OBJECT-ORIENTED PROGRAMMING

CL-1004

LAB 01

Introduction to C++ Programming: Program Structure, Basic Input/Output, Data Types, Functions, and Arrays.

National University of Computer and Emerging Sciences–NUCES – Karachi

Instructor: Talha Shahid

Contents

1. Introduction to IDE (Visual Studio Code)

1. First, you have to download and install the Visual Studio. For that, you can refer to:

   Visual Studio

2. After you installed the IDE, you need to configure the VS Code for the C++ compiler.

1.1. MinGW for C++ Compiler

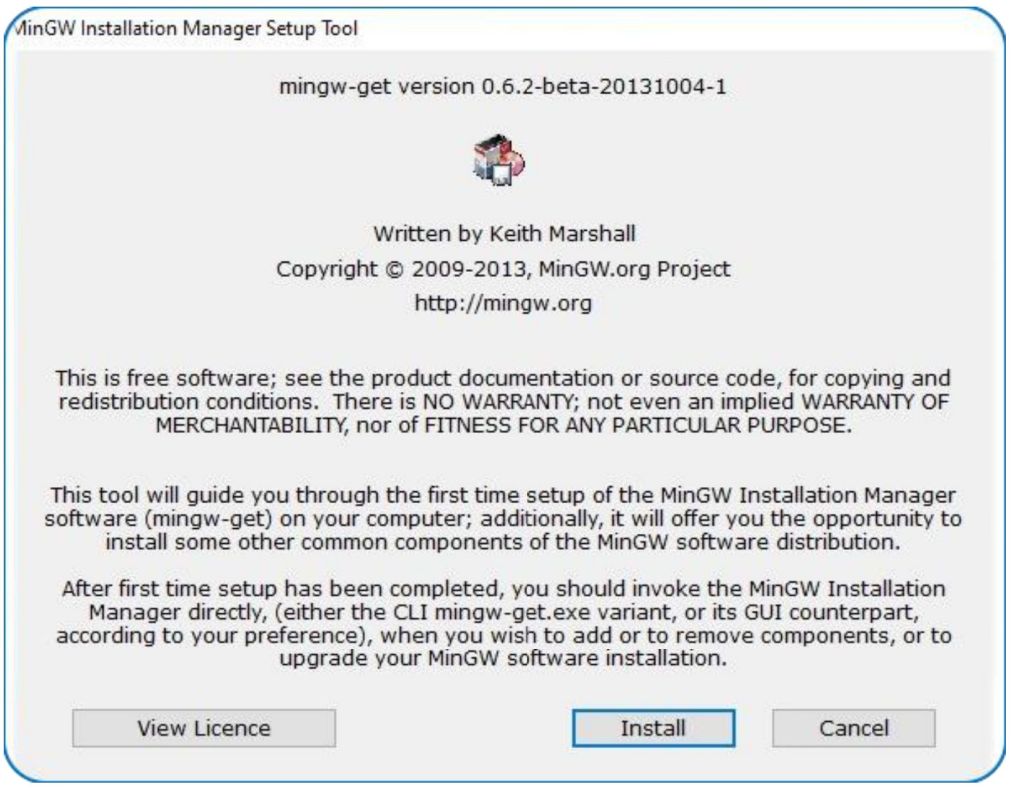
MinGW software consists of three basic components: the assembler, linker, and compiler. It is sufficient to begin designing applications with this minimal tool set.

1.1.1. MinGW C++ Download and Installation

1. First you have to download and install the MinGW for the C++ compiler. For that, you can refer to

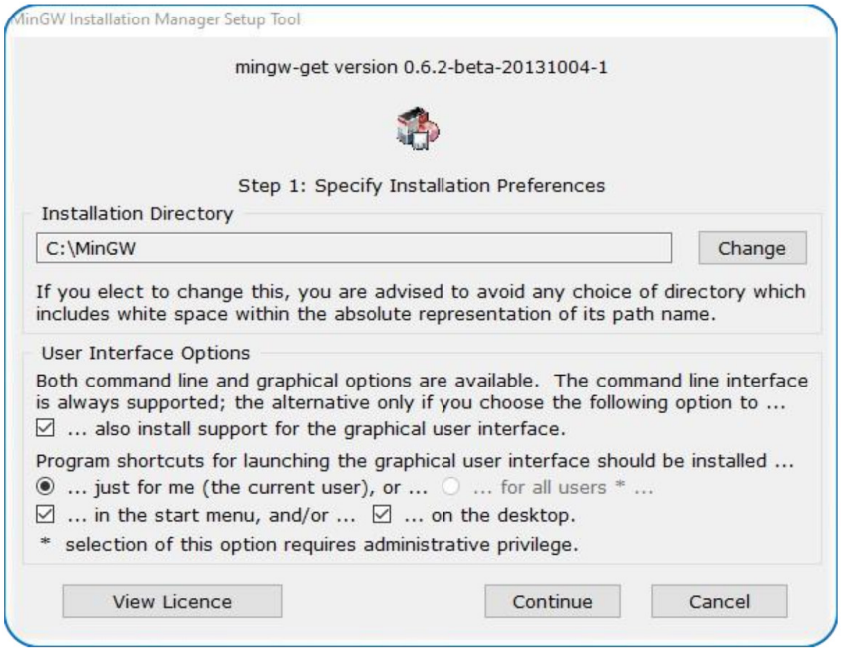
   C++ Compiler

2. After you download the MinGW, Run the setup and then following window will appear on your screen.



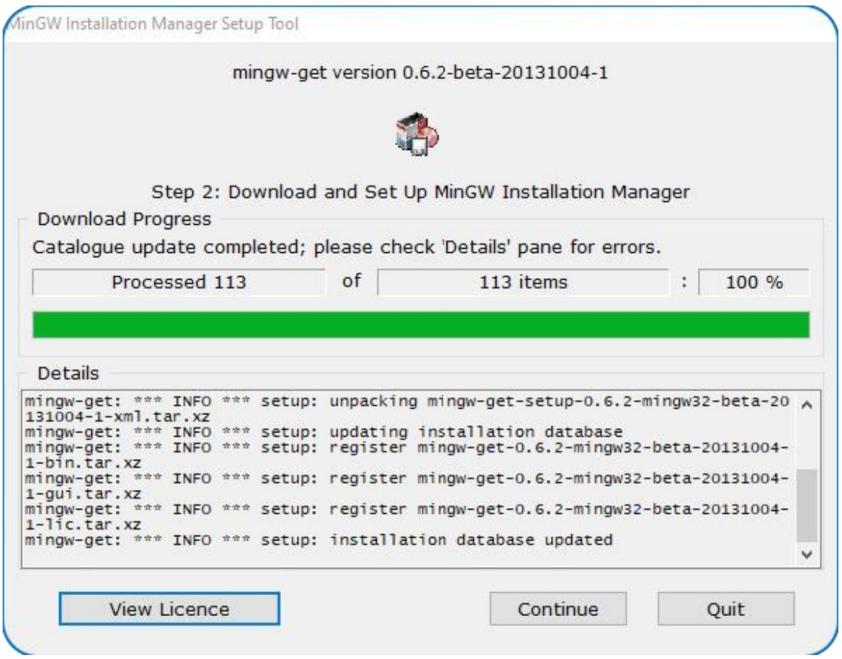
3. Click on **`Install.`**

The following pop-up window will appear.



