

C++ Notes

Part 1-2

**Slides were created from
CodewithMosh.com**

Formatting Output

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    cout << left;
    cout << setw( n: 10 ) << "Spring" << setw( n: 10 ) << "Nice"
        << setw( n: 10 ) << "Summer" << setw( n: 10 ) << "Hot";

    return 0;
}
```

left : Left alignment

right : Right alignment

setw : String formatting

Formatting Output

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    cout << fixed << setprecision( n: 10 ) << 12.34567;
}

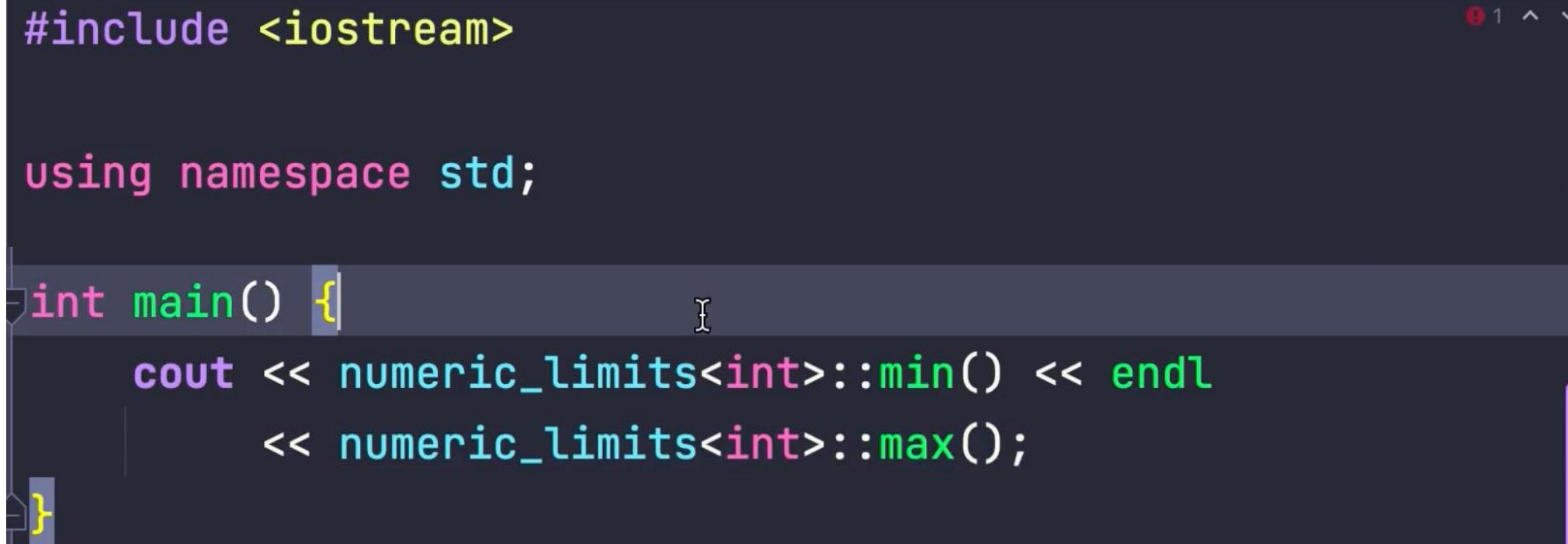
return 0;
}
```

fixed : To force fixed point notation.

- 6 digits after the decimal point.
- Stays in effect until changed.

setprecision : To use certain number of digits after the decimal point.

Data Types Size and Limits



```
#include <iostream>
using namespace std;

int main() {
    cout << numeric_limits<int>::min() << endl
        << numeric_limits<int>::max();
}
```

A screenshot of a code editor window showing a C++ program. The code includes an include directive for `<iostream>`, a `using namespace std;` statement, and a `main` function. Inside `main`, the `cout` object is used to output the minimum and maximum values for the `int` type by calling `numeric_limits<int>::min()` and `numeric_limits<int>::max()`. The code editor has a dark theme with syntax highlighting for different language elements.

numeric_limits : Generic class

Data Types: Overflowing

```
#include <iostream>

using namespace std;

int main() {
    int number = numeric_limits<int>::max();
    number++;
    cout << number;
}
```

numeric_limits : Generic class

Data Types: Underflowing

```
#include <iostream>

using namespace std;

int main() {
    int number = numeric_limits<int>::min();
    number--;
    cout << number;
}
```

numeric_limits : Generic class

Booleans

```
#include <iostream>

using namespace std;

int main() {
    bool isNewUser = false;
    cout << boolalpha << isNewUser;
    return 0;
}
```

boolalpha: Display the value of a boolean variable as true/false

noboolalpha: Display the value of a boolean variable as 1/0 (Default)

Characters

```
#include <iostream>

using namespace std;

int main() {
    char ch = 'a';
    cout << ch;
    return 0;
}
```

```
int main() {
    char ch = 'a';
    cout << +ch;
    return 0;
}
```

```
int main() {
    char ch = 98;
    cout << ch;
    return 0;
}
```

Even this is allowed, it is not a good practice.

- + before the variable name tells the compiler to treat it as a number.

Strings

```
using namespace std;

int main() {
    string name;
    cout << "Enter your name: ";
    cin >> name;
    getline( &cin, &name );
    cout << "Hi " << name;
    return 0;
}
```

- Reading a string with space using “cin”function is not possible.
- Instead we need to use “getline” function.

```
#include <iostream>

using namespace std;

int main() {
    int numbers[5];
    numbers[1] = 10;
    cout << numbers[5]; // size - 1

    return 0;
}
```

```
int numbers[5] = { [0]: 10, [1]: 20};
```

Arrays

- Array index starts from 0 (zero)
- Referencing an array index outside of the range can cause problem.
 - Program is still compiled but will produce a garbage value.

Type Conversion (Casting)

```
int main() {  
    int x = 1;  
    double y = 2.0;  
    int z = x + (int)y;  
    cout << z;  
    return 0;  
}
```

```
int main() {  
    int x = 1;  
    double y = 2.0;  
    int z = x + static_cast<int>(y);  
    cout << z;  
    return 0;  
}
```

- C-style casting: Casting problem cannot be caught until the run time.

- C++-style casting: Casting problem can be caught during the compile time.

Type Conversion (Casting)

```
#include <iostream>

using namespace std;

int main() {
    int x = 10;
    int y = 3;
    double z = static_cast<double>(x) / y;
    cout << z;
    return 0;
}
```

Example

Decision Making

Comparison operators

Logical operators

If statements

Switch statements

Conditional operator

Comparison Operators

```
int main() {  
    int x = 10;  
    int y = 5;  
    bool result = x == y;  
    cout << boolalpha << result;  
    return 0;  
}
```

==

!=

>, >=

<, <=

```
int main() {  
    int x = 10;  
    double y = 5;  
    bool result = x == y;  
    cout << boolalpha << result;  
    return 0;  
}
```

- The compiler automatically casts smaller value (less precise) to larger (more precise) types.
- Int 10 value is casted to double type.

Comparison Operators

```
int main() {  
    char first = 'a';  
    char second = 'A';  
    bool result = first == second;  
    cout << result;  
    return 0;  
}
```

`==`

`!=`

`>, >=`

`<, <=`

Logical Operators

```
int main() {  
    int age = 20;  
    bool isEligible = age > 18 && age < 65;  
    cout << boolalpha << isEligible;  
    return 0;  
}
```

&& : AND
|| : OR
! : NOT

- Evaluation is from left-to-right.

```
// ()  
// !  
// &&  
// ||  
  
bool a = true;  
bool b = false;  
bool c = false;  
bool result = a || b && c;  
  
cout << boolalpha << result;  
  
return 0;  
}
```

Logical Operators

&& : AND
|| : OR
! : NOT

Example

- ❑ Order is (), ! , &&, ||

IF Statement

```
int main() {  
    int temperature = 70;  
    if (temperature < 60)  
        cout << "Cold";  
    else if (temperature < 90)  
        cout << "Nice";  
  
    return 0;  
}
```

- **If (condition){**
 statement;
 }
- **else if (condition) {**
 statements;
 }
- **else {**
 statement;
 }

- Controls the logic of the program.

Nested if Statements

```
bool caResident = true;
short tuition = 0;
// Outer if statement
if (isCitizen) {
    // Inner if statement
    if (!caResident) tuition = 1000;
}
else tuition = 3000;

return 0;
}
```

The Conditional Operator

- ❑ Use conditional variable instead.
- ❑ If the condition is satisfied, then the value after ? is assigned, otherwise, the value after : is assigned to the conditional variable.

```
int main() {  
    int sales = 11'000;  
    double commission;  
    if (sales > 10'000)  
        commission = .1;  
    else  
        commission = .05;  
  
    return 0;  
}
```

```
int main() {  
    int sales = 11'000;  
    double commission = (sales > 10'000) ? .1 : .05;  
    cout << commission;  
  
    return 0;  
}
```

The Switch Statement

Syntax

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

The Switch Statement

```
short input;  
cin >> input;  
  
if (input == 1)  
    cout << "You selected 1";  
else if (input == 2)  
    cout << "You selected 2";  
else  
    cout << "Goodbye!";
```

```
switch (input) {  
    case 1:    {  
        cout << "You selected 1";  
        break;  
    }  
    case 2:  
        cout << "You selected 2";  
        break;  
    default:  
        cout << "Goodbye!";  
}
```

The Switch Statement

```
int day = 4;
switch (day) {
    case 1:
        cout << "Monday";
        break;
    case 2:
        cout << "Tuesday";
        break;
    case 3:
        cout << "Wednesday";
        break;
    case 4:
        cout << "Thursday";
        break;
    case 5:
        cout << "Friday";
        break;
    case 6:
        cout << "Saturday";
        break;
    case 7:
        cout << "Sunday";
        break;
}
// Outputs "Thursday" (day 4)
```

Example

From:<https://www.w3schools.com/cpp/>

The Switch Statement

Example

```
using namespace std;

int main() {
    cout << "Enter two numbers: ";
    int first;
    int second;
    cin >> first >> second;

    cout << "Enter an operator: ";
    char op;
    cin >> op;

    switch (op) {
        case '+':
            cout << first + second;
            break;
        case '-':
            cout << first - second;
            break;
        default:
            cout << "Invalid operator!";
    }
}
```

Loops in C++

For loops

Range-based for loops

While loops

Do-while loops

Break and continue statements

The for Loop

```
#include <iostream>

using namespace std;

int main() {
    for (int i = 0; i < 5; i++) |
        cout << i << endl;

    return 0;
}
```

Syntax

```
for (statement 1; statement 2; statement 3) {
    // code block to be executed
}
```

The for Loop

```
int main() {  
    for (int i = 5; i > 0; i--) {  
        if (i % 2 != 0)  
            cout << i << endl;  
    }  
  
    return 0;  
}
```

Syntax

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Example

- Odd numbers between 5 and 1
- %: Modulo operator. It calculates the remainder of a division between two integers.

Compound Assignment Statement

C++

```
variable operator= expression;
```

This is equivalent to:

C++

```
variable = variable operator expression;
```

Example

x *=y

is equivalent to

x =x * y;

Range-based for Loops

```
int main() {
    int numbers[] = { [0]: 1, [1]: 2, [2]: 3, [3]: 4 };
    // sizeof(numbers): 16
    // sizeof(int): 4
    for (int i = 0; i < sizeof(numbers) / sizeof(int); i++)
        cout << numbers[i] << endl;

    return 0;
}
```

Range-based for Loops

```
int main() {  
    int numbers[] = { [0]: 1, [1]: 2, [2]: 3 };  
    // sizeof(numbers): 16  
    // sizeof(int): 4  
    for (int i = 0; i < sizeof(numbers) / sizeof(int); i++)  
        cout << numbers[i] << endl;  
  
    for (int number: numbers)  
        cout << number << endl;  
  
    return 0;
```

Range-based for Loops

```
int main() {
    int temperatures[] = { [0]: 60, [1]: 80, [2]: 90 };
    double sum = 0;
    for (int temperature: temperatures)
        sum += temperature;
    short count = sizeof(temperatures) / sizeof(int);
    double average = sum / count;
    cout << average;

    return 0;
}
```

Example

Average of temperatures

While Loop

```
int main() {  
    for (int i = 1; i <= 5; i++)  
        cout << i << endl;  
  
    int i = 1;  
    while (i <= 5) {  
        cout << i << endl;  
        i++;  
    }  
}
```

Syntax

```
while (condition) {  
    // code block to be executed  
}
```

- We cannot declare a variable in the while loop unlike the for loop.
- Useful when we do not know how many times we need to loop ahead of time.

