# ASSIGNMENT

# INTRODUCTION TO C


**BATCH: 2023-2026**

**BCA 1ST YEAR**

**SUBMITTED BY:**

**Drishti Juyal**

**SUBMITTED TO:**

**Mr. Rishi Kumar**

**Assistant Professor**

**CSIT, GEU,DEHRADUN**

**<u>Ques:1</u> What are Constants and Variable, Types of constants, keywords, Rules for Identifiers, int, float, char, double, long, void.**

**<u>Sol-</u> CONSTANTS** are the values that don't change during the program execution. **Examples** include numerical values like 5 or 3.14, are string literals.

-**VARIABLES** are containers for storing data values. They can be assigned different values during the program's execution.

**Types of constants includes:**

- Integer constants
- Floating point constants
- Character constants
- String constants

**KEYWORDS** are reserve words in a programming language, and they cannot be used as an identifier. **Examples** in C programming include (int, float, char, long and void.)

**RULES FOR IDENTIFIERS:**

It includes starting with a letter or underscore, being case-sensitive, and not using reserve words.

- **INT** is a keyword representing integer data type.
- **FLOAT** represents floating-point numbers (decimal numbers)
- **CHAR** is for character data type, storing a single character

- **DOUBLE** is used for double precision floating-point numbers.
- **LONG** is used for longer integers.
- **VOID** is a keyword used to indicate that a function does not return any value.

**Ques 2- Explain with examples arithmetic operators, increment and decrement operators, relational operators, logical operators, bitwise operator, conditional operators, type conversion, and expression, precedence, and associativity of operators.**

**Sol- ARITHMETIC OPERATORS** perform basic mathematical operations.

EXAMPLE:

```
#include<stdio.h
Int main () {
Int a=10, b=5;
Int sum=a+b ;
//addition
Int difference=a-b;
//subtraction
Int product=a*b;
//multiplication
Int quotient=a/b;
//division
```

```
Return 0;}
```

**INCREMENT AND DECREMENT OPERATORS:** increment (++) adds 1 to a variable, and decrement (--) subtracts 1.

**EXAMPLE:**

```
#include <stdio.h>

Int main () {

Int x=5;

X++;//increment x by 1

Int y=10;

y--;//decrement y by 1


Return 0;

}
```

**RELATIONAL OPERATORS:** Relational operators compare values and return true or false.

**EXAMPLE:**

```
bool a = true, b = false;
bool logicalAnd = (a && b); //
Logical AND
bool logicalOr = (a || b); //
Logical OR
bool logicalNot = !a; // Logical NOT
```

**BITWISE OPERATORS:** Bitwise operators perform operations at a bit level.

**EXAMPLE:**

```
int m = 5, n = 3;
int bitwiseAnd = (m & n); // Bitwise
AND
int bitwiseOr = (m | n); // Bitwise
OR
int bitwiseXor = (m ^ n); // Bitwise
XOR
```

**CONDITIONAL OPERATOR:** The conditional operator (? :) is shorthand for an if –else statements.

**EXAMPLE:**

```
int p = 10, q = 5;
int max = (p > q) ? p : q; // If p >
q, max = p, else max = q
```

## TYPES OF CONVERSION AND EXPRESSION:

- **IMPLICIT CONVERSION:** Automatically done by compiler.

```
int a = 5;
float b = a; // Implicit
conversion from int to float
```

- **EXPLICIT CONVERSION:** Programmer specifies the conversion.

```
float x = 3.14;
int y = (int)x; // Explicitly
converting float to int
```

- **EXPRESSIONS:** Combinations of values, variables, and operators that can be evaluated.