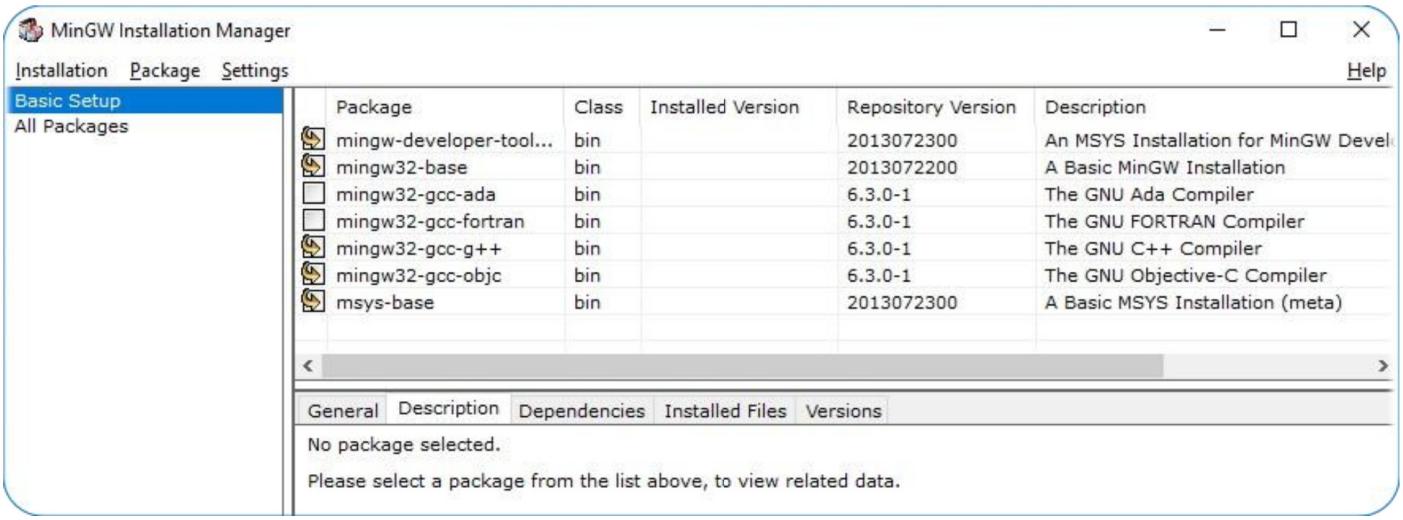
You can install this software anywhere, but it is recommended to install it in the default directory: **`C:\MinGW`**

4. Click on **`Continue.`** The following pop-up window will appear, showing the downloading progress. After about a minute, it should appear as follows.

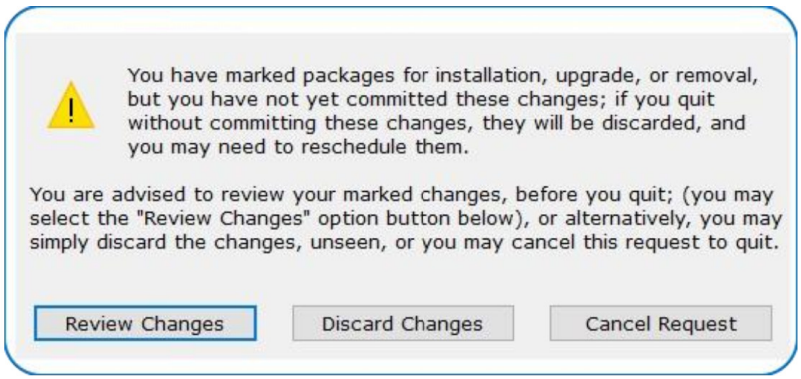


5. Click on **Continue.**

The following pop-up window will appear. Ensure on the left that **Basic Setup** is highlighted. Click the five boxes indicated below: **mingw- developer-tool, mingw32-base, mingw32-gcc-g++, mingw32-gcc-obj, msys-base.** After clicking each, select **Mark for selection.** This window should appear as follows.

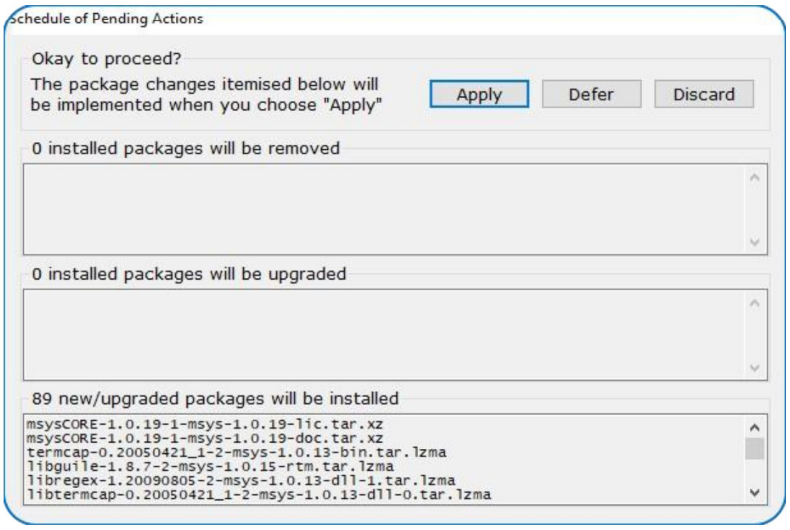


6. Terminate (click X on) the MinGW Installation Manager. The following pop- up window should appear.



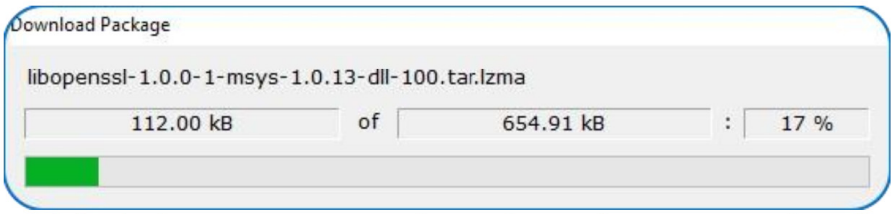
7. Click on **`Review Changes`**

The following pop-up window should appear.



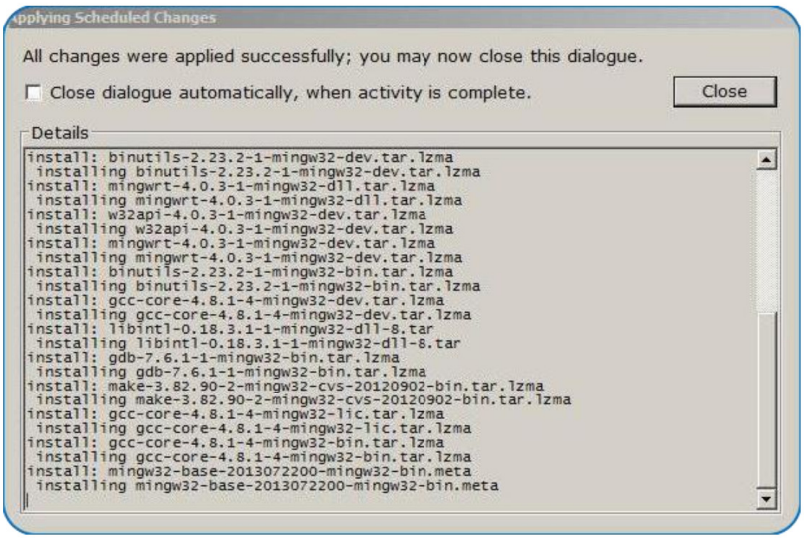
8. Click on **`Apply`**

The following pop-up window will appear, showing the downloading progress.



After a while (a few minutes to an hour, depending on your download speed), it should start extracting the downloaded files. A few minutes after that, the following pop-up window should appear.