While Loop

```
main() {  
    int number = 0;  
    while (number < 1 || number > 5) {  
        cout << "Number: ";  
        cin >> number;  
        if (number < 1 || number > 5)  
            cout << "Enter a number between 1 and 5!" << endl;  
    }  
}
```

Do-while Loop

```
int main() {  
    int number;  
    do {  
        cout << "Number: ";  
        cin >> number;  
    } while (number < 1 || number > 5);  
  
    return 0;  
}
```

Syntax

```
do {  
    // code block to be executed  
}  
while (condition);
```

The **do/while** loop is a variant of the **while** loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Break and Continue Statements

```
#include <iostream>

using namespace std;

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i % 3 != 0)
            cout << i << endl;
    }

    return 0;
}
```

break: to break out of a loop

continue: to skip an iteration

```
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i % 3 == 0)
            continue;
        cout << i << endl;
    }

    return 0;
}
```

Break and Continue Statements

break: to break out of a loop

continue: to skip an iteration

```
while (true) {  
    cout << "Number: ";  
    int number;  
    cin >> number;  
    if (number >= 1 && number <= 5)  
        break;  
    cout << "Error! Enter a number between 1 and 5." <<  
}  
  
return 0;
```

Nested Loops

```
int main() {
    for (int x = 1; x <= 5; x++) {
        for (int y = 1; y <= 5; y++)
            cout << "(" << x << ", " << y << ")" << endl;
    }

    return 0;
}
```

Nested Loops

```
int main() {
    cout << "Rows: ";
    int rows;
    cin >> rows;

    for (int i = 1; i <= rows; i++) {
        for (int j = 0; j < i; j++)
            cout << "*";
        cout << endl;
    }
}
```

Example

```
Rows: 5

*
**
***
****
*****
```

Functions

Define and call functions

Assign parameters a default value

Overload functions

Pass arguments by value or reference

Local vs global variables

Organize functions in different files

```
#include <iostream>
using namespace std;

void greet() {
    cout << "Hello World" << endl;
}

int main() {
    // Calling - invoking - executing
    greet();

    cout << "Done";
    return 0;
}
```

Functions

- No parameters and no returns.
- Use **void** (No returns)

Functions

□ With parameters and return.

```
#include <iostream>

using namespace std;

void greet(string firstName, string lastName) {
    cout << "Hello " << firstName << " " << lastName << endl
}

string fullName(string firstName, string lastName) {
    // Concatenating (combining)
    return firstName + " " + lastName;
}

int main() {
    // Calling - invoking - executing
    string name = fullName(firstName: "Mosh", lastName: "Hamedani");
    greet(name);

    cout << "Done";

    return 0;
}
```

```
void greet(string name) {
    cout << "Hello " << name << endl;
}

int main() {
    // Calling - invoking - executing
    greet(name: fullName(firstName: "Mosh", lastName: "Hamedani"));
    cout << "Done";

    return 0;
}
```

```
#include <iostream>

using namespace std;

int max(int first, int second) {
    if (first > second)
        return first;
    else
        return second;
}

#include <iostream>

using namespace std;

int max(int first, int second) {
    if (first > second)
        return first;
    return second;
}
```

Functions

Example

A function that receives two integers and returns the maximum of the two.

Version 1

```
int max(int first, int second) {
    return (first > second) ? first : second;
}
```

Version 3

Version 2

Functions

```
#include <iostream> ^2 ^  
  
using namespace std;  
  
int max(int first, int second) {  
    return (first > second) ? first : second;  
}  
  
int main() {  
    int larger = max(first: 1, second: 2);  
    cout << larger  
  
    return 0;  
}
```

Example

A function that receives two integers and returns the maximum of the two.

Parameters with a Default Value

```
#include <iostream>

using namespace std;

double calculateTax(double income, double taxRate = .2) {
    return income * taxRate;
}

int main() {
    double tax = calculateTax(income: 10'000);
    cout << tax;

    return 0;
}
```

- If an argument is not provided for a parameter with a default value, then the default value is used.
- **NOTE:** Argument(s) with default value should be defined after the argument(s) without default value. Otherwise, a compiler error will occur.

Overloading Functions

```
void greet(string name) {  
    cout << "Hello " << name;  
}  
  
// Signature = name + (number and type of parameters)  
void greet(string title, string name) {  
    cout << "Hello " << title << " " << name;  
}
```

- ❑ Each function should have a unique signature.

Passing Arguments by Value or Reference

```
#include <iostream>

using namespace std;

void increasePrice(double price) {
    price *= 1.2;
}
```

Same

```
void increasePrice(double price) {
    double price;
    price *= 1.2;
}
```

- ❑ The parameter variable “price” is local to the function. (Its scope)

Passing Arguments by Value or Reference

```
#include <iostream>

using namespace std;

void increasePrice(double price) {
    double price;
    price *= 1.2;
}

int main() {
    double price = 100;
    increasePrice(price);
    cout << price;
```

- The parameter variable “price” is local to the function. (Its scope)



The scope of the `price` variable is different than the scope of argument `price`.

Passing Arguments by Value or Reference

```
#include <iostream>

using namespace std;

double increasePrice(double price) {
    price *= 1.2;
    return price;
}

int main() {
    double price = 100;
    price = increasePrice(price);
    cout << price;
}
```

- The parameter variable “price” is local to the function. (Its scope).
- By default, arguments are passed-by-value in C++
- The value is copied to the local variable in the function.

Solution 1:

- Return the value from the function.
- Use pass by value.

Passing Arguments by Value or Reference

```
#include <iostream>
```

```
using namespace std;
```

```
void increasePrice(double& price) {  
    price *= 1.2;  
}
```

```
int main() {  
    double price = 100;  
    increasePrice( & price );  
    cout << price;  
  
    return 0;
```

Reference parameter



- Pass by reference is appropriate when the argument data is large.

```
double increasePrice(double price) {  
    price *= 1.2;  
    return price;  
}
```

```
int main() {  
    double price = 100;  
    price = increasePrice(price);  
    cout << price;
```

Solution 2:

- Use pass by reference.

Passing Arguments by Value or Reference

```
#include <iostream>

using namespace std;
    void greet(string& name) {
        cout << "Hello " << name << endl;
        name = "a";
}

int main() {
    string name = "Mosh";
    greet( &name );
    cout << name;

    return 0;
}
```

- Pass-by-reference is appropriate when the argument data is large.

Local versus Global Variables

```
#include <iostream>

using namespace std;

// Global variable (global scope)
const double taxRate = .2;

double calculateTax(int sales) {
    return sales * taxRate;
}
```

- Constant variables that are used by multiple functions should be declared as global variables.

```
int main() {
    // Local variable (local scope)
    int sales = 10'000;
    double tax = calculateTax(sales);
    cout << tax;

    return 0;
}
```

- Variables are accessible within the block that they are declared.

Declaring Functions

```
#include <iostream>

using namespace std;

int main() {
    greet("Mosh");
    return 0;
}

void greet(string name) {
    cout << "Hello " << name;
}
```

- By default, in C++, functions should be **defined** before the main function.

Declaring Functions

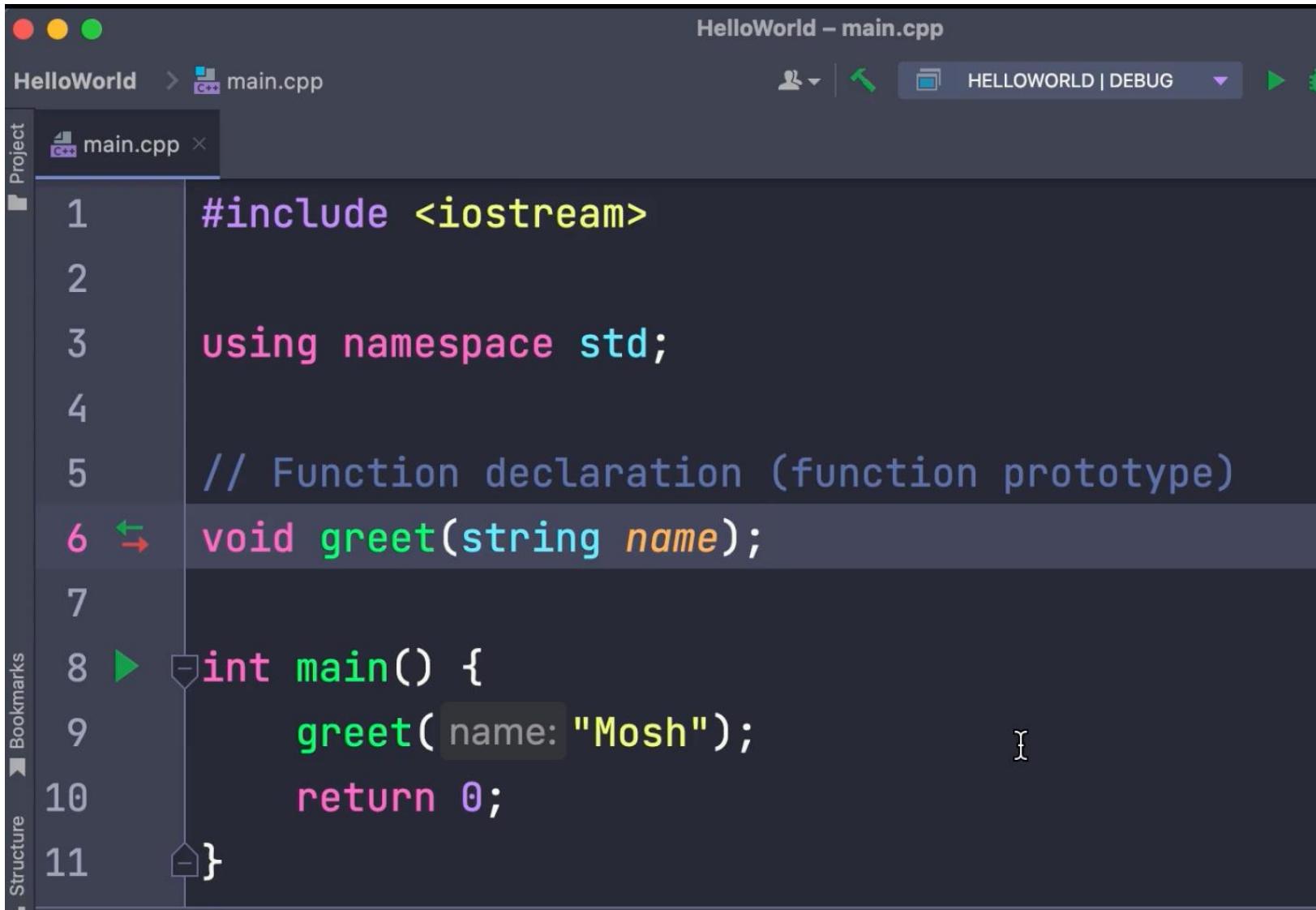
```
// Function declaration (function prototype)
void greet(string name);

int main() {
    greet( name: "Mosh");
    return 0;
}

// Function definition
void greet(string name) {
    cout << "Hello " << name;
}
```

- ❑ If we want to **define** a function after the main function, then we should **declare** the function first.