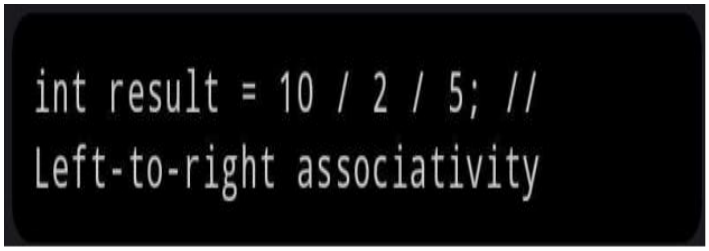
```
int a = 5, b = 3;
int result = a + b * 2; //
Expression involving addition and
multiplication
```

## PRECEDENCE AND ASSOCIATIVITY OF OPERATORS:

- **PRECEDENCE:** Determines the order in which operators are evaluated.

```
int result = 2 + 3 * 4; //
Multiplication has higher
precedence than addition
```

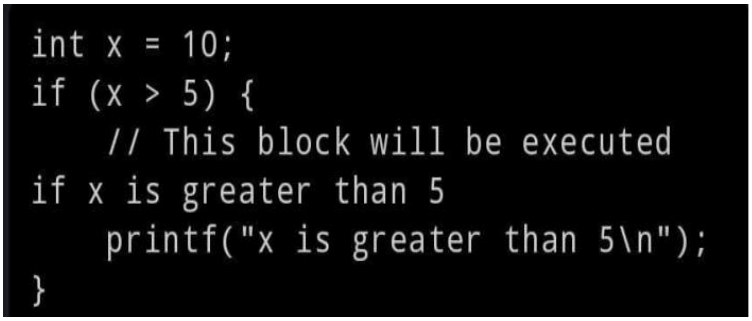
- **ASSOCIATIVITY:** Determines the order in which operators of the same precedence are evaluated.




**Ques 3-Explain with examples conditional statements if, if-else, elseif, nested if else.**

**Sol-** Conditional statements allow you to control the flow of a program based on certain conditions.

1-**IF STATEMENT:** The basic if statements execute a block of code if a specified condition is true.




2-**IF-ELSE STATEMENTS:** An if else statement executes one block of code if the condition is true and another if it’s false.

```
int y = 3;
if (y > 5) {
    printf("y is greater than 5\n");
} else {
    // This block will be executed
if y is not greater than 5
    printf("y is not greater than
5\n");
}
```

**3- ELSE IF STATEMENT:** An else-if statement allows you to check multiple condition in sequence.

```
int y = 3;
if (y > 5) {
    printf("y is greater than 5\n");
} else {
    // This block will be executed
if y is not greater than 5
    printf("y is not greater than
5\n");
}
```

**4. Nested If-else Statement:** You can nest if-else statements inside each other to handle more complex .

```
int z = 7;
if (z > 10) {
    printf("z is greater than
10\n");
} else if (z > 5) {
    // This block will be executed
if z is not greater than 10 but
greater than 5
    printf("z is greater than 5 and
less than or equal to 10\n");
} else {
    printf("z is 5 or less\n");
}
```

**Que.4- Explain switch case statement with example.**

**Solu.** The switch case statement is another way to make decisions in programming, especially when you have a

single expression that you want to compare against multiple possible values.

```c
#include <stdio.h>

int main() {
    int choice;

    // Prompt user for input
    printf("Enter a number (1-3): ");
    scanf("%d", &choice);

    // Switch statement
    switch (choice) {
        case 1:
            printf("You chose One\n");
            break;
        case 2:
            printf("You chose Two\n");
            break;
        case 3:
            printf("You chose Three\n");
            break;
        default:
            printf("Invalid choice\n");
    }

    return 0;
}
```

**Que.5- Explain loops, for loops, while loops, do while loop with example.**

**Solu.** Loops in programming allow you to execute a block of code repeatedly. There are three common types of loops: for, while, and do-while.

