AssignMent file

Foundation of c programming



BCA(Hons.) AI & DS

BATCH 2023-2027

SUBMITTED BY:- SUBMITTED TO:-

VAIDANSH THAPLIYAL MR. RISHI KUMAR

23091264

ASSIGNMENT

**QS-1** what are constant and variable , Types of constant , keywords, rules for identifiers , int , float , char , double , long , void .

**ANS:-**

**Constant and Variables** :

* **Constants** : Fixed values that do not change during program execution . Examples include numerical values like 5, 3.14 , or character literals like ‘A’
* **Variables** : Storage locations with a symbolic name ( an identifier) that can hold different values during the execution of a program.

**Types of Constants :**

* **Integers Constant :** Whole number without a decimal point , e.g., 5, -10.
* **Floating Point Constant :** Numbers with a decimal point or in exponential form, e.g., 3.14,-0.001.
* **Character Constants :** Single character enclosed in a single quotes ,e.g., ‘A’, ‘5’ .
* **String constant :** Sequence of character enclosed in double quotes , e.g., “Hello”.

**Keywords :** Reserved words with predefined meaning in a programming language. Examples include ‘int’ , ‘float’, ‘if’, ‘else’, etc.

**Rules for Identifiers :**

* 1. Must begin with a letter (uppercase or lowercase) or an underscore.
  2. The subsequent characters can be letters, digits, or underscores.
  3. Case-sensitive (e.g., count and Count are different identifiers).
  4. Should not be a keyword.
  5. Should not contain spaces or special characters (except underscore).

**Data Types:**

* **int:** Represents integers (whole numbers).
* **float**: Represents floating-point numbers (numbers with a decimal point).
* **char:** Represents a single character.
* **double:** Represents double-precision floating-point numbers.
* **long:** Represents long integers.
* **void:** Denotes the absence of a data type (used for functions that do not return a value).

**Q2**. Explain with examples Arithmetic Operators, Increment and

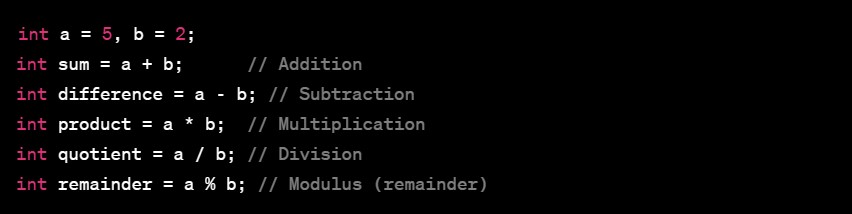
Decrement Operators, Relational Operators, Logical Operators, Bitwise Operators, Conditional Operators, Type Conversions, and expressions, Precedence, and associativity of operators.

**ANS :**

1. **Arithmetic Operators:**

Arithmetic operators perform basic mathematical operations.

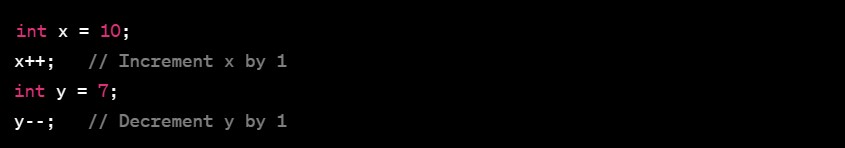
* + **Examples:**



1. **Increment and Decrement Operators:**

These operators increase or decrease the value of a variable.

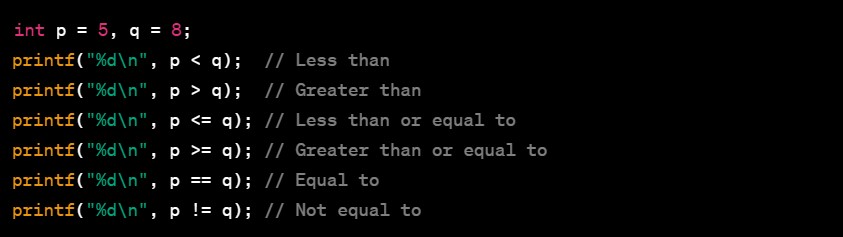
* + **Examples:**



1. **Relational Operators:**

Relational operators compare two values and return a Boolean result.

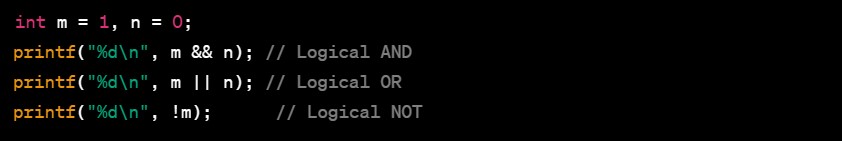
* + **Examples:**



1. **Logical Operators:**

Logical operators perform logical operations and return a Boolean result.

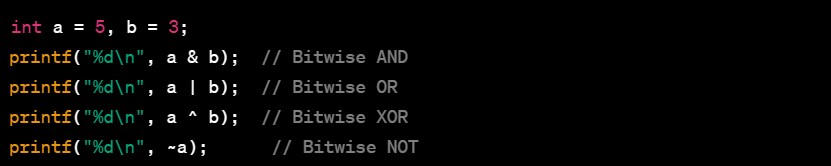
* + **Examples:**



1. **Bitwise Operators:**

Bitwise operators perform operations at the bit level.

* + **Examples:**



1. **Conditional Operator:**

The conditional operator is a shorthand for the if-else statement.

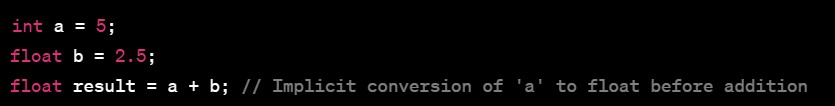
* + **Example:**



1. **Type Conversions and Expressions:**

Implicit and explicit type conversions may occur in expressions.

* + **Examples:**



1. **Precedence and Associativity of Operators:**

Operators have different precedence levels, and associativity determines the order of evaluation.

