Fundamentals of Computer Programming

TICT 1134

K.D.I.Karunathilaka
2020/ICTS/140

Content…..

1. Computer Programming

What’s a Computer?

A computer is a general purpose machines which can perform many computational task and the modern day computers that we have they can perform billions or trillions of calculations within a fraction of second.

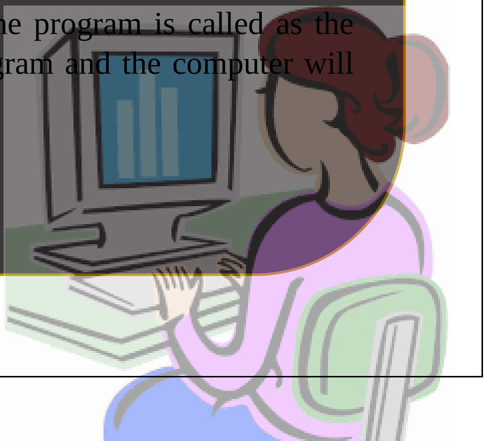
What’s a Computer Program?

- These computer’s they can’t really do anything on their own.
- So for a computer to do something one has to give the instructions to it and these instructions will contain step by information to perform a specific task and these are called as program
- **Programming** is the process of creating a set of instruction that tell a computer how to perform a task

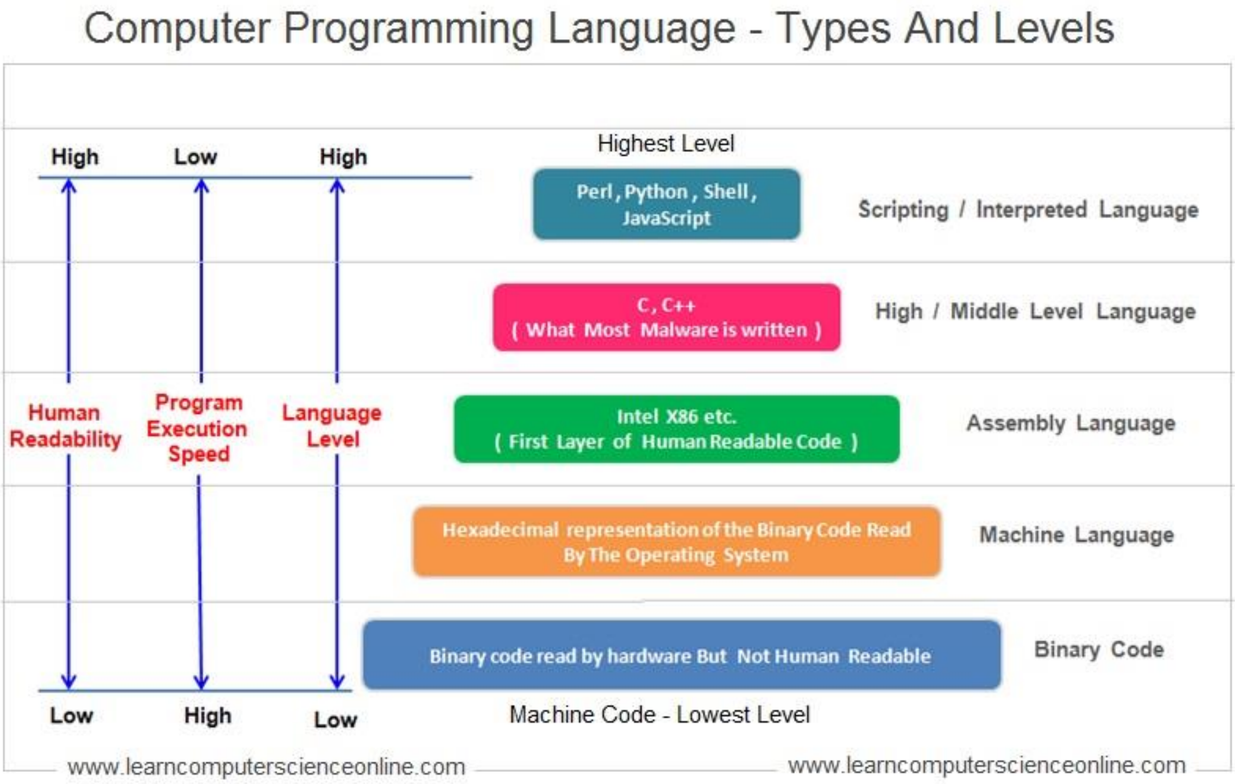
- **Programming** can be done using a variety of computer programming languages, such as JavaScript, Python and C++

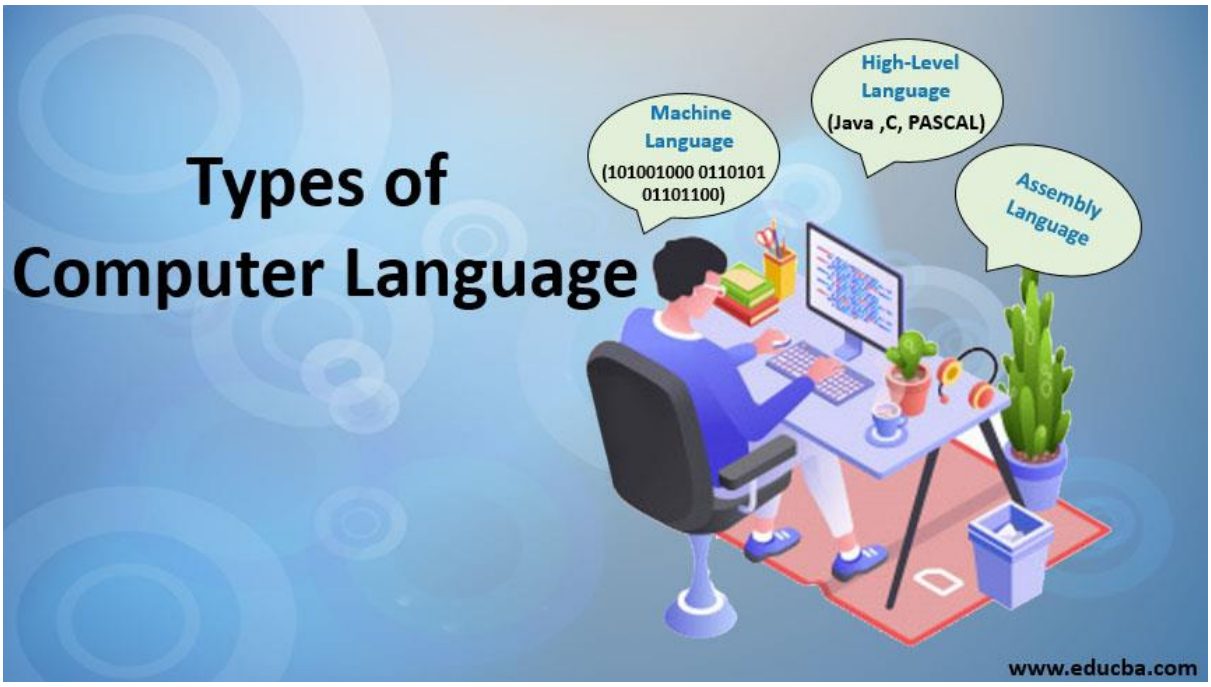
Who’s a Computer Programmer?

The person who is going to write these instructions or the program is called as the programmer and here the programmer will write the program and the computer will work on that instructions.

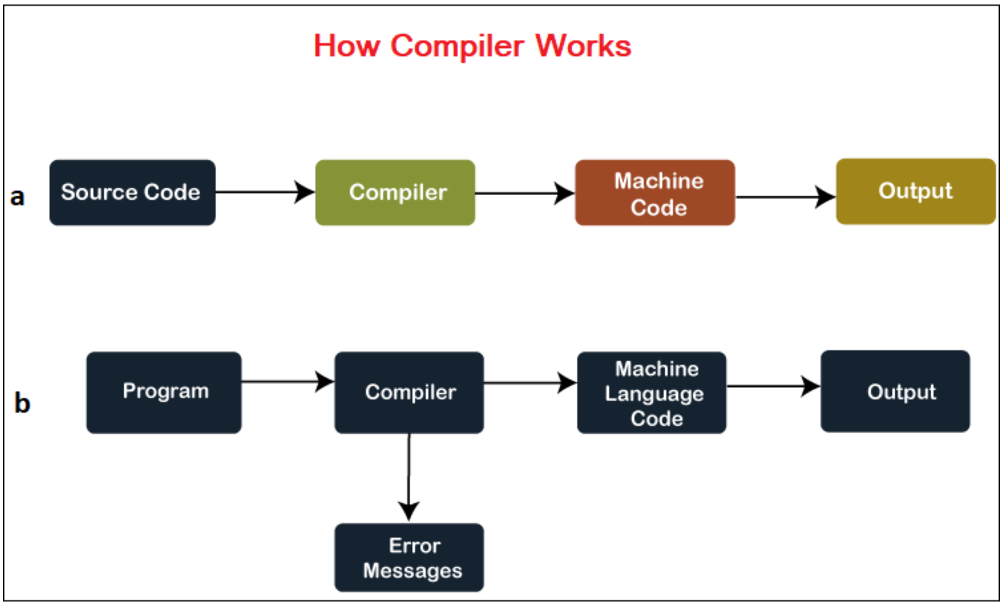


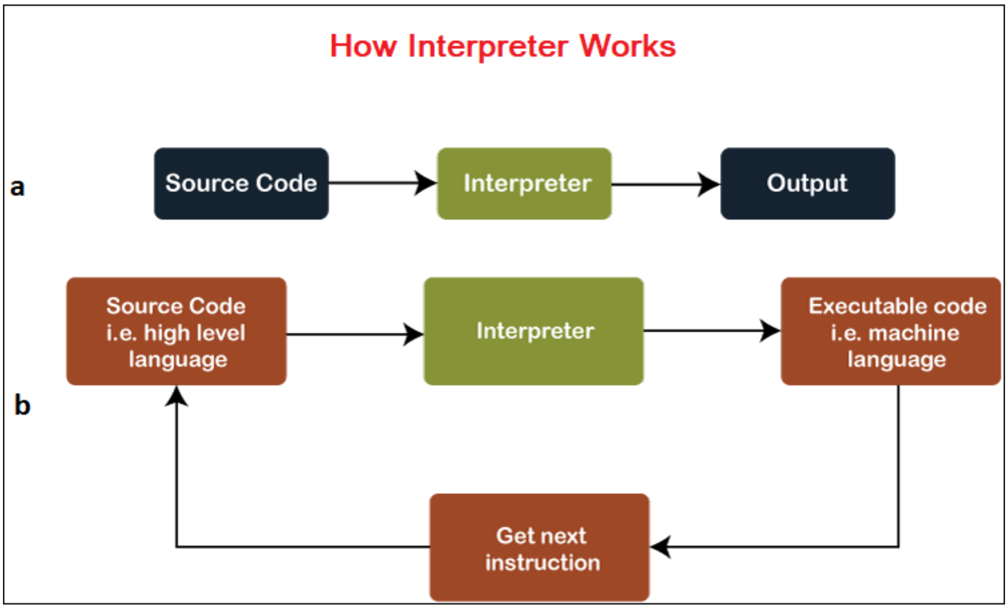
2. Types of Computer Programming


Computer Programming Language - Types And Levels
High
Low
High
Human Readability
Program Execution Speed
Language Level
Low
High
Low
Highest Level
Perl , Python , Shell , JavaScript
Scripting / Interpreted Language
C , C++ (What Most Malware is written)
High / Middle Level Language
Intel X86 etc. (First Layer of Human Readable Code)
Assembly Language
Hexadecimal representation of the Binary Code Read By The Operating System
Machine Language
Binary code read by hardware But Not Human Readable
Binary Code
Machine Code - Lowest Level
www.learncomputerscienceonline.com
www.learncomputerscienceonline.com


Types of Computer Language
Machine Language (101001000 0110101 01101100)
High-Level Language (Java ,C, PASCAL)
Assembly Language
www.educba.com