1. **For loop-**The for loop is used when you know in advance how many times the loop should run.

```c
#include <stdio.h>

int main() {
    // Example: Print numbers 1 to 5
using while loop
    int i = 1;
    while (i <= 5) {
        printf("%d ", i);
        i++;
    }

    return 0;
}
```

Output:

```
1 2 3 4 5
```

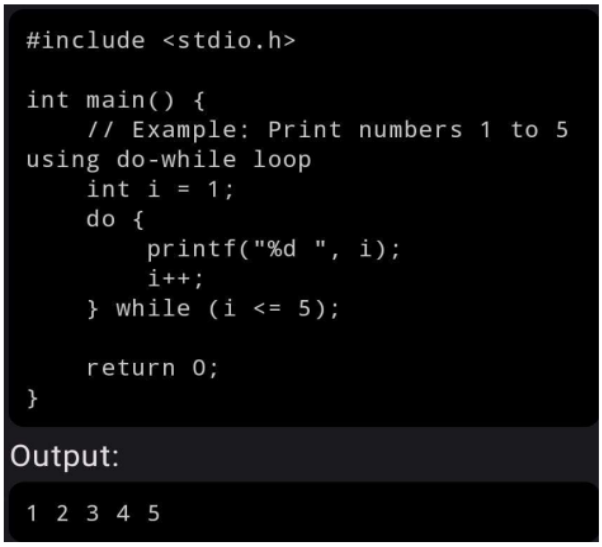
2. **While Loop:** The while loop continues to execute a block of code if the specified condition is true.

```c
#include <stdio.h>

int main() {
    // Example: Print numbers 1 to 5
using while loop
    int i = 1;
    while (i <= 5) {
        printf("%d ", i);
        i++;
    }

    return 0;
}
```

Output:

```
1 2 3 4 5
```

**3.Do-While Loop**: The do-while loop is like the while loop, but it always executes the block of code at least once before checking the condition.



```c
#include <stdio.h>

int main() {
    // Example: Print numbers 1 to 5
using do-while loop
    int i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i <= 5);

    return 0;
}
```

Output:

```
1 2 3 4 5
```

**Que.6- Explain with examples debugging importance, tools common errors: syntax, logic, and runtime errors, debugging, and Testing C programs.**

**Solu.** Debugging is the critical part of the software development process that involves identifying and fixing errors or bugs in code.

**Importance of debugging:**

 **Ensures Correct Functionality:** Debugging helps ensure that your program function as intended and produces the correct output.

 **Saves Time and Effort:** Identifying and fixing errors early in the development process saves time and effort in the long run.
 **Improves Code Quality:** Debugging promotes code quality by addressing issues such as syntax errors, logic errors, and runtime errors.

**<u>Common Types of Erros:</u>**

- **Syntax Errors:** These are errors that violate the rules of the programming language. They are usually caught by the compiler and must be fixed before the program can be executed.

**Example: int main () { printf(“hello, world!”)**

**}**

- **Logic Errors:** Logic errors occur when the code is syntactically correct but does not produce the expected results due to flawed logic.

**Example: int sum (int a,int b) {**

**Return a-b;//incorrect subtraction instead of addition**

- **Runtime Errors:** These errors occur during program execution and can lead to unexpected behavior, crashes, or abnormal termination.

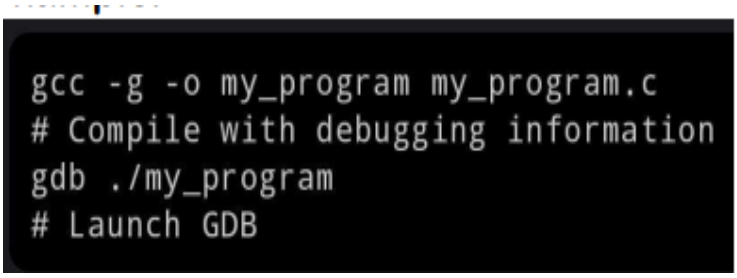
**Example: int main(){ int x=5,y=10; int result= x/y;**

**Debugging and Testing:**

**1. Print Statements:** Use *printf or fprintf* statements to print variable value at different points in your code for tracking program flow and variable values.

**2. GDB (GNU Debugger):** GDB is a powerful command-line debugger for C programs. It allows you to set breakpoints, inspect variables, and step through your code.

**Example:**




**3. IDE Debugger:** Integrated Development Environments (IDE's) such as Visual Studio Code, Eclipse or Code:: Blocks often come with built in debuggers that simplify the debugging process.

