Module-2

Introduction to programming

Overview of C Programming

Q.1 THEORY EXERCISE:

Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

1. Origins (1960s):
    - Developed at Bell Labs by Ken Thompson and Dennis Ritchie.
    - Evolved from the B programming language, which was a derivative of BCPL.
    - Created to facilitate the development of the Unix operating system.
2. Rise in Popularity (1970s):
    - Gained traction with the release of Unix in 1971.
    - The publication of "The C Programming Language" by Ritchie and Brian Kernighan in 1978 standardized C's syntax and usage.
3. Standardization (1980s):
    - The need for a standardized version arose due to variations in implementations.
    - ANSI C (C89/C90) was established in 1989, introducing function prototypes and standard libraries.
4. Evolution in the 1990s:
    - Emergence of C++ introduced object-oriented features, but C remained relevant for systems programming.
    - C99 standard introduced new features like variable-length arrays and inline functions.
5. Modern Updates (2000s and Beyond):

- C11 introduced multi-threading support and improved Unicode handling.
- C18 focused on bug fixes and clarifications rather than new features.

6. Current Relevance:
    - C is foundational in operating systems, embedded systems, and high-performance applications.
    - Continues to influence modern languages (e.g., C++, C#, Objective-C).
    - Remains a popular choice for teaching programming concepts due to its simplicity and power.
7. Legacy:
    - C's adaptability and efficiency ensure its ongoing significance in the programming landscape.

Explain its importance and why it is still used today.

1. Foundation for Modern Languages:
    - Influences many languages (C++, C#, Java, Python), providing a solid base for learning.
2. Efficiency and Performance:
    - Known for low-level memory manipulation, making it ideal for system programming and performance-critical applications.
3. Portability:
    - Code can be compiled and run on various hardware platforms with minimal changes.
4. System-Level Programming:
    - Extensively used in operating systems, device drivers, and embedded systems due to direct hardware interaction.
5. Rich Ecosystem and Libraries:

- A vast array of libraries and frameworks enhances functionality and speeds up development.

6. Simplicity and Control:
   - Offers a straightforward syntax and high control over system resources, aiding debugging and optimization.
7. Educational Value:
   - Commonly used in computer science education to teach fundamental programming concepts and algorithms.
8. Community and Support:
   - A large, active community provides resources, tutorials, and forums for problem-solving.
9. Legacy Systems:
   - Many existing systems are written in C, necessitating ongoing use for maintenance and updates.
10. Continued Development:
    - Regular updates (C11, C18) introduce new features, ensuring relevance in modern programming.

• LAB EXERCISE:

o Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development

1. Embedded Systems: C is widely used in automotive control systems, medical devices, and consumer electronics, enabling efficient hardware interaction and real-time processing.
2. Operating Systems: C is fundamental in developing operating systems, including creating kernels, device drivers, and system libraries, due to its low-level capabilities.
3. Game Development: C is utilized in game engines and graphics programming, providing the performance needed for real-time rendering and complex simulations.

2. Setting Up Environment •

THEORY EXERCISE:

o Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.

Installing GCC Compiler

For Windows:

1. Download MinGW:
    - Go to the MinGW-w64 website.
    - Download the installer (e.g., mingw-w64-install.exe).
2. Run the Installer:
    - Choose the architecture (e.g., x86_64 for 64-bit).
    - Select the threads model (e.g., posix) and exception model (e.g., seh).
    - Choose the installation directory (e.g., C:\mingw-w64).
3. Add to System PATH:
    - Right-click on "This PC" or "My Computer" and select "Properties."
    - Click on "Advanced system settings."
    - Click on "Environment Variables."
    - Under "System variables," find the Path variable and click "Edit."
    - Add the path to the bin directory of MinGW (e.g., C:\mingw-w64\bin).
4. Verify Installation:
    - Open Command Prompt and type gcc --version. If installed correctly, it will display the GCC version.

