**UNIVERTY OF PORT HARCOURT**

**FACULTY OG COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE**

**C++ COMPUTER PROGRAMMING**

**NAME: ROBERT BARIVURE BLESSED**

**MAT NO: U2022/5570055**

**DEPT.: COMPUTER SCIENCE**

**CODE: CSC 287 PRACTICAL**

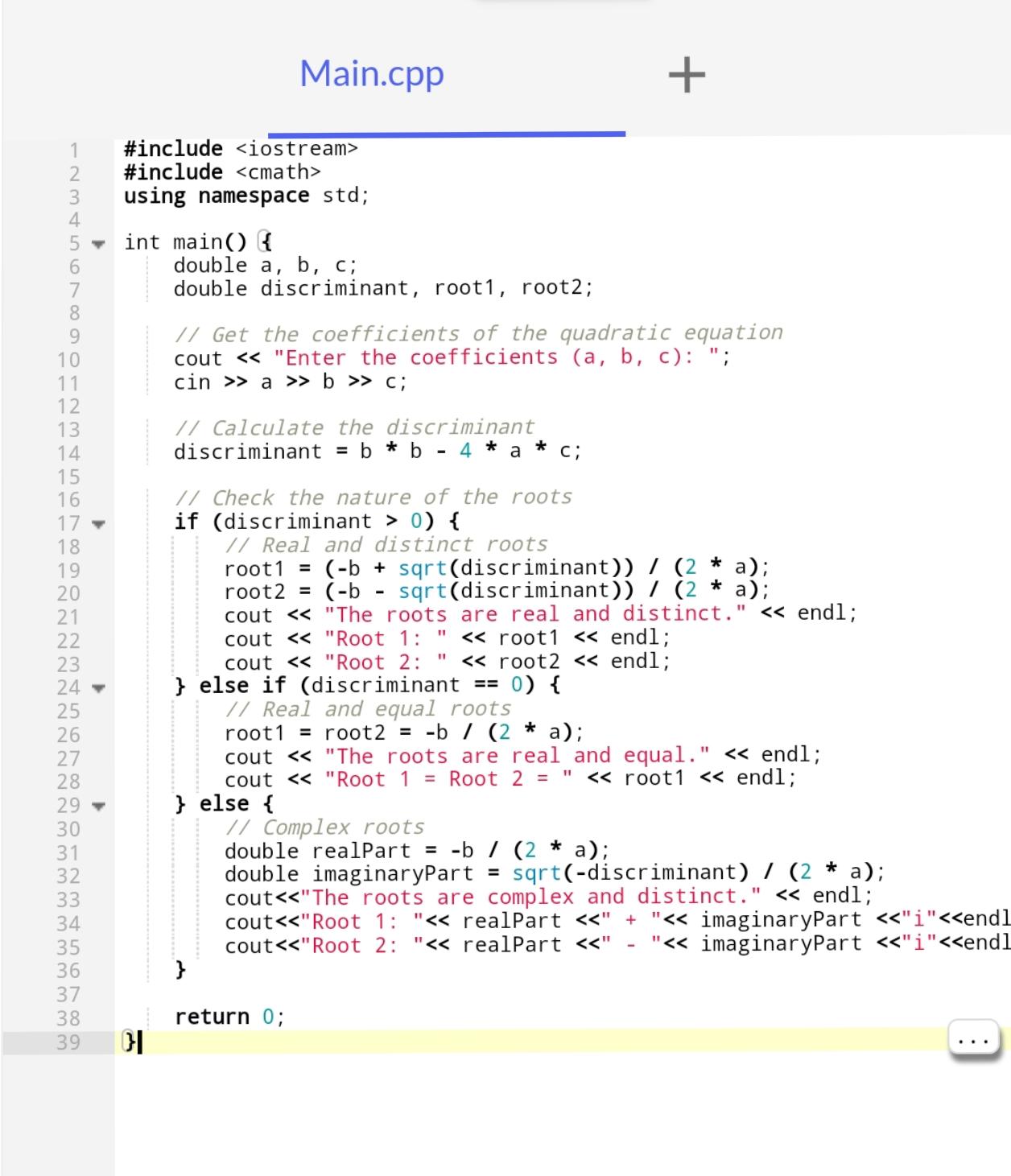
**DATE: 18TH SEPTEMBER, 2024**

***QUESTION 1***

1. **A C++ program to distinguish between two types of known variables.**



1. **Develop a C++ program to compute the root of a quadratic equation.**

****

1. **Write a comprehensive note on a** **comment in C++ programming language.**

In C++, comments are essential for enhancing code readability and maintainability. They serve as annotations that explain the purpose and functionality of code segments, making it easier for developers to understand and collaborate on projects.