**4. Valgrind:** Valgrind is a tool for detecting memory leaks , memory corruption, and undefined memory used in c programs.

**Example: valgrind ./my_prpgram.c**

**5. Static Code Analyzers:** Tools like cppcheck or clang static analyzer can help identify potential issues in your code without executing it.

**Example: cppheck my_prpgram.c**

## Debugging Process:

**1.Reproduce the Issue:** Understand and reproduce the issue by identifying and reproduce the issue by identifying the conditions under which it occur.

**2.Isolate the Problem:** Narrow down the problem by using print statements or debugging tools to identify the specific code causing the issue.

**3.Use Debugging Tools:** Employ debugging tools like GDB,IDE debuggers, or Valgrind to step through the code , inspect variables, and identify errors.

**4.Fix the Issue:** Once the issue is identified ,modify the code to fi the problem.

**5.Test the Fix:** Test the modified code to ensure that the issue is resolved and that the changes do not introduce new erros.

Debugging is an iterative process , and the use of various tools and techniques can significantly improve the efficiency of identifying and fixing errors in the c.

**Que.7 What is the user defined and pre-defined functions. Explain with example call by value and call by reference.**

**Sol**. **User Defined Functions:** User defined functions are functions created by the programmer to perform specific tasks. They enhance code modularity and reusability.

**EXAMPLE:**

```
#include <stdio.h>

// User-defined function to calculate
the square of a number
int square(int num) {
    return num * num;
}

int main() {
    int result = square(5);
    printf("Square: %d\n", result);  //
Output: 25

    return 0;
}
```

**Here** ,square is a user defined function that calculates the square of a number.

**Predefined Function:** Pre defined function are built-in functions provided by the c programming language or its libraries. They offer common functionalities and are available for use without the need for programmer to implement them.

**EXAMPLE:**

```
#include <stdio.h>

int main() {
    char text[] = "Hello, World!";
    int length = strlen(text);
    printf("Length: %d\n", length);  //
Output: 13

    return 0;
}
```

In this ,strlen() is a predefined function that calculates the length of string.

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

```
#include <stdio.h>

// Function to swap two numbers using
call by value
void swapByValue(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    swapByValue(x, y);
    printf("After swap: x=%d, y=%d\n",
x, y);  // Output: x=5, y=10

    return 0;
}
```

In this ,swap By Value function does not modify the original values of x and y.

**Call by Reference:** In call by Reference , the memory addresses (references) of actual parameters and passed to the function. Changes made to the parameters inside the function affect the original values outside the function.

**EXAMPLE:**

```c
#include <stdio.h>

// Function to swap two numbers using
call by reference
void swapByReference(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    swapByReference(&x, &y);
    printf("After swap: x=%d, y=%d\n",
x, y);  // Output: x=10, y=5

    return 0;
}
```

**<u>Que.8-</u> 1) Explain with Passing and returning arguments 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 operations. A) Display a concatenated output of S1 and S2. B) Count the number of characters and empty spaces in S1 and S2.**

**<u>Solu.1-</u> Passing Arguments to a Function :** when you call a function in C , you can pass data to it through parameters.

These parameters act as placeholders for the values you provide when making the function call.

**Example:**

```c
#include <stdio.h>

// Function with parameters (arguments)
void printSum(int a, int b) {
    int sum = a + b;
    printf("Sum: %d\n", sum);
}

int main() {
    // Calling the function with
arguments
    printSum(3, 5);
    return 0;
}
```

**Returning Values from a Function in C**: Function in C return value to the calling code using the return statement. The return type is specified in the function declaration.

**EXAMPLE:**

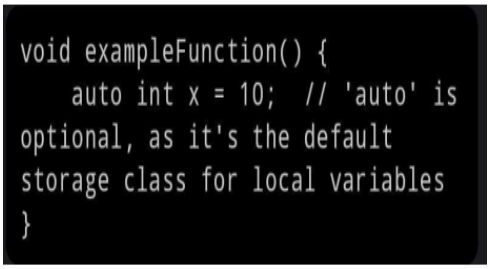
```c
#include <stdio.h>

// Function that returns the product of
two numbers
int multiply(int a, int b) {
    int result = a * b;
    return result;
}

int main() {
    // Calling the function and storing
the result
    int product = multiply(4, 6);
    printf("Product: %d\n", product);
    return 0;
}
```

## 2-Automatic Storage Class:

o Variables declared with auto keyword have automatic storage class.

o They are created when the block in which they are declared is entered, and destroyed when the block is exited.