Setting Up IDEs

1. DevC++

1. Download DevC++:
    - Go to the Dev-C++ website.
    - Download the latest version.
2. Install DevC++:
    - Run the installer and follow the prompts to install.
3. Configure Compiler:
    - Open DevC++.
    - Go to "Tools" > "Compiler Options."
    - Ensure that the selected compiler is GCC.
4. Create a New Project:
    - Go to "File" > "New" > "Project" to start coding in C.

2. Visual Studio Code (VS Code)

1. Download VS Code:
    - Go to the Visual Studio Code website.
    - Download and install the appropriate version for your OS.
2. Install C/C++ Extension:
    - Open VS Code.
    - Go to the Extensions view by clicking on the Extensions icon in the Activity Bar or pressing Ctrl +Shift +X.
    - Search for "C/C++" and install the extension provided by Microsoft.
3. Configure Build Tasks:
    - Create a new file with a .c extension.
    - Press Ctrl + Shift +B to configure build tasks.
    - Select "C/C++: gcc build active file" to create a tasks.json file.
4. Run Your Code:

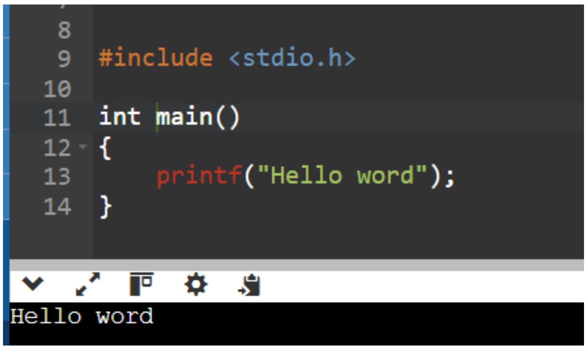
    - Use the terminal in VS Code to compile and run your C programs using GCC commands.

3. Code::Blocks

   1. Download Code::Blocks:
      - Go to the Code::Blocks website.
      - Download the version that includes MinGW (e.g., "codeblocks-20.03mingw-setup.exe").
   2. Install Code: :Blocks:
      - Run the installer and follow the prompts to install.
   3. Configure Compiler:
      - Open Code: :Blocks.
      - Go to "Settings" > "Compiler."
      - Ensure that the selected compiler is GCC.
   4. Create a New Project:
      - Go to "File" > "New" > "Project" and select "Console Application" to start coding in C.

● LAB EXERCISE:

o Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.



```c
#include <stdio.h>

int main()
{
        printf("Hello word");
}
```

```
Hello word
```

3.Basic Structure of a C Program

• THEORY EXERCISE:

O Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

- Basic Structure of C Program

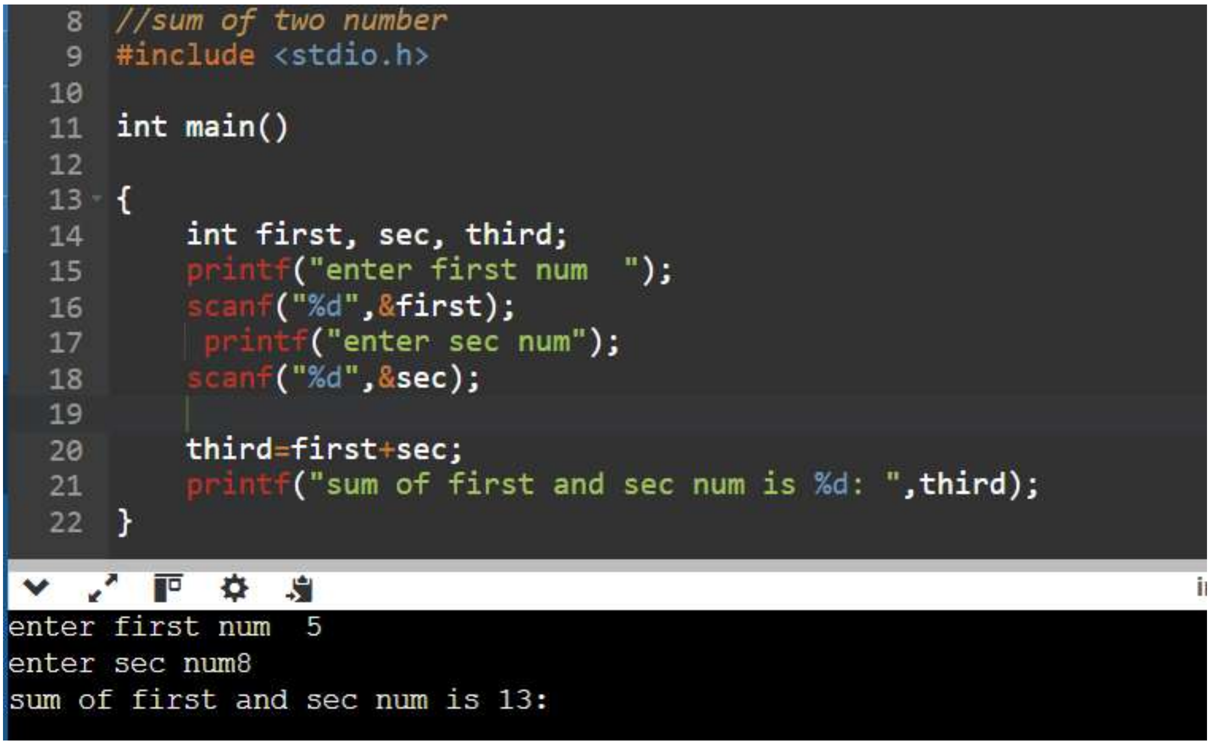
----

1) header file adding (to add library)

#include<stdio.h>
# : preprocessor
include : keyword to add some library file
<> : brackets to add header file.
stdio.h : Standard Input Output header file
            one header file for input & output

2) main()
-Function to start the execution of the code from here.

