfsd ameCS 201
Your letter grade : F
[aydamir.mirzayev@dijkstra ~]$
```

4. Running a project with multiple header and cpp files.

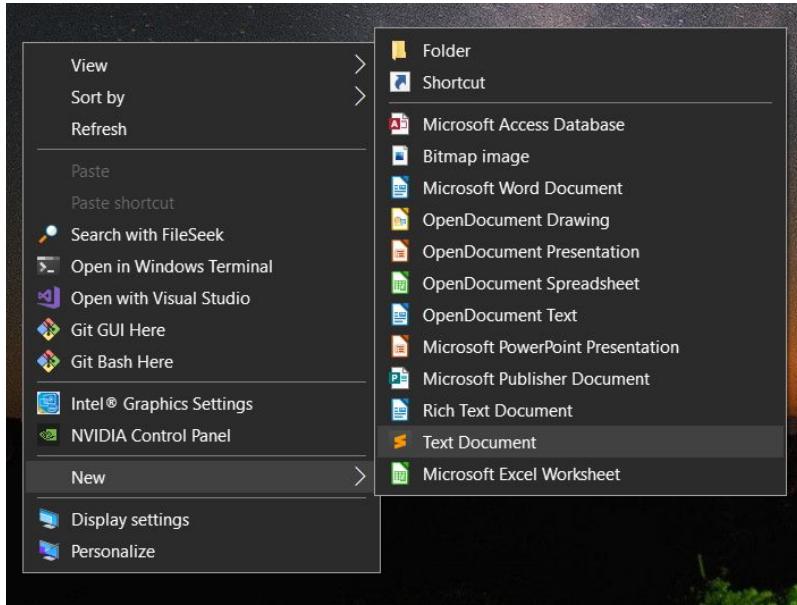
- That's it. If you reached this point. You are ready to test your code on the server before submitting it.
- Always make sure that you test your code on the server before you submit it
- Always make sure that you are **ONLY** using 'g++' compiler.
- Uploading your code to the server is not same as submitting it. Your code on the server is seen by you and only you. You still need to upload your code to the Moodle. And that is what we will use for grading.
- Next I will discuss an extra trick that might make your life a bit easier. It is called Makefile.

5. Briefly about Makefile

- Makefile is often used for making compiling large projects a bit easier.
- You now know that we need to type the names of all of our cpp files when we compile a project on the server.
- You can imagine how on a project with 10-15 cpp files compile command might get messy: `g++ main.cpp car.cpp house.cpp person.cpp ...`
- Makefile comes in handy in similar situations.
- Best way to explain this is to demonstrate and example. So let's do it.

5. Briefly about Makefile

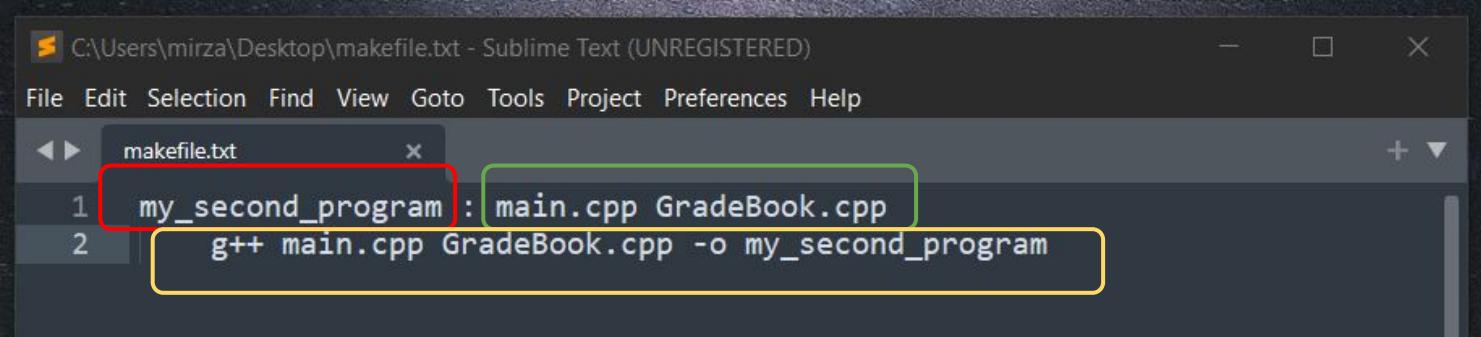
On your computer create a txt file called 'makefile.txt'. Open that txt file and type the following:



C:\Users\mirza\Desktop\makefile.txt - Sublime Text (UNREGISTERED)

```
File Edit Selection Find View Goto Tools Project Preferences Help  
makefile.txt x  
1 my_second_program : main.cpp GradeBook.cpp  
2 g++ main.cpp GradeBook.cpp -o my_second_program
```

5. Briefly about Makefile



The screenshot shows a Sublime Text window with the title "C:\Users\mirza\Desktop\makefile.txt - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu is a tab bar with "makefile.txt". The code editor contains two lines of text:

```
1 my_second_program : main.cpp GradeBook.cpp
2 g++ main.cpp GradeBook.cpp -o my_second_program
```

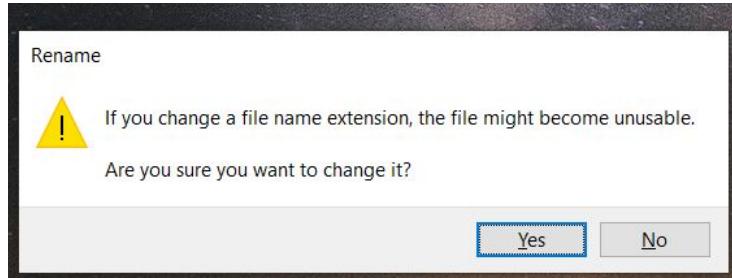
Annotations highlight specific parts of the code:

- A red box surrounds the output name "my_second_program" in the first line.
- A green box surrounds the list of source files "main.cpp GradeBook.cpp" in the first line.
- A yellow box surrounds the command "g++ main.cpp GradeBook.cpp -o my_second_program" in the second line.

- Red section on the left is the name of your output
- Green section on the right contains the names of the files you will use to generate the output
- And the yellow section at the bottom is the really long command that you don't want to type over and over again on the command prompt

5. Briefly about Makefile

Save and close the file. After you closed it rename it such that it no longer has the .txt extension.



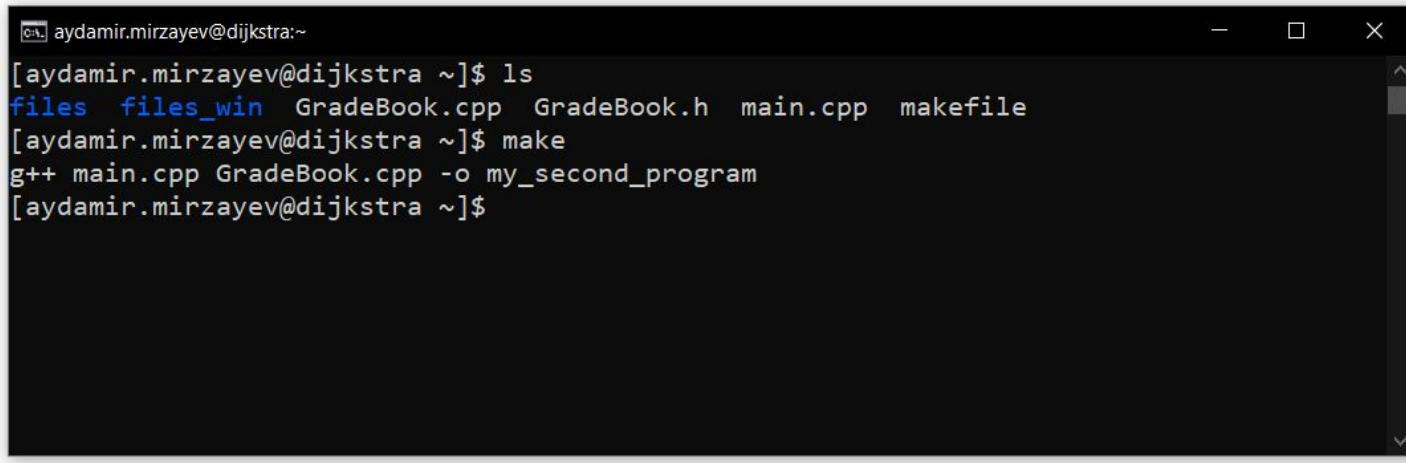
You will get a warning. Just press yes. Now upload this file to the server just as you uploaded your code. After you upload the makefile go to the command prompt and type 'ls' to observe the makefile in the directory.

```
[aydamir.mirzayev@dijkstra:~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp makefile
[aydamir.mirzayev@dijkstra ~]$
```

A screenshot of a terminal window. The command 'ls' is run, showing a list of files: 'files', 'files_win', 'GradeBook.cpp', 'GradeBook.h', 'main.cpp', and 'makefile'. The 'makefile' entry is highlighted with a red rectangle. The terminal window has a dark theme with light-colored text.

5. Briefly about Makefile

Now, to execute the same '`g++ main.cpp Gradebook.cpp -o my_second_output`' command you simply need to type '`make`'.



A screenshot of a terminal window titled 'aydamir.mirzayev@dijkstra:~'. The window contains the following text:

```
[aydamir.mirzayev@dijkstra ~]$ ls
files files_win GradeBook.cpp GradeBook.h main.cpp makefile
[aydamir.mirzayev@dijkstra ~]$ make
g++ main.cpp GradeBook.cpp -o my_second_program
[aydamir.mirzayev@dijkstra ~]$
```

And now your long command has been executed by just typing '`make`'

5. Briefly about Makefile

- In all honesty though, Makefile is a very powerful tool used by professional teams. It does more than just simplify commands. It optimizes the compilation process as well.
- But that is outside the scope of this course. But we do encourage you to read up on the Makefile.
- Please refer to <https://makefiletutorial.com/> for more information on usage of Makefile

6. Debugging

- In debugging mode you can track your code as it is being executed
- Instead of running the entire code at once you can watch step-by-step as each command is being executed
- Let's open the project GradeBook and try each step

6. Debugging

The screenshot shows the Code::Blocks IDE interface with the following details:

- Title Bar:** GradeBook.cpp [Project1] - Code::Blocks 13.12
- Menu Bar:** File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
- Toolbar:** Includes icons for New, Open, Save, Build, Run, Stop, and others.
- Management View:** Shows the Projects, Symbols, and Files tabs. Under Project1, Sources contains GradeBook.cpp.
- Code Editor:** Displays the GradeBook.cpp file with the following content:

```
36         cout << "The grades are " << midtermGrade << ", " << finalGrade << ", " << quizGrade << ", " << hwGrade << endl;
37
38     overallGrade = midtermGrade * .3 + finalGrade * .35 + quizGrade * .2 + hwGrade * .15;
39     cout << "Overall grade : " << overallGrade << endl;
40
41     if (overallGrade > 100 || overallGrade < 0)
42         return 'U';
43     else if (overallGrade >= 90)
44         return 'A';
45     else if (overallGrade >= 80)
46         return 'B';
47     else if (overallGrade >= 70)
48         return 'C';
49     else if (overallGrade >= 60)
50         return 'D';
51     else
52         return 'F';
53
54
55 private:
56     string courseName;
57     double midtermGrade, finalGrade, quizGrade, hwGrade;
58 };
59
60
61 int main()
62 {
63     GradeBook gb ("CS 201");
64     char letterGrade;
65     string courseName;
66
67     courseName = gb.getCourseName();
68     letterGrade = gb.computeFinalGrade();
69
70     cout << "Course created with course name " << courseName << endl;
71     cout << "Your letter grade : " << letterGrade << endl;
72
73     return 0;
74 }
```
- Breakpoint:** A red circle highlights line 67, indicating it is a breakpoint. A callout box labeled "Breakpoint" points to this circle.
- Logs & others:** Shows the status "Debugger finished with status 0".
- Bottom Status Bar:** C:\Users\Tunc\CS201\Project1\GradeBook.cpp, Windows (CR+LF), WINDOWS-1254, Line 47, Column 24, Insert, Read/Write, default.

6. Debugging

The screenshot shows the Code::Blocks IDE interface with the title "GradeBook.cpp [Project1] - Code::Blocks 13.12". The menu bar is visible with options like File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. A context menu is open under the Debug menu, specifically the "Active debuggers" submenu, listing various debugging commands with their keyboard shortcuts.

The main code editor window displays the "GradeBook.cpp" file. A red circular marker indicates a breakpoint is set on line 67. The code includes a private section with variables and a main function that creates a GradeBook object, gets its course name, computes a final grade, and outputs the results. The code editor has syntax highlighting for C++.

The bottom status bar shows the path "Windows (CR+LF) / WINDOWS-1254" and the current position "Line 47, Column 24".

```
GradeBook.cpp [Project1] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Active debuggers
Start / Continue F8
Break debugger Shift-F8
Stop debugger F4
Run to cursor F4
Next line F7
Step into Shift-F7
Step out Ctrl-F7
Next instruction Alt-F7
Step into instruction Alt-Shift-F7
Set next statement.
Toggle breakpoint F5
Remove all breakpoints
Add symbol file
Debugging windows Information
Attach to process...
Detach
Send user command to debugger