* + **Example:**



**Q3.** Explain with example conditional statement if , if-else , else if , nested if else .

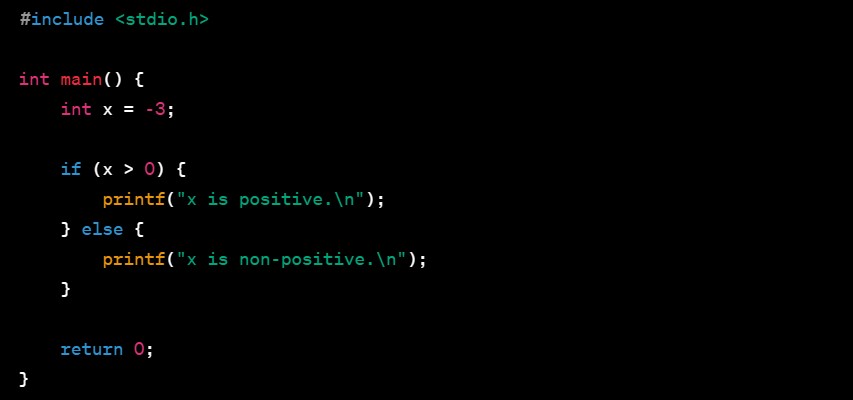
**ANS:**

Conditional statements are used in programming to make decisions based on certain conditions. These statements control the flow of a program by executing different blocks of code depending on whether a specified condition evaluates to true or false. Here are explanations and examples of various conditional statements in programming:

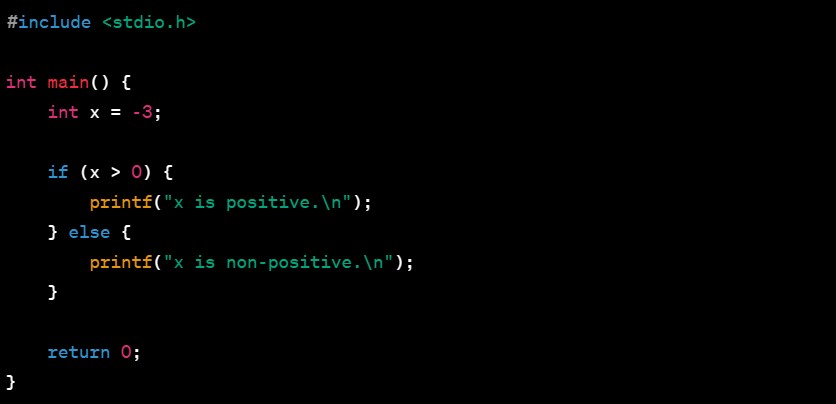
* 1. **if statement:** The **if** statement is used to execute a block of code if a specified condition is true.



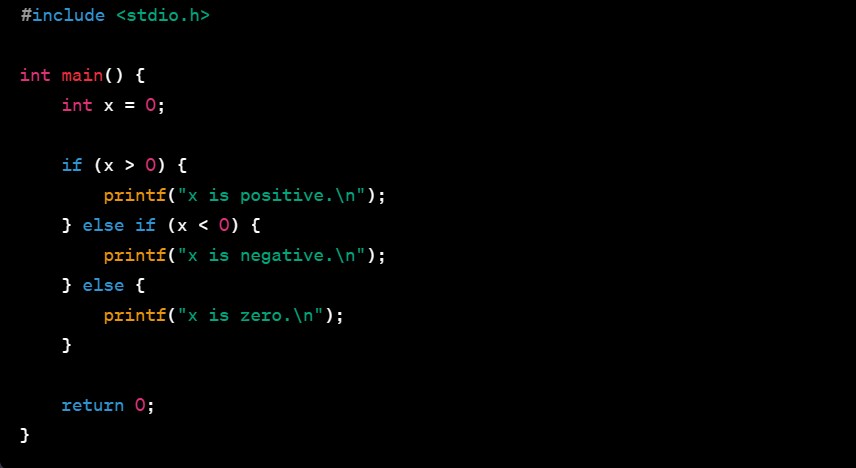
* 1. **if-else statement:** The if**-**else statement allows you to execute one block of code if a condition is true and another block if the condition is false.



* 1. **else if statement:** The elif statement allows you to check multiple conditions in sequence. If the first **if** condition is false, it checks the next condition, and so on.



* 1. **Nested if-else statement:** You can nest conditional statements within each other to create more complex decision structures.

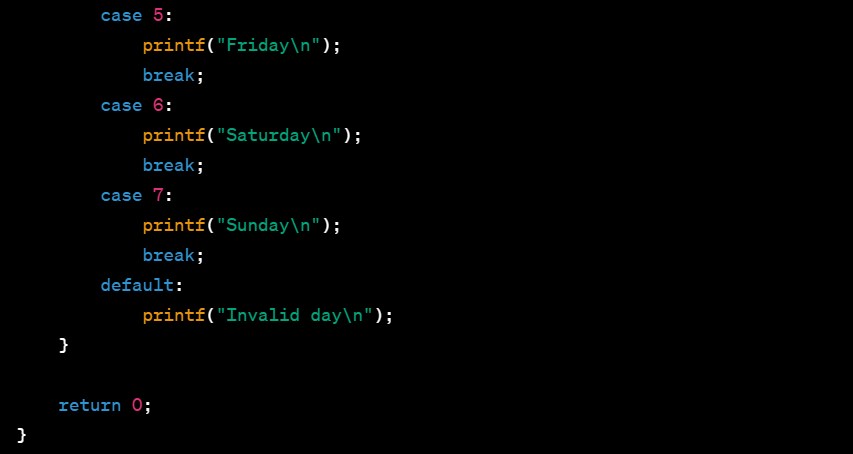


In the examples above, the conditions are simple numerical comparisons, but in real-world programming, conditions can involve more complex expressions and logical operations. Conditional statements are fundamental for controlling the flow of a program and making it respond to different scenarios.

**Q4.** Explain Switch case statement with example.

**ANS :**

The switch statement in programming is used to select and execute a block of code from multiple options based on the value of an expression. It provides an alternative to a series of if-else if statements when dealing with multiple conditions. Here's an explanation with an example in C programming:



In this example:

* The **‘switch’** statement evaluates the value of the variable day.
* The **‘case’** statements provide different options based on the possible values of day.
* The **‘break’** statements are crucial to exit the switch block after a case is matched. If a break statement is omitted, the control will "fall through" to the next case.
* The **‘default’** case is optional and is executed when none of the case values match the value of the expression.

If ‘day’ is 3, the output will be "Wednesday." If ‘day’ is not in the range 1 to 7, it will print "Invalid day" due to the default case.