Types of Comments

C++ supports two primary types of comments:

1. \*\*Single-line Comments\*\*: These begin with `//` and extend to the end of the line. They are typically used for brief explanations or notes. For example:

// This is a single-line comment

cout << “Hello, World!”; // Print greeting

1. \*\*Multi-line Comments\*\*: These start with `/\*` and end with `\*/`, allowing for comments that span multiple lines. They are useful for more detailed explanations. For example:

/\* This is a multi-line comment

That spans multiple lines \*/

Purpose of Comments

Comments are used for various reasons:

- \*\*Code Explanation\*\*: They clarify complex code sections, aiding in understanding.

- \*\*Documentation\*\*: Comments serve as inline documentation for functions, classes, and algorithms.

- \*\*Debugging Aid\*\*: Developers can comment out code segments to isolate issues during debugging.

- \*\*Maintainability\*\*: Well-commented code is easier to modify and update, especially for new developers.

- \*\*Collaboration\*\*: Comments facilitate communication among team members by providing context for code decisions.

Best Practices

- Use comments to explain \*\*why\*\* something is done, rather than \*\*how\*\* it is done.

- Keep comments concise and relevant to the code they annotate.

- Avoid over-commenting; strive for clean, self-explanatory code where possible.

By incorporating comments effectively, developers can create more understandable and maintainable codebases, benefiting both current and future collaborators.

1. **What do you mean when you say C++ language is case sensitive.**

When we say that C++ is a \*\*case-sensitive\*\* programming language, it means that the language distinguishes between uppercase and lowercase letters in identifiers, such as variable names, function names, and class names. This characteristic implies that `Variable`, `variable`, and `VARIABLE` are considered three distinct identifiers in C++.

Key Points about Case Sensitivity in C++:

1. \*\*Identifier Uniqueness\*\*: In C++, identifiers that differ only in case are treated as separate entities. For example:

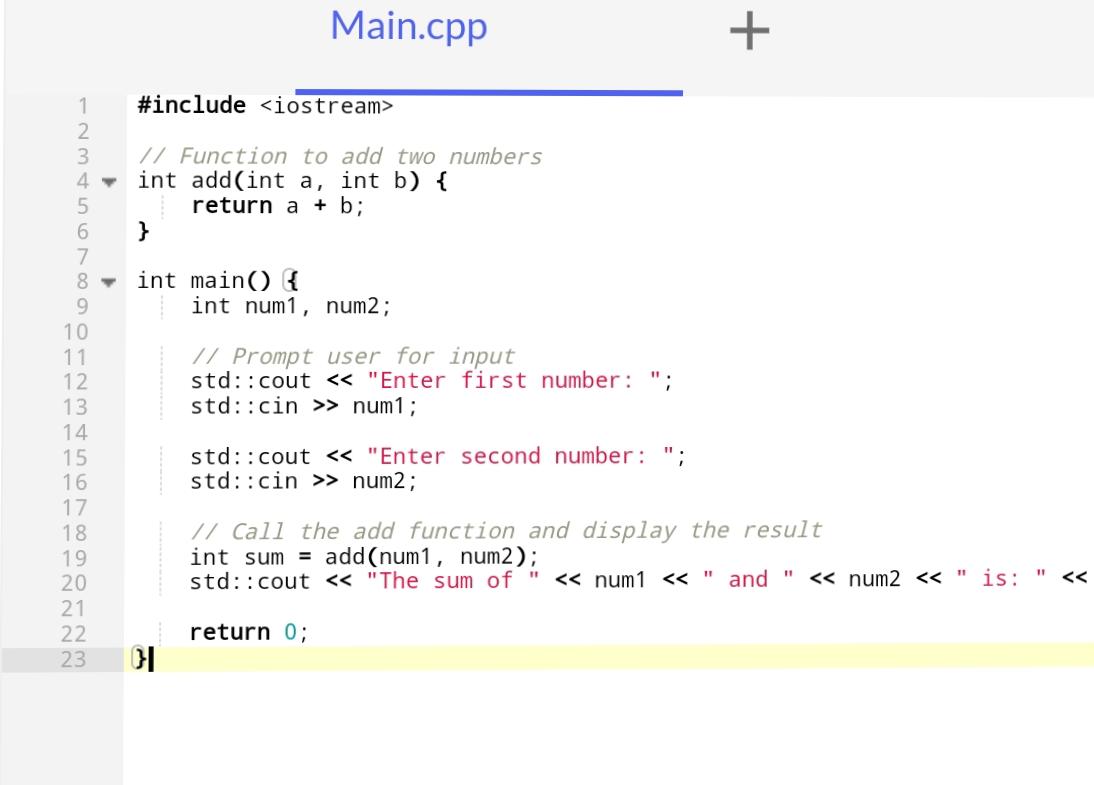
Int myVariable = 5;