3. Compiler vs. Interpreter





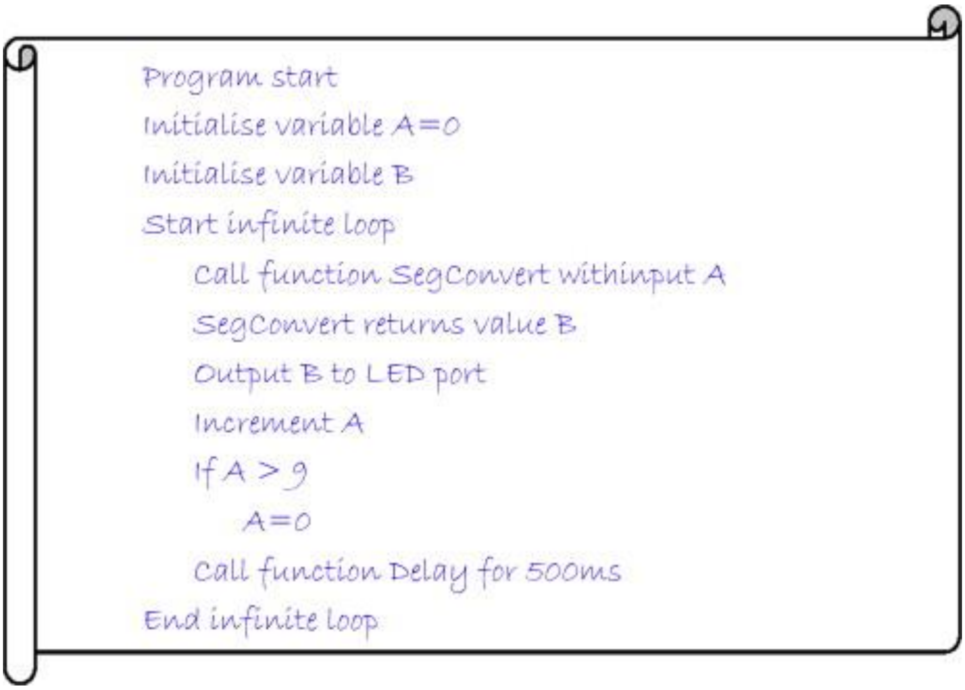
Commonly, there are two types of compilers available in computer languages. Those are compilers and interpreters. Simply Compilers compile all source code simultaneously to machine code, but interpreters compile code line by line. That is the main difference between compilers and interpreters. Most of the time, compilers covert the code to machine code much faster than interpreters. C / C++ and Java are the common examples that use compilers. Python and JavaScript are examples of interpreters.

4. Designing a Program with Pseudo code

- Pseudocode is a technique used to describe an algorithm in a way that is easy to understand for anyone with basic programming knowledge.

- Pseudocode can’t understand by computer.

- It’s used to help programmers while they are planning an algorithms.



```
Program start
Initialise variable A=0
Initialise variable B
Start infinite loop
    Call function SegConvert withinput A
    SegConvert returns value B
    Output B to LED port
    Increment A
    If A > 9
        A=0
    Call function Delay for 500ms
End infinite loop
```

Here is an example of a simple pseudo-code algorithm for finding the most significant number in a list:

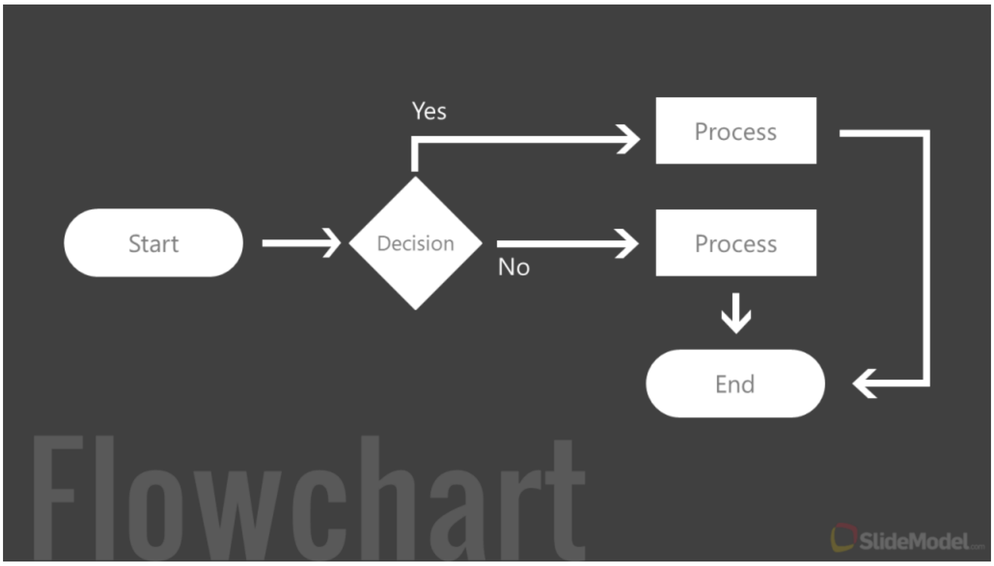
1. Initialize a variable called "largest" to the first number in the list.
2. Loop through the rest of the numbers in the list.
3. For each number in the list, compare it to the current value of "largest".
4. If the current number is more extensive than "largest", update the value of "largest" to the current number.
5. After looping through all the numbers, "largest" will contain the most significant number in the list.

Pseudo code can be written in various styles, but the key is to focus on the logic of the program rather than the specific syntax of a particular programming language.

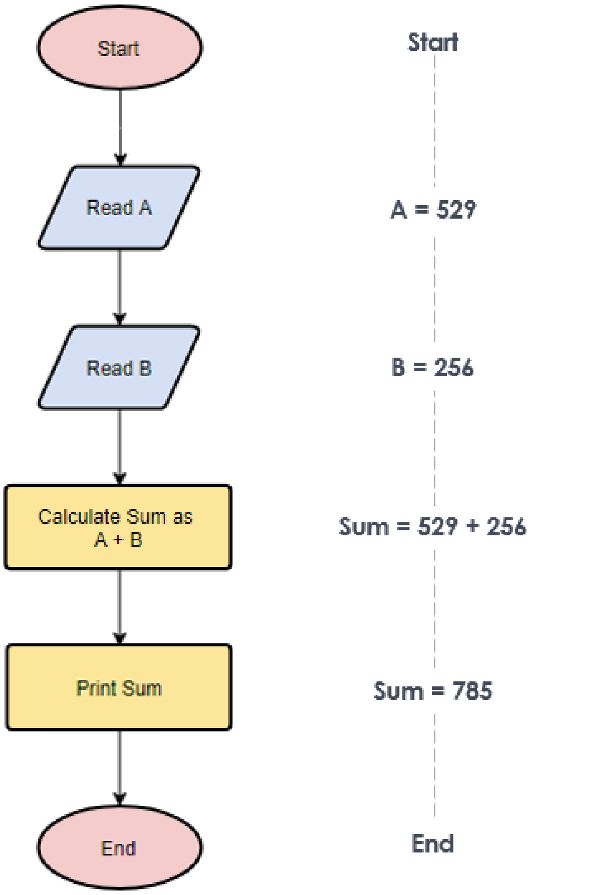
5. Flowchart

A flowchart is a diagrammatic representation of the solution to a problem.

It is a tool programmers, analysts, and other stakeholders use to understand, document, and improve processes. A flowchart comprises various symbols, each representing a specific action or decision point in the program's logic.



Find the sum of 529 and 256



6. Control /Logic Structure

All computer programs no matter how simple or complex are written using one or more of 3 basic structures

- Sequence
- Selection
- Repetition

The Sequence Structure

- We should follow step by instructions from beginning to end.

- In a computer program the sequence structure directs the computer to process the program instruction one after another in the order listed in the program.

The Selection / Decision Structure

- Based on the Conditions Indicates that a decision needs to be made, followed by an appropriate action derived from that decision.
- This based on either true or false.

The Selection / Decision Structure

- Indicates one or more instructions need to be repeated until some condition is met.
- In here we used loop or an iteration.

7. Declaring the memory location

- Reserving a memory location.
- Programmers have to assign a name, a data type and an initial value to the location.
- The name allows the programmer to refer to the memory location using one or more descriptive words.

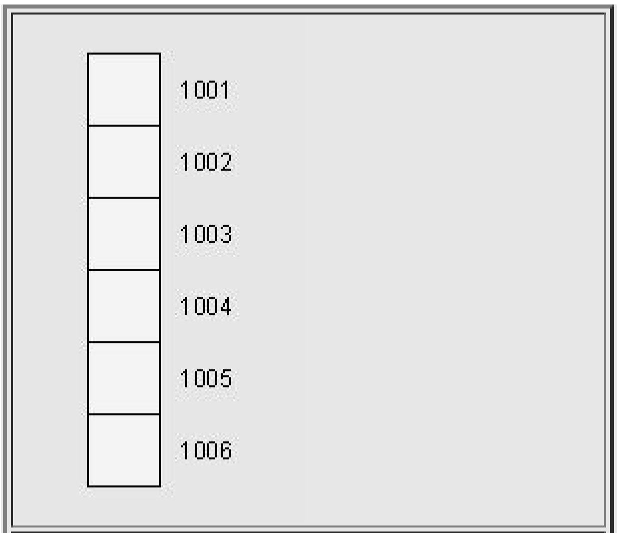


figure 1. Representation of computer memory.

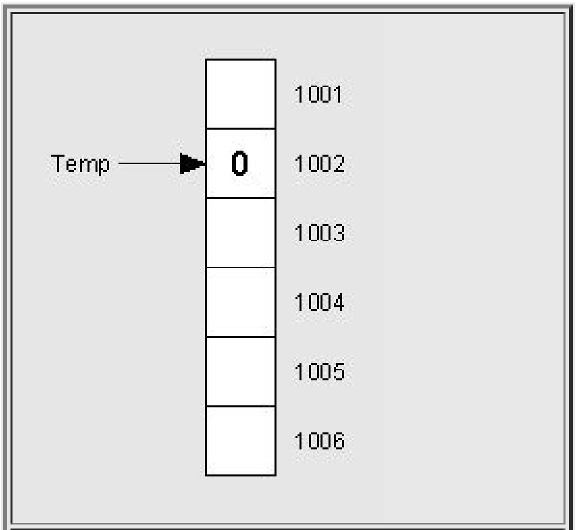


figure 2. Memory after declaring a variable.

- There are two types of memory locations that a programmer can declare
  - Variable
  - Named Constant

8. Variable vs. Constant

- Variable is a memory location whose value can change during runtime, which is when a program.
- Most of the memory locations declared in a program are variables.
- Named Constant is memory location whose value can't change during runtime.

  Ex: - Calculate an area of a circle pi - constant and radius – variables

Selecting a name for a memory location

- **Identifier** – Every memory location that a programmer declares must be assigned a name.

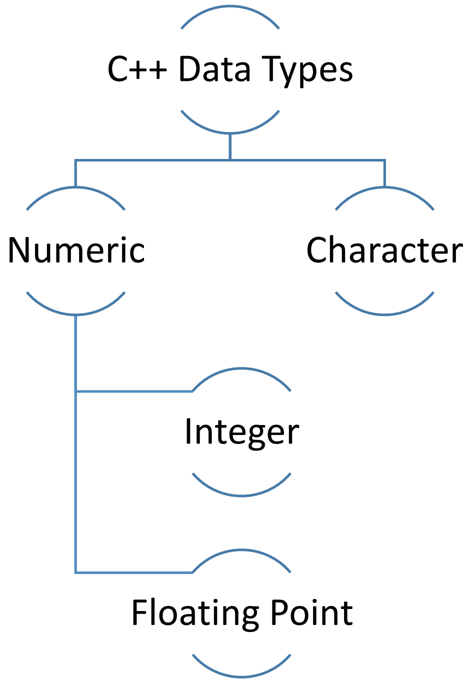
- The name must be begin with a letter.