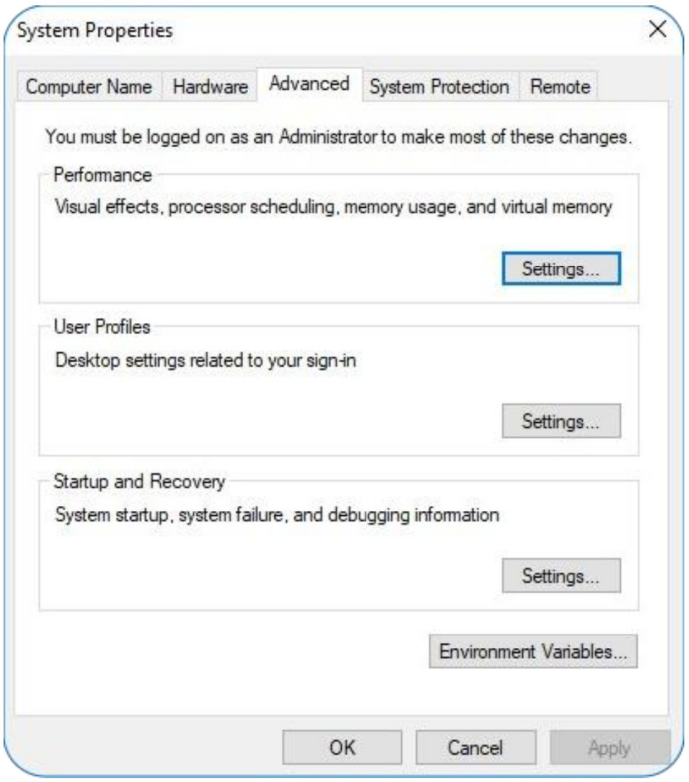
9. Click on **`Close.`**



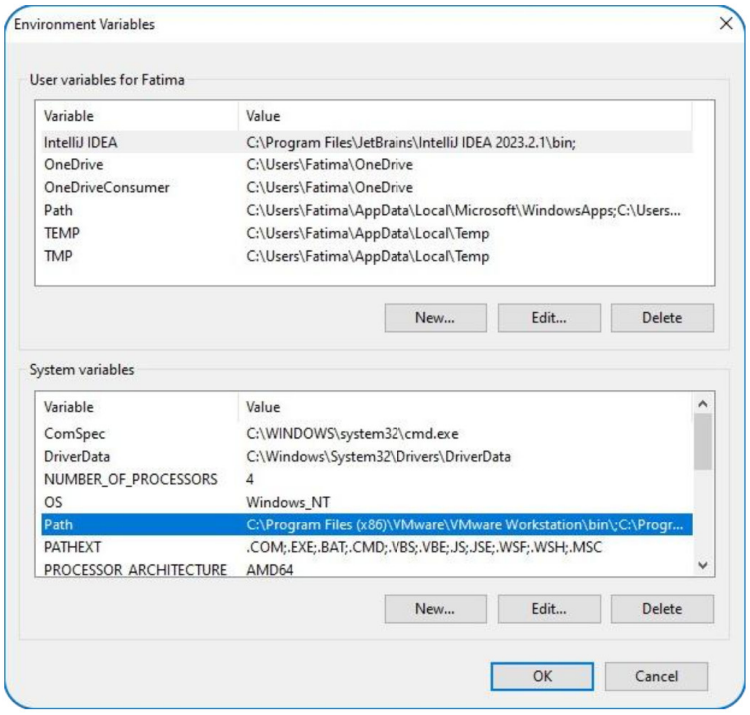
10. Edit **`Path`** so that the **`MinGW`** and **`MSYSM`** software is findable by VS Code.

a. Click on **`This PC`**
b. Click on **`Local disk(C)`**
c. Click on **`MinGW`**
d. Click on **`bin`**
e. Copy on **`Path`**

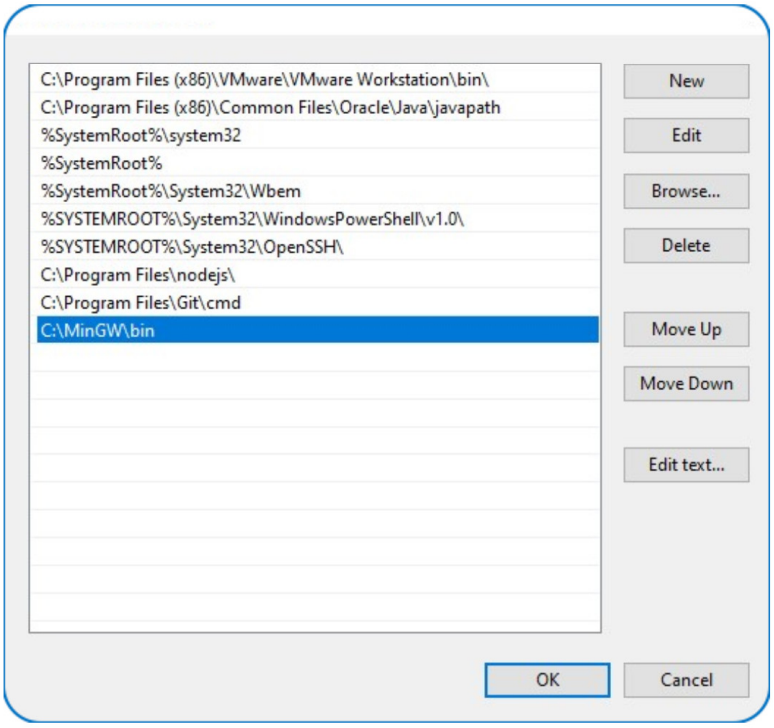
11. For setting path, Click on **`Properties`** of **`This PC`**, then go to **`Advance system setting`**.