Int MyVariable = 10; // This is a different variable

1. \*\*Error Prone\*\*: Case sensitivity can lead to errors if a programmer accidentally uses the wrong case. For instance, if a variable is declared as `myVar`, using `MyVar` later in the code will result in a compilation error, as it is not recognized as the same identifier.
2. \*\*Consistency in Naming\*\*: To avoid confusion, it is essential to adopt consistent naming conventions. Many developers use specific casing styles, such as camelCase or snake\_case, to improve code readability and maintainability.
3. \*\*Comparison with Case-Insensitive Languages\*\*: In contrast, case-insensitive languages treat identifiers without regard to case. For example, in a case-insensitive language, `myVariable`, `MyVariable`, and `MYVARIABLE` would all refer to the same identifier.
4. \*\*Influence on Language Design\*\*: The case sensitivity in C++ is inherited from its predecessor, C, and is a common feature in many modern programming languages, including Java and C#. This design choice allows for a broader range of identifier names but can also complicate debugging.

Understanding case sensitivity is crucial for writing correct and efficient C++ code, as it directly impacts how identifiers are interpreted and managed within the program.

1. **Write a function code in C++r to add two numbers**

****

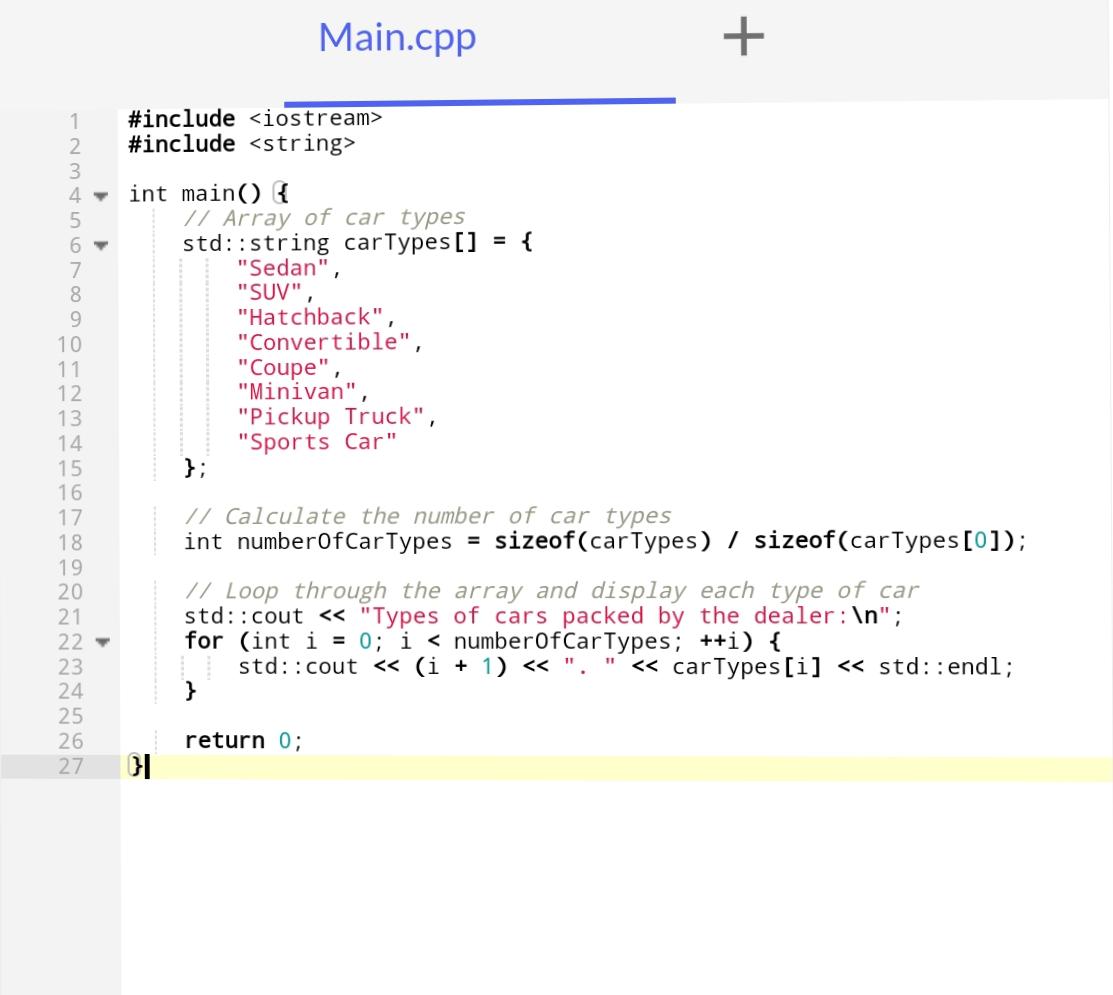
1. **With the aid of diagram explain C++ data types**

|  |
| --- |
| | Data Types |  | Primitive Types |   * int * char * bool * float * double * void   | Derived Types |   * Array * Function * Pointer   | User-defined Types |   * Class * Struct * Union   +-----------------------+ |

1. **Develop a C++ program to implement a multiplication table using the nested for loop technique.**

***QUESTION 2***

* 1. **Write an array to loop through the types of cars packed by a car dealer.**



.