- The name can contain only letters, Numbers and underscore characters (_).No punctuation marks, spaces or other special characters are allowed in the name.

- The name can't be a keyword.

- Names in C++ are case sensitive.

8. Data Types



In C++ mainly we have 2 data types. Numeric and Characters. Also we can categorize Numeric data types to floating points and integers. Also we can categorize these as follows.

1. Integer types: C++ supports various integer types, such as int, short which are used to store whole numbers. The size of these types varies depending on the platform and compiler.
2. Floating-point types: C++ supports float, double which are used to store decimal numbers with varying levels of precision.
3. Character types: C++ supports char which store single characters.
4. Boolean type: C++ supports bool, which stores logical values of true or false.
5. Invalid type: C++ supports void, which represents the absence of a value. It is commonly used as a return type for functions that do not return a value.
6. Enumerated types: C++ supports enumerations, which define a set of named constants. Enumerations help improve code readability and maintainability.

9. C++ programming language and its structure

Developing system, application, and games with C++ is a popular choice due to its powerful capabilities as an object-oriented programming language. Originating in the early 1980s, Bjarne Stroustrup extended the C programming language to create the language known for its impressive performance, ability to execute low-level control, and support for varying programming paradigms including procedural and object-oriented methods.

C++ has a variety of everyday applications, which include:

1. System programming: C++ is used to develop operating systems, device drivers, and other low-level system software.
2. Application software: C++ is used to develop large-scale applications, such as financial, multimedia, and scientific applications.
3. Games: C++ used in the game development industry for its performance and low-level control.
4. Libraries: C++ to develop software libraries that other programmers can use to build applications.

The structure of a C++ computer program generally follows these basic components:

1. Preprocessor directives: This is where the program includes header files, which contain predefined functions and variables used in the program.
2. Global declarations: This is where the program declares global variables and functions that can used throughout the program.
3. Main function: This is the starting of the program, where the program execution begins.
4. Functions: These are blocks of code that perform specific tasks, which can asked from other program parts.
5. Statements: These are individual instructions that perform specific operations, such as assignments, calculations, or control flow.
6. Comments: These are explanatory notes added by the programmer to clarify the purpose or operation of parts of the program.

C++ programs are generally written using an Integrated Development Environment (IDE) or a text editor, and then compiled into machine code that the computer can execute. The C++ compiler translates the program's code into executable machine code.

10. Numeric Data Types

Numerical data types in C++ are integers, floating-point numbers, and complex numbers.

- Integer types: represent whole numbers, such as -1, 0, 1, 2, 3, etc. C++ supports several integer types, including:

  short: 2 bytes

  int: 4 bytes

- Floating-point types: represent decimal values with varying levels of precision. C++ supports three floating-point types:

  float: 4 bytes

  double: 8 bytes

- Complex types: represent complex numbers, which have a real part and an imaginary part. C++ supports two complex types:

  complex: contains two floating-point numbers representing the real and imaginary parts

  complex long double: contains two long double floating-point numbers representing the real and imaginary parts

11. Variable Declaration and Initialization

Variable Declaration

Creating a variable, It should be declared before used it.

```
int i;
char num;
float val;
```

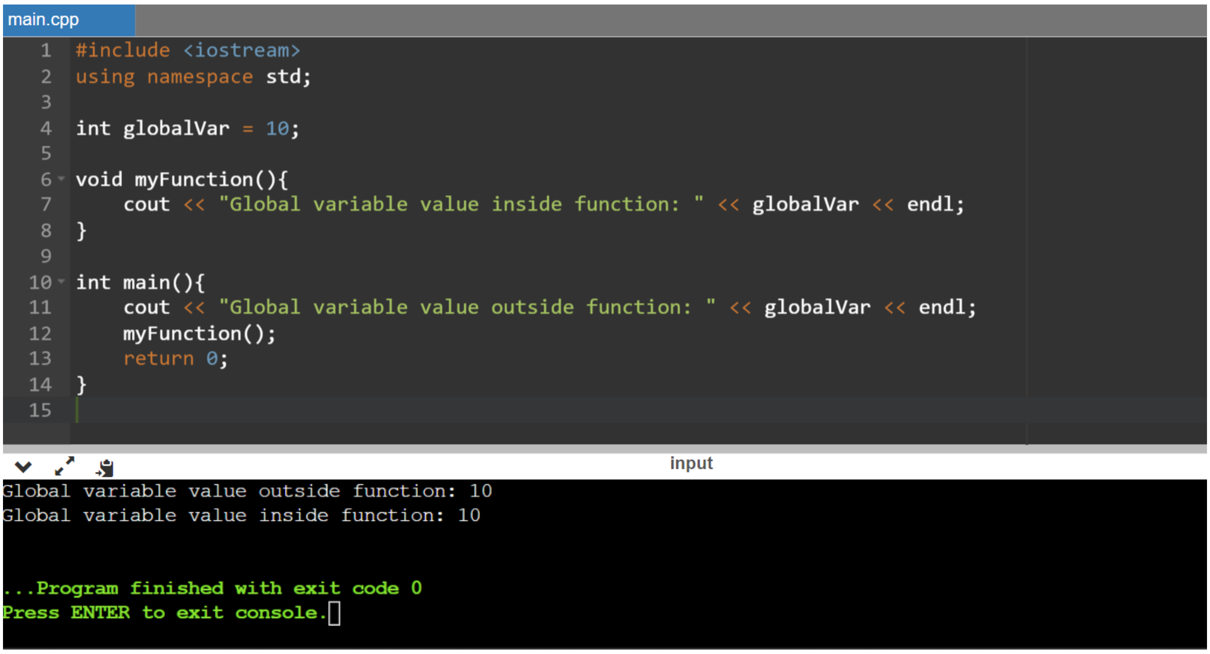
Variable Initialization

Assign a value to the variable.

```
int i = 12;
char num = ‘v’;
float val = 12.12;
```

12. Types of Variables

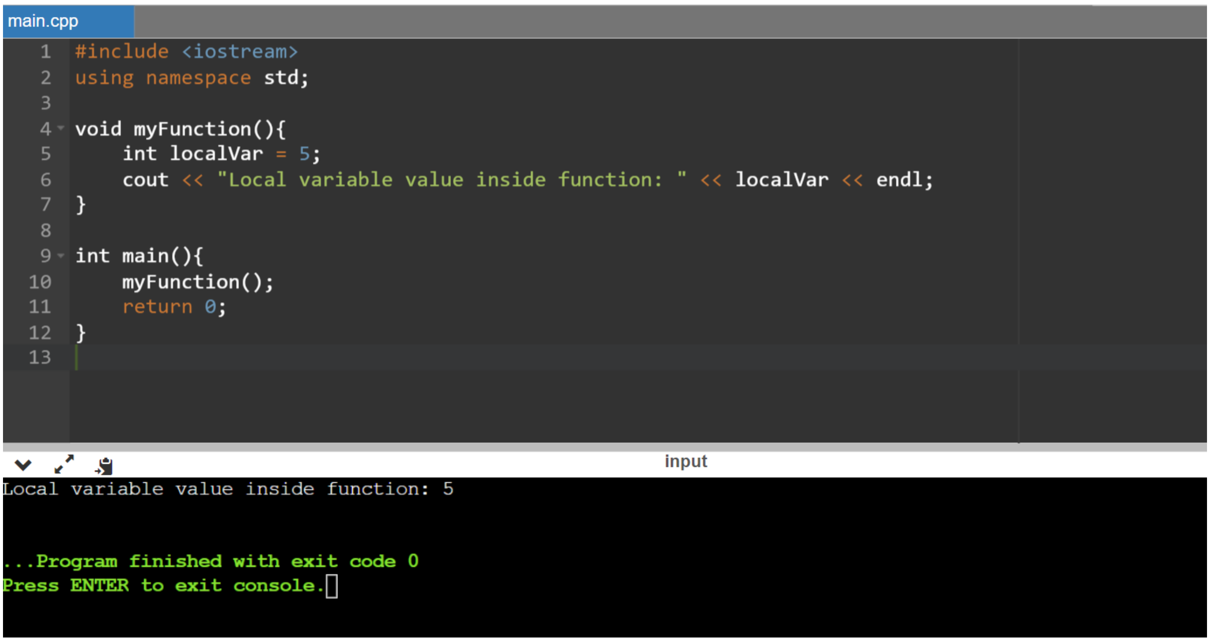
Global variables: These are declared outside of any function or block, and can be accessed and modified by any function or block within the program. Global variables have a lifetime that spans the entire execution of the program.



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

int globalVar = 10;

void myFunction(){
    cout << "Global variable value inside function: " << globalVar << endl;
}

int main(){
    cout << "Global variable value outside function: " << globalVar << endl;
    myFunction();
    return 0;
}

```

```
Global variable value outside function: 10
Global variable value inside function: 10


...Program finished with exit code 0
Press ENTER to exit console.
```

Local variables: These are declared inside a function or block, and can only be accessed and modified within that function or block. Local variables have a lifetime that spans only the execution of the function or block in which they are declared.



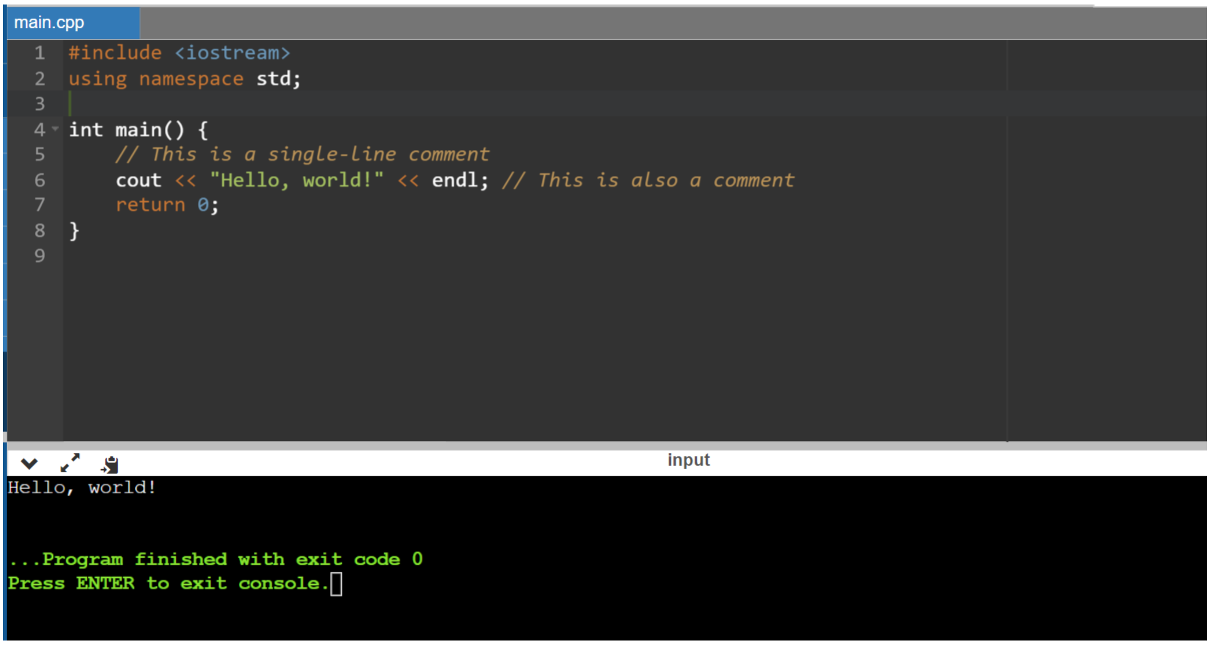
```cpp
#include <iostream>
using namespace std;

void myFunction(){
    int localVar = 5;
    cout << "Local variable value inside function: " << localVar << endl;
}