Switch statements are particularly useful when you have a single expression to evaluate against multiple possible values, providing a cleaner and more concise structure than a series of if-else statements.

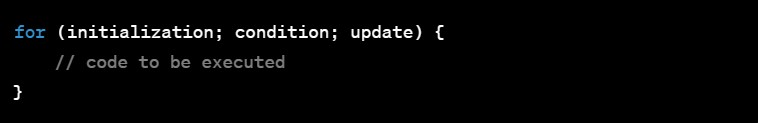
**Q5.** Explain Loops, for loops , while loop , do while loop with example.

Loops in programming are used to repeatedly execute a block of code as long as a certain condition is true. There are three main types of loops: **for** loop, **while** loop, and **do-while** loop.

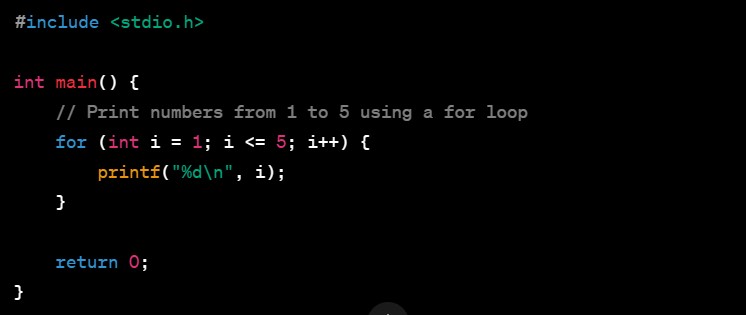
1. **for Loop:**

The **for** loop is used when you know the number of iterations in advance.

**Syntax:**



**Example (in C programming):**

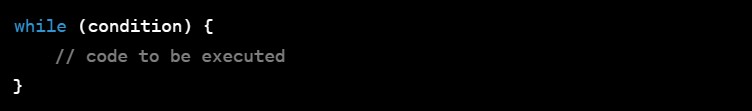


This program uses a **for** loop to print numbers from 1 to 5.

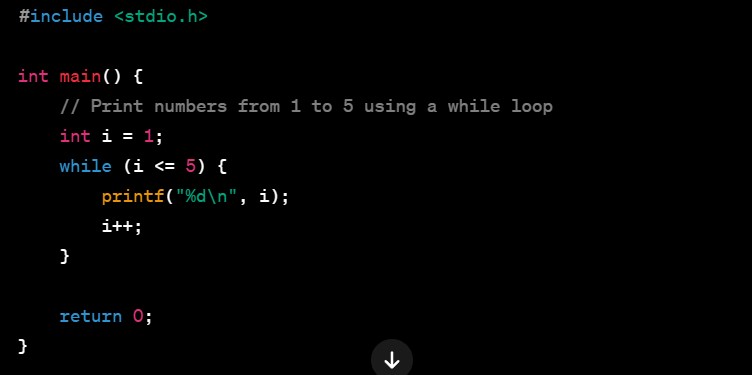
1. **while Loop:**

The **while** loop is used when you don't know the number of iterations in advance, and the loop continues as long as a condition is true.

**Syntax:**



**Example (in C programming):**

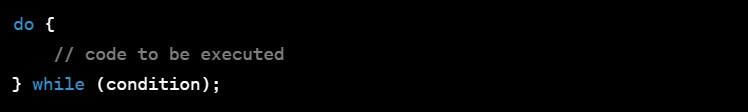


This program uses a **while** loop to print numbers from 1 to 5.

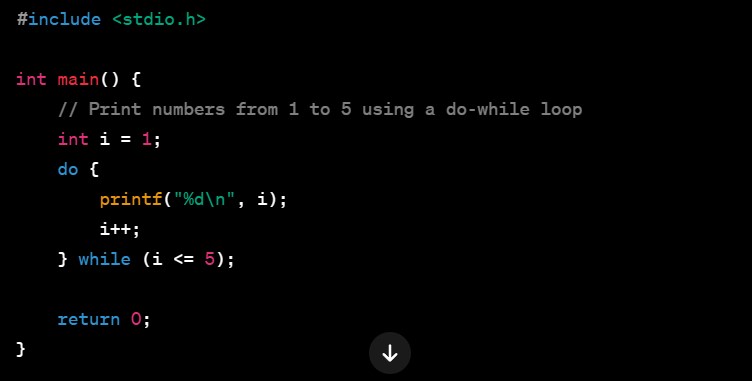
1. **do-while Loop:**

The **do-while** loop is similar to the **while** loop, but it guarantees that the block of code is executed at least once, as the condition is checked after the execution of the block.

**Syntax:**



**Example (in C programming):**



This program uses a **do-while** loop to print numbers from 1 to 5.

Loops are essential for repetitive tasks in programming, and the choice of which loop to use depends on the specific requirements of the task at hand.

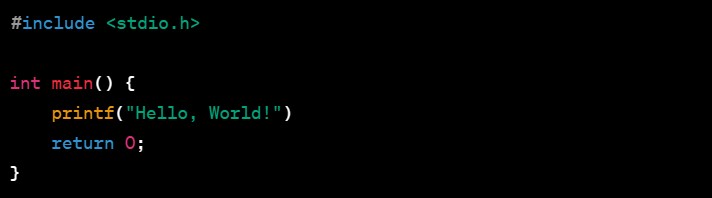
**Q6.** Explain with example debugging Importance, tools common errors : syntax logic , and runtime errors , debugging , and testing C programs.