* 1. **What is the difference between parameter a argument.**

## The Difference Between Parameters and Arguments in C++

The main difference between parameters and arguments in C++ is:

* + \*\*Parameters\*\* are the variables defined in the function declaration, while \*\*arguments\*\* are the actual values passed to the function when it is called.

Here’s a more detailed explanation:

Parameters

- Parameters are the variables listed in the function declaration, inside the parentheses.

- They act as placeholders for the values that will be passed to the function.

- Parameters are used to receive the arguments when the function is called.

- They are also known as \*\*formal parameters\*\*.

Arguments

- Arguments are the actual values passed to the function when it is called.

- They are used to initialize the parameters of the function.

- Arguments are placed inside the parentheses in the function call.

- They are also known as \*\*actual parameters\*\*.

Here’s an example to illustrate the difference:

Void myFunction(int x, float y) { // x and y are parameters

// function body

}

Int main() {

myFunction(5, 3.14); // 5 and 3.14 are arguments

return 0;

}

In this example:

- `x` and `y` are parameters in the function declaration.

- `5` and `3.14` are arguments passed to the function when it is called in `main()`.

The number, type, and order of arguments must match the number, type, and order of parameters in the function declaration. If they don’t match, a compiler error will occur.

So, in summary, \*\*parameters\*\* are the variables used in the function definition, while \*\*arguments\*\* are the actual values passed to the function when it is called.

* 1. **In C++ how can you describe a pointer.**

In C++, a \*\*pointer\*\* is a variable that stores the memory address of another variable. This allows for efficient memory management and direct access to memory locations, which is a powerful feature of the language.

Key Characteristics of Pointers

1. \*\*Memory Address Storage\*\*: Pointers hold the address of a variable rather than its value. For example, if you have an integer variable, you can create a pointer that points to its memory address.

2. \*\*Declaration\*\*: A pointer is declared using the asterisk (`\*`) symbol. The syntax is:

data\_type \*pointer\_name;

Here, `data\_type` indicates the type of data the pointer will point to, and `pointer\_name` is the name of the pointer variable.

3. \*\*Initialization\*\*: You can assign a pointer the address of a variable using the address-of operator (`&`). For example:

int var = 10;

int \*ptr = &var; // ptr now holds the address of var

4. \*\*Dereferencing\*\*: To access or modify the value stored at the address pointed to by a pointer, you use the dereference operator (`\*`). For example:

int value = \*ptr; // Gets the value at the address stored in ptr

5.\*\*Pointer Arithmetic\*\*: You can perform arithmetic operations on pointers, such as incrementing or decrementing them, which allows for easy traversal through arrays.

* 1. **With the aid of appropriate illustration demonstrate how pointers are declared in C++**

Here’s a simple example demonstrating pointers in C++:

#include <iostream>

int main() {

int num = 42; // An integer variable

int \*ptr = &num; // Pointer declaration and initialization

std::cout << "Address of num: " << ptr << std::endl; // Prints address of num

std::cout << "Value of num: " << \*ptr << std::endl; /\* Dereferences ptr to get value \*/

\*ptr = 100; // Modifies num via pointer

std::cout << "New value of num: " << num << std::endl; // Prints modified value

return 0;

}

Output

Address of num: 0x7ffeedb2c6bc

Value of num: 42

New value of num: 100In this example, `ptr` holds the address of `num`, and through dereferencing, we can access and modify `num` directly via its pointer.

* 1. **Explain the meaning of the following escape sequence \n, \r, \t, \v, \b, \f,\a, \’, \”,\?, \\**

Escape sequences in C++ are special character combinations that represent certain characters or actions within string and character literals. They begin with a backslash (`\`) followed by one or more characters. Here’s a breakdown of the specified escape sequences:

| Escape Sequence | Meaning

`\n` Newline: Moves the cursor to the beginning of the next line.

`\r` Carriage Return: Moves the cursor to the beginning of the current line without advancing to the next line.

`\t` Horizontal Tab: Inserts a tab space, typically equivalent to 4 or 8 spaces.

`\v` Vertical Tab: Moves the cursor down to the next vertical tab stop (less commonly used).

`\b` Backspace: Moves the cursor back one position, effectively deleting the character at that position.

`\f` Form Feed: Advances the cursor to the next page (used in printing).