55
56     private:
57         string courseName;
58         double midtermGrade, finalGrade, quizGrade, hwGrade;
59     };
59
60
61 int main() {
62     GradeBook gb ("CS 201");
63     char letterGrade;
64     string courseName;
64
65     courseName = gb.getCourseName();
66     letterGrade = gb.computeFinalGrade();
67     cout << "Course created with course name " << courseName << endl;
68     cout << "Your letter grade : " << letterGrade << endl;
69
70     return 0;
71 }
72
73 
```

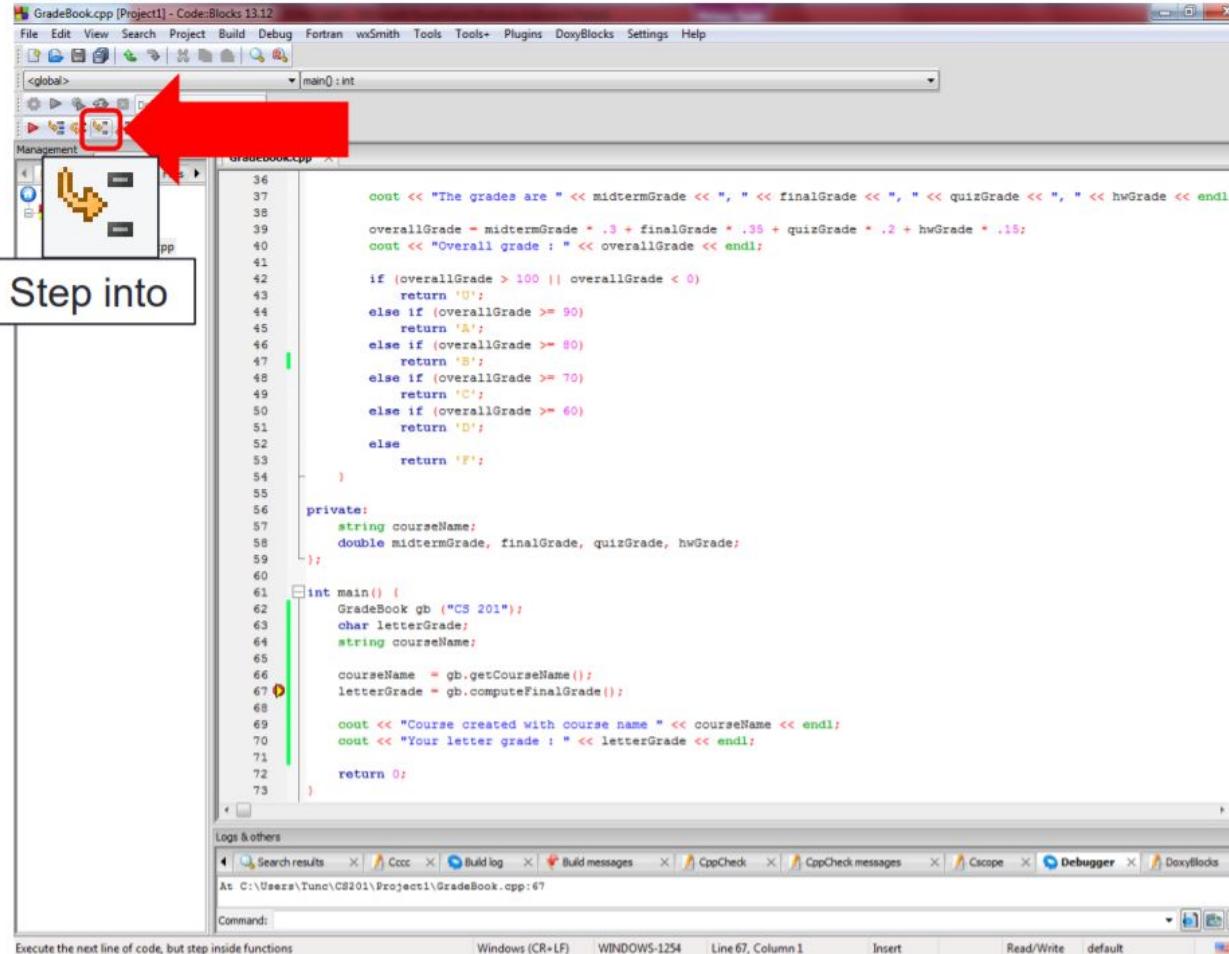
Logs & others

Debugger finished with status 0

Command:

Windows (CR+LF) WINDOWS-1254 Line 47, Column 24 Insert Read/Write default

6. Debugging



The screenshot shows the Code::Blocks IDE interface. A large red arrow points from the text "Step into" to the "Step into" button in the toolbar. The main window displays the code for GradeBook.cpp, which includes functions for calculating overall grades and returning letter grades based on those scores. The code uses standard C++ syntax with if-else statements and string manipulations. The toolbar at the top includes various icons for file operations, search, and build. The status bar at the bottom provides information about the current file and line.

```
GradeBook.cpp [Project1] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
<global> main(): int
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

cout << "The grades are " << midtermGrade << ", " << finalGrade << ", " << quizGrade << ", " << hwGrade << endl;
overallGrade = midtermGrade * .3 + finalGrade * .35 + quizGrade * .2 + hwGrade * .15;
cout << "Overall grade : " << overallGrade << endl;

if (overallGrade > 100 || overallGrade < 0)
    return 'D';
else if (overallGrade >= 90)
    return 'A';
else if (overallGrade >= 80)
    return 'B';
else if (overallGrade >= 70)
    return 'C';
else if (overallGrade >= 60)
    return 'D';
else
    return 'F';

private:
    string courseName;
    double midtermGrade, finalGrade, quizGrade, hwGrade;
};

int main() {
    GradeBook gb ("CS 201");
    char letterGrade;
    string courseName;

    courseName = gb.getCourseName();
    letterGrade = gb.computeFinalGrade();

    cout << "Course created with course name " << courseName << endl;
    cout << "Your letter grade : " << letterGrade << endl;