**ANS :**

Debugging is a crucial aspect of programming that involves identifying and fixing errors in your code. The importance of debugging cannot be overstated, as it helps ensure that your program functions correctly and produces the expected output. There are several types of errors that programmers commonly encounter: syntax errors, logic errors, and runtime errors. Here's an overview along with examples and tips on debugging C programs:

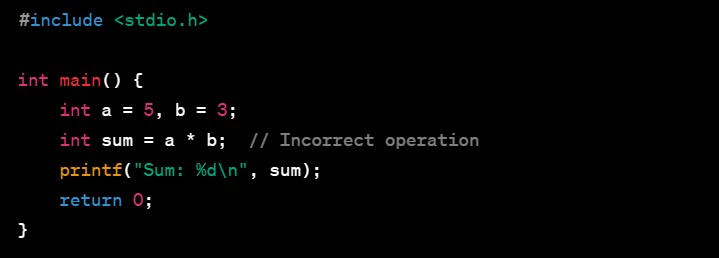
**Types of Errors:**

1. **Syntax Errors:** 
   * Description: Syntax errors occur when the rules of the programming language are not followed. These errors are caught by the compiler and prevent the program from being executed.
   * **Example:**



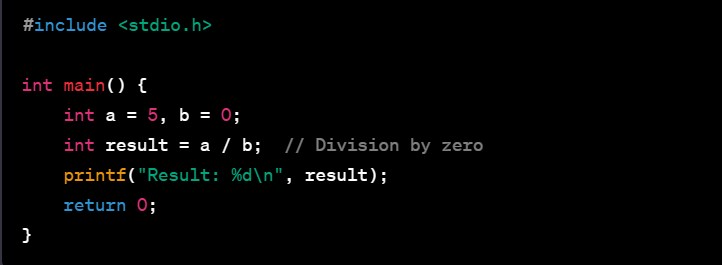
In this example, the semicolon is missing at the end of the printf statement, leading to a syntax error.

1. **Logic Errors:** 
   * Description: Logic errors occur when the code is syntactically correct but does not produce the intended results due to faulty logic.
   * **Example:**



Here, the programmer intended to calculate the sum of a and b, but the multiplication operator is used instead of the addition operator.

1. **Runtime Errors:** 
   * Description: Runtime errors occur during the execution of the program and can cause it to terminate abnormally. Examples include division by zero, accessing invalid memory, etc.
   * **Example:**



In this example, attempting to divide by zero will result in a runtime error.

**Debugging and Testing:**

1. **Use Debugging Tools:** 
   * Debuggers like GDB (GNU Debugger) can be invaluable for stepping through your code, setting breakpoints, and inspecting variables.
2. **Print Statements:** 
   * Use printf statements strategically to print the values of variables and identify where the code is not behaving as expected**.**
3. **Code Reviews:** 
   * Having another set of eyes review your code can help catch errors or suggest improvements.
4. **Testing:** 
   * Implement thorough testing, including unit tests and integration tests, to ensure that your program works correctly under various conditions.
5. **Learn from Errors:** 
   * When you encounter an error, understand its cause and use it as a learning opportunity to improve your coding skills.

Debugging is an ongoing process that requires patience and attention to detail. The ability to effectively debug code is a valuable skill for any programmer. Regular testing and debugging practices contribute to the creation of robust and reliable software.

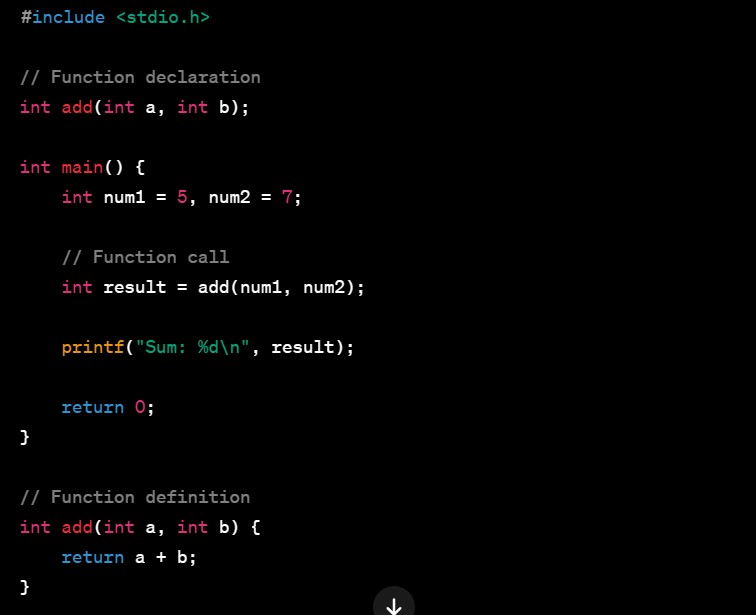
**Q7.** What is the User defined and pre-defined functions. explain with example call by value and call by reference.

**ANS :**

**User-Defined Functions:** User-defined functions in programming allow you to create your own functions to perform specific tasks.

These functions are defined by the programmer and provide modularity to the code. You can call these functions whenever needed.

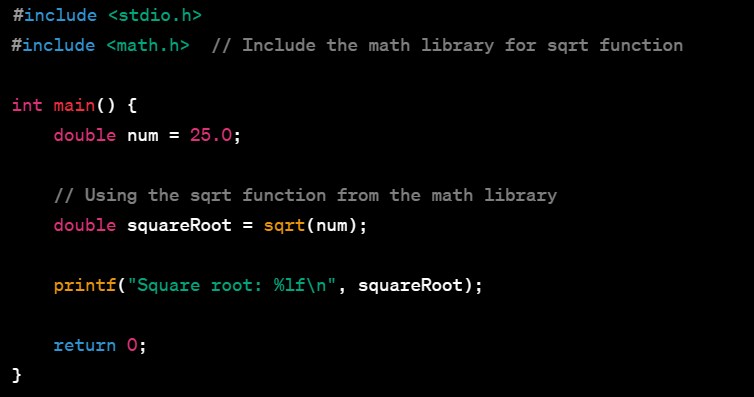
**Example of User-Defined Function:**



In this example, the add function is user-defined. It takes two parameters (a and b) and returns their sum. The add function is then called from the main function.

**Pre-Defined Functions**: Pre-defined functions, also known as built-in functions or standard library functions, are functions that come with the programming language or libraries. These functions are already defined and can be used by programmers to perform common tasks.

**Example of Pre-Defined Function:**

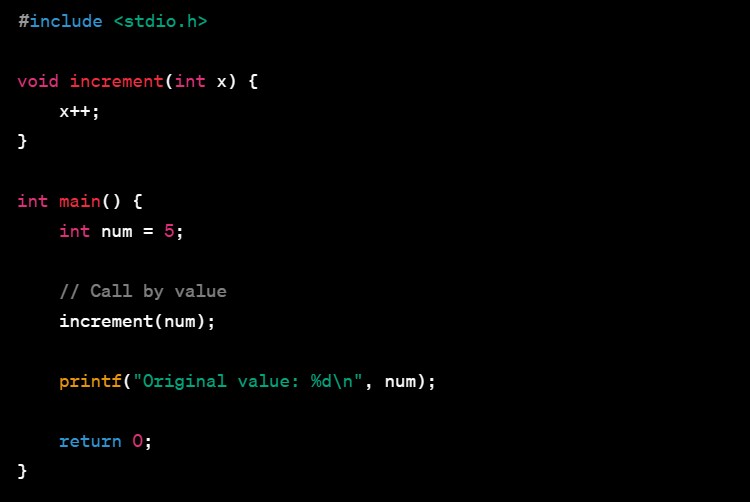


In this example, the sqrt function is a pre-defined function from the math library. It calculates the square root of a given number.