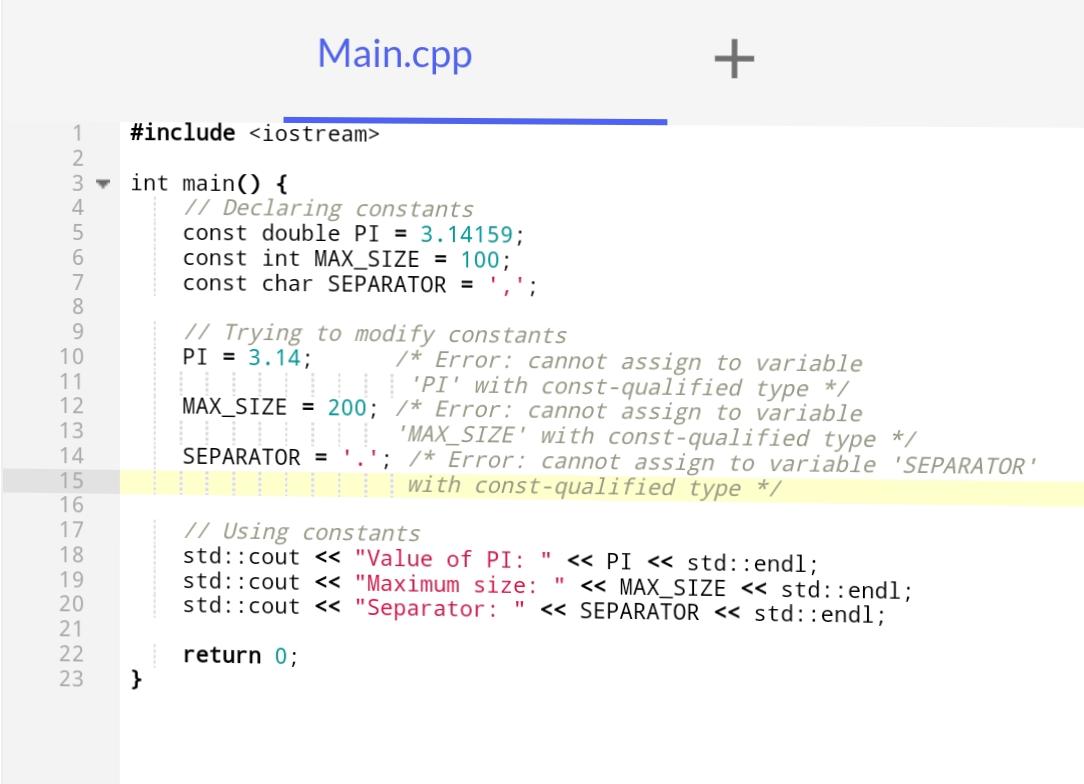
`\a` Alert (Bell): Triggers an audible alert or beep sound, if supported by the system.

`\’` Single Quote: Inserts a single quote character into a string.

`\”` Double Quote: Inserts a double quote character into a string.

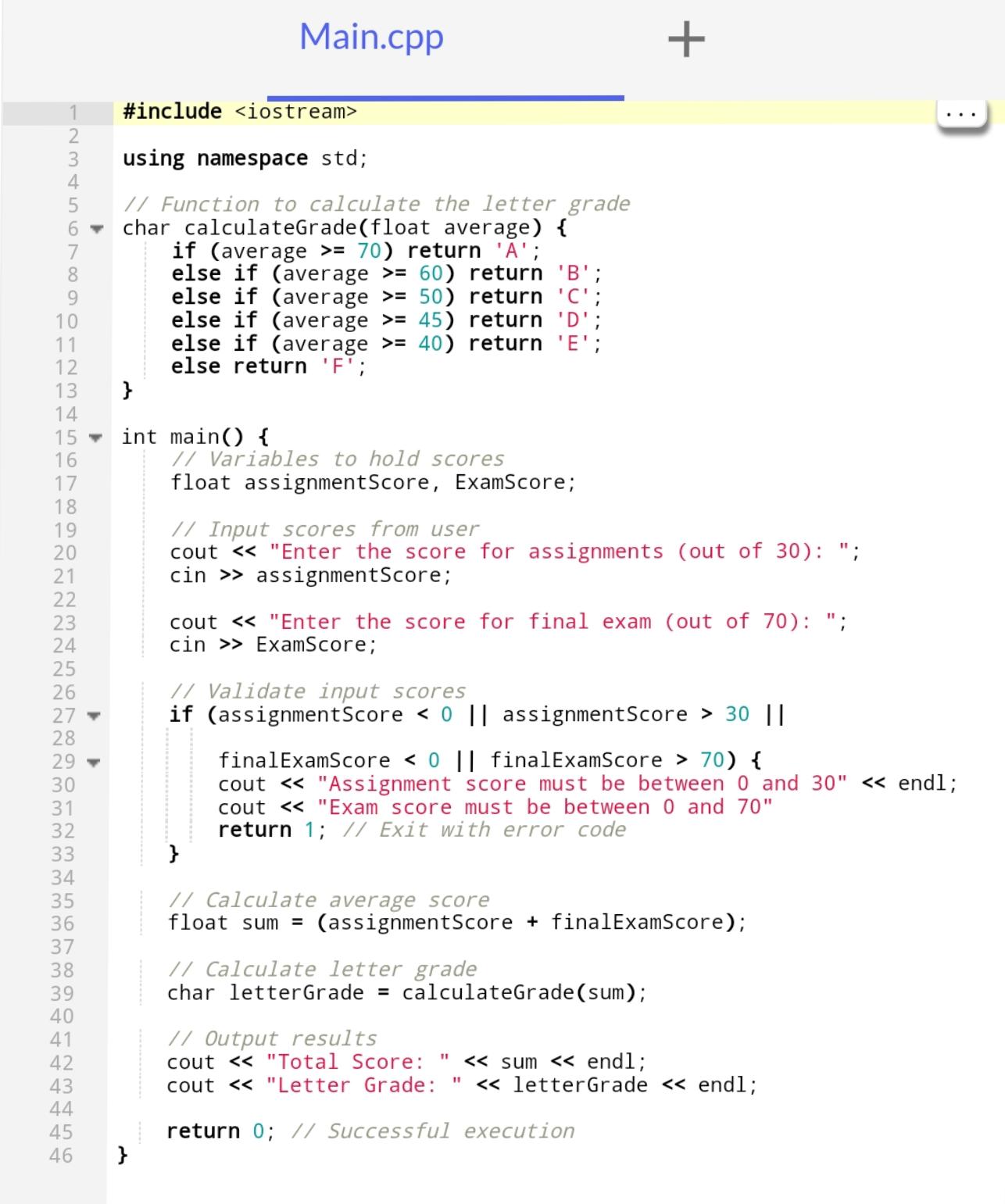
`\?` Question Mark: Used to avoid trigraphs; it inserts a question mark character.