**EXAMPLE:**



```
void exampleFunction() {
    auto int x = 10;  // 'auto' is
optional, as it's the default
storage class for local variables
}
```

- **Static Storage Class**:

o Variables declared with static keyword have static storage class.

o They are created when the program starts and exit throughout the program's execution.

o They retain their values between function calls.

o Example:

```
void exampleFunction() {
    auto int x = 10;  // 'auto' is
optional, as it's the default
storage class for local variables
}
```

- **Register Storage Class:**

o Variables declared with register keyword have register storage class.

o They are stored in CPU registers for quick access

o The use of register is a hint to the compiler, and it's optional- the compiler may choose to ignore it.

**Example:**

```
void exampleFunction() {
    register int z = 7;
}
```

- **External Storage Class:**

o Variables declared with extern keyword have external storage class.

o They are defined outside of any function and can be accessed globally.

o They are often used when a variable needs to be shared among multiple source files.

**Example:**

```
// File1.c
extern int globalVar;  //
Declaration of globalVar

// File2.c
int globalVar = 20;    //
Definition of globalVar
```

**3.Write a program for two strings S1 and S2. Develop a c program for the following operations.**

**(a) Display a concatenated output of S1 and s2.**

**(b) Count the number of characters and empty spaces in S1 and S2.**

**INPUT**

```
#include <stdio.h>
#include <string.h>
int main() {
    char s1[100], s2[100];
    printf("My name is Drishti Juyal");
    printf("Enter the first string: ");
    (s1);
    printf("Enter the second string: ");
    (s2);
    printf("Length of the first string:
        %lu\n", strlen(s1));
    printf("Length of the second string:
        %lu\n", strlen(s2));
    strcat(s1, s2);
    printf("Concatenated string: %s\n", s1
        );
    int cmpResult = strcmp(s1, s2);
    if (cmpResult == 0) {
        printf("The strings are equal.\n"
            );
    } else {
        printf("The strings are not equal
            .\n");
    }
    return 0;
}
```

Run

## OUTPUT

```
/tmp/Isi9foWK2A.o
My name is Drishti JuyalEnter the first string:
    Enter the second string: Length of the first
    string: 0
Length of the second string: 2
Concatenated string: @
The strings are equal.
```

## A- INPUT

```c
#include <stdio.h>
#include <string.h>
int main() {
    char s1[100], s2[100],
        concatenated[200];
    printf("My name is Drishti Juyal\n");
    printf("Enter the first string:\n ");
    (s1);
    printf("Enter the second string:\n ");
    (s2);
    strcpy(concatenated, s1);
    strcat(concatenated, s2);
    printf("Concatenated string: %s\n",
        concatenated);
    return 0;
}
```

## OUTPUT

```
My name is Drishti Juyal
Enter the first string:
Enter the second string:
Concatenated string: @
```

## B- INPUT

```c
#include <string.h>
int main() {
    char s1[100], s2[100];
    printf("My name is Drishti Juyal\n");
    printf("Enter the first string: ");
    (s1);
    printf("Enter the second string: ");
    (s2);
    int charCountS1 = 0, spaceCountS1 = 0;
    for (int i = 0; s1[i] != '\0'; i++) {
        if (s1[i] == ' ') {
            spaceCountS1++;
        } else {
            charCountS1++;
        }
    }
    int charCountS2 = 0, spaceCountS2 = 0;
    for (int i = 0; s2[i] != '\0'; i++) {
        if (s2[i] == ' ') {
            spaceCountS2++;
        } else {
            charCountS2++;
        }
    }
    printf("Number of characters in S1: %d\n", charCountS1);
```

**OUTPUT**

```
My name is Drishti Juyal
Enter the first string: Enter the second string: Number of characters in S1: 0
Number of empty spaces in S1: 0
Number of characters in S2: 2
Number of empty spaces in S2: 0
```

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

**Sol- 1D Array:** A 1D array is a linear collection of elements of the same data type, stored in contiguous memory locations. It is often used to represent a list or a sequence of elements.