12. Click on **`Environment Variables`**. Following screen will appear on your screen.

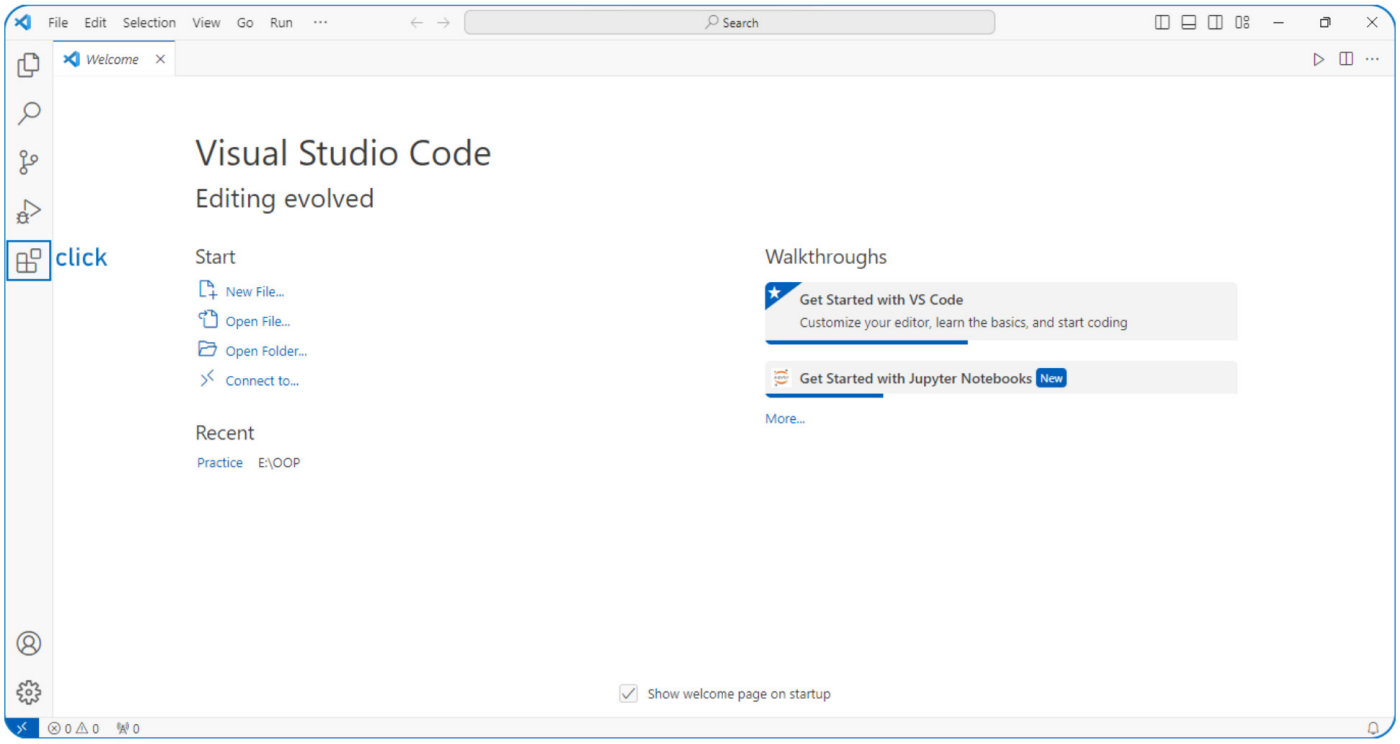


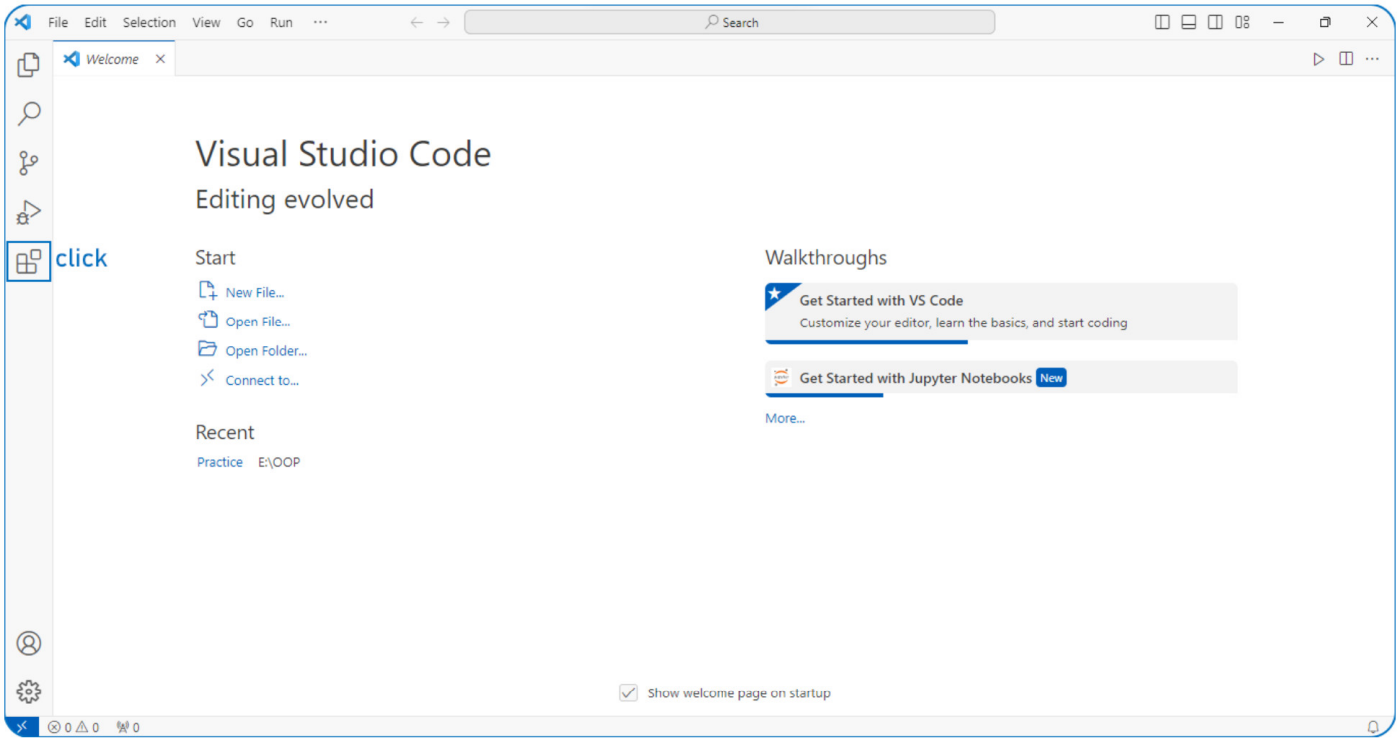
13. Select **`Path`** and then click on **`Edit.`**
14. Click on **`New`** and then paste the path.



15. Click on **`OK`**.
16. **`MinGW`** is now installed.

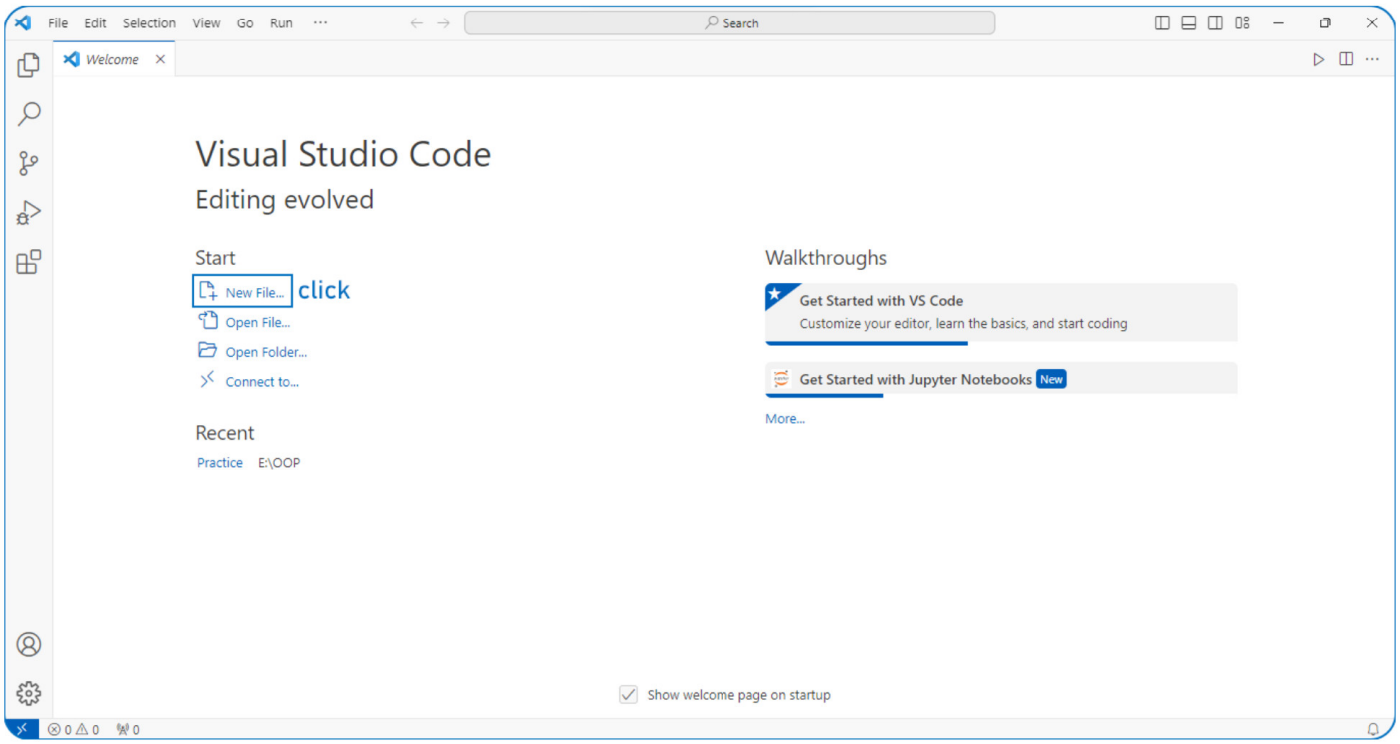
1.2. How to Install Extensions in VS Code