3) {
        ..... Block of code
  }

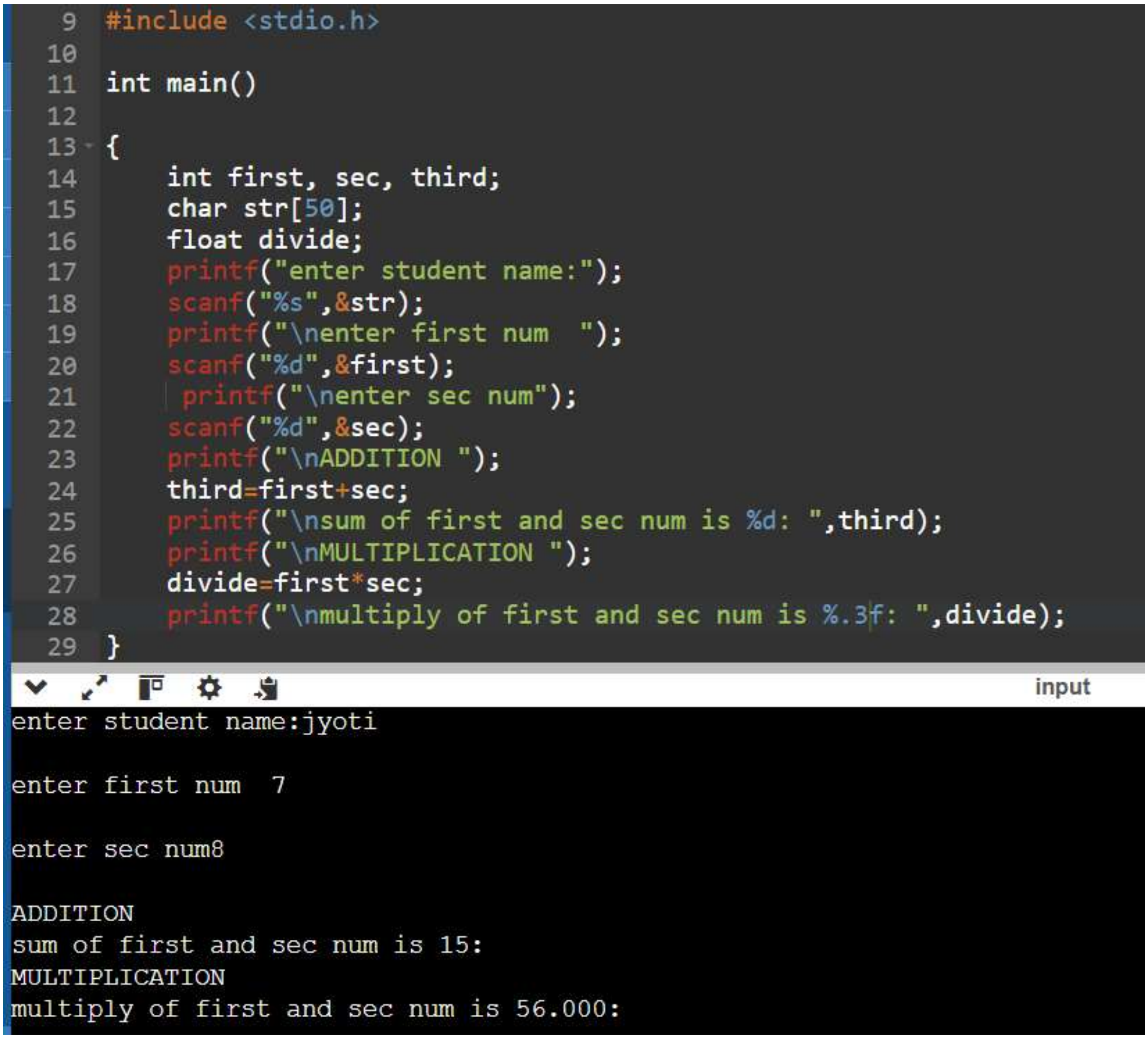


```c
//sum of two number
#include <stdio.h>

int main()

{
    int first, sec, third;
    printf("enter first num  ");
    scanf("%d",&first);
     printf("enter sec num");
    scanf("%d",&sec);

    third=first+sec;
    printf("sum of first and sec num is %d: ",third);
}
```

```
enter first num  5
enter sec num8
sum of first and sec num is 13:
```

• LAB EXERCISE:

o Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.



```c
#include <stdio.h>

int main()

{
    int first, sec, third;
    char str[50];
    float divide;
    printf("enter student name:");
    scanf("%s",&str);
    printf("\nenter first num  ");
    scanf("%d",&first);
     printf("\nenter sec num");
    scanf("%d",&sec);
    printf("\nADDITION ");
    third=first+sec;
    printf("\nsum of first and sec num is %d: ",third);
    printf("\nMULTIPLICATION ");
    divide=first*sec;
    printf("\nmultiply of first and sec num is %.3f: ",divide);
}
```

```
enter student name:jyoti

enter first num  7

enter sec num8

ADDITION
sum of first and sec num is 15:
MULTIPLICATION
multiply of first and sec num is 56.000:
```

4.Operators in C

• THEORY EXERCISE:

o Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

Operators :

1) Arithmetic Operators

+, -, *, /

2) Assignment / Short hand Operator.

+=, -=, *=, /=, %=

a=a+10 (a+=10)

3) Increment/Decrement Op. (inc by 1, dec. by 1)

++, --

unary op. : a++ (a=a+1)

----

one operand (a), one operator (+)

binary op. : a+b

Two operands (a, b) , one operator (+)

Increment:

Postfix : a++ (store->increment)

Prefix : ++a (increment->store)

Decrement :

Postfix : a-- (store->increment)

Prefix : --a (increment->store)

4) Relational operators/conditional/comparision

>, <, <=, >=, ==, !=   (= & ==)

a>b

a<b

a>=b

a >=50

a<=b

a==b

a!=b

5) Logical operators

-to combine expressions by one condition

&& - and :

 - all the conditions have to be true.