Organizing Functions in Files

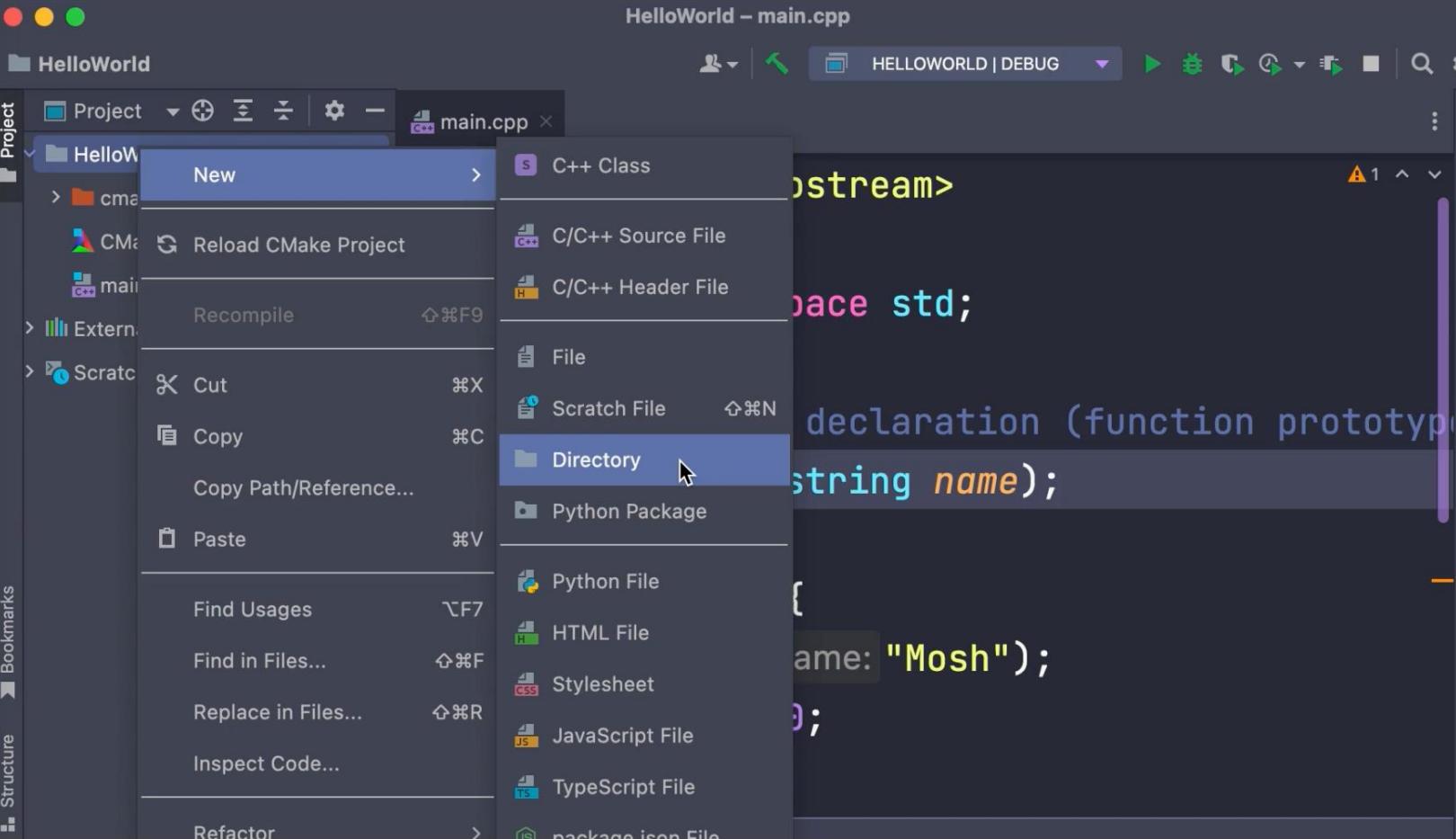


The screenshot shows a C++ development environment with a dark theme. The title bar reads "HelloWorld – main.cpp". The project navigation bar shows "HelloWorld" and "main.cpp". The toolbar includes icons for user profile, save, undo, redo, build configuration "HELLOWORLD | DEBUG", run, and terminal. The left sidebar has "Project" selected, showing a tree with "main.cpp" expanded. The main editor area contains the following code:

```
1 #include <iostream>
2
3 using namespace std;
4
5 // Function declaration (function prototype)
6 ↵ void greet(string name);
7
8 ► int main() {
9     greet( name: "Mosh" );
10    return 0;
11 }
```

The code includes a function declaration for `greet` and a call to it within the `main` function. The code editor uses color coding for syntax: `#include`, `using namespace std;`, `void`, `int`, and `return` are in blue; `iostream`, `greet`, and `name` are in green; and `<>`, `>`, and `<` are in orange.

Organizing Functions in Files



The screenshot shows a dark-themed IDE interface with a project named "HelloWorld". In the center, there is a code editor window titled "HelloWorld – main.cpp" containing the following C++ code:

```
#include <iostream>
#include <string>

using namespace std;

// Function declaration (function prototype)
void greet(string name);
```

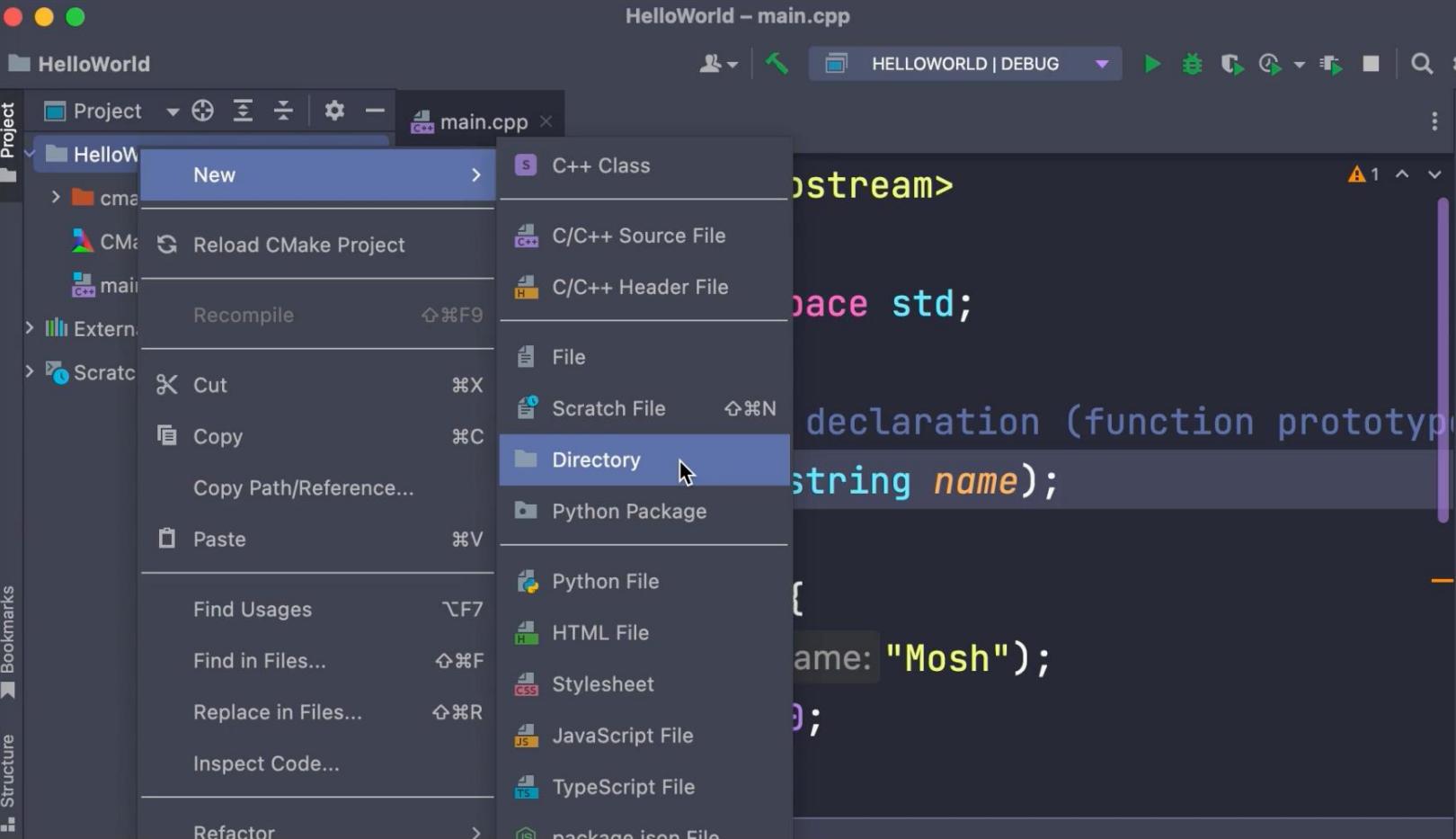
A context menu is open over the word "greet". The menu has a "New" submenu expanded, showing options like "C++ Class", "C/C++ Source File", "C/C++ Header File", "File", "Scratch File", "Directory" (which is highlighted in blue), and "Python Package". Other options in the "New" submenu include "Python File", "HTML File", "Stylesheet", "JavaScript File", and "TypeScript File". The main menu bar at the top includes "File", "Edit", "View", "Project", "Tools", "Help", and "File" again. The status bar at the bottom shows "package icon File".

Organizing Functions in Files

The screenshot shows the CLion IDE interface with a project named "HelloWorld". The "main.cpp" file is open in the editor. A context menu is displayed at line 6, with the option "New Directory" selected. A sub-menu shows the path "HelloWorld/main.cpp" and the directory name "utils" is highlighted. The code in the editor is:

```
#include <iostream>
using namespace std;
// Function declaration (function prototype)
int greet(string name);
int main() {
    greet(name: "Mosh");
    return 0;
}
```

Organizing Functions in Files



The screenshot shows a dark-themed IDE interface with a project named "HelloWorld". In the center, there is a code editor window titled "HelloWorld – main.cpp" containing the following C++ code:

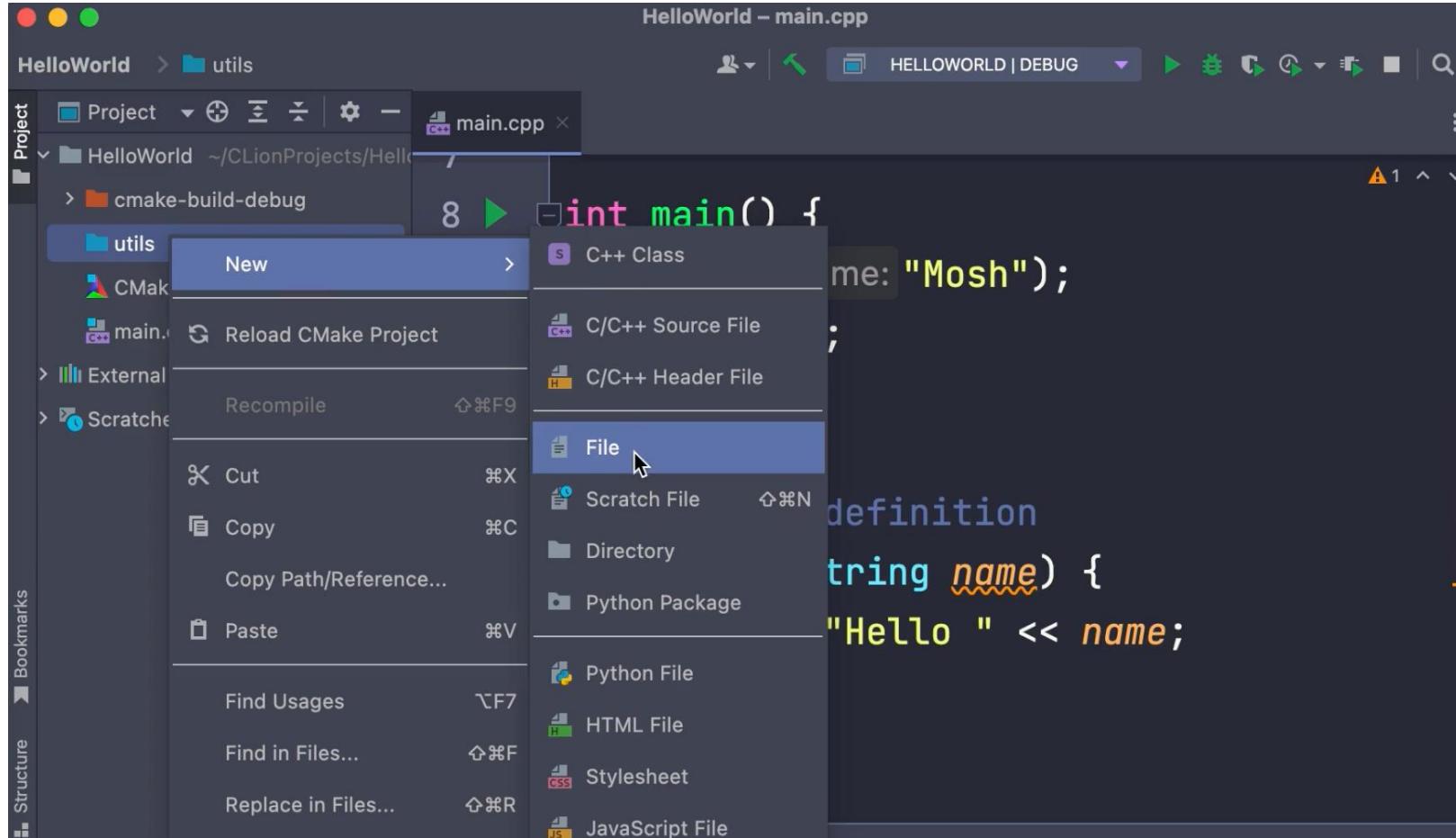
```
#include <iostream>
#include <string>

using namespace std;

// Function declaration (function prototype)
void greet(string name);
```

A context menu is open over the word "greet". The menu has a "New" submenu expanded, showing options like "C++ Class", "C/C++ Source File", "C/C++ Header File", "File", "Scratch File", "Directory" (which is highlighted in blue), and "Python Package". Other options in the "New" submenu include "Python File", "HTML File", "Stylesheet", "JavaScript File", and "TypeScript File". The main menu bar at the top includes "File", "Edit", "View", "Project", "Tools", "Help", and "About". The status bar at the bottom shows "package icon File".

Organizing Functions in Files



Organizing Functions in Files

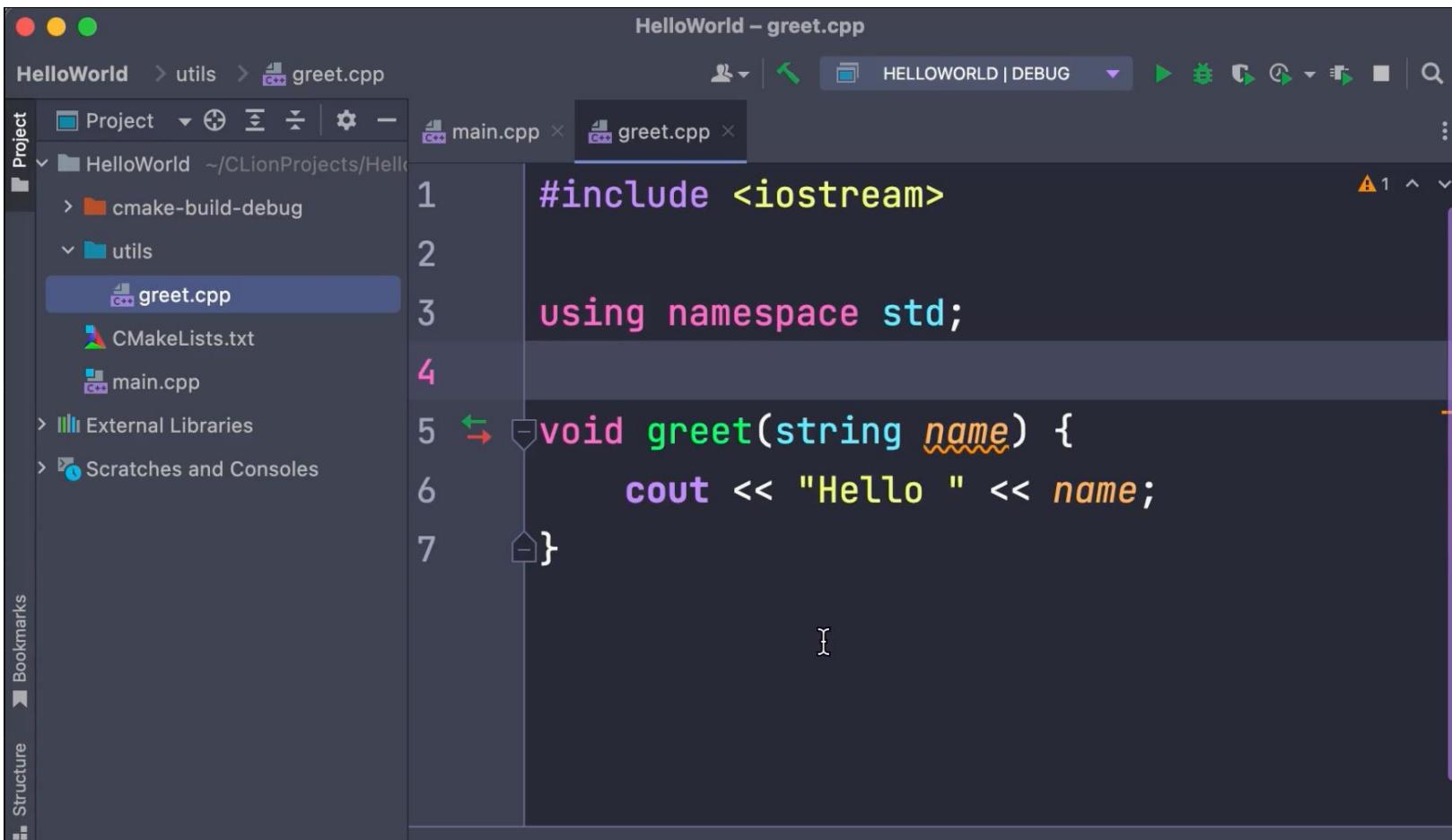
The screenshot shows the CLion IDE interface with a project named "HelloWorld". The "utils" folder is selected in the Project tool window. The main editor window displays the file "main.cpp" with the following code:

```
int main() {
    greet(name: "Mosh");
    return 0;
}

New File
greet.cpp
void greet(string name) {
    cout << "Hello " << name;
}
```

A context menu is open over the word "greet" at line 14, with the option "New File" highlighted. This indicates the process of extracting the function "greet" into its own file.

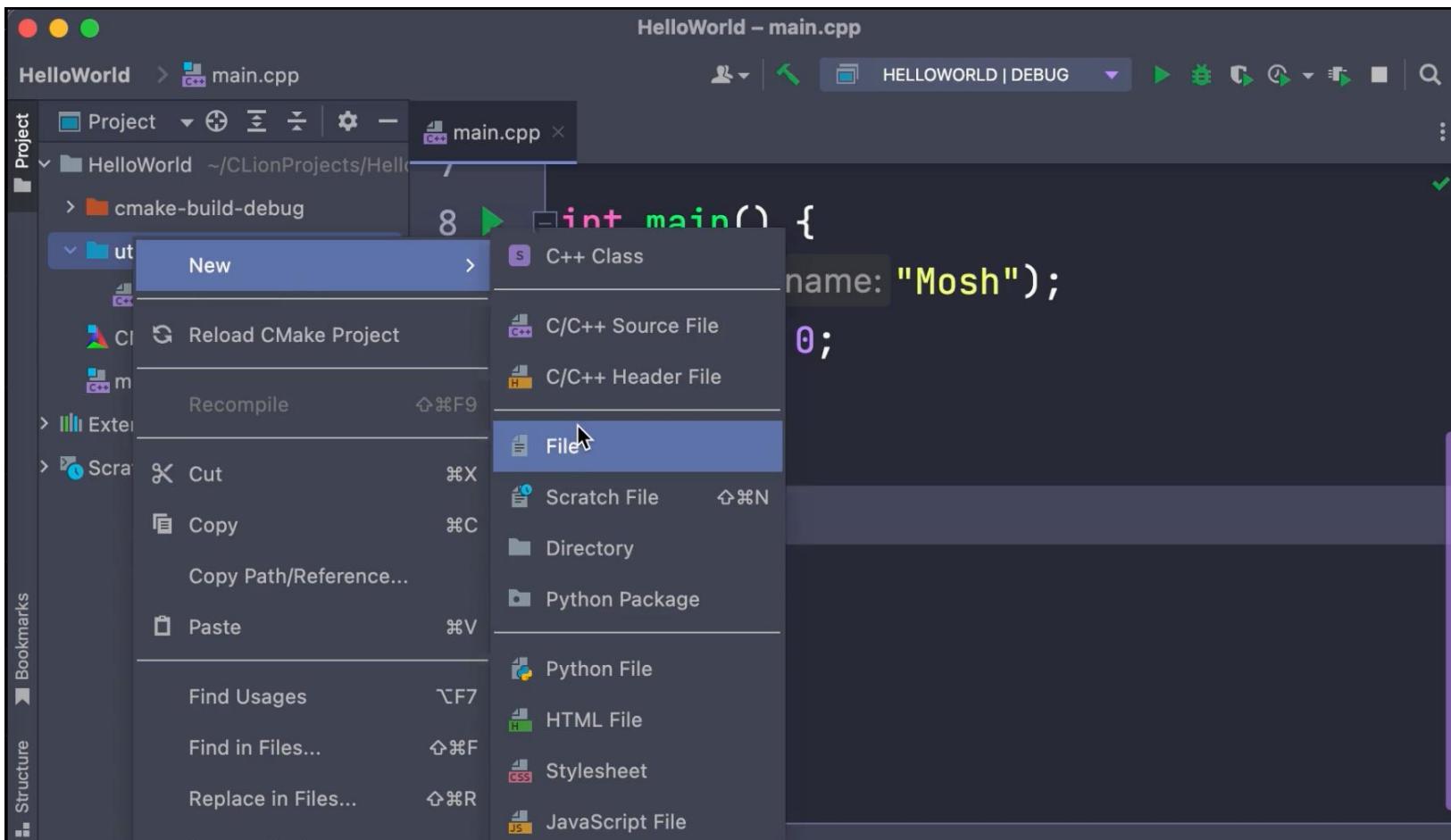
Organizing Functions in Files



The screenshot shows the CLion IDE interface with the project "HelloWorld" open. The "utils" directory contains three files: "greet.cpp", "main.cpp", and "CMakeLists.txt". The "greet.cpp" file is currently selected in the Project view and is being edited in the main editor window. The code in "greet.cpp" is as follows:

```
#include <iostream>
using namespace std;
void greet(string name) {
    cout << "Hello " << name;
}
```