int main(){
    myFunction();
    return 0;
}

```

```
Local variable value inside function: 5


...Program finished with exit code 0
Press ENTER to exit console.
```

13. Comments

Adding extra information to a program's code is possible with C++ comments. Although ignored by the compiler, comments can make understanding a program's purpose easier, explaining complicated logic, or providing additional resources for other programmers. In the context of C++, two types of comments exist: explanatory or descriptive.

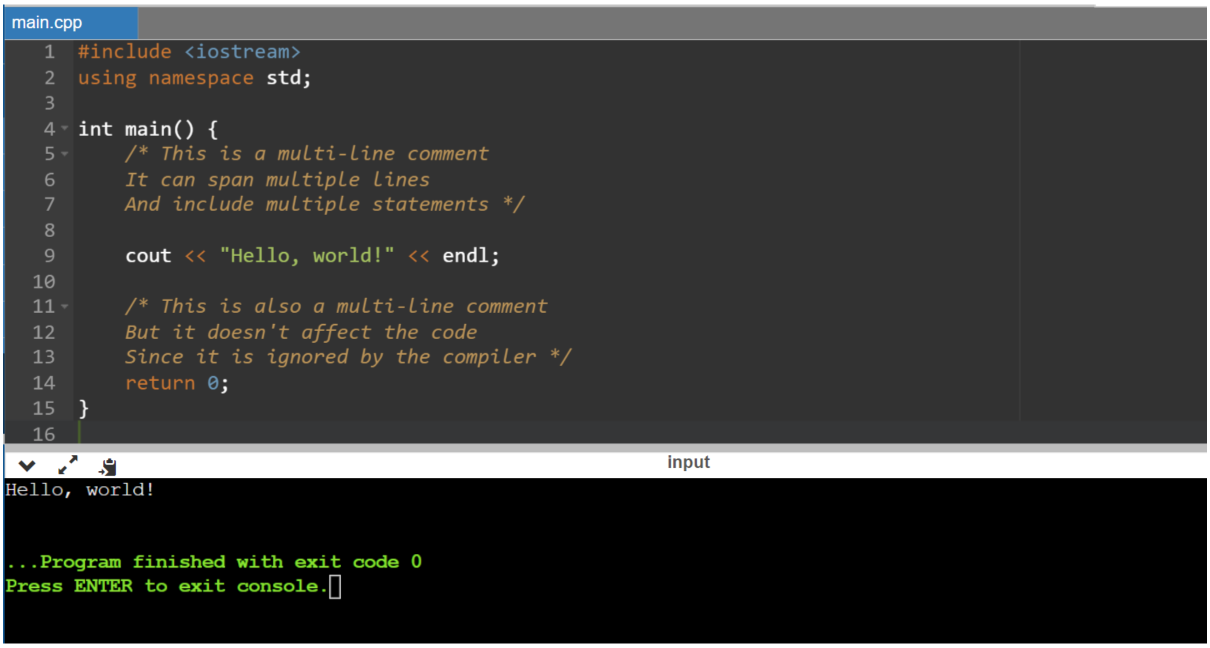
Single-line comments: These comments start with two forward slashes // and extend to the end of the line. Anything after // on the same line is ignored by the compiler.



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

int main() {
    // This is a single-line comment
    cout << "Hello, world!" << endl; // This is also a comment
    return 0;
}

```

```
Hello, world!


...Program finished with exit code 0
Press ENTER to exit console.
```

Multi-line comments: These comments start with /* and end with */. Anything between these two symbols is ignored by the compiler.

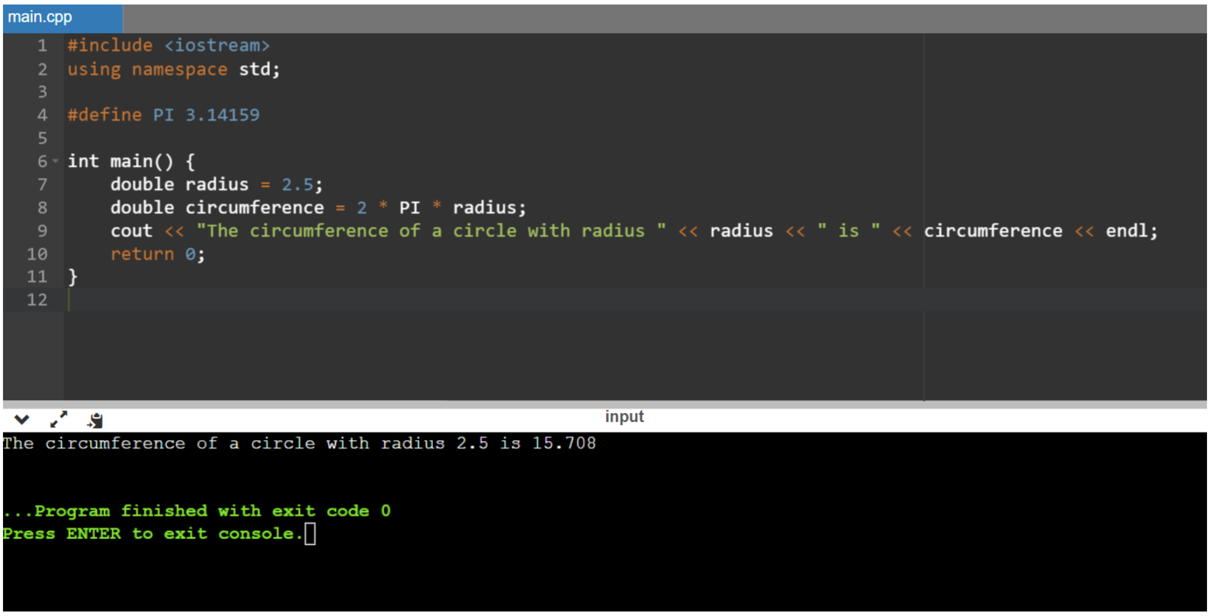


```cpp
#include <iostream>
using namespace std;

int main() {
    /* This is a multi-line comment
    It can span multiple lines
    And include multiple statements */

    cout << "Hello, world!" << endl;

    /* This is also a multi-line comment
    But it doesn't affect the code
    Since it is ignored by the compiler */
    return 0;
}

```

```
Hello, world!


...Program finished with exit code 0
Press ENTER to exit console.
```

14. Main Keywords in C++

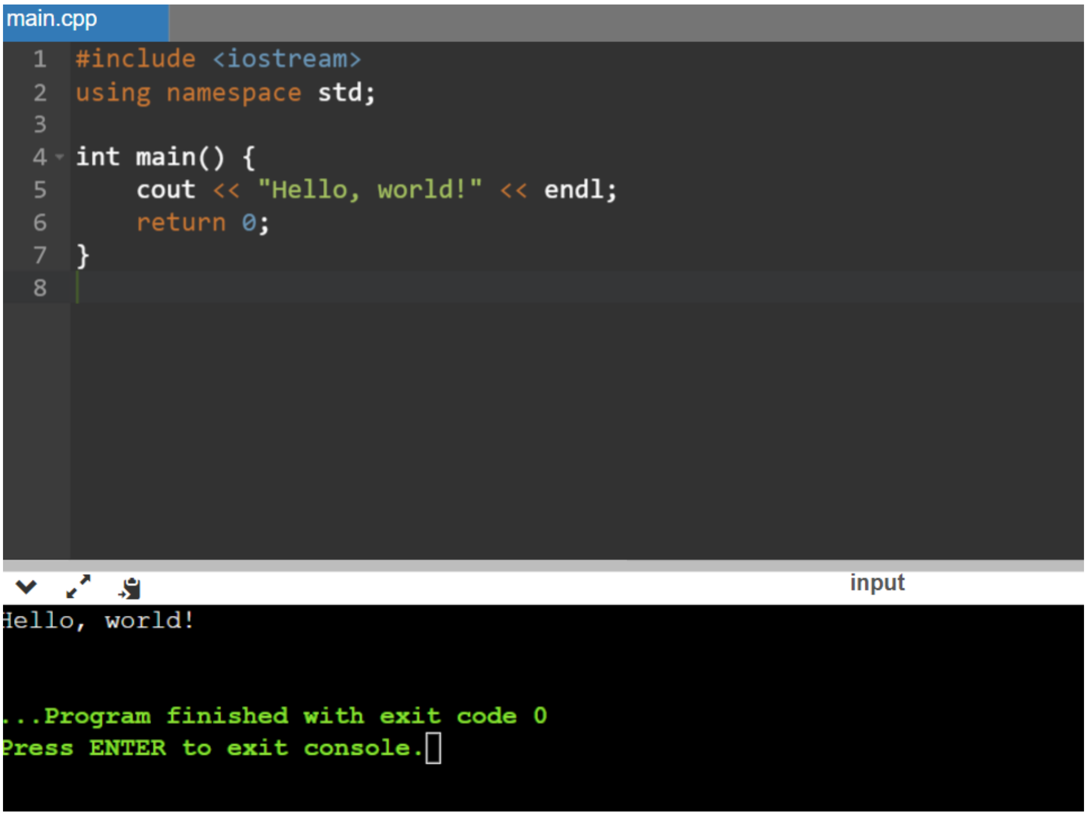
Preprocessor Directives: These are used to give instructions to the compiler about how to handle certain parts of the code. They begin with a hash symbol # and are executed before the main program starts. Examples of preprocessor directives in C++ include #include, #define, and #ifdef.



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

#define PI 3.14159

int main() {
    double radius = 2.5;
    double circumference = 2 * PI * radius;
    cout << "The circumference of a circle with radius " << radius << " is " << circumference << endl;
    return 0;
}

```

```
The circumference of a circle with radius 2.5 is 15.708


...Program finished with exit code 0
Press ENTER to exit console.
```

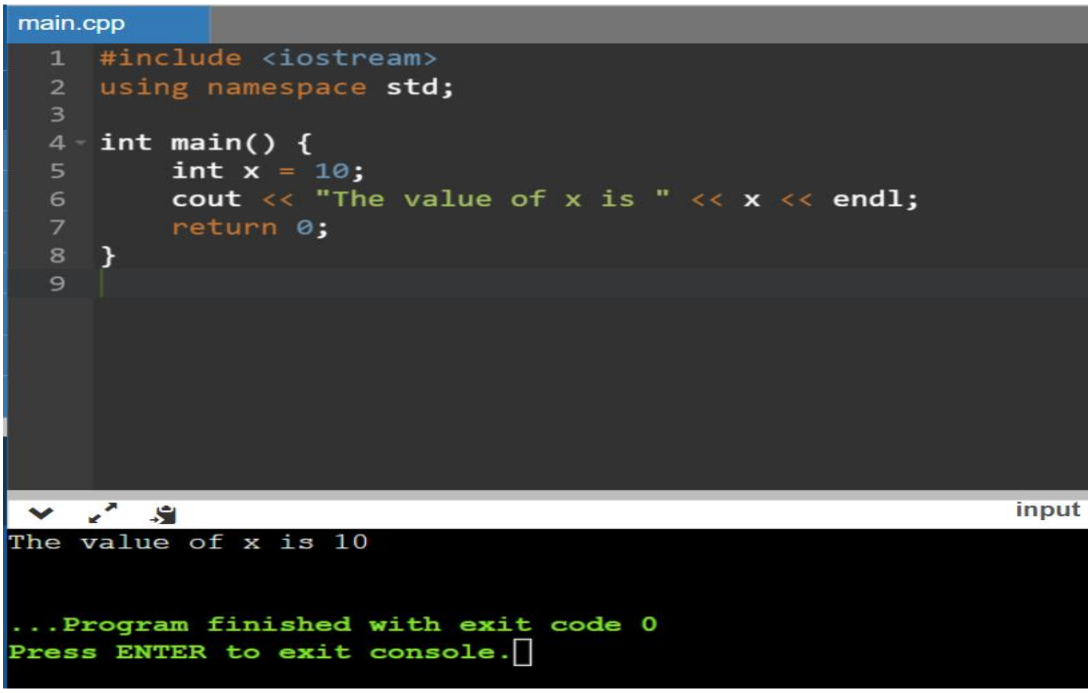
using namespace std;: This statement is used to declare that the program will use the std namespace, which contains functions and objects provided by the C++ standard library. This is often included at the top of a C++ program.



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

int main() {
    cout << "Hello, world!" << endl;
    return 0;
}