**Example: #**include <stdio.h>

```
int main () {
int numbers [5] = {1, 2, 3, 4, 5};
printf("Elements of the 1D array: ");
for (int i = 0; i < 5; i++) {
printf("%d ", numbers[i]);
}
return 0;
}
```

In this example, numbers is a 1D array of integers with five elements. The program initializes the array with values 1, 2, 3, 4, and 5, and then it prints each element in a loop.

**Multidimensional Array:**

A multidimensional array is an array of arrays. It can be thought of as a table or a matrix with rows and columns, where each element is identified by its indices in multiple dimensions.

**Example:**

```
#include <stdio.h>
int main () {
int matrix [3][3] = {
{1, 2, 3},
{4, 5, 6},
```

```
    {7, 8, 9}
};
printf("Elements of the 2D array:\n");
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d ", matrix[i][j]);
    }
    printf("\n");
}
return 0;
}
```

In this example, matrix is a 2D array representing a 3x3 matrix of integers. The program initializes the array with values 1 to 9, and then it prints each element in a nested loop to display the matrix.

**Q10. Explain with example with Structure, Declaration, and Initialization, Structure Variables, Array of Structures, and Use of typedef, Passing Structures to Functions. Define union declaration, and Initialization Passing structures to functions. 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**

**program to read 10 people's details and display the record with the highest bank balance.**

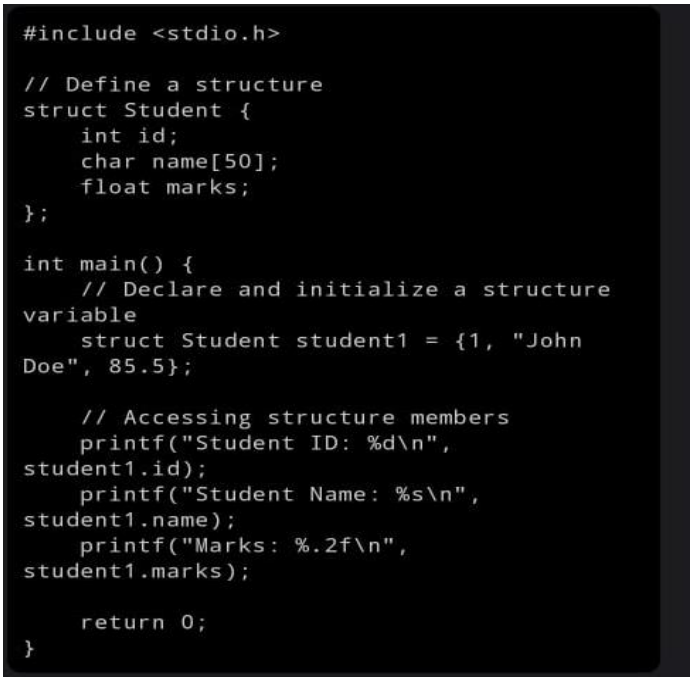
**<u>Sol-</u> STRUCTURE DECLARATION AND INITALIZATION:**

**Structure Declaration:**

struct Person {: Declares a structure named "Person" with three members: name (character array), age (integer), and height (floating-point number).