Organizing Functions in Files



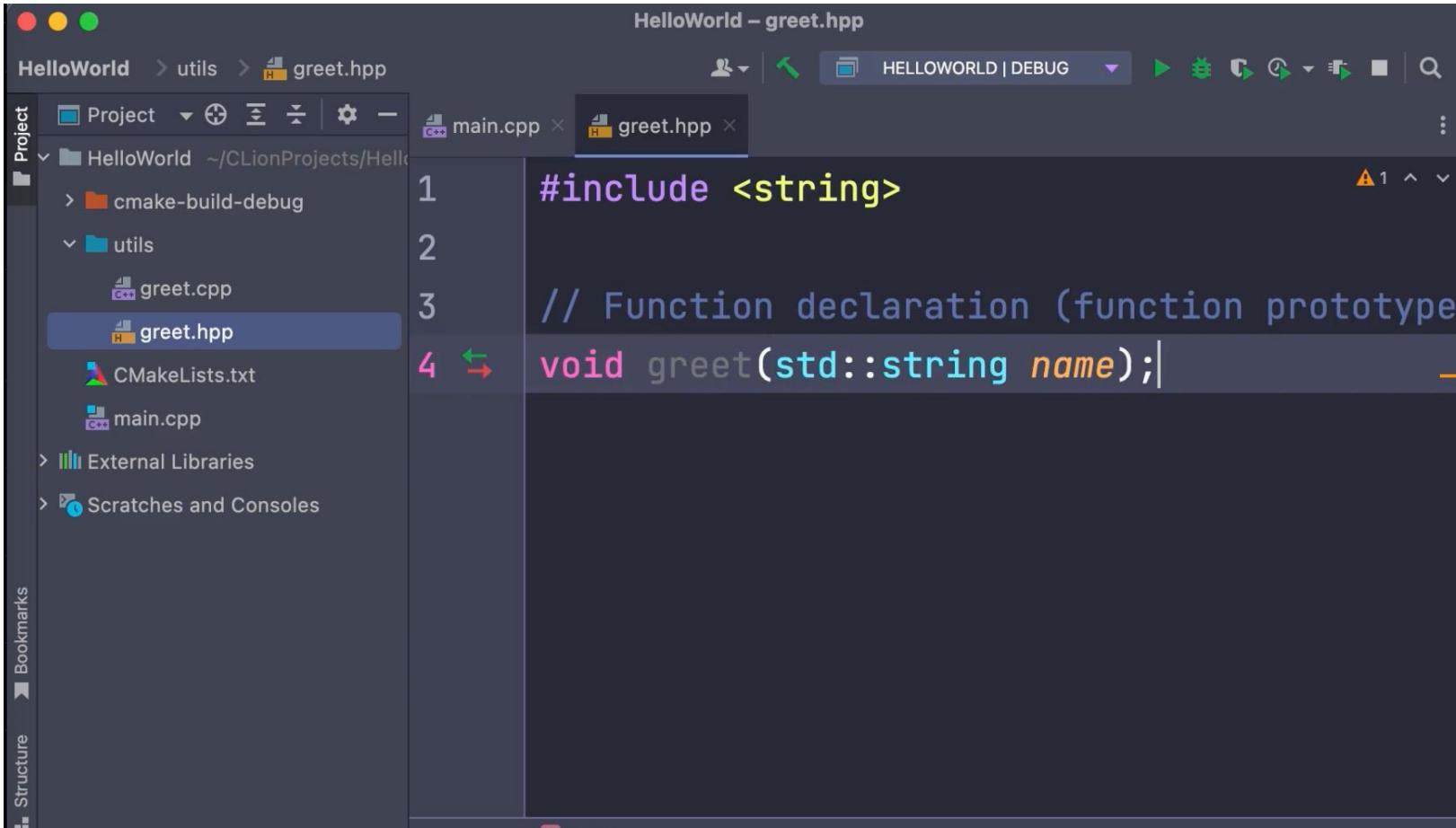
Organizing Functions in Files

The screenshot shows the CLion IDE interface with the following details:

- Title Bar:** HelloWorld – main.cpp
- Toolbar:** Includes icons for file operations, build, run, and debug.
- Project Bar:** HelloWorld > main.cpp
- Project View:** Shows the project structure:
 - Project
 - HelloWorld (~CLionProjects/HelloWorld)
 - cmake-build-debug
 - utils
 - greet.cpp
 - CMakeLists.txt
 - main.cpp
 - External Libraries
 - Scratches and Consoles
- Code Editor:** HelloWorld – main.cpp
- Code Content:**

```
int main() {
    greet(name: "Mosh");
    return 0;
}
```
- Tooltips:** A tooltip labeled "New File" is displayed above the cursor, which is positioned over the word "greet". Another tooltip labeled "greet.hpp" is shown in a dropdown menu below the cursor.

Organizing Functions in Files

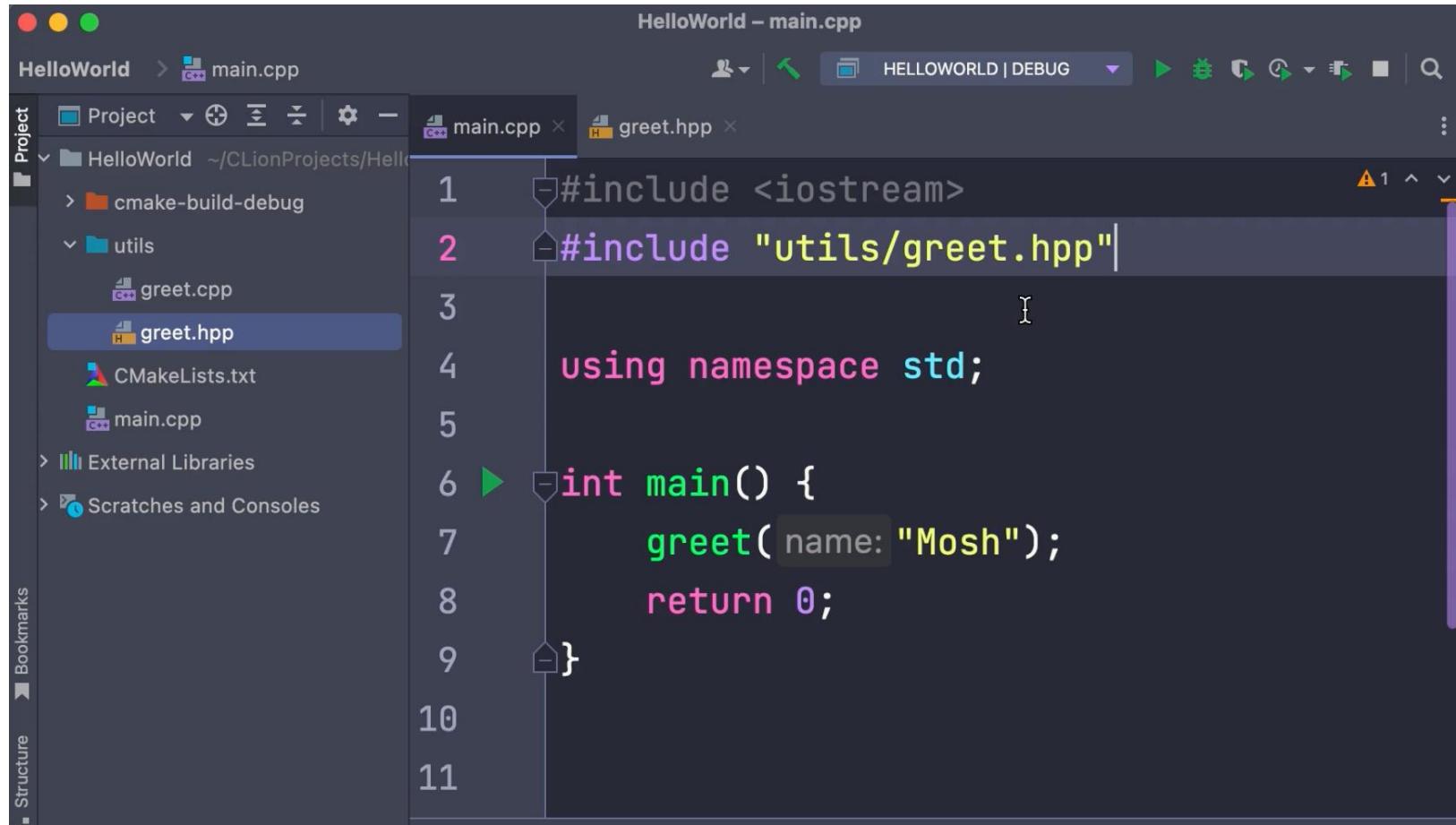


The screenshot shows the CLion IDE interface with a project named "HelloWorld". The "utils" directory contains two files: "greet.cpp" and "greet.hpp". The "greet.hpp" file is currently open in the editor, displaying the following code:

```
#include <string>
// Function declaration (function prototype)
void greet(std::string name);
```

The "Project" tool window on the left shows the file structure of the "HelloWorld" project.

Organizing Functions in Files



The screenshot shows the CLion IDE interface with the following details:

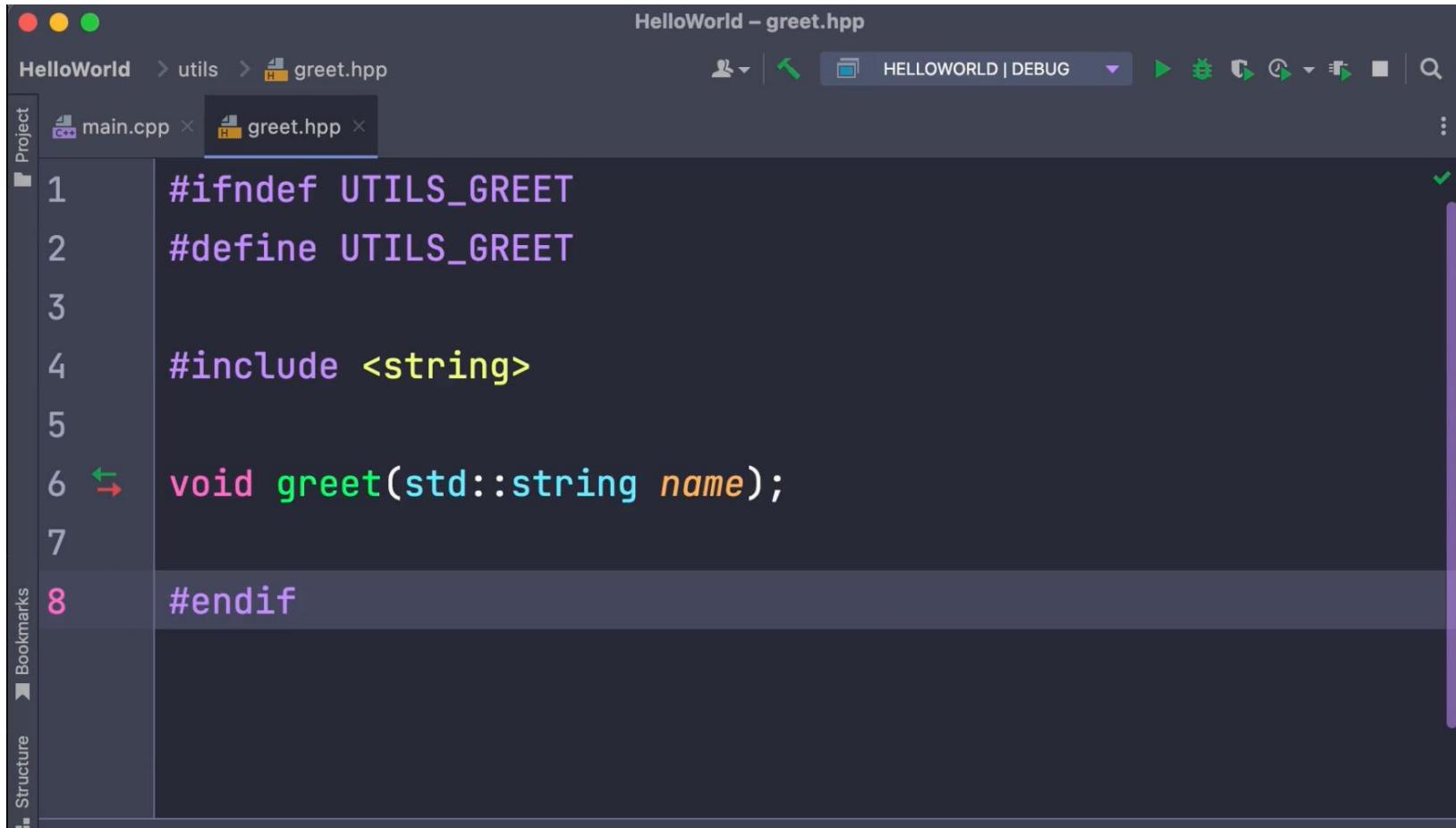
- Title Bar:** HelloWorld – main.cpp
- Toolbars:** Standard toolbar with icons for file operations, search, and navigation.
- Project View:** Shows the project structure under "HelloWorld".
 - cmake-build-debug
 - utils
 - greet.cpp
 - greet.hpp (selected)
 - CMakeLists.txt
 - main.cpp
 - External Libraries
 - Scratches and Consoles
- Code Editor:** The main window displays the content of main.cpp.

```
#include <iostream>
#include "utils/greet.hpp"

using namespace std;

int main() {
    greet(name: "Mosh");
    return 0;
}
```
- Status Bar:** Shows "HELLOWORLD | DEBUG" and other build-related information.