```

```
Hello, world!


...Program finished with exit code 0
Press ENTER to exit console.
```

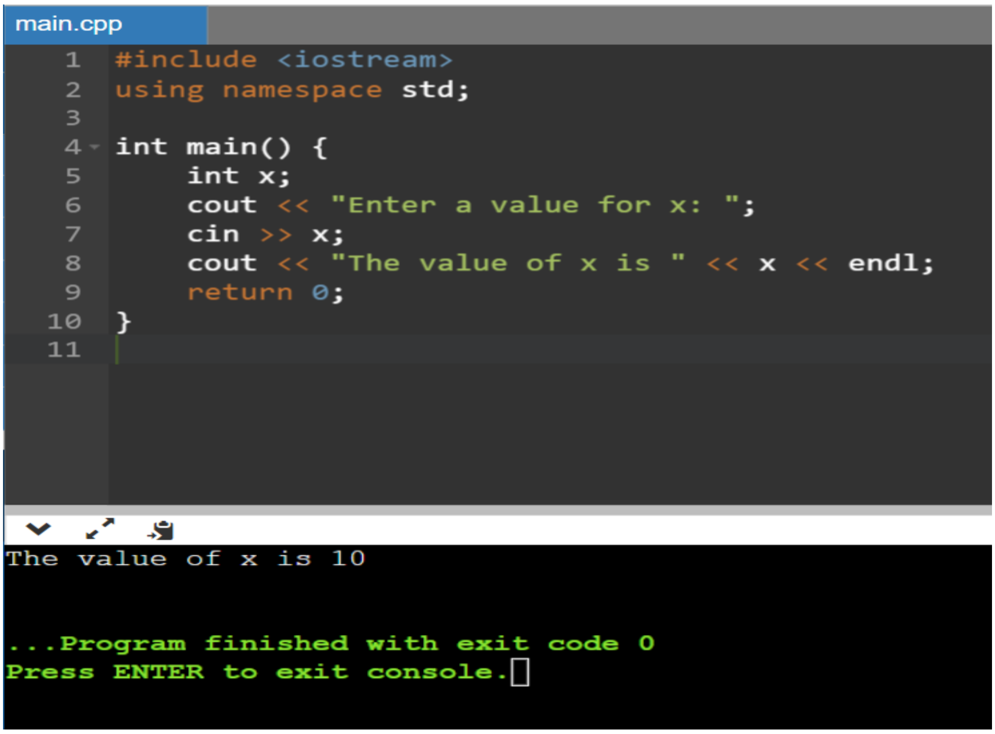
int main(): This is the main function of a C++ program. It is where the program begins execution, and must always return an integer value. The int keyword indicates that the function returns an integer, while main() is the name of the function.
cout: This is an output stream object that is used to display text on the console or terminal. It is used in combination with the insertion operator << to output text or variables.



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

int main() {
    int x = 10;
    cout << "The value of x is " << x << endl;
    return 0;
}

```

```
The value of x is 10


...Program finished with exit code 0
Press ENTER to exit console.
```

cin: This is an input stream object used to read input from the console or terminal. It is combined with the extraction operator >> to read input into variables.



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

int main() {
    int x;
    cout << "Enter a value for x: ";
    cin >> x;
    cout << "The value of x is " << x << endl;
    return 0;
}

```

```
The value of x is 10


...Program finished with exit code 0
Press ENTER to exit console.
```

15 . Literals

In C++, literals are values directly specified in a program's source code. They can represent values of various data types, such as integers, floating-point numbers, characters, and strings. Here are some examples of literals in C++:

1. Integer Literals: These are used to represent integer values. They can be expressed in decimal, hexadecimal, octal, or binary format.

```
int x = 42;       // decimal integer

int y = 0x2a;     // hexadecimal integer

int z = 052;      // octal integer

int w = 0b101010; // binary integer
```

2. Floating-Point Literals: These are used to represent real numbers. They can be expressed in decimal or exponential notation, including a decimal point and/or an exponent.

```
float a = 3.14;         // decimal floating-point
float b = 6.022e23;     // exponential notation
double c = 1.23456789;  // double-precision decimal
double d = 1.23e-4;     // small floating-point value
```

16. C++ String Class

C++ standard library provides a special string class for manipulate strings. It is define using #include<iostrem> preprocessor.

17. Truncated

whilst the floating point fee is assigned to an integer variable , the factorial part

of the fee is discared.

```
int val;

val=7.8;

int intno;

double doubleNo=7.9;

intNo=doubleNo;
```

18. Operators

1. Arithmetic Operators: These are used to perform mathematical operations, such as addition, subtraction, multiplication, and division.

```
int x = 10, y = 5;
int z = x + y; // addition
int w = x - y; // subtraction
int a = x * y; // multiplication
int b = x / y; // integer division
double c = x / (double)y; // floating-point division
int d = x % y; // modulus (remainder)
```

2. Comparison Operators: These are used to compare two values and produce a boolean result (true or false).

```
int x = 10, y = 5;
bool result1 = x == y; // equal to
bool result2 = x != y; // not equal to
bool result3 = x > y; // greater than
bool result4 = x < y; // less than
bool result5 = x >= y; // greater than or equal to
bool result6 = x <= y; // less than or equal to
```

3.Logical Operators: These are used to combine boolean expressions and produce a boolean result.

```
bool a = true, b = false;
bool result1 = a && b;  // logical AND
bool result2 = a || b;  // logical OR
bool result3 = !a;      // logical NOT (negation)
```

4.Bitwise Operators: These are used to perform bitwise operations on integer values.

```
int x = 0b1010, y = 0b1100;
int result1 = x & y;  // bitwise AND
int result2 = x | y;  // bitwise OR
int result3 = ~x;     // bitwise NOT (complement)
int result4 = x ^ y;  // bitwise XOR (exclusive OR)
int result5 = x << 2; // left shift
int result6 = y >> 1; // right shift
```

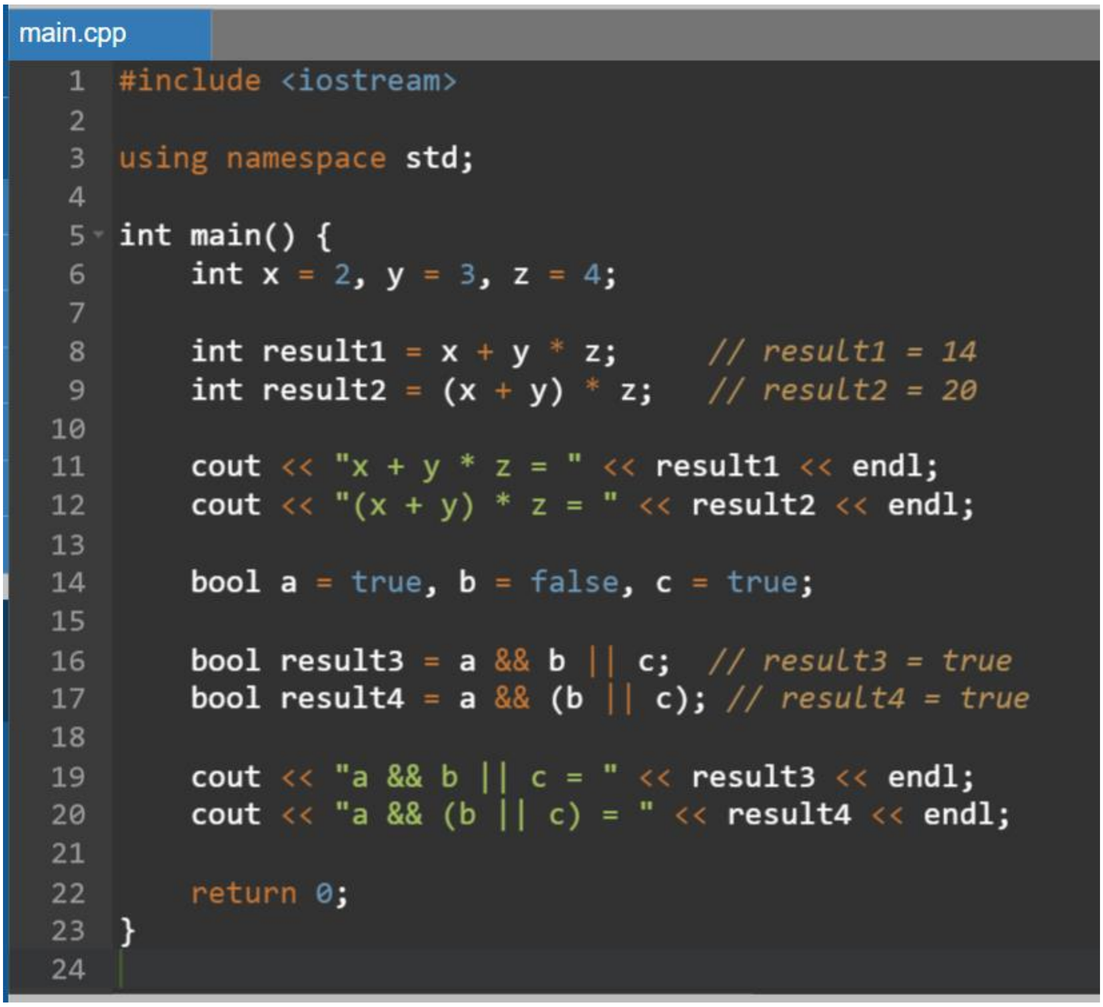
5.Assignment Operators: These are used to assign a value to a variable and perform a specific operation on it at the same time.

```
int x = 10;
x += 5;  // equivalent to x = x + 5;
x -= 2;  // equivalent to x = x - 2;
x *= 3;  // equivalent to x = x * 3;
x /= 2;  // equivalent to x = x / 2;
x %= 4;  // equivalent to x = x % 4;
```

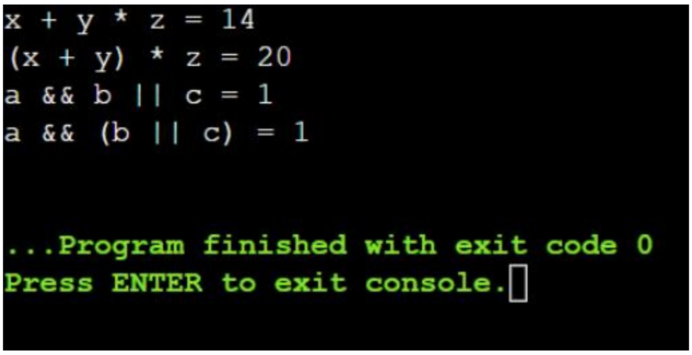
Operators Precedence

1. () (parentheses) - used to group expressions and force their evaluation before other operations.
2. ++ -- (postfix increment and decrement) - used to increment or decrement a variable after an expression is evaluated.
3. ++ -- (prefix increment and decrement) - used to increment or decrement a variable before an expression is evaluated.
4. + - (unary plus and minus) - used to indicate positive or negative numbers.
5. ! (logical NOT) - used to negate a boolean expression.
6. * / % (multiplication, division, and modulus) - used to perform arithmetic operations.
7. + - (addition and subtraction) - used to perform arithmetic operations.
8. < > <= >= (comparison operators) - used to compare two values.
9. == != (equality and inequality) - used to test equality or inequality of two values.
10. && (logical AND) - used to combine two boolean expressions.
11. || (logical OR) - used to combine two boolean expressions.
12. = (assignment) - used to assign a value to a variable.