**Call by Value and Call by Reference:**

1. **Call by Value:** 
   * In call by value, the actual values of the arguments are passed to the function. Changes made to the parameters inside the function do not affect the original values outside the function.

**Example:**



In this example, the increment function takes num by value. Changes made to x inside the function do not affect the original value of num.

1. **Call by Reference:** 
   * In call by reference, the memory address (reference) of the actual arguments is passed to the function. Changes made to the parameters inside the function directly affect the original values outside the function.

**Example:**



In this example, the increment function takes the address of num as a parameter. Changes made to \*x inside the function directly affect the original value of num.

**Q8.** Explain with passing and returning argument to and from function. 2) Explain storage classes , automatic , static , register, external. 3) Write a program for two strings S1 and S2. Develop a C program for the following questions. A) Display a concatenated output of S1 and S2. B) Count the number of character and space in S1 and S2

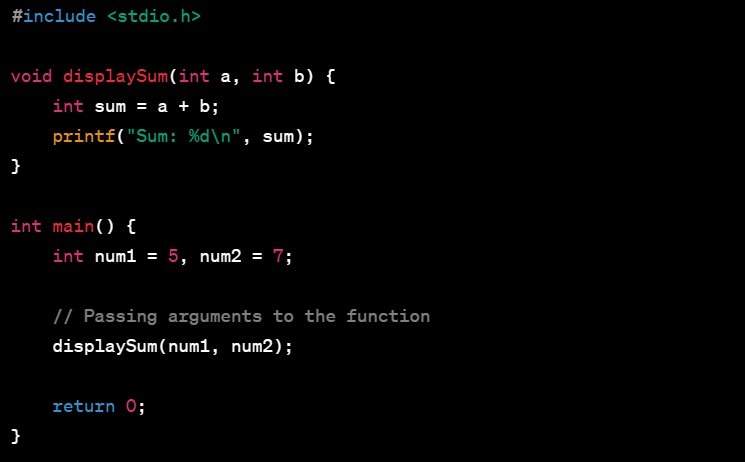
**ANS :**

**1) Passing and Returning Arguments to/from Functions:**

**Passing Arguments:**

When you pass arguments to a function, you are providing values or variables to be used inside the function**.**

**Example:**

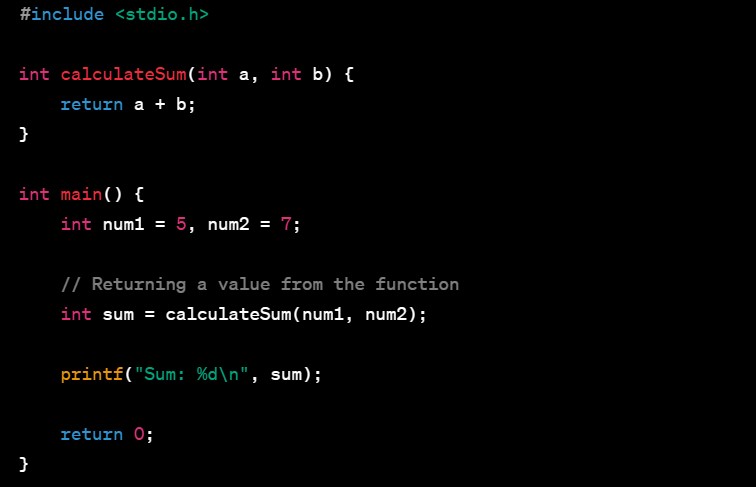


In this example, the display Sum function takes two parameters (a and b) as arguments.

**Returning Values:**

Functions in C can return values using the return statement.

**Example:**



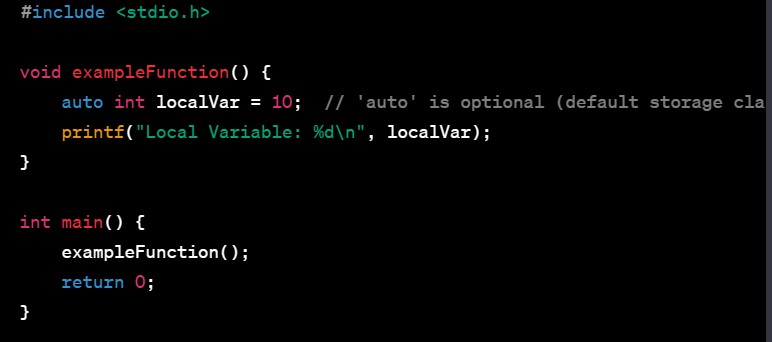
In this example, the calculate Sum function returns the sum of two numbers.

1. **Storage Classes:**

**Automatic Storage Class:**

* + Variables declared inside a function without using any storage class specifier have automatic storage.
  + They are created when the function is called and destroyed when it exits.

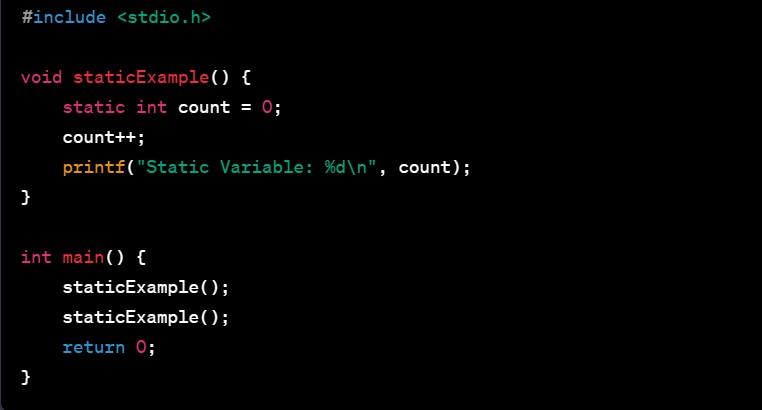
**Example:**



**Static Storage Class :**

* + The static storage class is used to declare variables that persist throughout the program's lifetime.
  + Static variables retain their values between function calls.