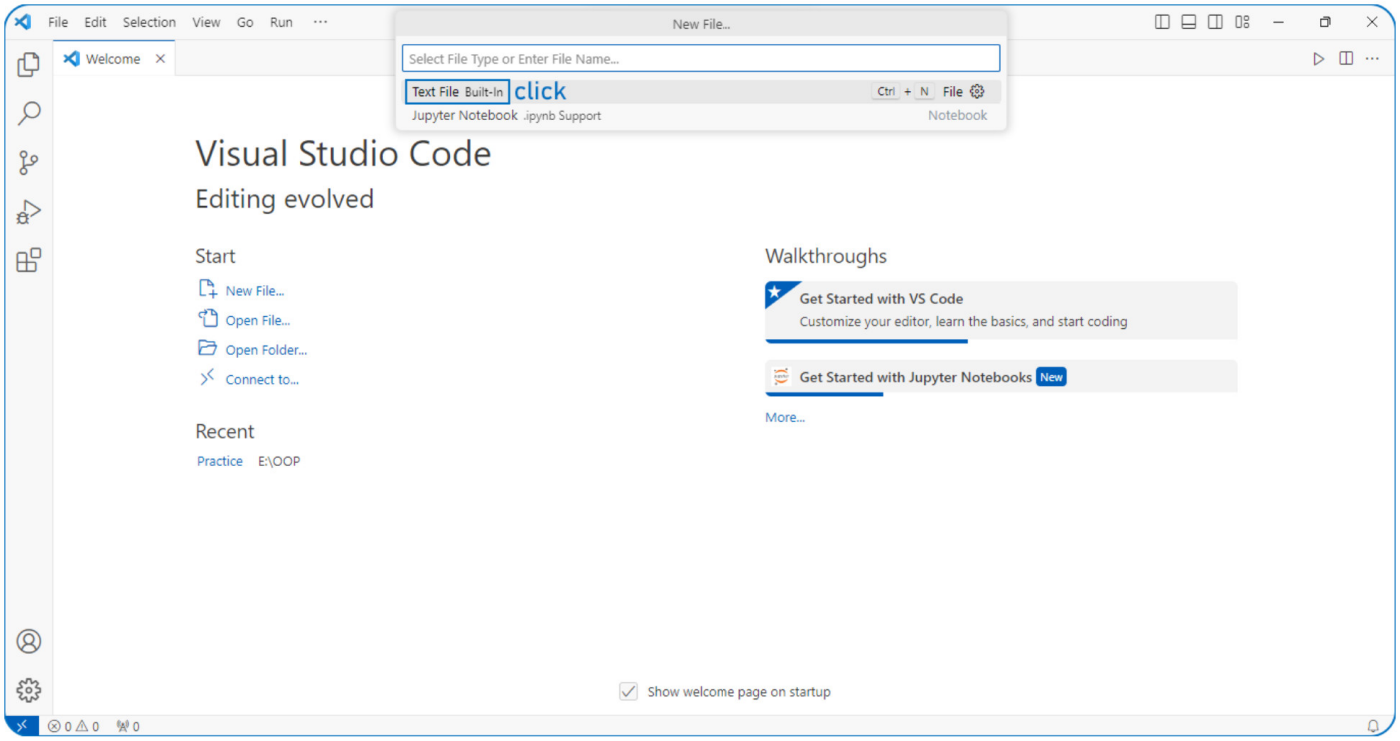


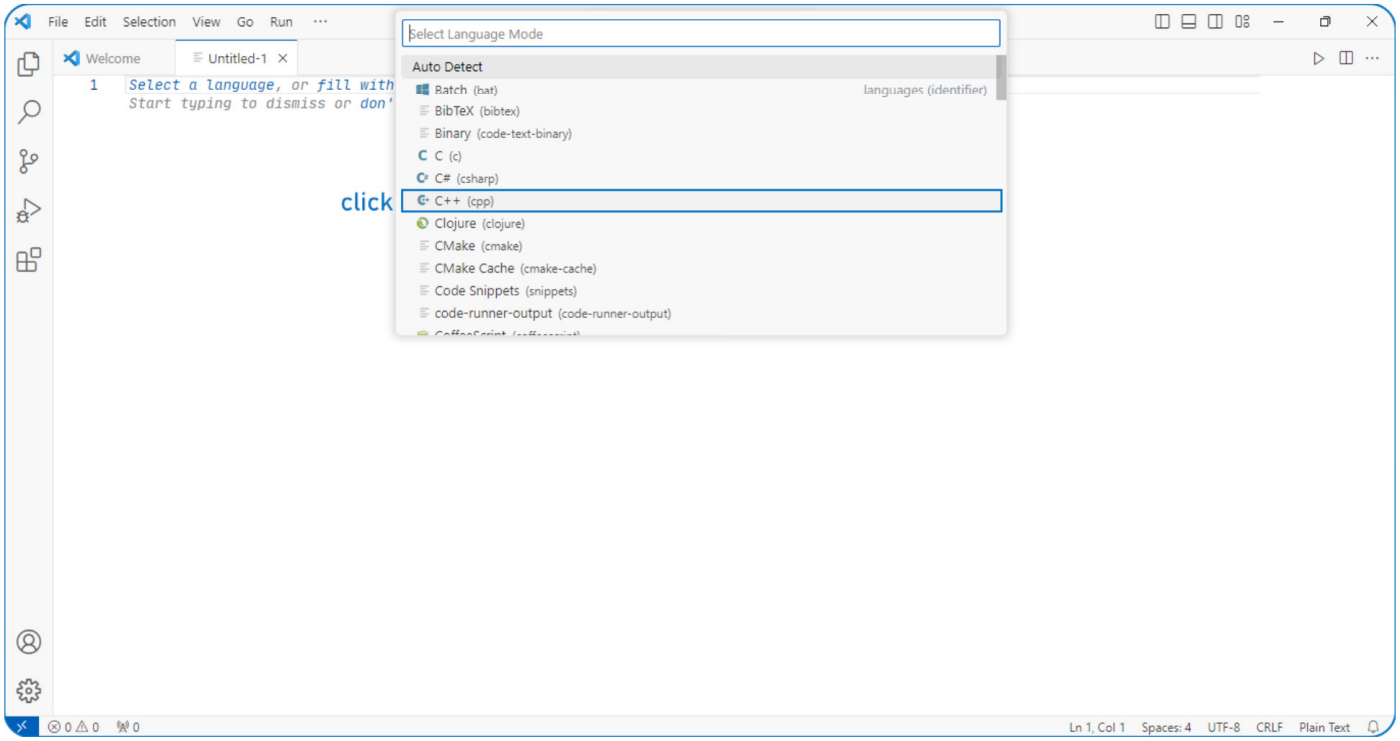


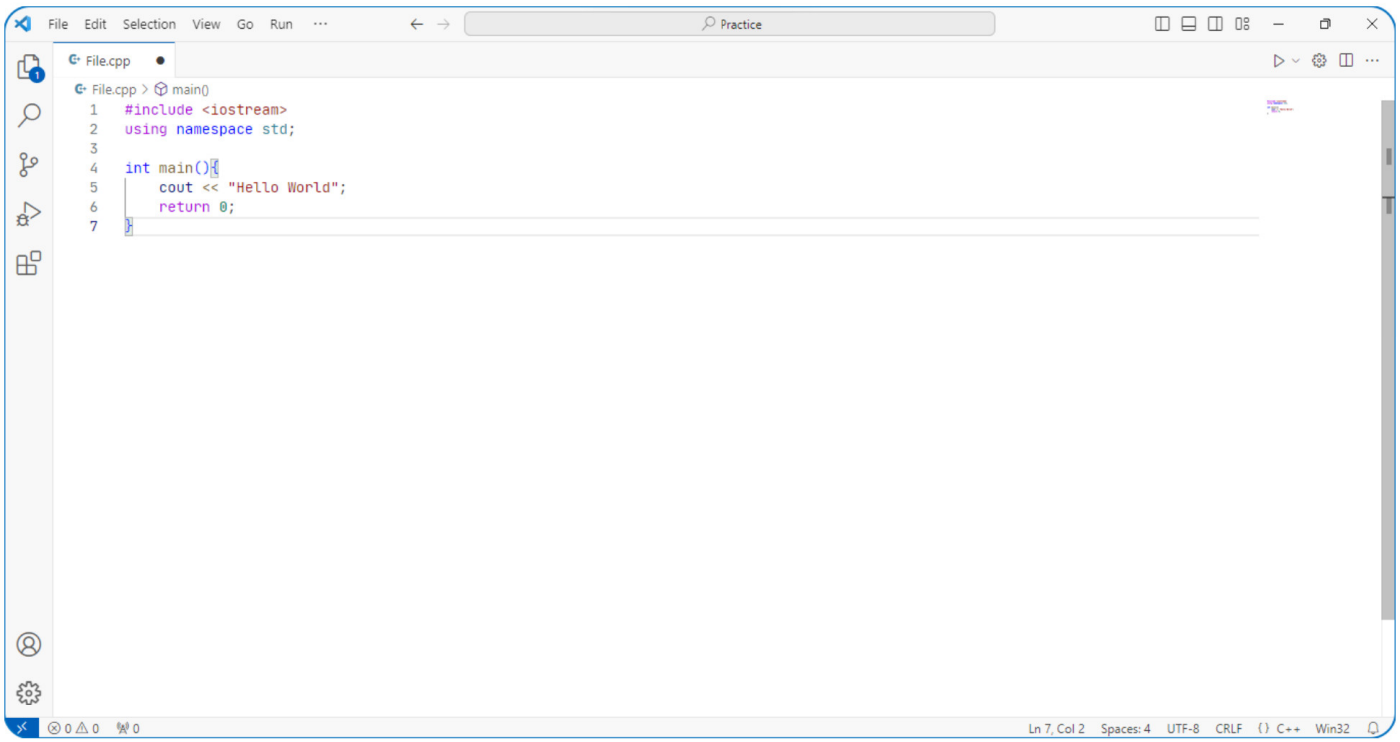
1.3. Create New Project

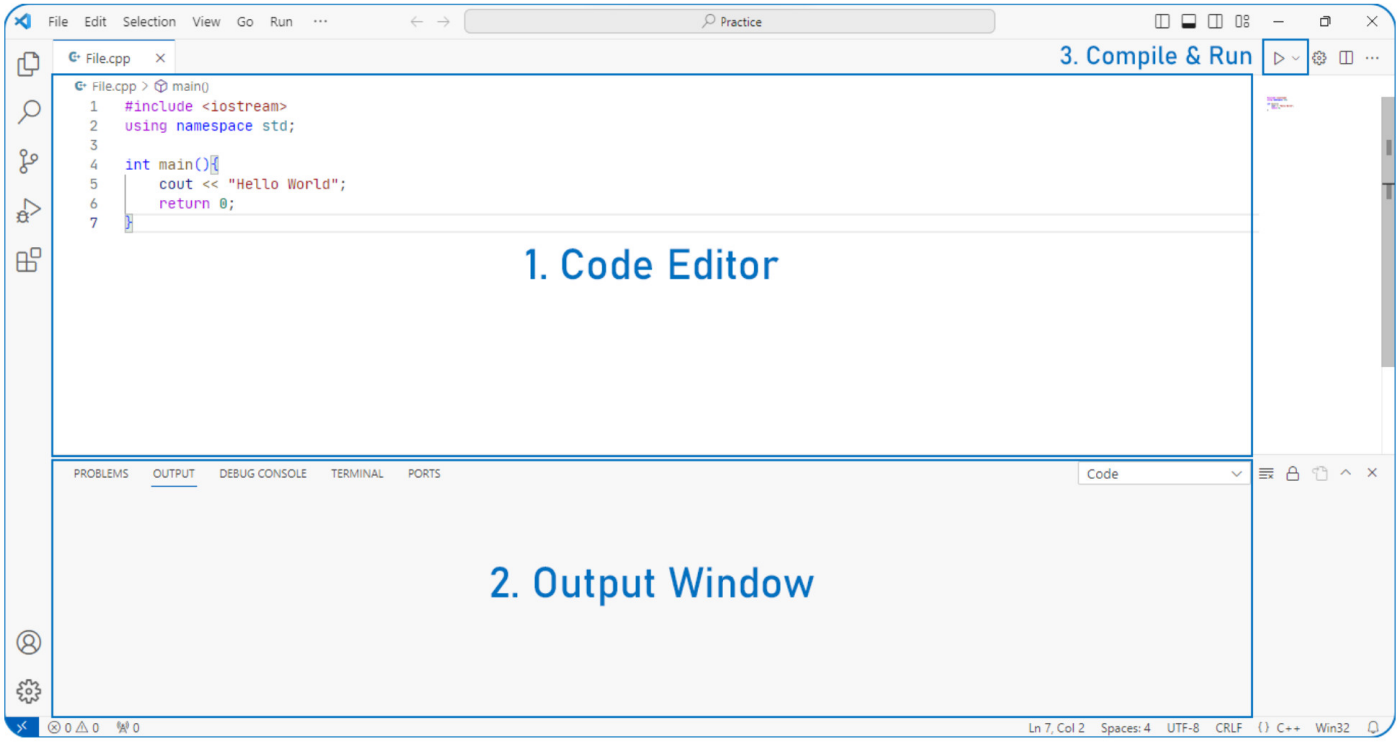
Create new and save this file with extension **.cpp** i.e., **File.cpp.**






File Edit Selection View Go Run
Welcome
Untitled-1
Select a language, or fill with
Start typing to dismiss or don'
click
Select Language Mode
Auto Detect
Batch (bat)
languages (identifier)
BibTeX (bibtex)
Binary (code-text-binary)
C (c)
C# (csharp)
C++ (cpp)
Clojure (clojure)
CMake (cmake)
CMake Cache (cmake-cache)
Code Snippets (snippets)
code-runner-output (code-runner-output)
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text


File Edit Selection View Go Run
Practice
File.cpp
File.cpp > main()
#include <iostream>
using namespace std;
int main(){
cout << "Hello World";
return 0;
}
Ln 7, Col 2 Spaces: 4 UTF-8 CRLF C++ Win32



- **`Code Editor`**: Where the user will write code.
- **`Output Window:`** Here the Visual Studio shows the outputs, compiler warnings, error messages and debugging information.

2. Skeleton of C++ Program

A C++ program is structured in a specific and particular manner. In C++, a program is divided into the following three sections:

1. Standard Libraries Section
2. Main Function Section
3. Function Body Section

For example, let's look at the implementation of the `Hello World` program:

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

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

2.1. Standard Libraries Section

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

- `#include` is a specific preprocessor command that effectively copies and pastes the entire text of the file, specified between the angle brackets, into the source code.
- The file `<iostream>`, which is a standard file that should come with the C++ compiler, is short for input-output streams. This command contains code for displaying and getting an input from the user.
- `namespace` is a prefix that is applied to all the names in a certain set. iostream file defines two names used in this program - `cout` and `endl`.