    return 0;
}

Logs & others
At C:\Users\Tunc\CS201\Project1\GradeBook.cpp:67
Command:
```

Execute the next line of code, but step inside functions

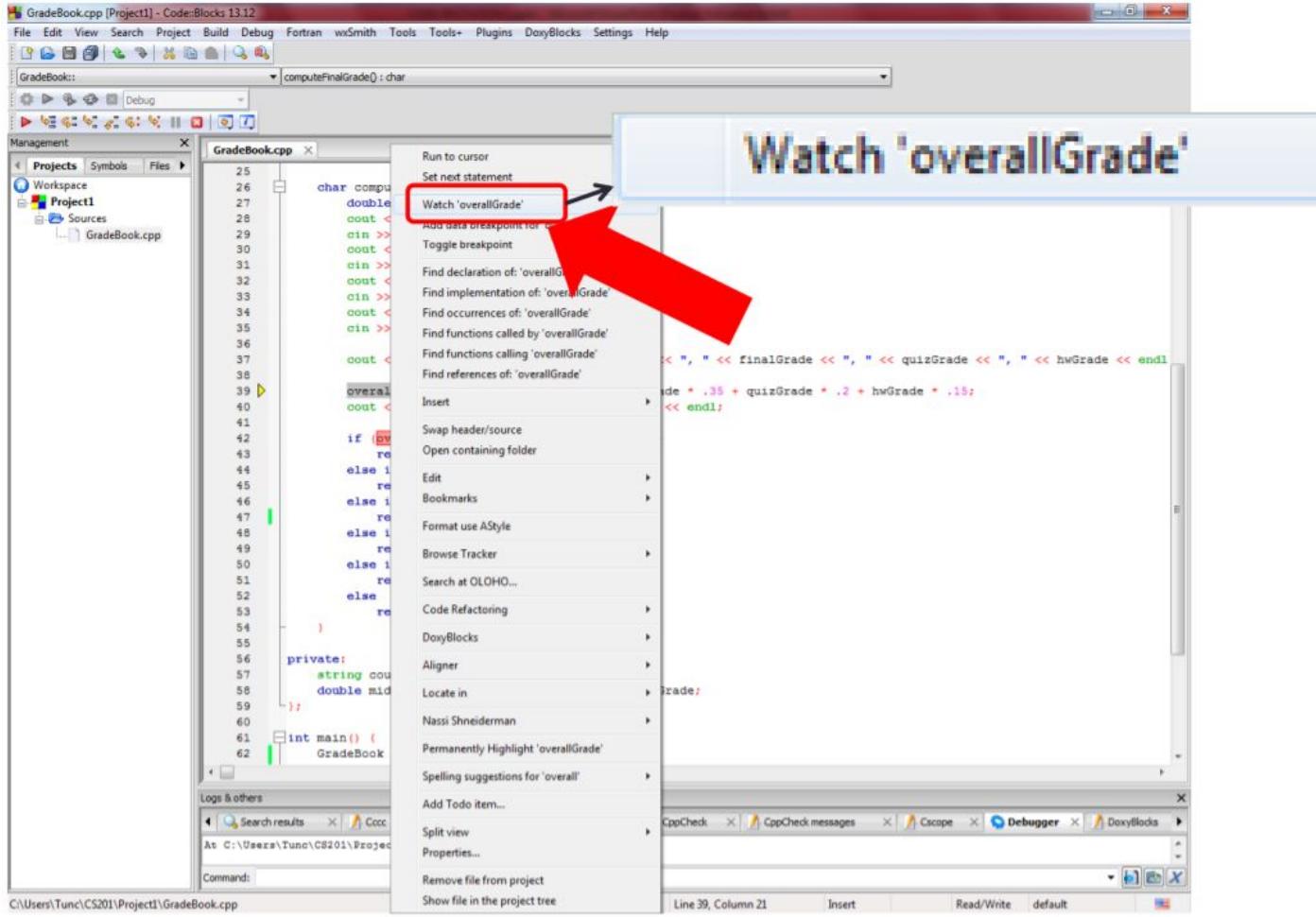
Windows (CR+LF) WINDOWS-1254 Line 67, Column 1 Insert Read/Write default

6. Debugging

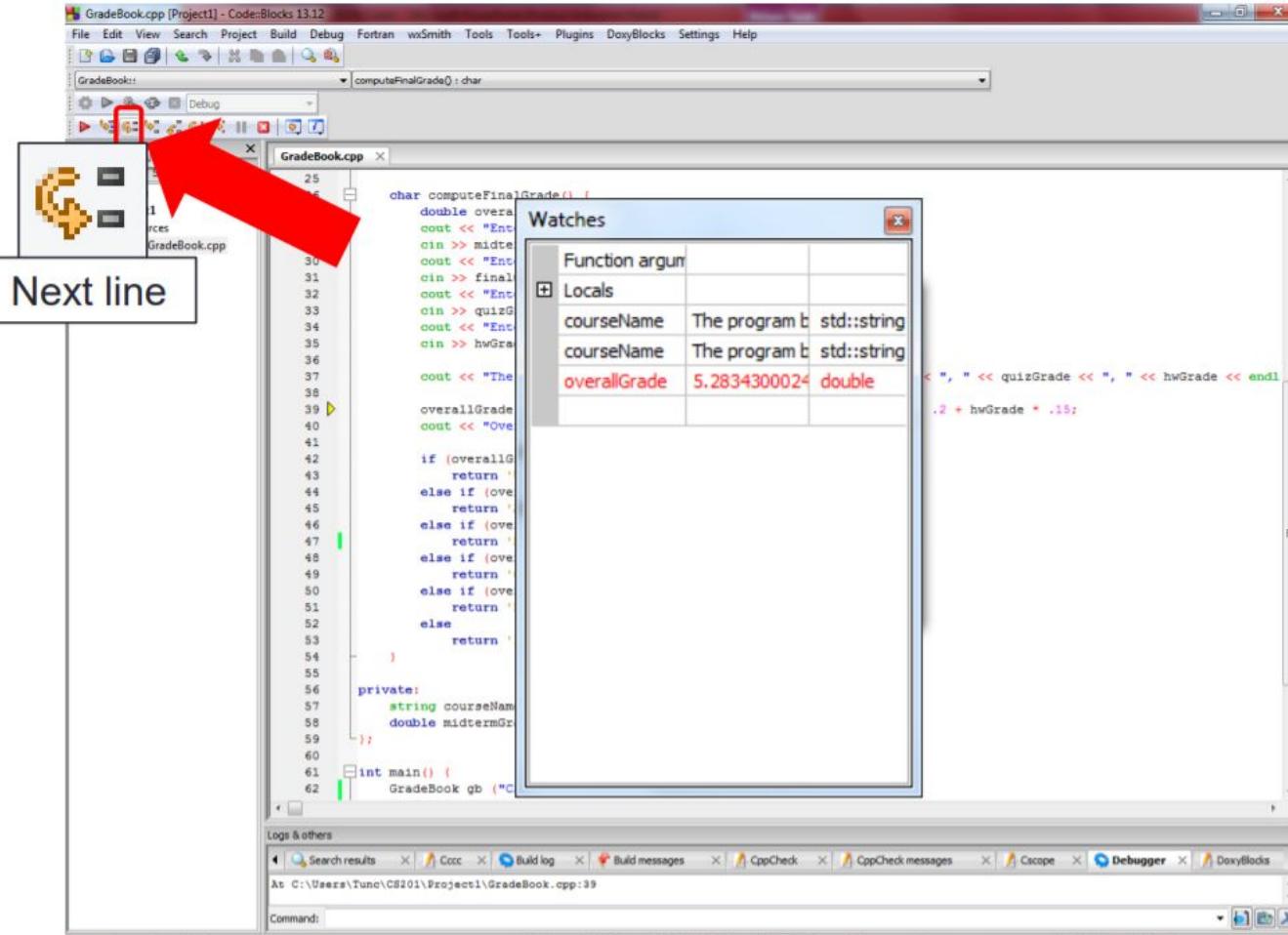
The screenshot shows the Code-Blocks IDE interface with the following details:

- Title Bar:** GradeBook.cpp [Project1] - Code-Blocks 13.12
- Menu Bar:** File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, Help
- Toolbar:** Includes icons for Open, Save, Build, Run, Stop, and others.
- Management View:** Shows the Project tree under Project1, with Sources and GradeBook.cpp selected.
- GradeBook.cpp Editor:** Displays the C++ code for the GradeBook class. The code includes methods for computing final grades based on midterm, final, quiz, and homework grades, and a main function that creates a GradeBook object for course CS 201.
- Logs & Others:** A tab bar at the bottom showing various logs: Search results, Ccc, Build log, Build messages, CppCheck, CppCheck messages, Cscope, Debugger, and DoxyBlocks.
- Status Bar:** At C:\Users\Tunc\CS201\Project1\GradeBook.cpp:28, Line 50, Column 36, Insert, Read/Write, default.

6. Debugging



6. Debugging



6. Debugging

The screenshot shows the Code::Blocks IDE interface during debugging. The main window displays the `GradeBook.cpp` file with the following code:

```
25     char computeFinalGrade() {
26         double overa
27         cout << "Ent
28         cin >> midte
29         cout << "Ent
30         cin >> final
31         cout << "Ent
32         cin >> quizG
33         cout << "Ent
34         cin >> hwGra
35
36         cout << "The
37
38         overallGrade
39         cout << "Ove
40
41         if (overallG
42             return '
43         else if (ove
44             return '
45         else if (ove
46             return '
47         else if (ove
48             return '
49         else if (ove
50             return '
51         else if (ove
52             return '
53         else
54             return '
55
56     private:
57         string courseNam
58         double midtermGr
59     };
60
61     int main() {
62         GradeBook gb ("C
```

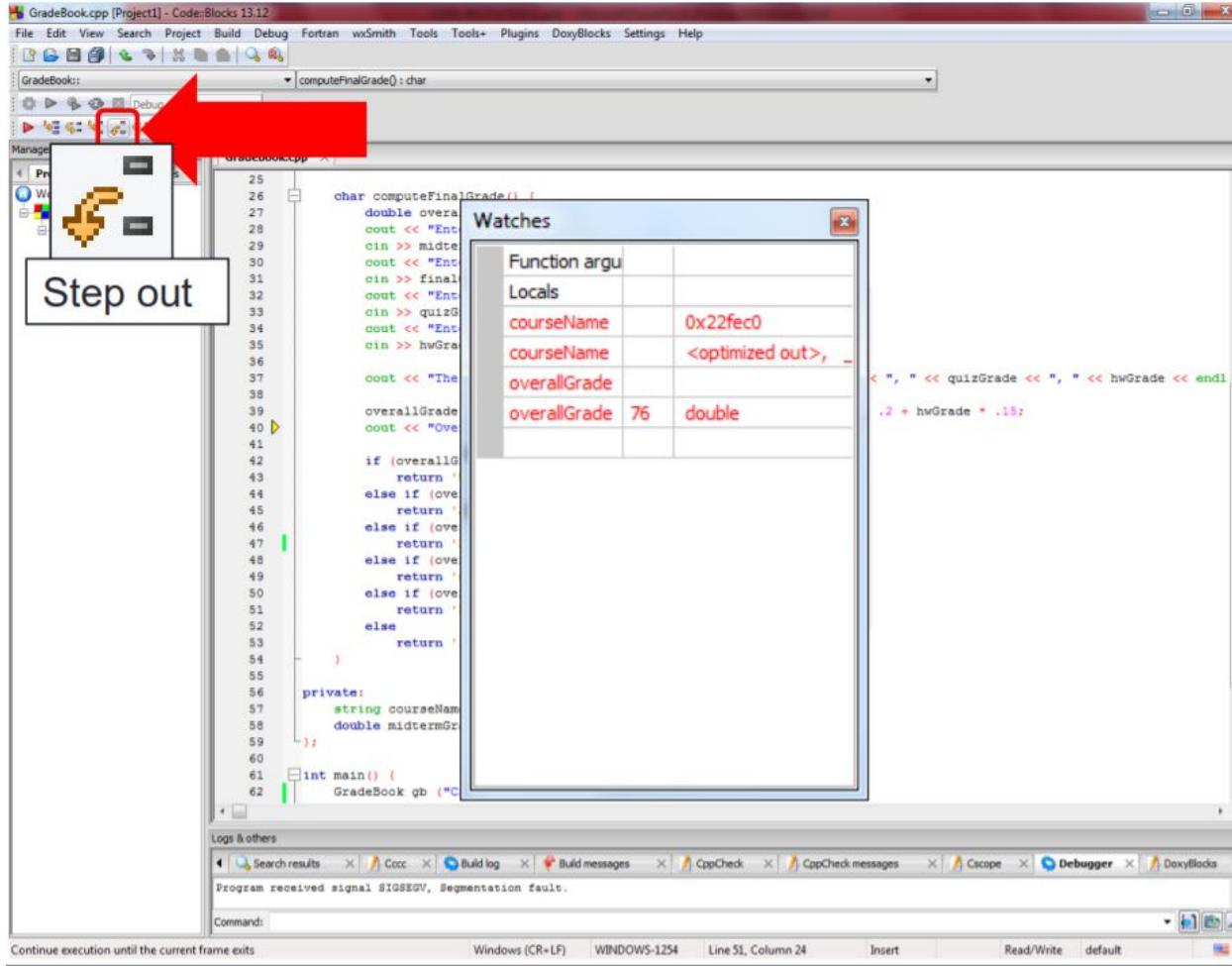
The `Watches` window is open, showing the state of variables:

Function argu	
Locals	
courseName	0x22fec0
courseName	<optimized out>
overallGrade	76
overallGrade	double

A large red arrow points to the `overallGrade` entry in the `Locals` section of the `Watches` window.

The status bar at the bottom shows the error message: `Program received signal SIGSEGV, Segmentation fault.`

6. Debugging



6. Debugging

GradeBook.cpp [Project1] - Code::Blocks 13.12

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

<global> main() : int

Management Projects Sources GradeBook.cpp

36 cout << "The grades are " << midtermGrade << ", " << finalGrade << ", " << quizGrade << ", " << hwGrade << endl;

37 overallGrade = .2 + hwGrade * .15;

38 cout << "Over

39 if (overallGrade >= 90) return 'A';

40 else if (overallGrade >= 80) return 'B';

41 else if (overallGrade >= 70) return 'C';

42 else if (overallGrade >= 60) return 'D';

43 else return 'F';

44

45 private:

46 string className;

47 double midtermGrade;

48

49 string courseName;

50

51 double hwGrade;

52

53

54 }

55

56

57

58

59 ;

60

61 int main() {

62 GradeBook gb ("CS101");

63 char letterGrade;

64 string courseName;

65

66 courseName = gb.courseName();

67 letterGrade = gb.letterGrade();

68

69 cout << "Course name: " << courseName;

70 cout << "Your letter grade: " << letterGrade;

71

72 return 0;

73

Watches

Function argu

Locals