**Structure Variable Declaration and Initialization:**

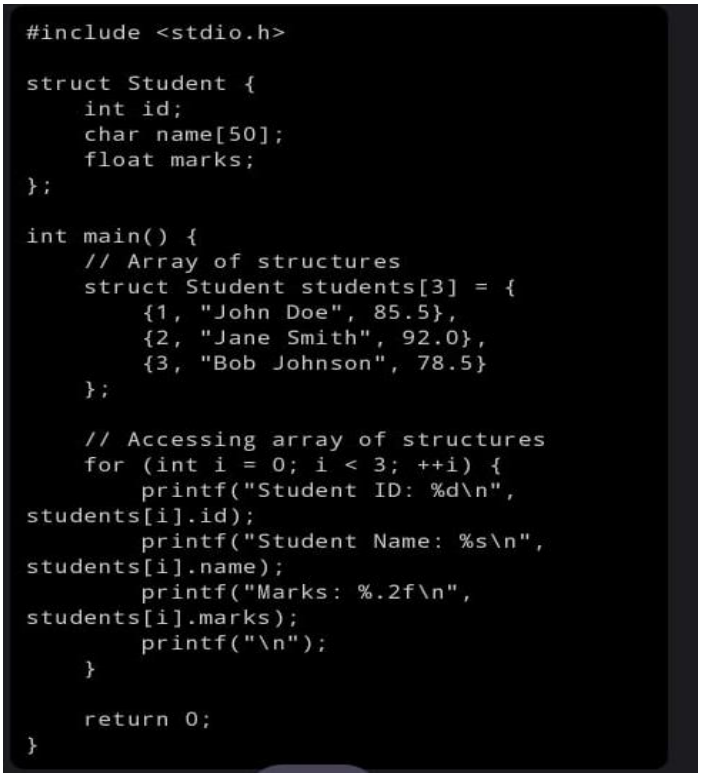
struct Person person1 = {"John Doe", 25, 175.5};Declares a structure variable named person1 of type struct Person. Initializes the structure members with the values provided in the curly braces {}**.** **EXAMPLE:**



```
#include <stdio.h>

// Define a structure
struct Student {
    int id;
    char name[50];
    float marks;
};

int main() {
    // Declare and initialize a structure
variable
    struct Student student1 = {1, "John
Doe", 85.5};

    // Accessing structure members
    printf("Student ID: %d\n",
student1.id);
    printf("Student Name: %s\n",
student1.name);
    printf("Marks: %.2f\n",
student1.marks);

    return 0;
}
```

**ARRAY OF STRUCTURE: struct** Person people [3];: Declares an array of structures named people with three elements, each of type struct Person.

**EXAMPLE:**



```c
#include <stdio.h>

struct Student {
    int id;
    char name[50];
    float marks;
};

int main() {
    // Array of structures
    struct Student students[3] = {
        {1, "John Doe", 85.5},
        {2, "Jane Smith", 92.0},
        {3, "Bob Johnson", 78.5}
    };

    // Accessing array of structures
    for (int i = 0; i < 3; ++i) {
        printf("Student ID: %d\n",
students[i].id);
        printf("Student Name: %s\n",
students[i].name);
        printf("Marks: %.2f\n",
students[i].marks);
        printf("\n");
    }

    return 0;
}
```

**USE OF TYPEDEF:** typedef struct Point Point;: Introduces an alias "Point" for the structure. Now, you can use Point instead of struct Point to declare variables. **EXAMPLE:**

```c
#include <stdio.h>

// Define a structure with typedef
typedef struct {
    int day;
    int month;
    int year;
} Date;

int main() {
    // Declare and initialize a structure
variable using typedef
    Date today = {27, 11, 2023};

    // Accessing structure members
    printf("Today's date: %d/%d/%d\n",
today.day, today.month, today.year);

    return 0;
}
```

## PASSING STRUCTURE TO FUNCTIONS:

## EXAMPLE:

```c
#include <stdio.h>

struct Point {
    int x;
    int y;
};

// Function to print the coordinates of a
point
void printPoint(struct Point p) {
    printf("Coordinates: (%d, %d)\n", p.x,
p.y);
}

int main() {
    // Declare and initialize a structure
variable
    struct Point point1 = {3, 5};

    // Pass structure to function
    printPoint(point1);

    return 0;
}
```

- **<u>Union Declaration and Initialization:</u>**

In C, a union is a user-defined data type that allows storing different data types in the same memory location