Organizing Functions in Files



```
HelloWorld – greet.hpp
HelloWorld > utils > greet.hpp
Project main.cpp × greet.hpp × ...
1 #ifndef UTILS_GREET
2 #define UTILS_GREET
3
4 #include <string>
5
6 ↗ void greet(std::string name);
7
8 #endif
```

- Update the greet.hpp header file so that it will not be build multiple times by the compiler if it is included in multiple files.

Organizing Functions in Files



The screenshot shows a code editor interface with the following details:

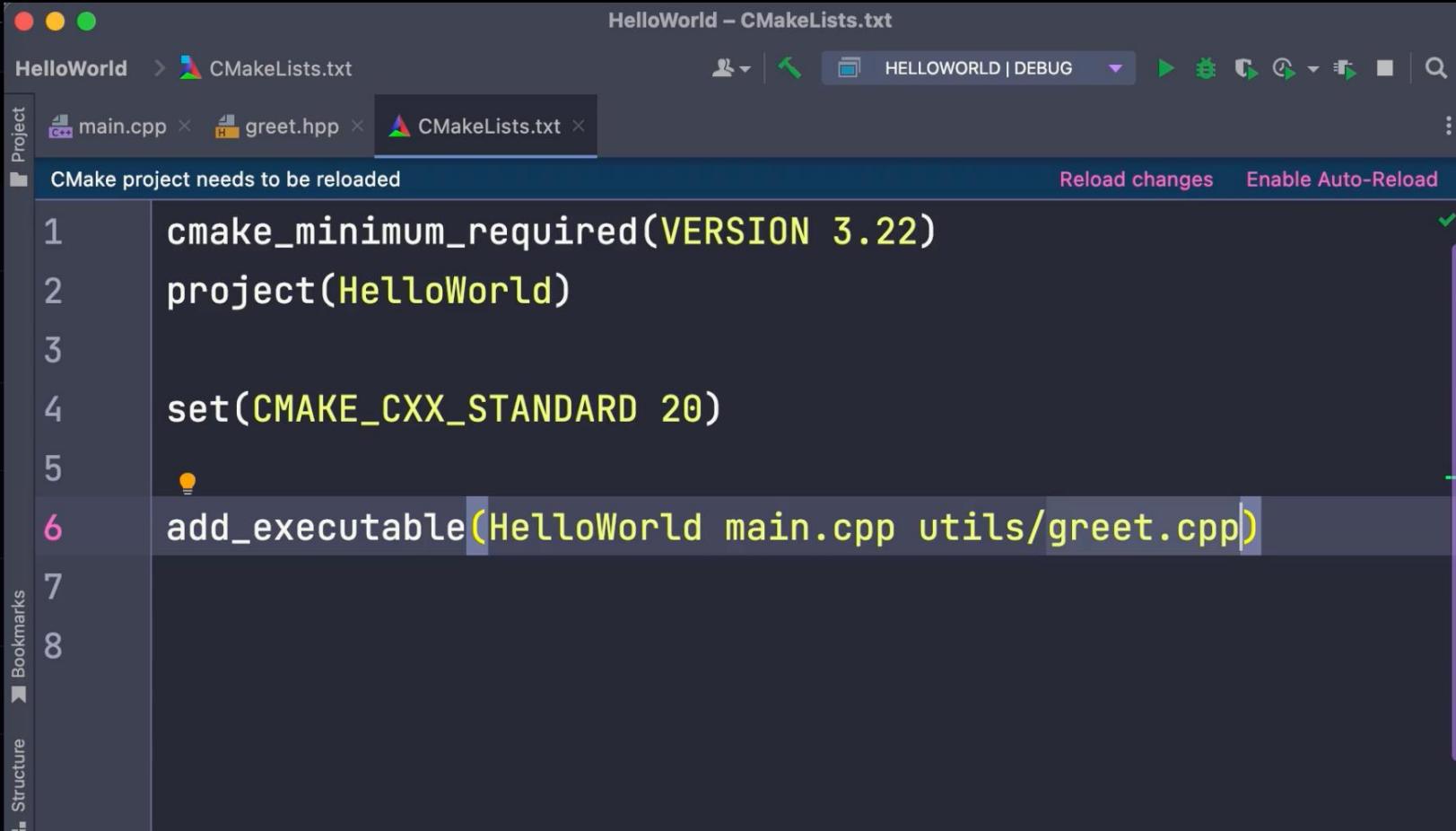
- Project Bar:** HelloWorld > utils > greet.hpp
- Toolbars:** Standard icons for file operations.
- Status Bar:** HELLOWORLD | DEBUG
- Code Editor:** File: greet.hpp. Content:

```
1 #ifndef UTILS_GREET
2 #define UTILS_GREET
3
4 #include <string>
```
- Messages Tab:** Build tab. Content:

```
"greet(std::__1::basic_string<char, std::__1::char_traits<cha
_main in main.cpp.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to
ninja: build stopped: subcommand failed.
```
- Sidebar:** Project, Bookmarks, Structure.

- ☐ When we run the project, we will get this error message!
- ☐ One more thing to do!

Organizing Functions in Files



The screenshot shows a code editor window titled "HelloWorld – CMakeLists.txt". The window displays the following CMake configuration code:

```
cmake_minimum_required(VERSION 3.22)
project(HelloWorld)

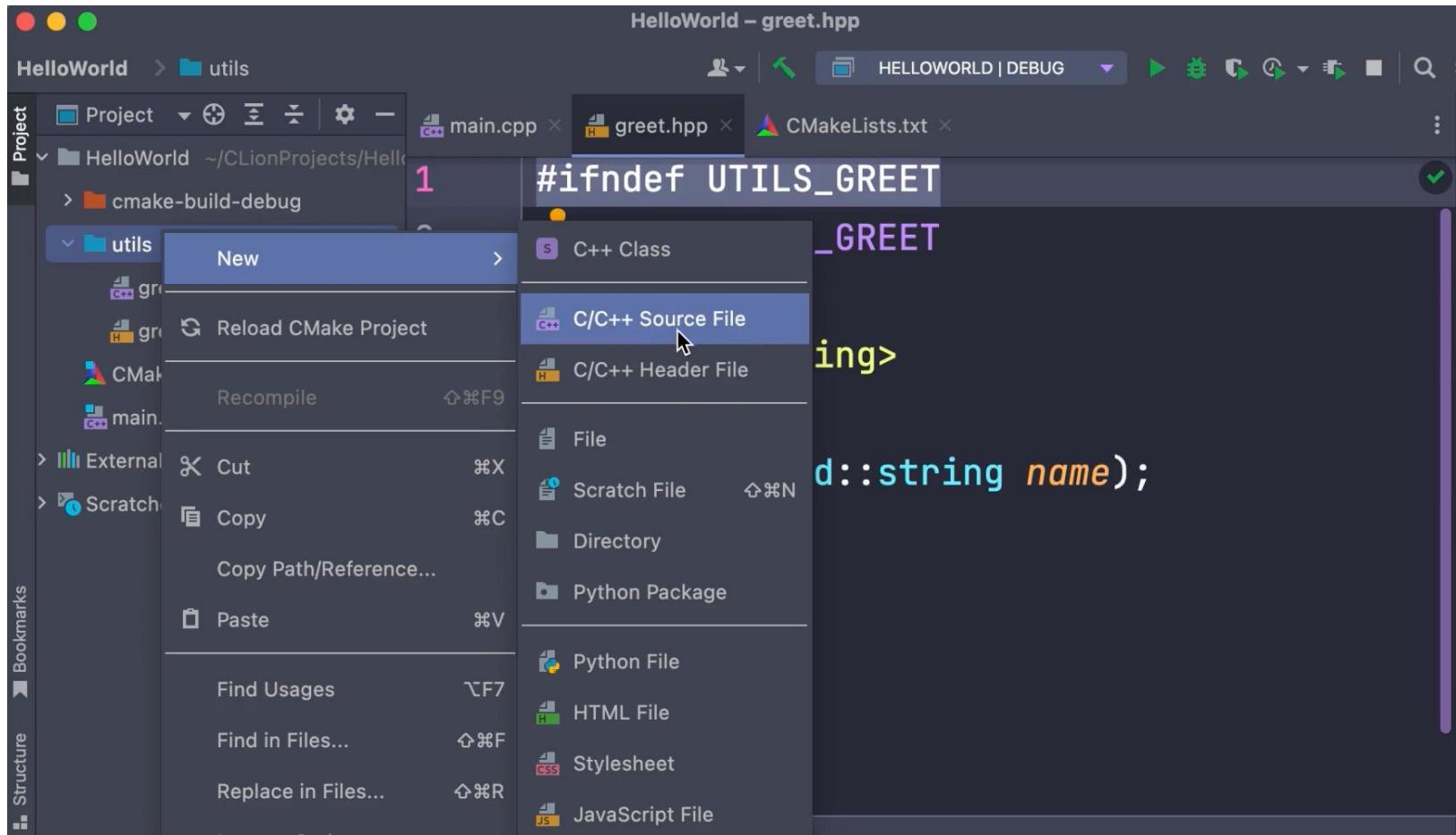
set(CMAKE_CXX_STANDARD 20)

add_executable(HelloWorld main.cpp utils/greet.cpp)
```

The code editor interface includes a toolbar at the top with icons for user, refresh, build, run, and search. Below the toolbar is a tab bar with "main.cpp", "greet.hpp", and "CMakeLists.txt". A status bar at the bottom indicates "CMake project needs to be reloaded" with buttons for "Reload changes" and "Enable Auto-Reload". On the left side, there are navigation buttons for "Bookmarks" and "Structure".