`\\` Backslash: Inserts a literal backslash character into a string.

* 1. **With the aid of program construct explain how constant can be declared in a C++ program.**

***QUESTION 3***

* + 1. **Write a C++ code to compute the grade of computer science student**



* + 1. **How best can you differentiate between a reference operator deference operator .**

To differentiate between the reference operator and the dereference operator in programming languages like C and C++, it’s essential to understand their functions and usage:

Reference Operator (`&`)

* + The \*\*reference operator\*\* is denoted by `&`. It is used to obtain the \*\*address\*\* of a variable. For example, if you have a variable `x`, using `&x` will return the memory address where `x` is stored.
  + This operator is commonly used when you want to assign the address of a variable to a pointer. For instance, in the statement `p = &x;`, `p` is a pointer that now holds the address of `x`[2][5].

Dereference Operator (`\*`)

* + The \*\*dereference operator\*\* is represented by `\*`. It allows you to access or modify the \*\*value\*\* stored at the address that a pointer points to. For example, if you have a pointer `p`, using `\*p` will give you the value located at the address held by `p`.
  + This operator is crucial when you want to read or change the value of a variable through its pointer. For instance, in the expression `y = \*p;`, you are assigning the value pointed to by `p` to the variable `y`
  + In summary, use `&` to get an address (reference), and use `\*` to access or modify the value at that address (dereference).
    1. **Differentiate between a variable and a keyword**

Variable

* + A \*\*variable\*\* is a named storage location in memory that holds data. It can store different types of values (e.g., integers, floats, strings) and can be modified during program execution.

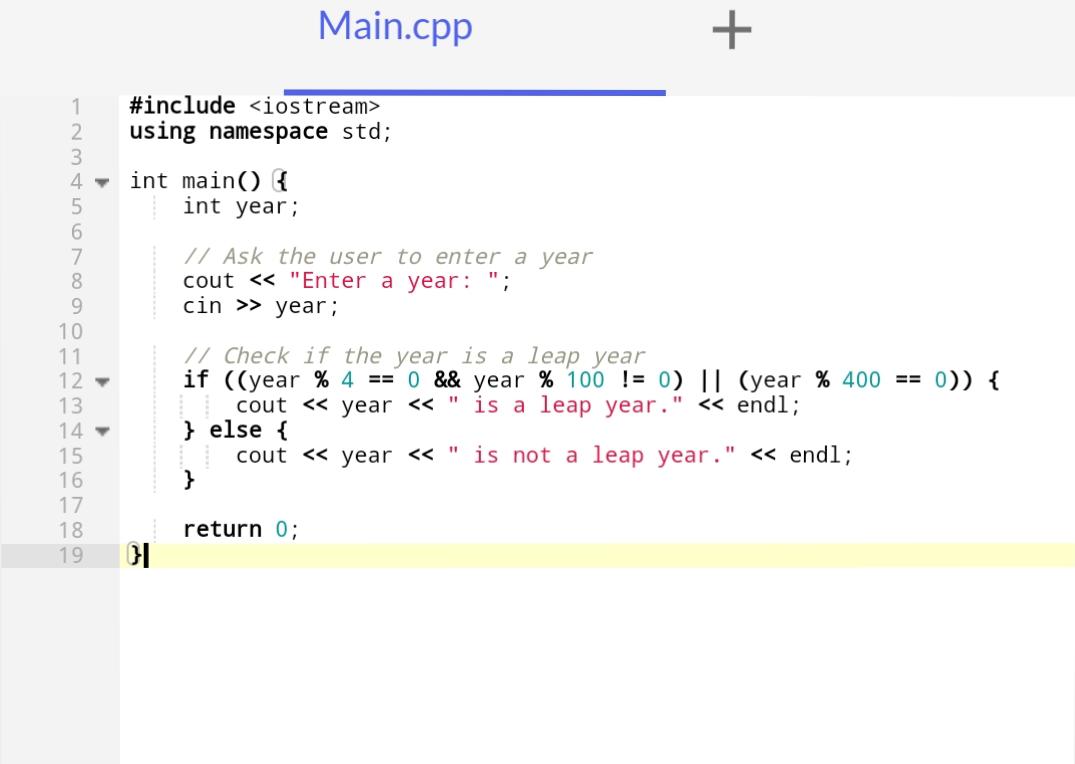
Int age = 25;