**EXAMPLE:**

```c
#include <stdio.h>

// Union declaration
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    // Union variable declaration and
initialization
    union Data data1 = {10}; // Initializes
the integer member
    union Data data2 = {3.14}; //
Initializes the float member
    union Data data3 = {.str = "Hello"}; //
Initializes the character array member

    // Accessing and displaying union
members
    printf("Value of integer: %d\n",
data1.i);
    printf("Value of float: %.2f\n",
data2.f);
    printf("String: %s\n", data3.str);

    return 0;
}
```

- **Passing Union to Functions:**

Passing a union to functions is similar to passing structures. You can pass the union by value or by using pointers**.**

## EXAMPLE:

```c
#include <stdio.h>

// Union declaration
union Data {
    int i;
    float f;
    char str[20];
};

// Function that takes a union by value
void printData(union Data d) {
    printf("Value of union: %d\n", d.i);
}

int main() {
    // Union variable declaration
    union Data data = {42};

    // Call the function and pass the union
by value
    printData(data);

    return 0;
}
```

# ***Difference Between Structure and Union:***

The key difference between a structure and a union lies in how they allocate memory:

## Structure:

Allocates memory for each member separately. Members share different memory locations. Useful when you need to store and access multiple pieces of information simultaneously.

## Union:

Allocates memory that is large enough to hold the largest member. All members share the same memory location. Useful when you only need to store and access one type of information at a time, conserving memory.

## INPUT

```c
#include <stdio.h>
#include <string.h>
struct BankAccount {
    int accountNumber;
    char accountHolderName[50];
    float balance;
};
void displayAccountDetails(struct BankAccount account) {
    printf("Account Number: %d\n", account.accountNumber);
    printf("Account Holder's Name: %s\n", account.accountHolderName);
    printf("Balance: %.2f\n", account.balance);
}
void deposit(struct BankAccount *account, float amount) {
    account->balance += amount;
    printf("Deposit of %.2f successful.\n", amount);
}
void withdraw(struct BankAccount *account, float amount) {
    if (amount <= account->balance) {
        account->balance -= amount;
        printf("Withdrawal of %.2f successful.\n", amount);
    } else {
        printf("Insufficient funds for withdrawal.\n");    }}
int main() {
    struct BankAccount myAccount = {123456, "Drishti juyal", 1000.0};
    printf("Initial Account Details:\n");
    displayAccountDetails(myAccount);
    deposit(&myAccount, 500);
    return 0;
```

## OUTPUT

```
Account Number: 123456
Account Holder's Name: Drishti juyal
Balance: 1000.00
Deposit of 500.00 successful.


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

## INPUT

```c
#include <stdio.h>
struct Person {
    char name[50];
    float balance;
};
int main() {
    struct Person people[10];
    for (int i = 0; i < 10; ++i) {
        printf("my name is drishti juyal\n");
        printf("Enter details for Person %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", people[i].name);
        printf("Balance: ");
        scanf("%f", &people[i].balance);
    }
    int maxIndex = 0;
    for (int i = 1; i < 10; ++i) {
        if (people[i].balance > people[maxIndex].balance) {
            maxIndex = i;
        }
    }
    printf("\nPerson with the Highest Balance:\n");
    printf("Name: %s\n", people[maxIndex].name);
    printf("Balance: %.2f\n", people[maxIndex].balance);

    return 0;
}
```

## OUTPUT

```
my name is drishti juyal
Enter details for Person 1:
Name: rishab
Balance: 300
my name is drishti juyal
Enter details for Person 2:
Name: aditi
Balance: 4500
my name is drishti juyal
Enter details for Person 3:
Name: ronak
Balance: 450
my name is drishti juyal
Enter details for Person 4:
Name: gaurav
Balance: 900
my name is drishti juyal
Enter details for Person 5:
Name: neeraj
Balance: 3500
my name is drishti juyal
Enter details for Person 6:
Name: devansh
Balance: 500
my name is drishti juyal
Enter details for Person 7:
Name: rishita
Balance: 560
my name is drishti juyal
Enter details for Person 8:
Name: nidhi
```