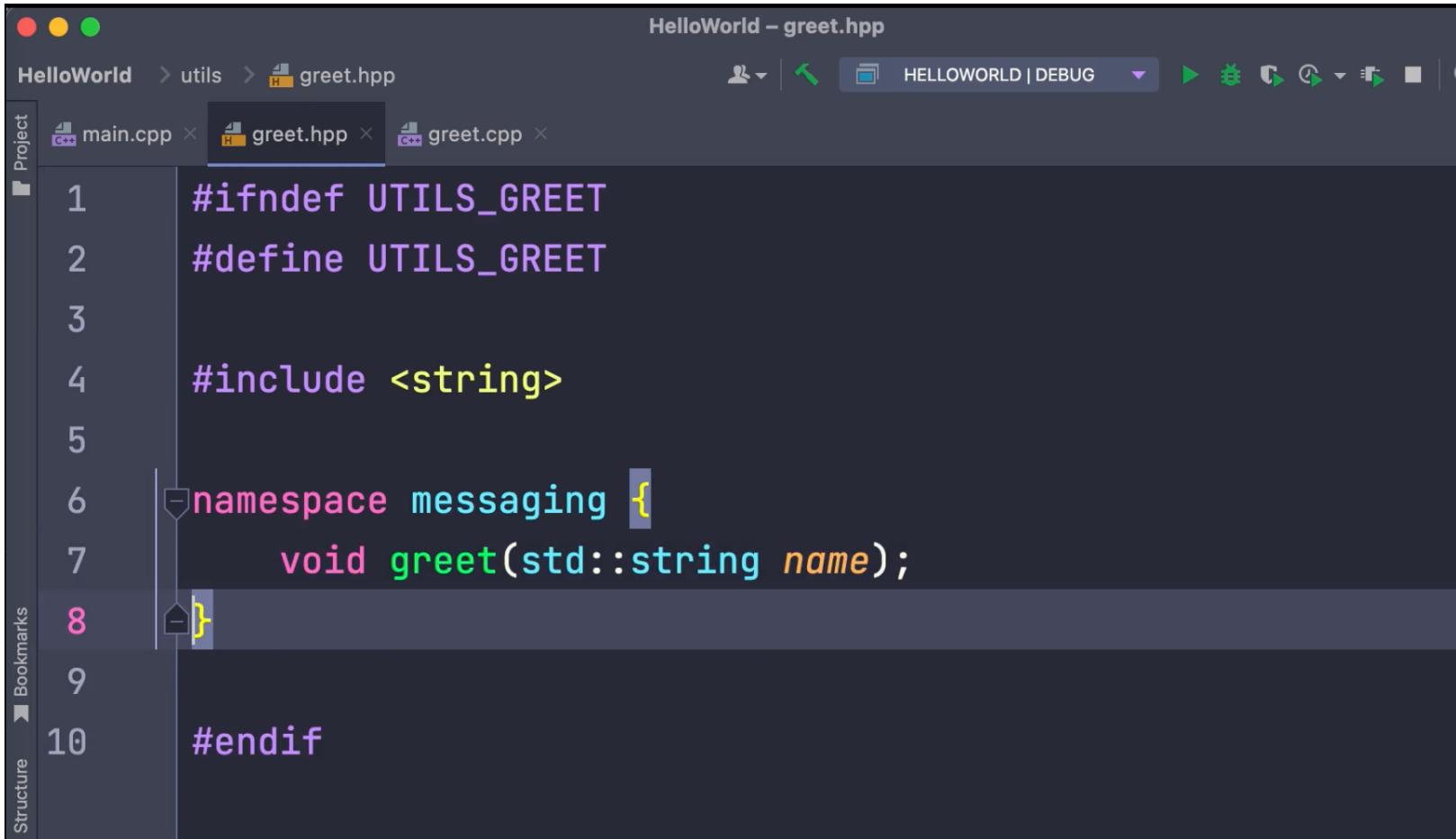
- Update the CMakeLists.txt file to include the greet.cpp file.

Organizing Functions in Files



- ❑ Shortcut in CLion.

Using Namespaces



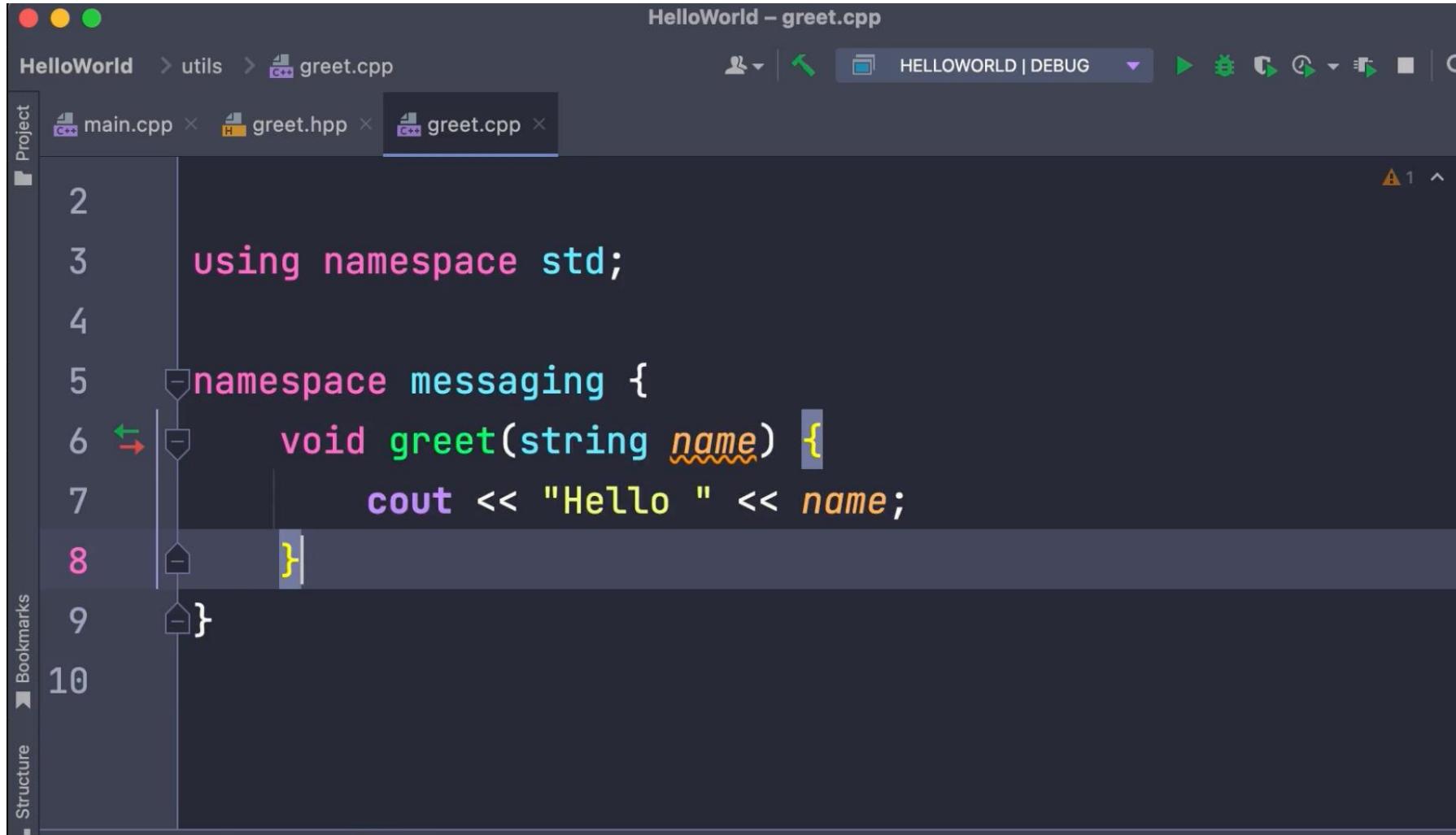
The screenshot shows a code editor window titled "HelloWorld – greet.hpp". The project navigation bar indicates the current file is "greet.hpp". The code editor displays the following content:

```
1 #ifndef UTILS_GREET
2 #define UTILS_GREET
3
4 #include <string>
5
6 namespace messaging {
7     void greet(std::string name);
8 }
9
10#endif
```

The code uses a namespace named "messaging" to encapsulate a function "greet". The code editor interface includes a "Project" sidebar on the left and various toolbars at the top.

- Namespaces are used to prevent conflict of having functions with the same name developed multiple times.
- It is like a bucket where we put all our functions in.

Using Namespaces



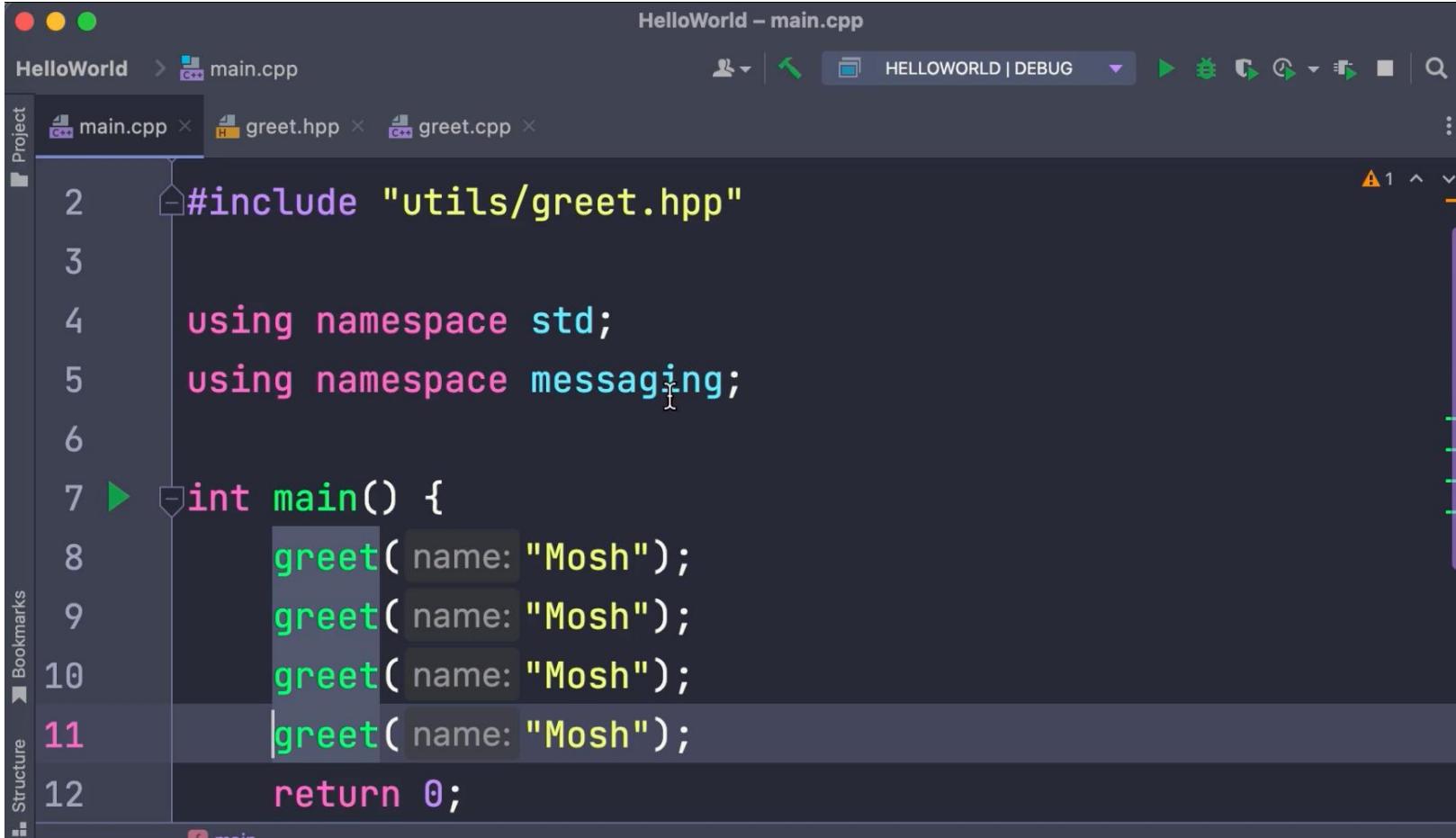
The screenshot shows a C++ development environment with the following details:

- Title Bar:** HelloWorld – greet.cpp
- Toolbar:** Includes icons for user, file, build, and search.
- Project Bar:** Shows main.cpp, greet.hpp, and greet.cpp. greet.cpp is the active file.
- Code Editor:** Displays the following C++ code:

```
2
3 using namespace std;
4
5 namespace messaging {
6     void greet(string name) {
7         cout << "Hello " << name;
8     }
9 }
10 }
```

The code uses the `using namespace std;` directive at the top. It defines a namespace `messaging` containing a function `greet` that prints a greeting to `cout`. The code editor highlights the `name` parameter with a wavy underline, indicating it is a variable.
- Sidebar:** Includes Project, Bookmarks, and Structure tabs.