Keyword

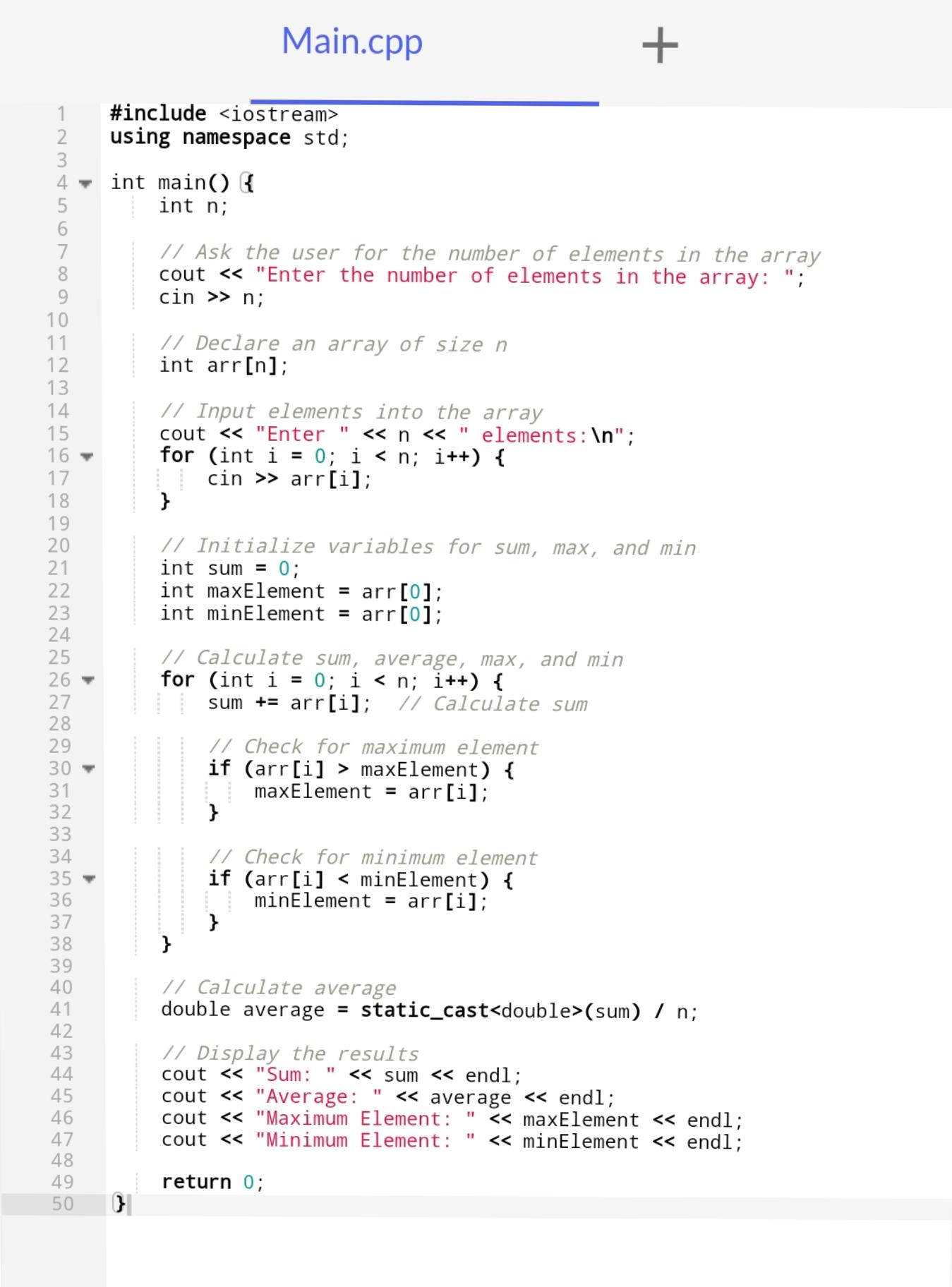
* + A \*\*keyword\*\* is a reserved word in a programming language that has a predefined meaning and cannot be used as an identifier (like variable names).
  + Examples of keywords in C include `int`, `return`, `if`, `else`, and `while`. These words serve specific functions within the language’s structure.