2.2. `main` Function Section

```
int main(){}
```

- The starting point of all C++ programs is the `main` function.
- This function is called by the operating system when your program is executed by the computer.
- `{` signifies the start of a block of code, and `}` signifies the end.

3. Basic I/O in C++

3.1. Input/Output in C++

- C++ is very similar to the C Language.
- For the input/output stream we use `<iostream>` library (in C it was `<stdio>`).
- For taking input and out we use `cout` and `cin` (in C it was `printf` and `scanf`).
  - `cout` uses insertion (`<<`) operator.
  - `cin` uses extraction (`>>`) operator.

Sample C++ Code

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

int main() {
```

```
    int var = 0;
    cout << "Enter an Integer value: ";
    cin >>var;
    return 0;

}
```

Sample Run	Enter an Integer value: 12

4. Variables & Data Types in C++

While writing a program in any language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

4.1 Primitive Built-in Types

C++ offers the programmer a rich assortment of built-in as well as user defined data types.

Following table lists down seven basic C++ data types:

Type	Keyword
Boolean	`bool`
Character	`char`
Integer	`int`
Floating Point	`float`
Double Floating Point	`double`
Valueless	`void`
Wide Character	`wchar_t`

Several of the basic types can be modified using one or more of these type modifiers:

- **`signed`**
- **`unsigned`**
- **`short`**
- **`long`**

The following table shows the variable type, how much memory it takes to store the value in memory, and what is maximum and minimum value which can be stored in such type of variables.

Type	Typical Bit Width	Typical Range
char	1 byte	-127 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-127 to 127
int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
signed int	4 bytes	-2147483648 to 2147483647
short int	2 bytes	-32768 to 32767
unsigned short int	2 bytes	0 to 65,535
signed short int	2 bytes	-32768 to 32767
long int	8 bytes	-9223372036854775808 to 9223372036854775807
signed long int	8 bytes	same as long int
unsigned long int	8 bytes	0 to 18446744073709551615
long long int	8 bytes	(-2^{63}) to $(2^{63}-1)$
unsigned long long int	8 bytes	0 to 18,446,744,073,709,551,615
float	4 bytes	6-7 decimal digits precision
double	8 bytes	15 decimal digits precision
long double	12 bytes	
wchar_t	2 or 4 bytes	1 wide character

The size of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example, which will produce correct size of various data types on your computer.

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

int main() {
    cout << "Size of char: " << sizeof(char) << endl;
    cout << "Size of int: " << sizeof(int) << endl;
    cout << "Size of short int: " << sizeof(short int) << endl;
    cout << "Size of long int: " << sizeof(long int) << endl;
    cout << "Size of float: " << sizeof(float) << endl;
    cout << "Size of double: " << sizeof(double) << endl;
    cout << "Size of wchar_t: " << sizeof(wchar_t) << endl;
   return 0;
}
```

Sample Run

```
Size of char:          1
Size of int:           4
Size of short int:     2
Size of long int:      4
Size of float:         4
Size of double:        8
Size of wchar_t:       2
```

This example uses **endl**, which inserts a new-line character after every line and **<<** operator is being used to pass multiple values out to the screen. We are also using **sizeof()** operator to get size of various data types.

Following is another example:

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

int main() {
    cout << "Int Min: " << numeric_limits<int>::min() << endl;
    cout << "Int Max: " << numeric_limits<int>::max() << endl;
    cout << "Unsigned Int Min: " << numeric_limits<unsigned int>::min() << endl;
    cout << "Unsigned Int Max: " << numeric_limits<unsigned int>::max() << endl;
    cout << "Long Int Min: " << numeric_limits<long int>::min() << endl;
    cout << "Long Int Max: " << numeric_limits<long int>::max() << endl;
    cout << "Unsigned Long Int Min: " << numeric_limits<unsigned  long
int>::min() <<endl;
    cout << "Unsigned Long Int Max: " << numeric_limits<unsigned  long
int>::max() << endl;
}
```

Sample Run	Int Min: -2147483648 Int Max: 2147483647 Unsigned Int Min: 0 Unsigned Int Max: 4294967295 Long Int Min: -2147483648 Long Int Max: 2147483647 Unsigned Long Int Min: 0 Unsigned Long Int Max: 4294967295

4.1.1. `typedef` Declarations

You can create a new name for an existing type using typedef. Following is the simple syntax to define a new type using **`typedef`**.

```
typedef type newname;
```

For example, the following tells the compiler that feet are another name for **`int`**.

```
typedef int feet;
```

Now, the following declaration is perfectly legal and creates an integer variable called **`distance`**.

```
feet distance;
```

5. C++ Functions

A function is a block of code that performs a specific task.

Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:

1. a function to draw the circle.
2. a function to color the circle.

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. There are two types of function:

1. Standard Library Functions: Predefined in C++
2. User-defined Function: Created by user.

5.1. C++ User-defined Function

C++ allows the programmer to define their own function.

A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).

When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

5.1.1. C++ Function Declaration

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

```
// function declaration
void greet() {
    cout << "Hello World";
}
```

Here, the name of the function is **`greet()`** the return type of the function is **`void`** the empty parentheses mean it doesn't have any parameters the function body is written inside **`{}`**

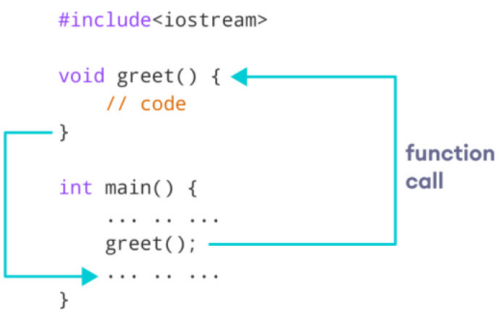
Note: We will learn about **`returnType`** and **`parameters`** later in this session.

5.1.2 Calling a Function

In the above program, we have declared a function named **`greet()`**. To use the **`greet()`** function, we need to call it.

Here's how we can call the above **`greet()`** function.

```
int main() {
    // calling a function
    greet();
}
```



5.1.3. How Function works in C++

Example 01: Display a Text

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

// declaring a function
void greet() {
    cout << "Hello there!";
}

int main() {

    // calling the function
    greet();
    return 0;
}
```

Sample Run	Hello there!

5.1.4. Function Parameters

As mentioned above, a function can be declared with parameters. A parameter is a value that is passed when declaring a function.

For example, let us consider the function below:

```
void printNum(int num) {
    cout << num;
}
```

Here, the `int` variable `num` is the function parameter.

We pass a value to the function parameter while calling the function.

```
int main() {
    int n = 7;
    // calling the function
    // n is passed to the function as argument
    printNum(n);
    return 0;
}
```

Example 02: Function with Parameters

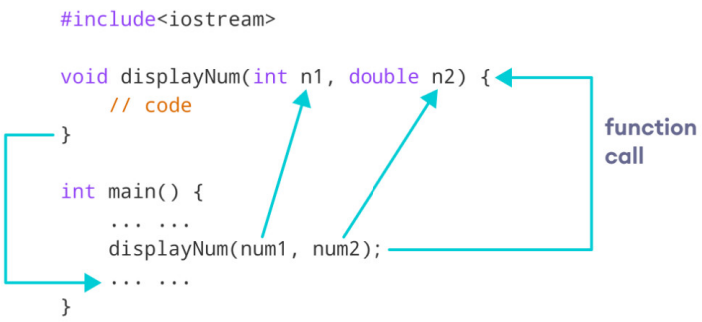
```cpp
#include <iostream>
using namespace std;

// display a number
void displayNum(int n1, float n2) {
    cout << "The int number is: " << n1 << endl;
    cout << "The double number is: " << n2;
}
int main() {
     int num1 = 5;
     double num2 = 5.5;
    // calling the function
    displayNum(num1, num2);
    return 0;
}
```

Sample Run	The int number is: 5 The double number is: 5.5

In the above program, we have used a function that has one **int** parameter and one **double** parameter.