Using Namespaces

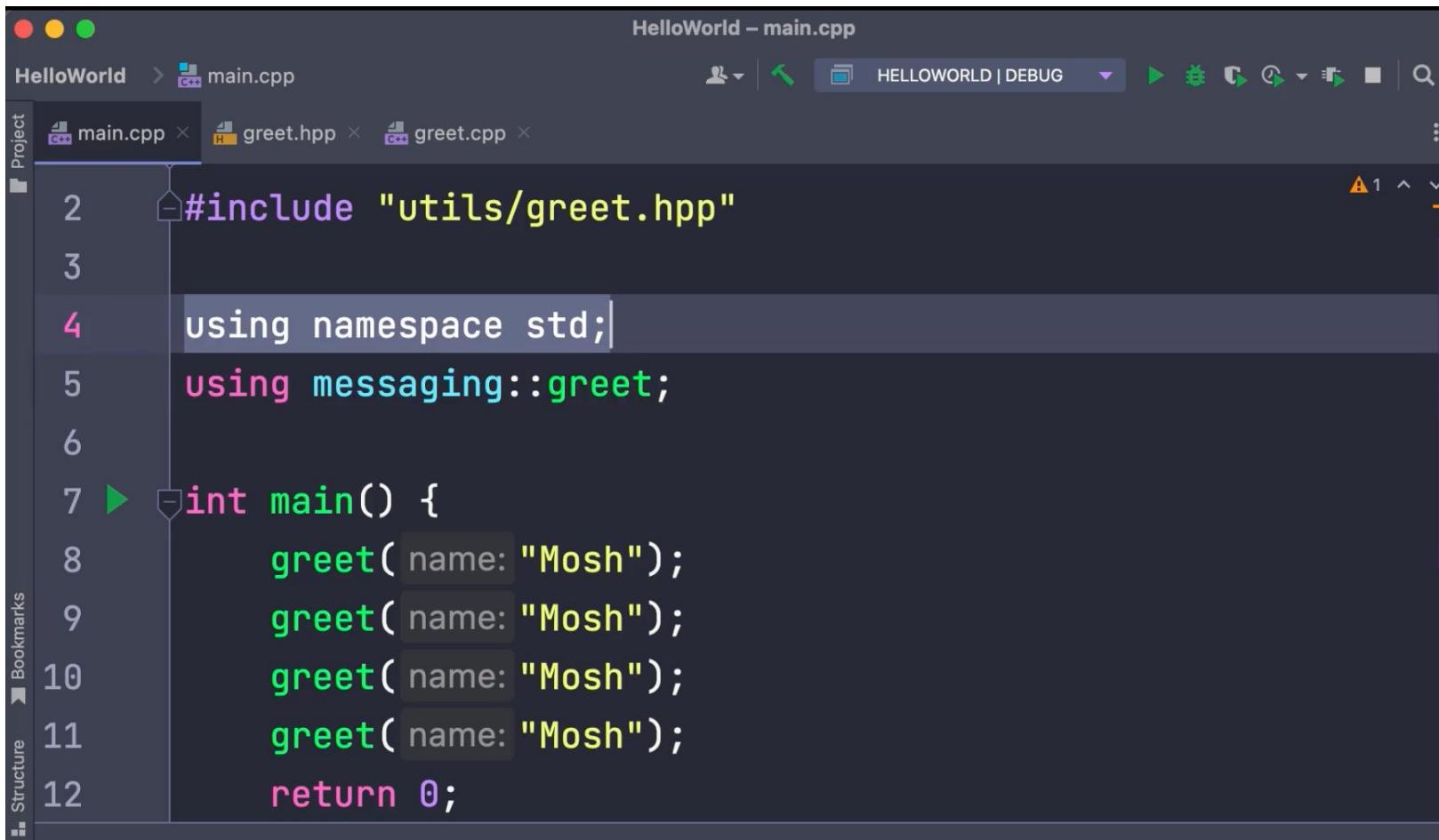


The screenshot shows a dark-themed IDE interface with the following details:

- Title Bar:** HelloWorld – main.cpp
- Toolbar:** Includes icons for user, file, and various build and run operations.
- Project View:** Shows the project structure with files main.cpp, greet.hpp, and greet.cpp.
- Code Editor:** Displays the main.cpp file content:

```
1 //include "utils/greet.hpp"
2
3
4 using namespace std;
5 using namespace messaging;
6
7 int main() {
8     greet( name: "Mosh");
9     greet( name: "Mosh");
10    greet( name: "Mosh");
11    greet( name: "Mosh");
12
13    return 0;
14}
```
- Status Bar:** Shows the file name main.cpp.

Using Namespaces



The screenshot shows a C++ development environment with a dark theme. The project is named "HelloWorld" and the active file is "main.cpp". The code editor displays the following content:

```
>HelloWorld - main.cpp
HelloWorld > C++ main.cpp
Project: main.cpp x greet.hpp x greet.cpp x
1 #include "utils/greet.hpp"
2
3
4 using namespace std;
5 using messaging::greet;
6
7 int main() {
8     greet(name: "Mosh");
9     greet(name: "Mosh");
10    greet(name: "Mosh");
11    greet(name: "Mosh");
12    return 0;
13 }
```

The code demonstrates the use of namespaces. It includes the "greet.hpp" header from the "utils" namespace. Inside the "main" function, it uses the `using namespace std;` directive and the `using messaging::greet;` directive to bring the `greet` function into the current scope. The `greet` function is called four times with the argument "Mosh".

- ❑ Include only greet function from the namespace instead of the entire messaging namespace functions.

Debugging

```
#include <iostream>  
using namespace std;
```

```
void printOddNumbers(int limit) {  
    for (int i = 0; i < limit; i++) {  
        if (i % 2 != 0) Change to  
            cout << i << endl;  
    }  
}
```

```
int main() {  
    printOddNumbers( limit: 10 );  
    return 0;  
}
```

```
void printOddNumbers(int limit) {  
    for (int i = 0; i < limit; i++) {  
        if (i % 2 != 0)  
            cout << i << endl;  
    }  
}  
  
int main() {  
    printOddNumbers( limit: 10 );  
    return 0;  
}
```

Debugging

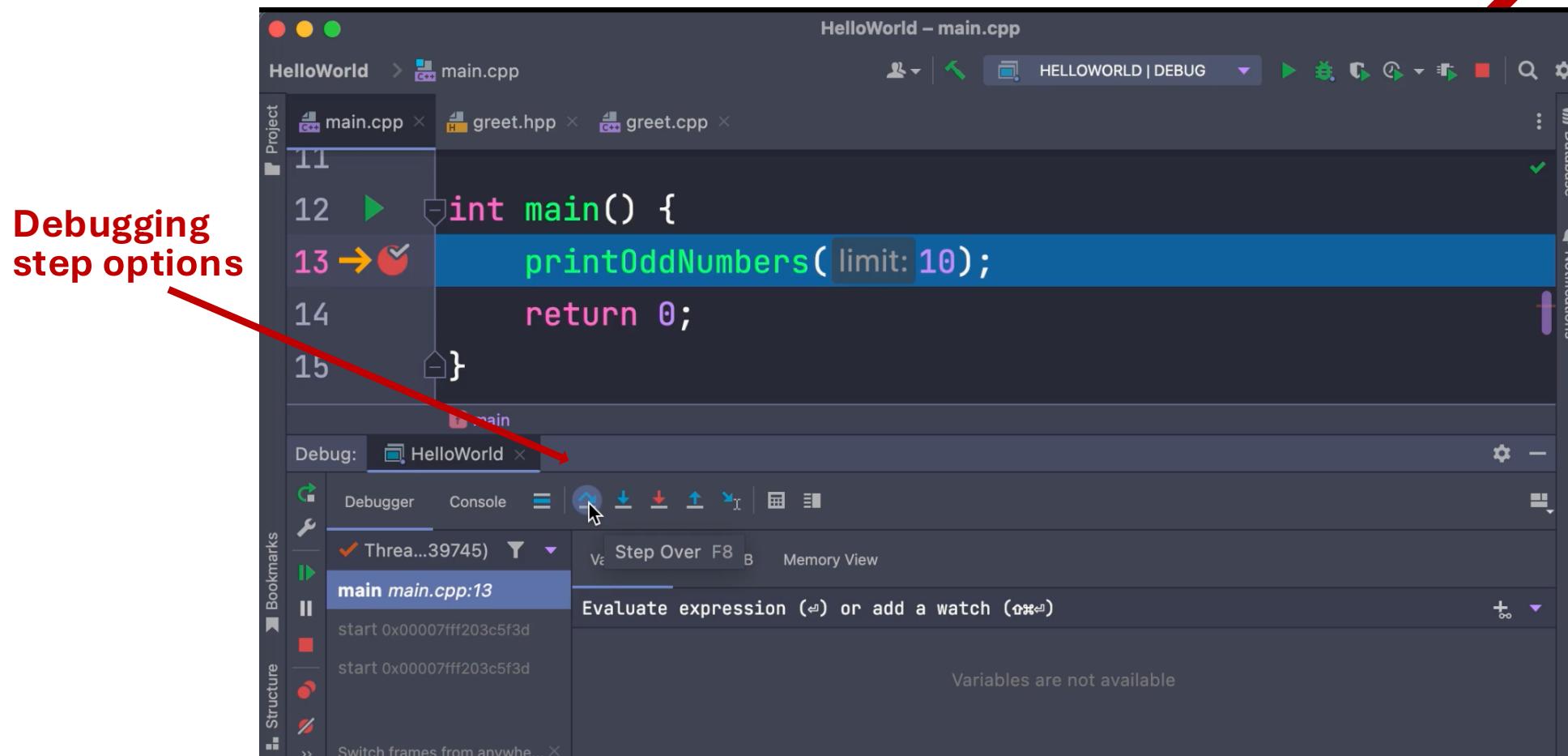
Debug button

The screenshot shows a C++ development environment with the following details:

- Title Bar:** HelloWorld – main.cpp
- Toolbar:** Includes icons for user, file, and various debug and build actions. The "HELLOWORLD | DEBUG" dropdown is selected.
- Project Explorer:** Shows files main.cpp, greet.hpp, and greet.cpp.
- Code Editor:** Displays the main.cpp code with line numbers 5 to 15. A red circle highlights a break point at line 13, which is also annotated with a red arrow labeled "Break point".
- Callout:** A red arrow points to the "Debug 'HelloWorld'" button in the toolbar, which is annotated with the text "Debug button".

```
5 void printOddNumbers(int limit) {  
6     for (int i = 0; i < limit; i++) {  
7         if (i % 2 != 0)  
8             cout << i << endl;  
9     }  
10    }  
11  
12 int main() {  
13     printOddNumbers( limit: 10);  
14     return 0;  
15 }
```

Debugging



Debugging

The screenshot shows a debugger interface with the following details:

- Code View:** The main window displays the C++ file `main.cpp`. A red arrow points from the text "Copy the expression into a watch in the debugger." to the line of code `i % 2 == 0`.
- Breakpoint:** A yellow circular breakpoint icon is visible on line 7.
- Variables View:** The bottom panel shows the `Variables` tab of the debugger. It lists variables `limit` and `i`, both currently set to 0.
- Watch List:** The `Watch` section of the Variables view has a button labeled `+ New Watch...`.

Copy the expression into a
watch in the debugger.

The screenshot shows a debugger interface with the following details:

- Code View:** The main window displays the C++ file `main.cpp`.
- Breakpoint:** A yellow circular breakpoint icon is visible on line 7.
- Variables View:** The bottom panel shows the `Variables` tab of the debugger. It lists variables `limit` and `i`, both currently set to 0.
- Watch List:** The `Watch` section of the Variables view has a button labeled `+ New Watch...`. A red arrow points to this button, indicating where to click to add a new watch.