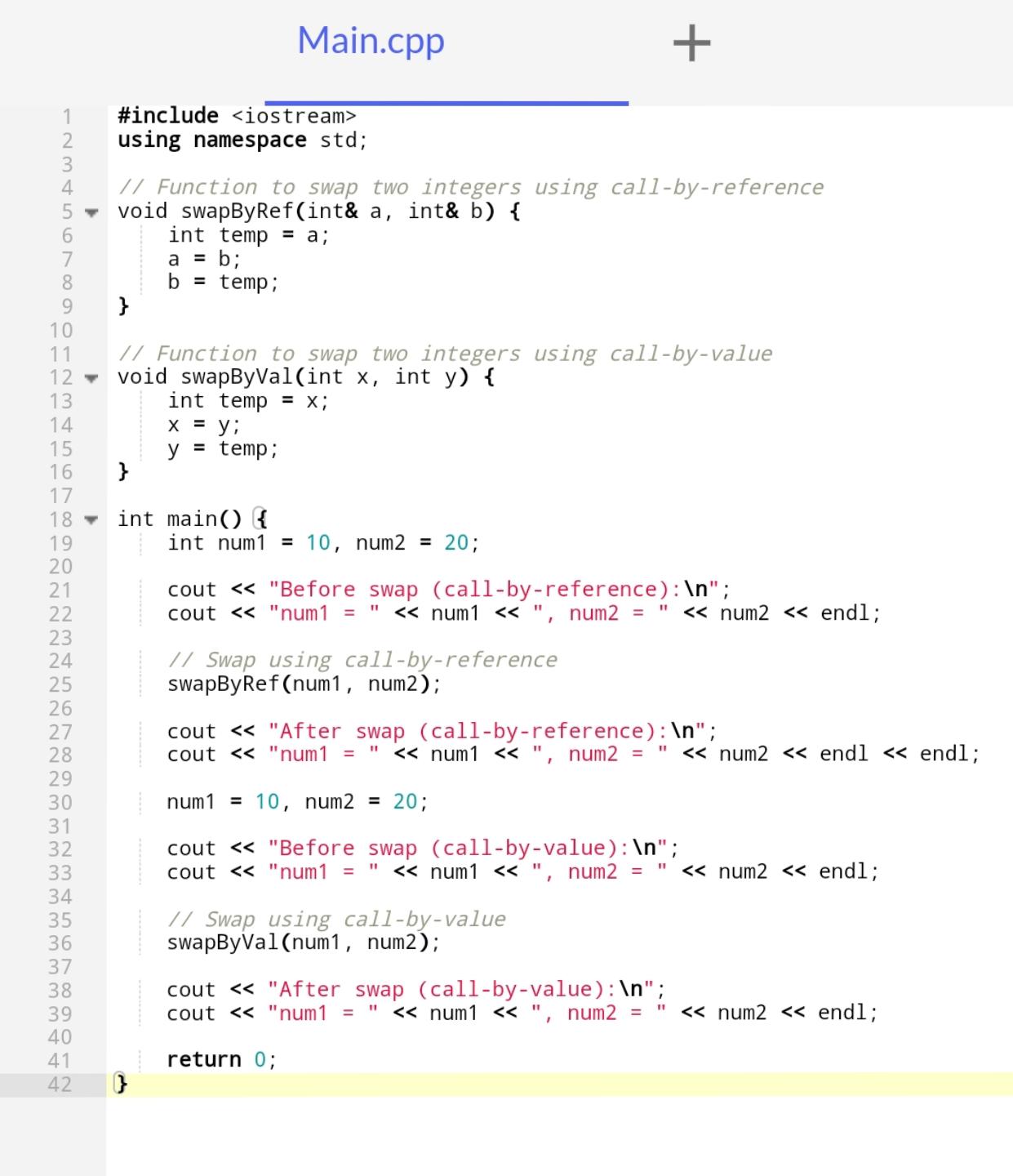
In summary, variables are user-defined names for data storage, while keywords are predefined terms with special meanings in the programming language.

* + 1. **Write a C++ program to check whether a given year is a leap year or not**

****

* + 1. **Write a C++ program to compute the sum, average and maximum or minimum element in an array.**

****

* + 1. **With the aid of a program construct demonstrate how parameter are passed to a function in C++**