We then pass **num1** and **num2** as arguments. These values are stored by the function parameters **n1** and **n2** respectively.



Note: The type of the arguments passed while calling the function must match with the corresponding parameters defined in the function declaration.

5.1.5 return Statement

In the above programs, we have used void in the function declaration. For example:

```cpp
void displayNumber() {
    // code
}
```

This means the function is not returning any value.

It's also possible to `return` a value from a function. For this, we need to specify the `returnType` of the function during function declaration.

Then, the `return` statement can be used to `return` a value from a function.

For example:

```
int add (int a, int b) {
   return (a + b);
}
```

Here, we have the data type `int` instead of `void`. This means that the function returns an `int` value.

The code return `(a + b)`; returns the sum of the two parameters as the function value.

The `return` statement denotes that the function has ended. Any code after return inside the function is not executed.

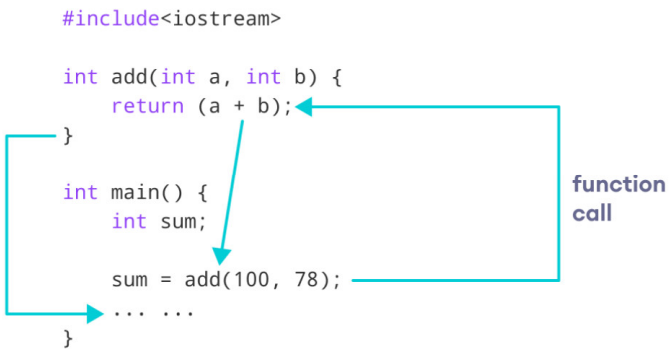
Example 03: Add Two Numbers

```
// program to add two numbers using a function
#include <iostream>
using namespace std;

// declaring a function
int add(int a, int b) {
    return (a + b);
}
int main() {
    int sum;
    int n1 = 100;
    int n2 = 78;
    // calling the function and storing
    // the returned value in sum
    sum = add(n1, n2);
    cout << "Sum = " << n1 << " + " << n2 << " = " << sum << endl;
    return 0;
}
```

Sample Run	Sum = 100 + 78 = 178

- In the above program, the `add()` function is used to find the sum of two numbers.
- We pass two int values, `100(n1)` and `78(n2)` while calling the function.
- We store the returned value of the function in the variable sum, and then we print it.



Notice that **sum** is a variable of **int** type. This is because the **return** value of **add()** is of **int** type.

5.1.6. Function Prototype

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example:

```
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.

The syntax of a function prototype is:

```
returnType functionName(dataType1, dataType2, ...);
```

Example 04: C++ Function Prototype

```cpp
// using function definition after main() function
// function prototype is declared before main()

#include <iostream>
using namespace std;

// function prototype
int add(int, int);

int main() {
    int sum;
    int n1 = 100;
    int n2 = 78;

    // calling the function and storing
    // the returned value in sum
    sum = add(n1, n2);
    cout << "Sum = " << n1 << " + " << n2 << " = " << sum << endl;
    return 0;
}

// function definition
int add(int a, int b) {
    return (a + b);
}
```

Sample Run	`Sum = 100 + 78 = 178`

The above program is nearly identical to Example 3. The only difference is that here, the function is defined **after** the function call.

That's why we have used a function prototype in this example.

Benefits of Using User-Defined Functions

- Functions make the code reusable. We can declare them once and use them multiple times.
- Functions make the program easier as each small task is divided into a function.
- Functions increase readability.

5.1.7. C++ Library Functions

- Library functions are the built-in functions in C++ programming.
- Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.
- Some common library functions in C++ are **`sqrt()`**, **`abs()`**, **`isdigit()`**, etc.

- In order to use library functions, we usually need to include the header file in which these library functions are defined.
- For instance, in order to use mathematical functions such as **sqrt()** and **abs()**, we need to include the header file **cmath**.

Example 05: C++ Program to find the Square Root of a number

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

int main() {
    double number, squareRoot;

    number = 25.0;

    // sqrt() is a library function to calculate the square root
    squareRoot = sqrt(number);

    cout << "Square root of " << number << " = " << squareRoot;

    return 0;
}
```

Sample Run	Square root of 25 = 5

In this program, the **sqrt()** library function is used to calculate the square root of a number.

The function declaration of **sqrt()** is defined in the **cmath** header file. That's why we need to use the code **#include <cmath>.**

6. 1D and 2D Array

6.1. Arrays

An Array is a collection of fixed number of elements of same data type.

6.1.1. 1-D Arrays

- 1-D Array is a form of array in which elements are arranged in a form of List.
- To declare a 1D array you need to specify the data type, name and array size.

```
dataType arrayName [ arraySize ] ;
```

- Following is the declaration of a 1D array.

```
int numArray[5];
```

where,

Data Type: Integers (`int`)

Array Name: `numArray`

Array Size: `5`

- To access array elements, you use the array name along with the index in subscript operator "`[ ]`".

```
numArray[0], numArray[1], numArray[2], numArray[3], numArray[4]
```

where,

Index of the array starts with zero '`0`'.

Index of the last element is always '`size - 1`' (in this case it is `4`).

Example Code for 1-D Array

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

int main() {
    int item[5]; // Declare an array 'item' of five components
    int sum = 0;
    int counter;
    cout << "Enter five numbers: ";

    // Input five numbers into the array and calculate the sum
    for (counter = 0; counter < 5; counter++) {
        cin >> item[counter];
        sum = sum + item[counter];
    }

    cout << endl;
    cout << "The sum of the numbers is: " << sum << endl;

    cout << "The numbers in reverse order are: ";

    // Print the numbers in reverse order
    for (counter = 4; counter >= 0; counter--)
        cout << item[counter] << " ";

    cout << endl;

    return 0;
}
```

Sample Run	Enter five number: 12 76 34 52 89 The sum of the numbers is: 263 The numbers in reverse order are: 89 52 34 76 12

6.1.2. 2-D Arrays

- 2-D Array is a collection of fixed collection of elements arranged in rows and columns.
- To declare a 2D array you need to specify the data type, name and no. of rows and columns.

```
dataType arrayName [ rowSize ][ columnSize ] ;
```

- Following is the declaration of a 2D array.

```
int numArray[5][5];
```

- To access array element you use the array name along with the row Index and column Index in subscript operator "`[ ][ ]`".

```
numArray[0][0], numArray[1][1], numArray[2][2], numArray[3][3],
numArray[4][4]
```

where,

Index for the rows and columns of the array starts with zero '`0`'.

Index of the last element in rows and columns is always '`sizeofRow - 1`' and

'`sizeofColumn -1`' respectively (in this case it is `4`).

Example Code for 2-D Array:

Program to read a 2D array of size 3x3 find the sum for each row, print the sum line by line.

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

int main() {
    int item[3][3]; //declare an array of size 3x3
    int sum = 0;
    int row, col;
    cout << "Enter array elements: " << endl;
    for (row = 0; row < 3; row++) {
        for (col = 0; col < 3; col++) {
            cin >> item[row][col];
            sum = sum + item[row][col];
            }
        cout << "The sum of row " << i << " : " << sum << endl;
        }
    cout << endl; return 0;
```

```
}
```

Sample Run	Enter array elements: 12 76 34 The sum of row 0 : 122 52 89 48 The sum of row 1 : 189 22 63 99 The sum of row 2 : 184