In this code we can learn this precedence very well.



```cpp
#include <iostream>

using namespace std;

int main() {
    int x = 2, y = 3, z = 4;

    int result1 = x + y * z;     // result1 = 14
    int result2 = (x + y) * z;   // result2 = 20

    cout << "x + y * z = " << result1 << endl;
    cout << "(x + y) * z = " << result2 << endl;

    bool a = true, b = false, c = true;

    bool result3 = a && b || c;  // result3 = true
    bool result4 = a && (b || c); // result4 = true

    cout << "a && b || c = " << result3 << endl;
    cout << "a && (b || c) = " << result4 << endl;

    return 0;
}

```

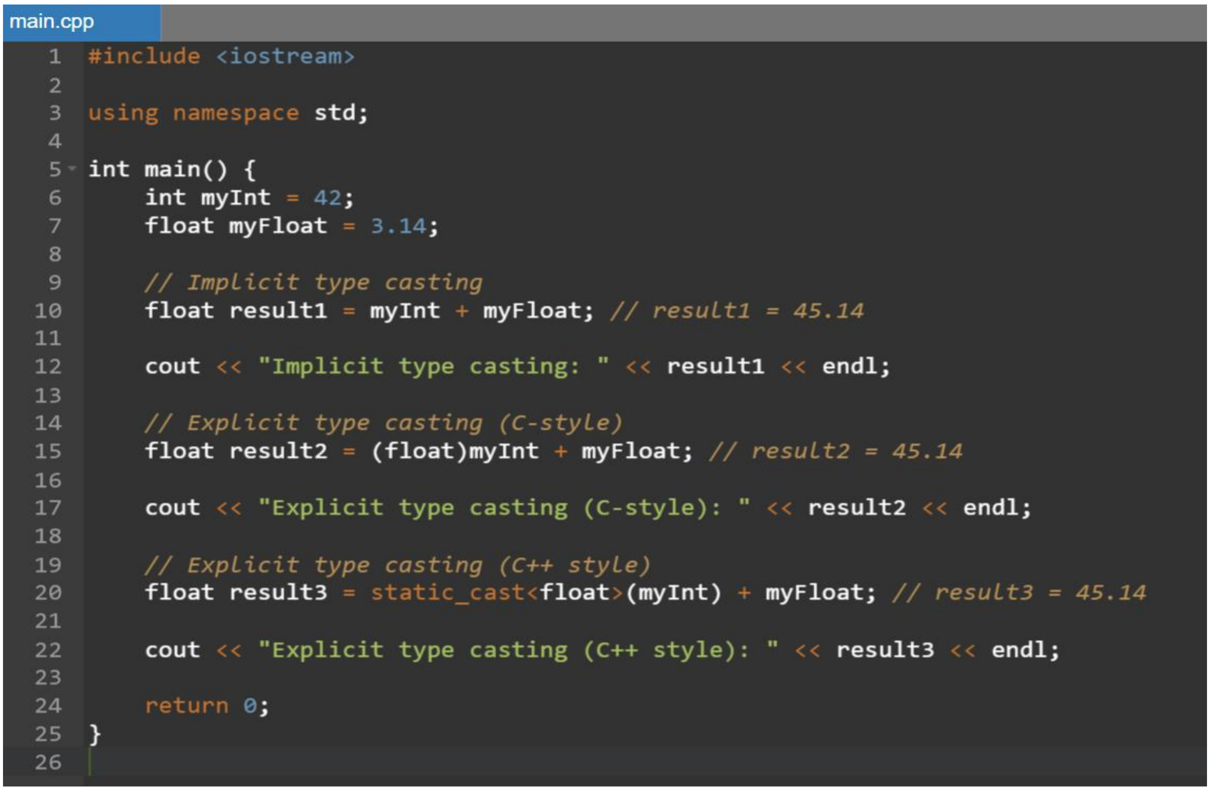


```
x + y * z = 14
(x + y) * z = 20
a && b || c = 1
a && (b || c) = 1


...Program finished with exit code 0
Press ENTER to exit console.
```

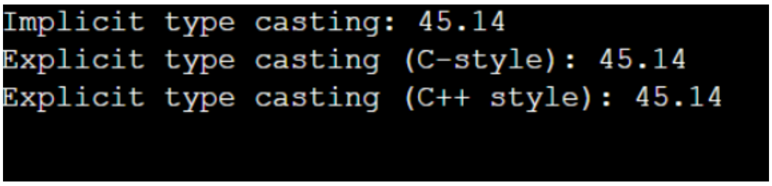
Type Casting

Explicit type casting is where you force a value to be converted to a different data type using the cast operator. There are two forms of explicit type casting in C++:

1. C-style type casting: This type of type casting uses the syntax (type)expression. For example, to convert an integer to a floating-point number, you can use (float)myInt.
2. C++ style type casting: This type casting type uses new keywords to perform the conversion. There are four types of C++ style type casting:
3. a. static_cast: This keyword is used to perform non-polymorphic conversions. For example, you can use static_cast<float>(myInt) to convert an integer to a floating-point number.
4. b. dynamic_cast: This keyword is used to perform downcasting (converting a pointer to a base class to a pointer to a derived class) in polymorphic class hierarchies.
5. c. const_cast: This keyword removes the const or volatile qualifiers from a variable.
6. d. reinterpret_cast: This keyword converts a pointer to any other type of pointer, even if the resulting pointer is not of the same type or alignment.

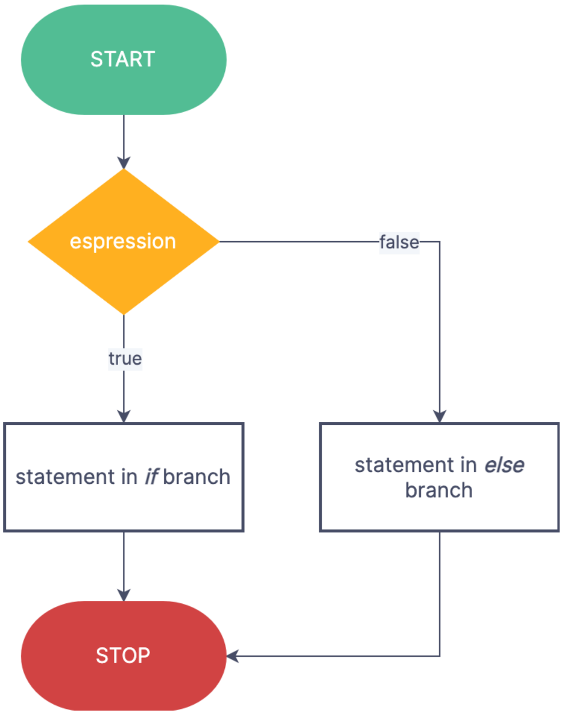


```cpp
#include <iostream>

using namespace std;

int main() {
    int myInt = 42;
    float myFloat = 3.14;

    // Implicit type casting
    float result1 = myInt + myFloat; // result1 = 45.14

    cout << "Implicit type casting: " << result1 << endl;

    // Explicit type casting (C-style)
    float result2 = (float)myInt + myFloat; // result2 = 45.14

    cout << "Explicit type casting (C-style): " << result2 << endl;

    // Explicit type casting (C++ style)
    float result3 = static_cast<float>(myInt) + myFloat; // result3 = 45.14

    cout << "Explicit type casting (C++ style): " << result3 << endl;

    return 0;
}

```

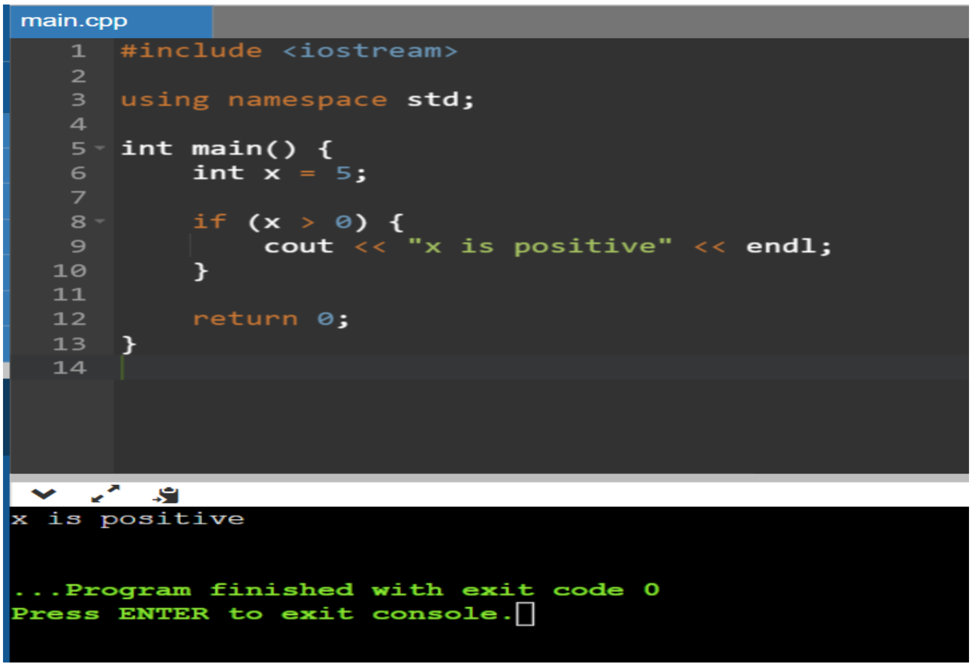


```
Implicit type casting: 45.14
Explicit type casting (C-style): 45.14
Explicit type casting (C++ style): 45.14
```

19. Statement and Conditions



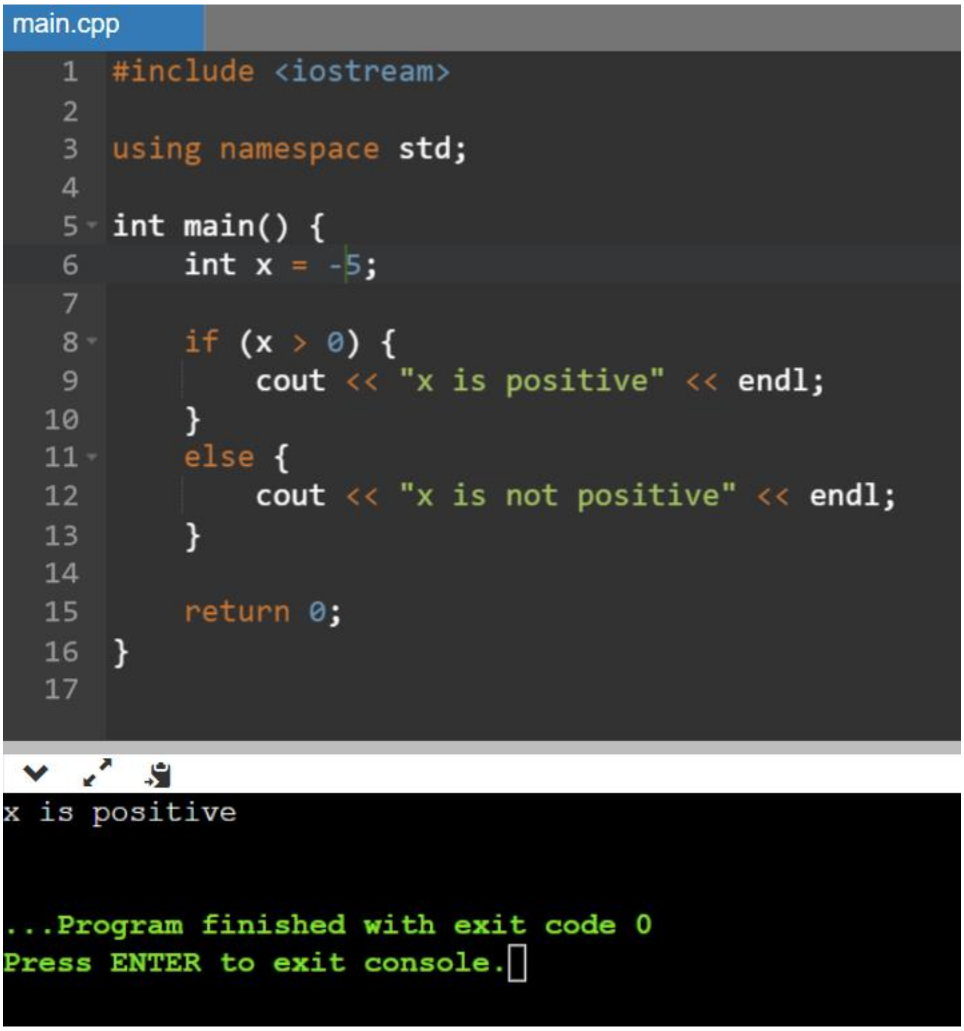
If Statement



```cpp
#include <iostream>

using namespace std;

int main() {
    int x = 5;

    if (x > 0) {
        cout << "x is positive" << endl;
    }

    return 0;
}