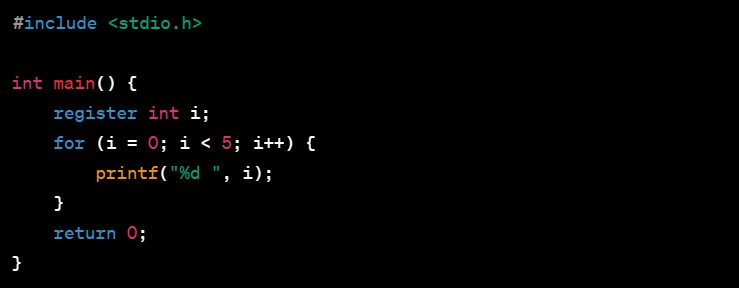
**Example:**



**Register Storage Class:**

* + The register storage class is used to define local variables that should be stored in a register.
  + The use of register is a hint to the compiler for optimization**.**

**Example:**

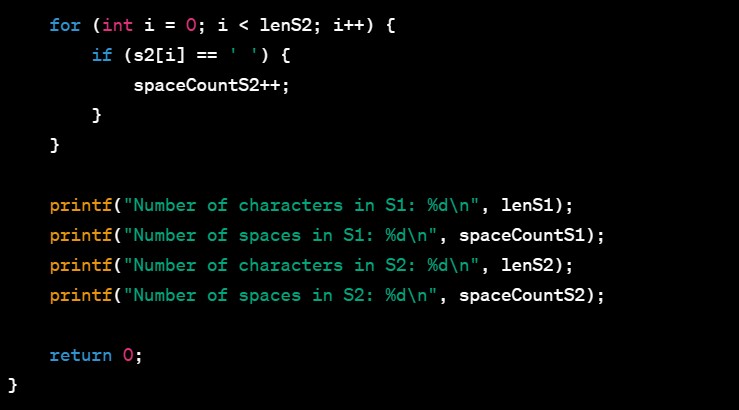
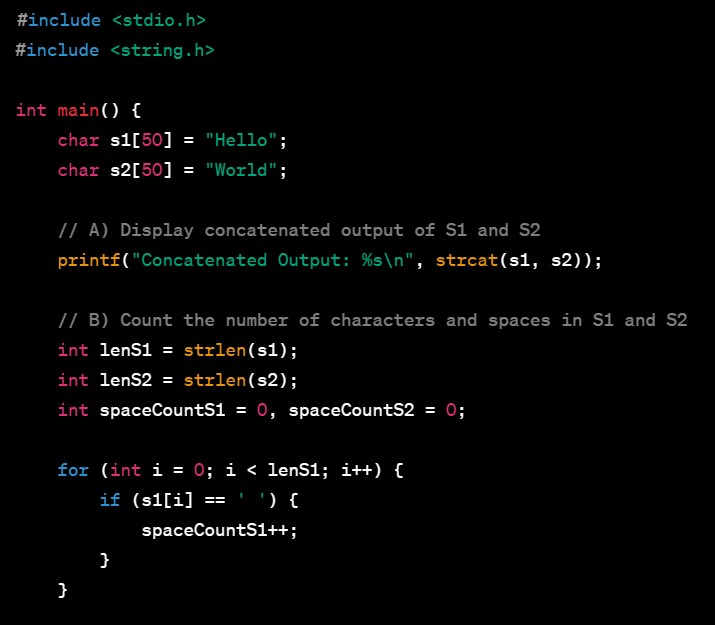


**External Storage Class:**

* + The extern storage class is used to declare a global variable that can be accessed by multiple source files.
  + **Example:**



1. **C Program for String Operations:**



This program performs the following tasks:

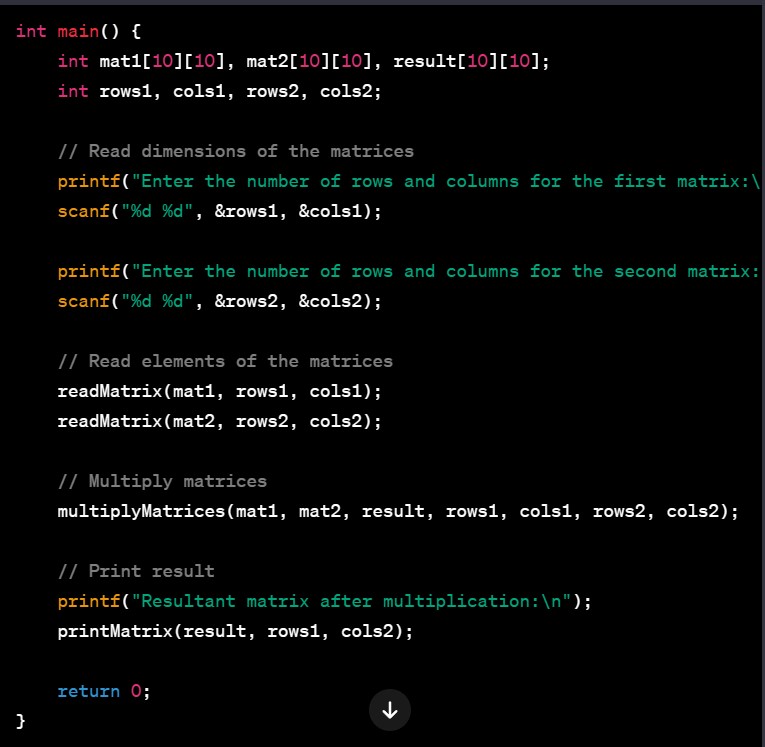
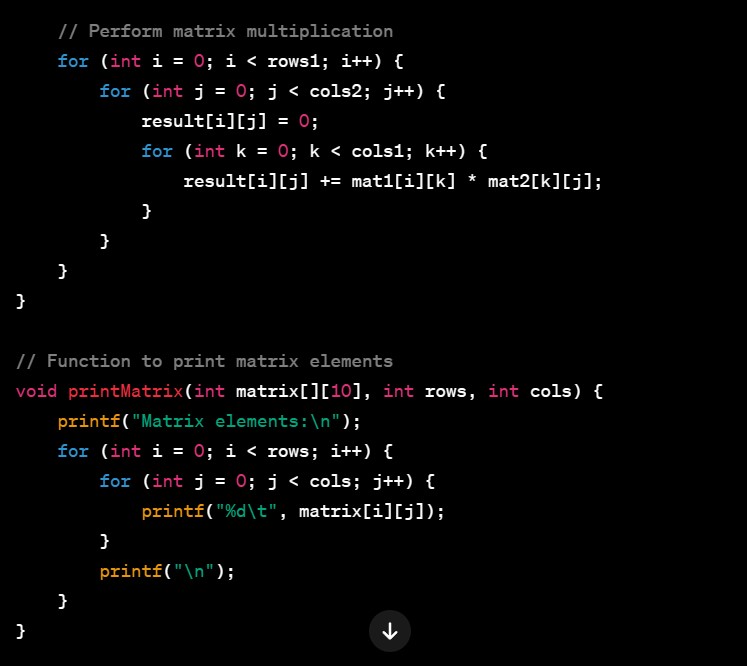
* + Concatenates **s1** and **s2** and displays the result.
  + Counts the number of characters and spaces in s1 and s2.