courseName	The p std::string
courseName	The p std::string
overallGrade	No symbol
overallGrade	Not a

Logs & others

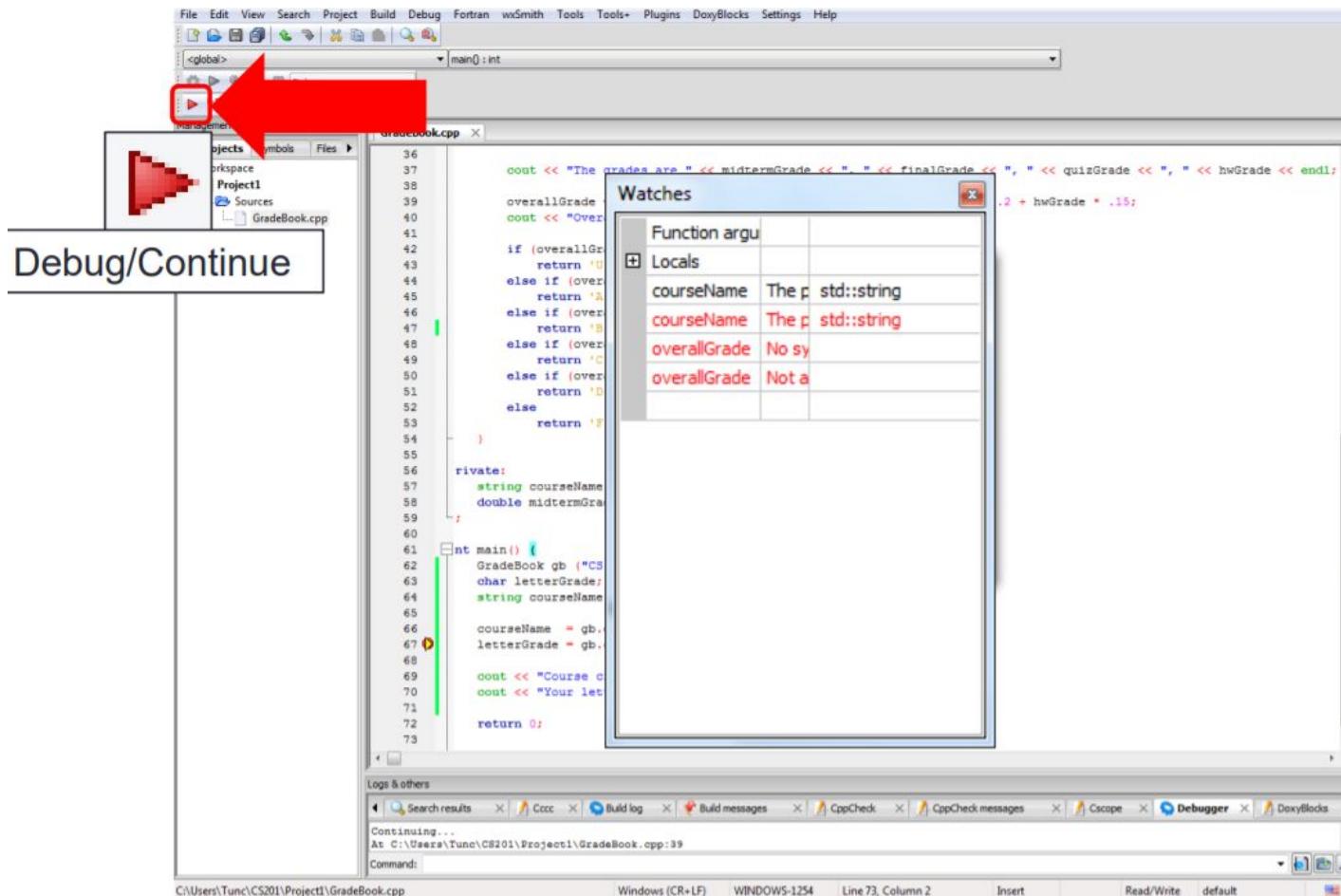
Continuing... At C:\Users\tunc\CS201\Project1\GradeBook.cpp:39

Command: C:\Users\tunc\CS201\Project1\GradeBook.cpp

Windows (CR+LF) WINDOWS-1254 Line 73, Column 2 Insert Read/Write default

A screenshot of the Code::Blocks IDE interface. The main window displays the 'GradeBook.cpp' file with several lines of C++ code. A tooltip window titled 'Watches' is overlaid on the code, showing the current values of variables: 'courseName' is listed twice as 'The p std::string', and 'overallGrade' is listed twice as 'No symbol'. In the code editor, a red circle highlights a breakpoint at line 71. The bottom status bar shows the command 'C:\Users\tunc\CS201\Project1\GradeBook.cpp'.

6. Debugging



6. Debugging

- Step Into :

- Runs the program until the next instruction is reached.

- Next Line :

- Runs the program until the next line of code is reached.

- Step Out :

- Runs the program until the current procedure is completed.

Step Out ≥ Next Line ≥ Step Into

6. Debugging

At Break Point

Output :
Instruction A1
-

```
1 #include <iostream>
2 using namespace std;
3 void procedureB () {
4     cout << "Instruction B1" << endl;
5     cout << "Instruction B2" << endl;
6 }
7 void procedureA () {
8     cout << "Instruction A1" << endl;
9     procedureB ();
10    cout << "Instruction A2" << endl;
11    cout << "Instruction A3" << endl;
12 }
13 int main () {
14     cout << "Output : " << endl;
15     procedureA ();
16     cout << "End of Procedure A" << endl;
17 }
```

Step Into



Output :
Instruction A1

```
3 void procedureB () {
4     cout << "Instruction B1" << endl;
5     cout << "Instruction B2" << endl;
6 }
```

Next Line



Output :
Instruction A1
Instruction B1
Instruction B2

```
7 void procedureA () {
8     cout << "Instruction A1" << endl;
9     procedureB ();
10    cout << "Instruction A2" << endl;
11    cout << "Instruction A3" << endl;
12 }
```

Step Out



Output :
Instruction A1
Instruction B1
Instruction B2
Instruction A2
Instruction A3

```
13 int main () {
14     cout << "Output : " << endl;
15     procedureA ();
16     cout << "End of Procedure A" << endl;
17 }
```