```

```
x is positive


...Program finished with exit code 0
Press ENTER to exit console.
```

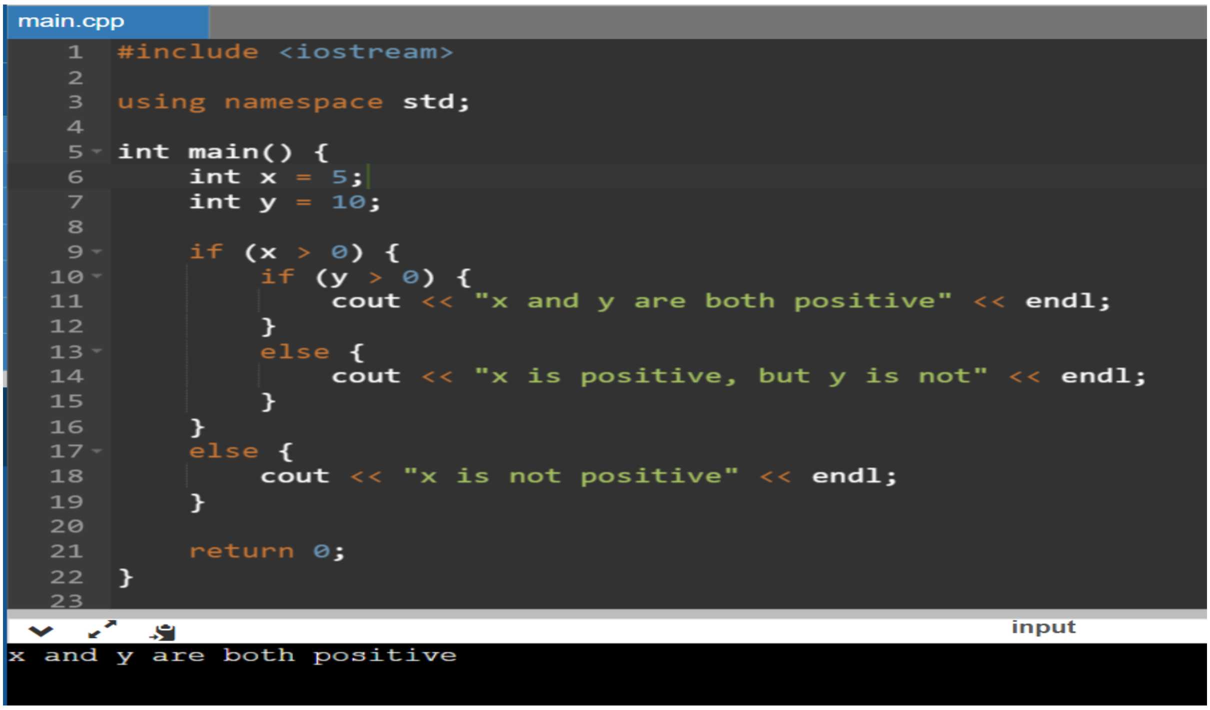
If else Statement



```cpp
#include <iostream>

using namespace std;

int main() {
    int x = -5;

    if (x > 0) {
        cout << "x is positive" << endl;
    }
    else {
        cout << "x is not positive" << endl;
    }

    return 0;
}

```

```
x is positive


...Program finished with exit code 0
Press ENTER to exit console.
```

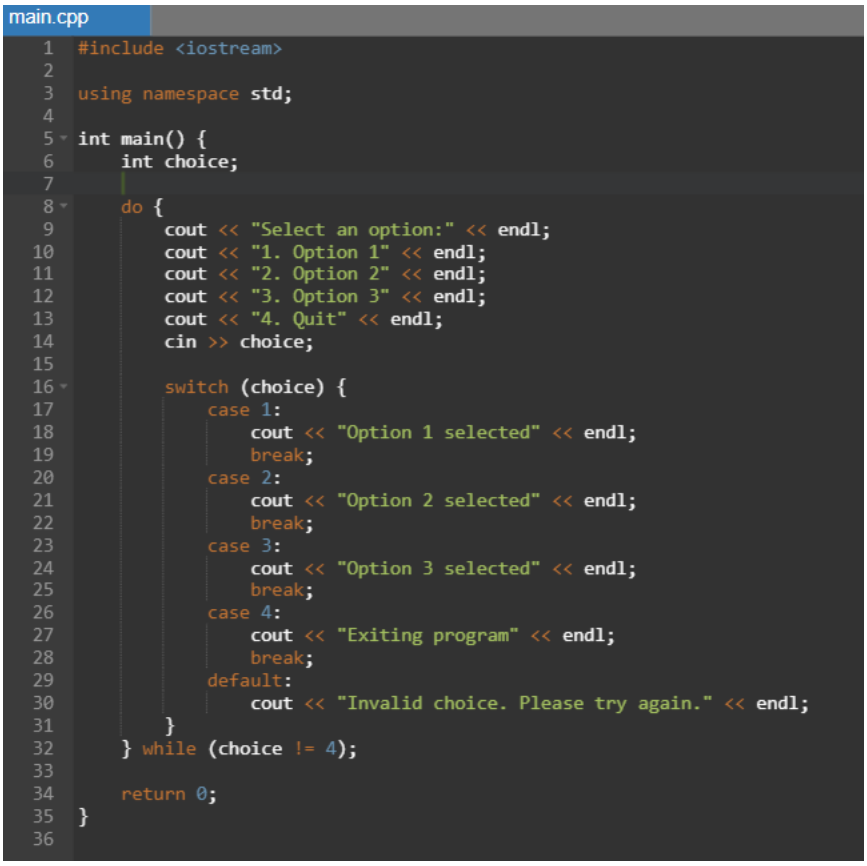
Nested if statement



```cpp
#include <iostream>

using namespace std;

int main() {
    int x = 5;
    int y = 10;

    if (x > 0) {
        if (y > 0) {
            cout << "x and y are both positive" << endl;
        }
        else {
            cout << "x is positive, but y is not" << endl;
        }
    }
    else {
        cout << "x is not positive" << endl;
    }

    return 0;
}
```

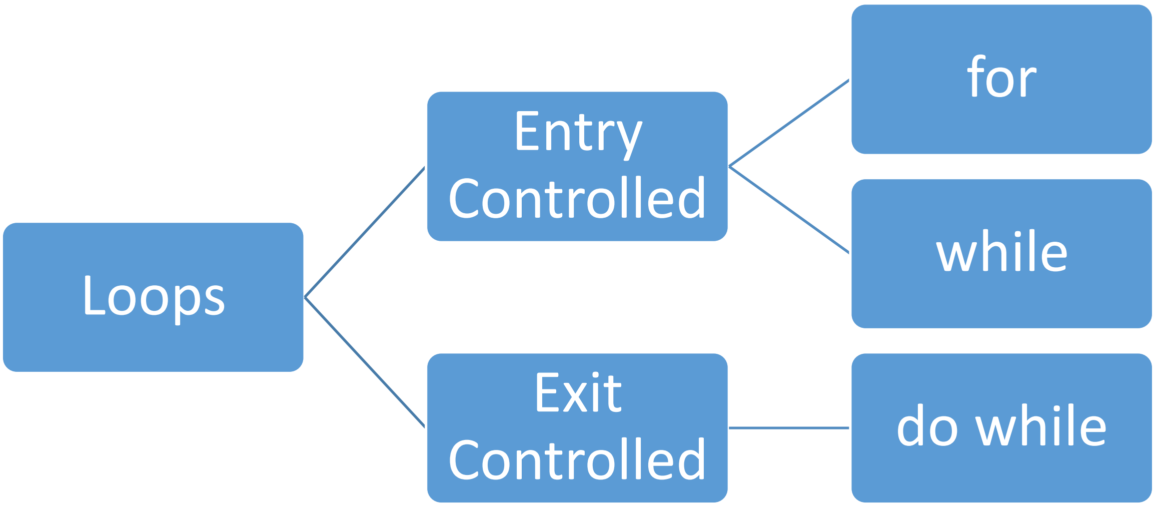
```
x and y are both positive
```

Menu Driven Program



```cpp
#include <iostream>

using namespace std;

int main() {
    int choice;

    do {
        cout << "Select an option:" << endl;
        cout << "1. Option 1" << endl;
        cout << "2. Option 2" << endl;
        cout << "3. Option 3" << endl;
        cout << "4. Quit" << endl;
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Option 1 selected" << endl;
                break;
            case 2:
                cout << "Option 2 selected" << endl;
                break;
            case 3:
                cout << "Option 3 selected" << endl;
                break;
            case 4:
                cout << "Exiting program" << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 4);

    return 0;
}
```

20. C++ Loops



Loops

Entry Controlled

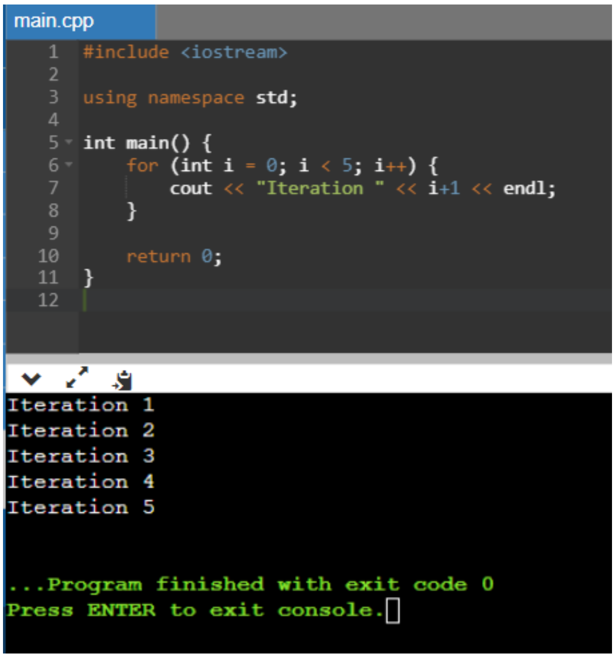
Exit Controlled

for

while

do while

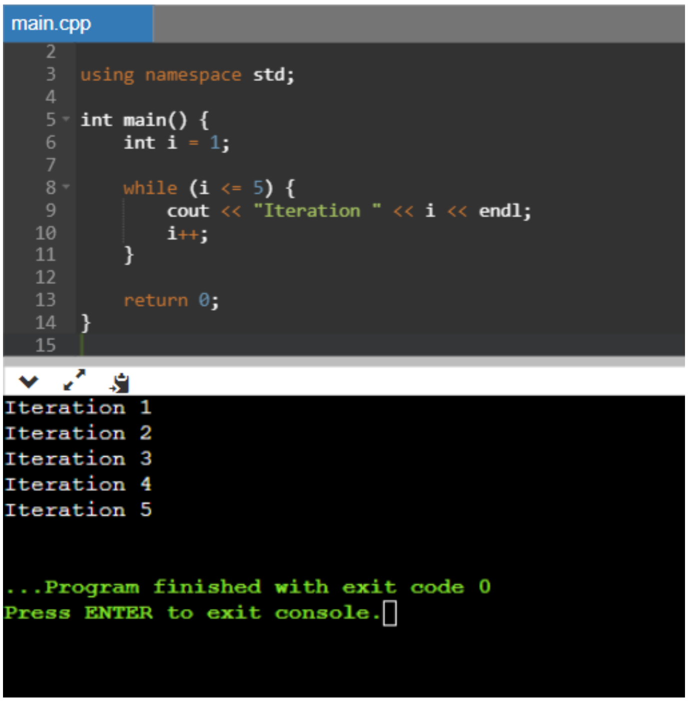
For Loop



```cpp
#include <iostream>

using namespace std;

int main() {
    for (int i = 0; i < 5; i++) {
        cout << "Iteration " << i+1 << endl;
    }

    return 0;
}
```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5