**Q9.** Explain with example ID array and multidimensional array. Consider two matrices of the size m & n. Implement matrix multiplication operation and display result using functions. Write three functions 1) Read matrix element 2) Matrix multiplication 3) Print matrix element.

ANS :

Sure, let's go through the implementation of matrix multiplication using functions in C. We'll define three functions: one for reading matrix elements, one for matrix multiplication, and one for printing matrix elements.





Explanation of the functions:

1. **Read Matrix function:**
   * Takes a matrix, number of rows, and number of columns as parameters.
   * Reads elements of the matrix from the user.
2. **Multiply Matrices function:**
   * Takes two matrices, their dimensions, and the resultant matrix as parameters.
   * Checks if matrices can be multiplied.
   * Performs matrix multiplication and stores the result in the resultant matrix. 3. **Print Matrix function:**
   * Takes a matrix and its dimensions as parameters.
   * Prints the elements of the matrix.

In the **main** function, the user is prompted to enter the dimensions and elements of two matrices. The matrix multiplication is performed using the **multiply Matrices** function, and the result is displayed using the **print Matrix** function.

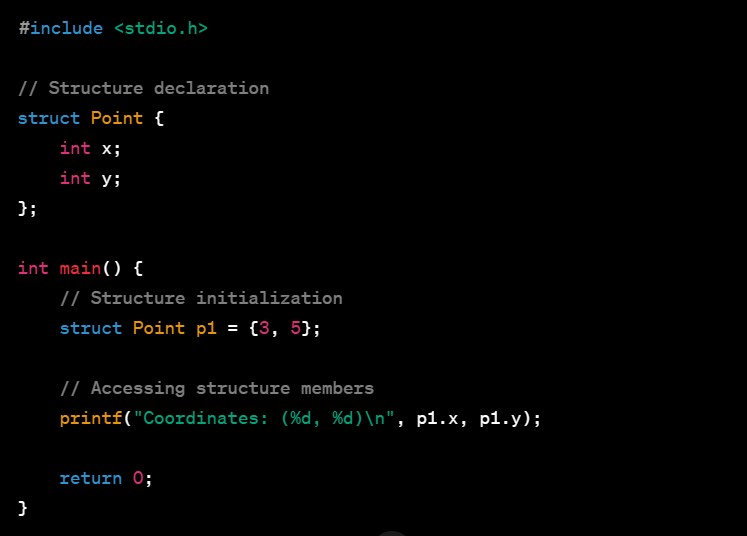
**Q10.** Explain with example with structure , declaration and initialisation, structure variables, array of structure, and use of typedef, passing structure to functions. Define union declaration, and Initialization passing structure to function. Explain difference between structure and union . Write a program on details of a bank account with the fields account number account holder’s name , and balance. Write a programme today 10 people‘s details and display the record with highest bank balance.

**ANS :**

**Structures in C:**

**Structure Declaration and Initialization:**

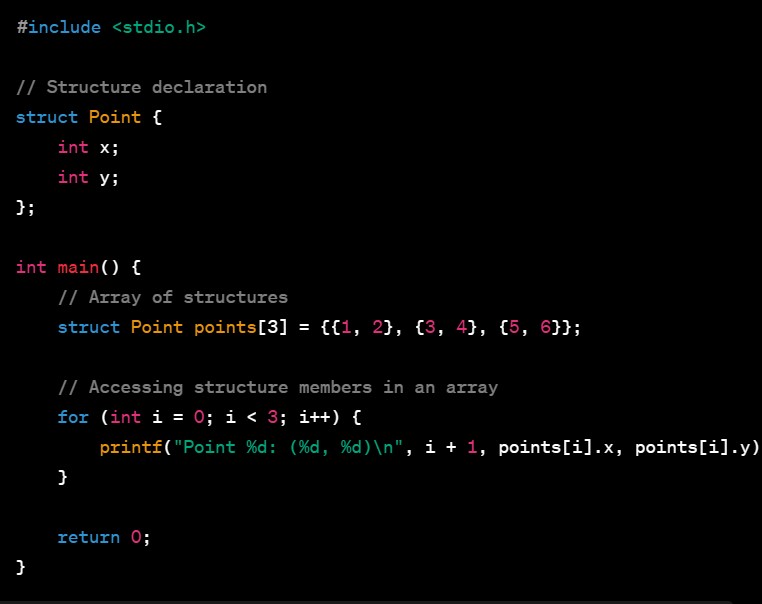
A structure in C is a composite data type that groups together variables under a single name. Here's an example of structure declaration and initialization:



In this example, a structure named Point is declared with two members (x and y). An instance of the structure p1 is declared and initialized with values.