...Program finished with exit code 0
Press ENTER to exit console.
```

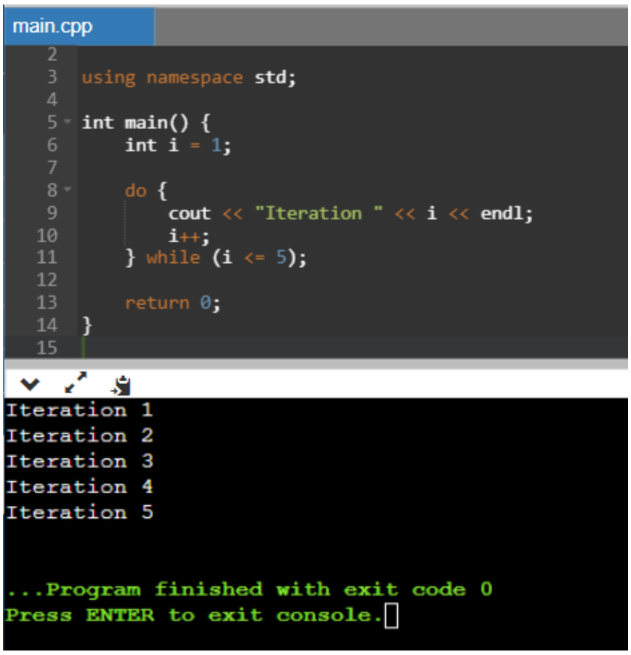
While Loop



```cpp

using namespace std;

int main() {
    int i = 1;

    while (i <= 5) {
        cout << "Iteration " << i << endl;
        i++;
    }

    return 0;
}

```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5


...Program finished with exit code 0
Press ENTER to exit console.
```

Do While Loop



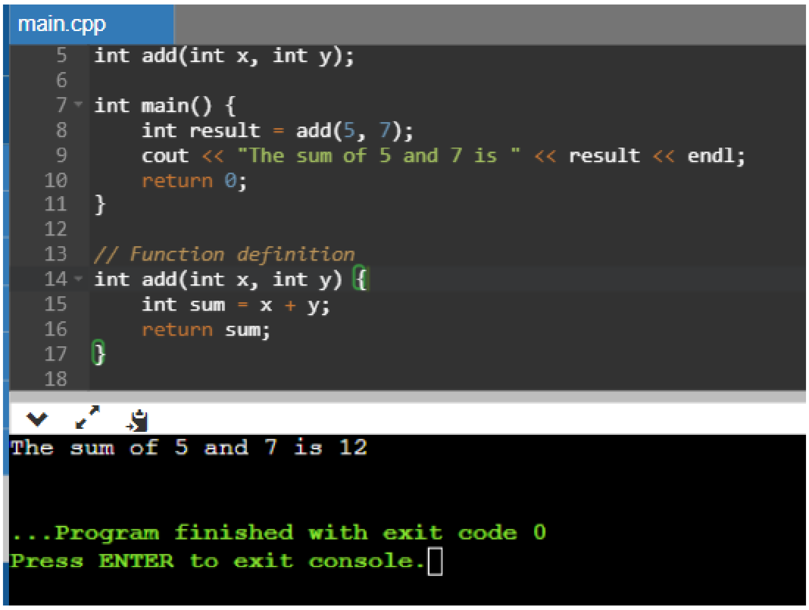
```cpp

using namespace std;

int main() {
    int i = 1;

    do {
        cout << "Iteration " << i << endl;
        i++;
    } while (i <= 5);

    return 0;
}

```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5


...Program finished with exit code 0
Press ENTER to exit console.
```

21. Function of C++

Functions allow you to break your code into small, reusable pieces, making it easier to read, debug, and maintain. In C++, you can create two types of functions: built-in functions and user-defined functions.

1. Built-in functions: These are functions that are already provided by the C++ language, such as sqrt, pow, and strlen. You can call these functions directly in your code without defining them yourself.
2. User-defined functions: These are functions that you define yourself in your C++ code. User-defined functions allow you to create your functions to perform specific tasks not provided by the built-in functions.

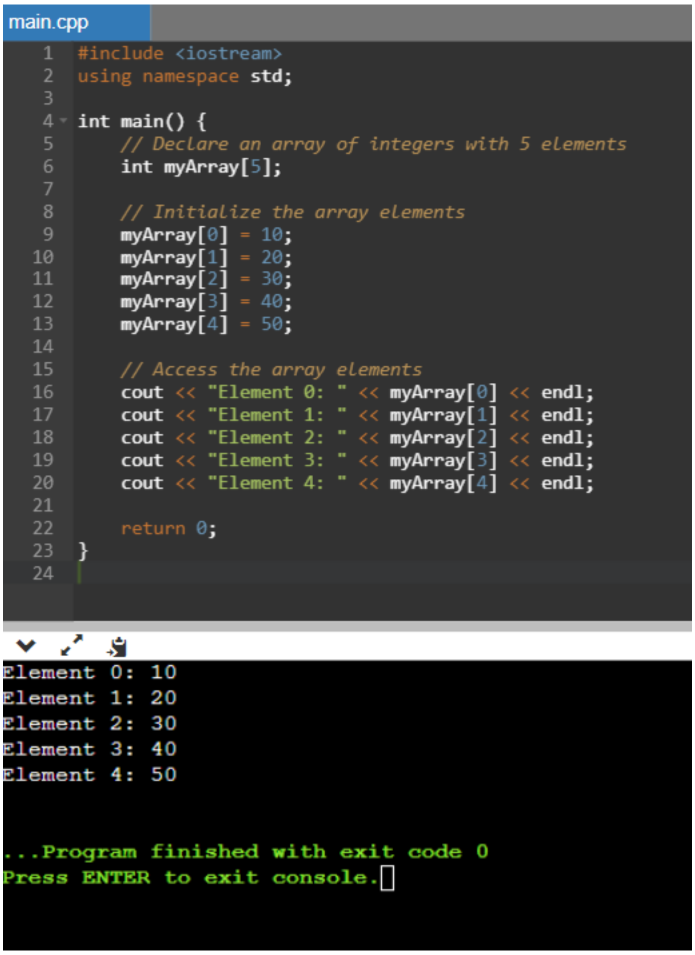


```cpp
int add(int x, int y);

int main() {
    int result = add(5, 7);
    cout << "The sum of 5 and 7 is " << result << endl;
    return 0;
}

// Function definition
int add(int x, int y) {
    int sum = x + y;
    return sum;
}

```

```
The sum of 5 and 7 is 12


...Program finished with exit code 0
Press ENTER to exit console.
```

22. Arrays in C++

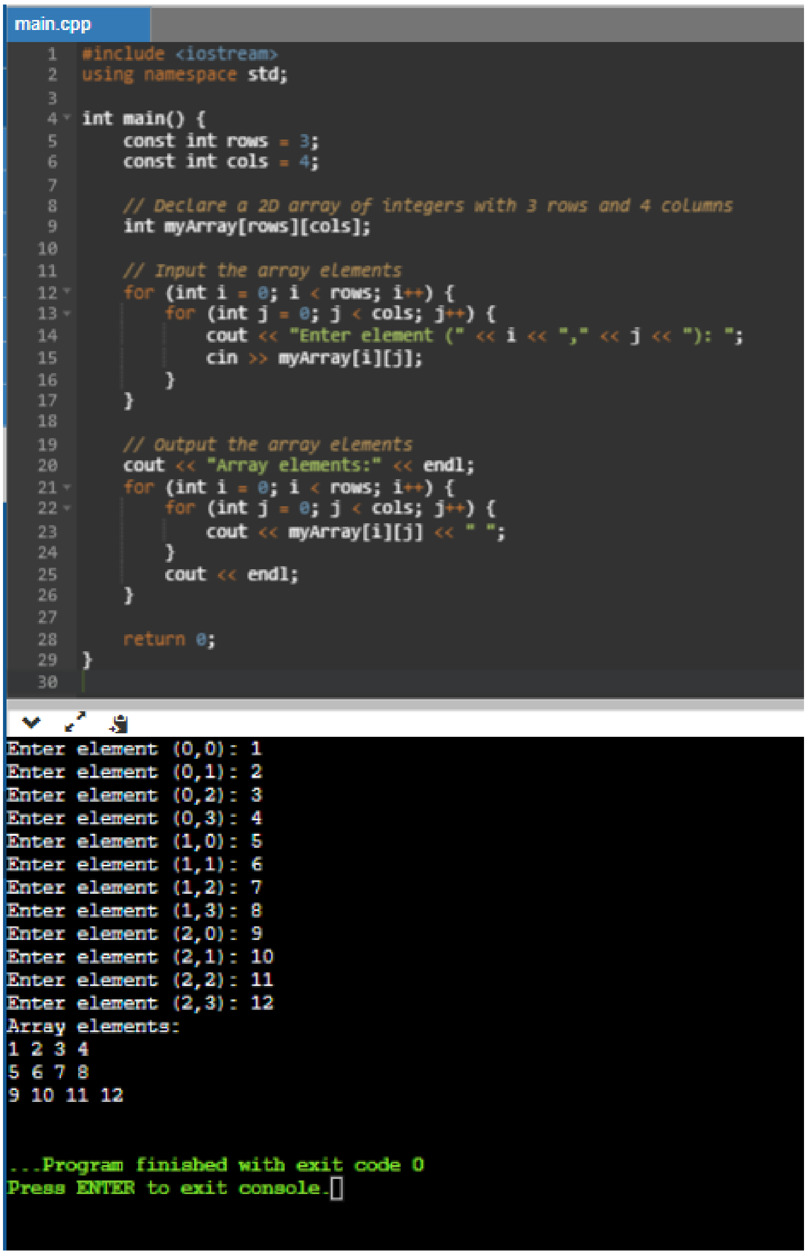
An array is a collection of elements of the same data type stored in memory. You can think of an array as a group of variables that are all the same type and are accessed using a single name.



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

int main() {
    // Declare an array of integers with 5 elements
    int myArray[5];

    // Initialize the array elements
    myArray[0] = 10;
    myArray[1] = 20;
    myArray[2] = 30;
    myArray[3] = 40;
    myArray[4] = 50;

    // Access the array elements
    cout << "Element 0: " << myArray[0] << endl;
    cout << "Element 1: " << myArray[1] << endl;
    cout << "Element 2: " << myArray[2] << endl;
    cout << "Element 3: " << myArray[3] << endl;
    cout << "Element 4: " << myArray[4] << endl;

    return 0;
}
```

```
Element 0: 10
Element 1: 20
Element 2: 30
Element 3: 40
Element 4: 50


...Program finished with exit code 0
Press ENTER to exit console.
```

22. 2D Array in C++



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

int main() {
    const int rows = 3;
    const int cols = 4;

    // Declare a 2D array of integers with 3 rows and 4 columns
    int myArray[rows][cols];

    // Input the array elements
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << "Enter element (" << i << "," << j << "): ";
            cin >> myArray[i][j];
        }
    }

    // Output the array elements
    cout << "Array elements:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << myArray[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}

```

```
Enter element (0,0): 1
Enter element (0,1): 2
Enter element (0,2): 3
Enter element (0,3): 4
Enter element (1,0): 5
Enter element (1,1): 6
Enter element (1,2): 7
Enter element (1,3): 8
Enter element (2,0): 9
Enter element (2,1): 10
Enter element (2,2): 11
Enter element (2,3): 12
Array elements:
1 2 3 4
5 6 7 8
9 10 11 12


...Program finished with exit code 0
Press ENTER to exit console.
```

END