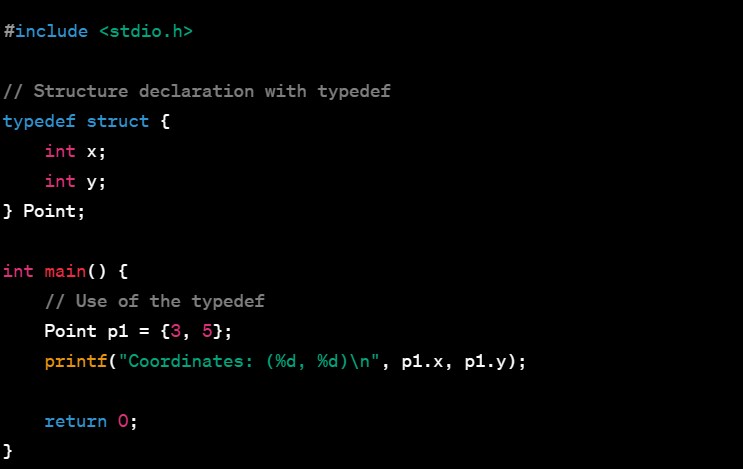
**Array of Structures:**

Structures can be used to create arrays, allowing you to store multiple instances of the same structure.



**Use of ‘typedef’ :**

typedef allows you to create aliases for data types, making your code more readable.



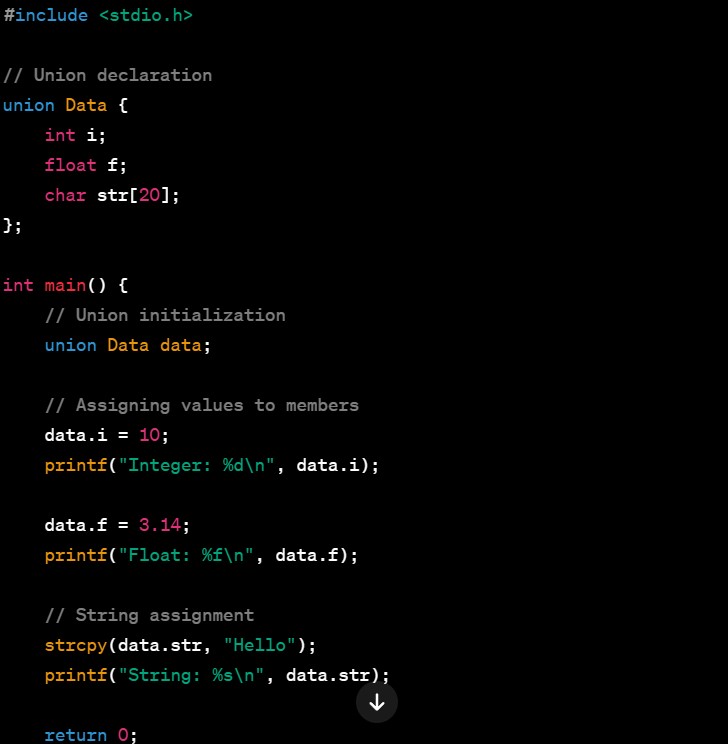
**Passing Structure to Functions:**

You can pass structures to functions, either by value or by reference.



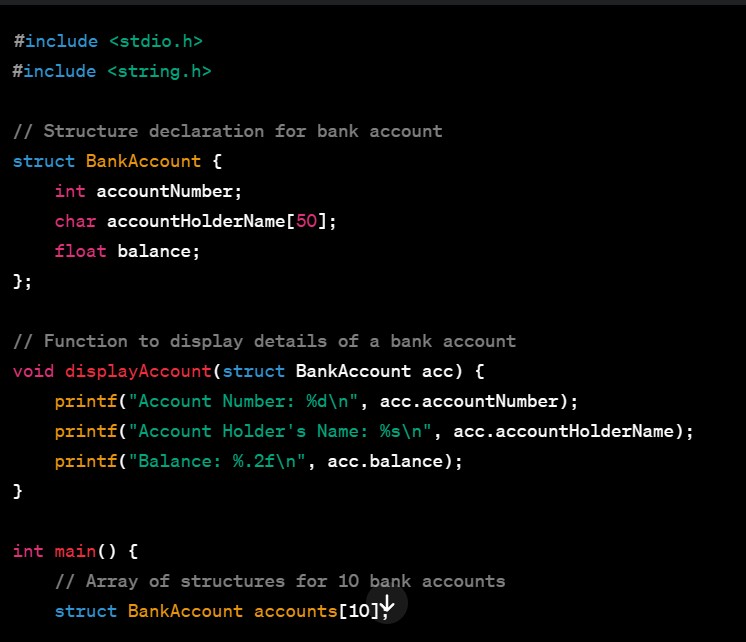
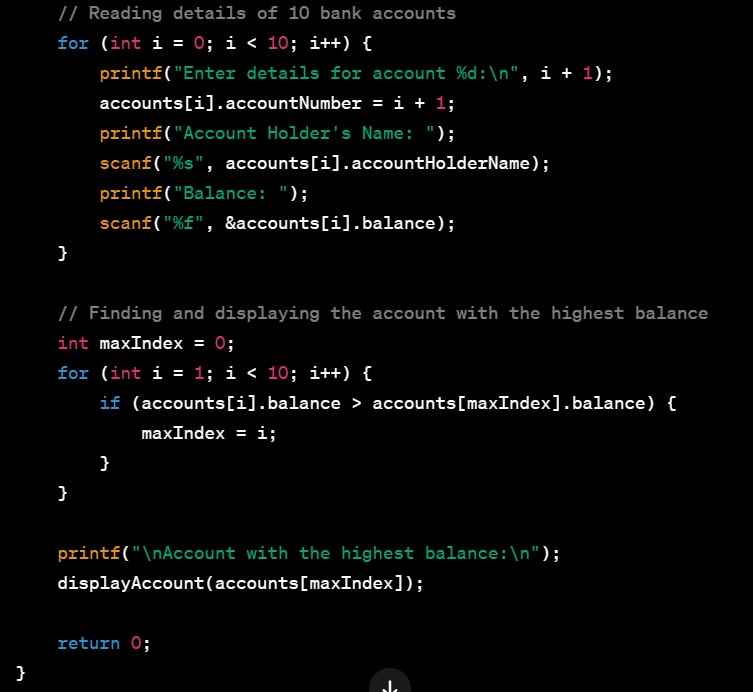
**Union in C:**

A union is similar to a structure, but it allows different data types to be stored at the same memory location. Only one member can be active at a time.



}

**Bank Account Program:**



In this program, a structure Bank Account is used to represent the details of a bank account. An array of 10 structures is used to store the details of 10 bank accounts. The program then finds and displays the account with the highest balance**.**