|| - or

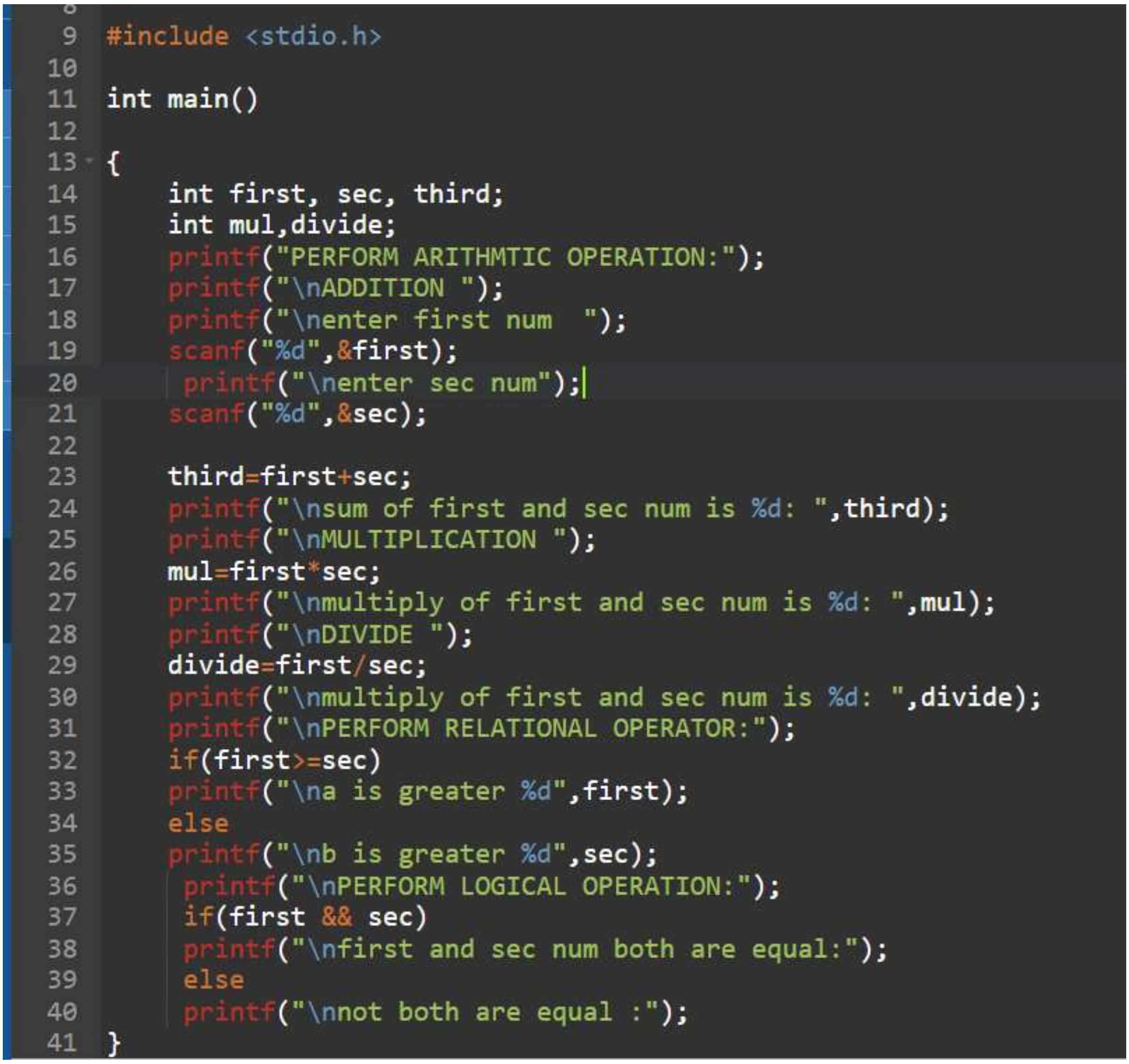
- One of the any condition has to be true.

!   - not

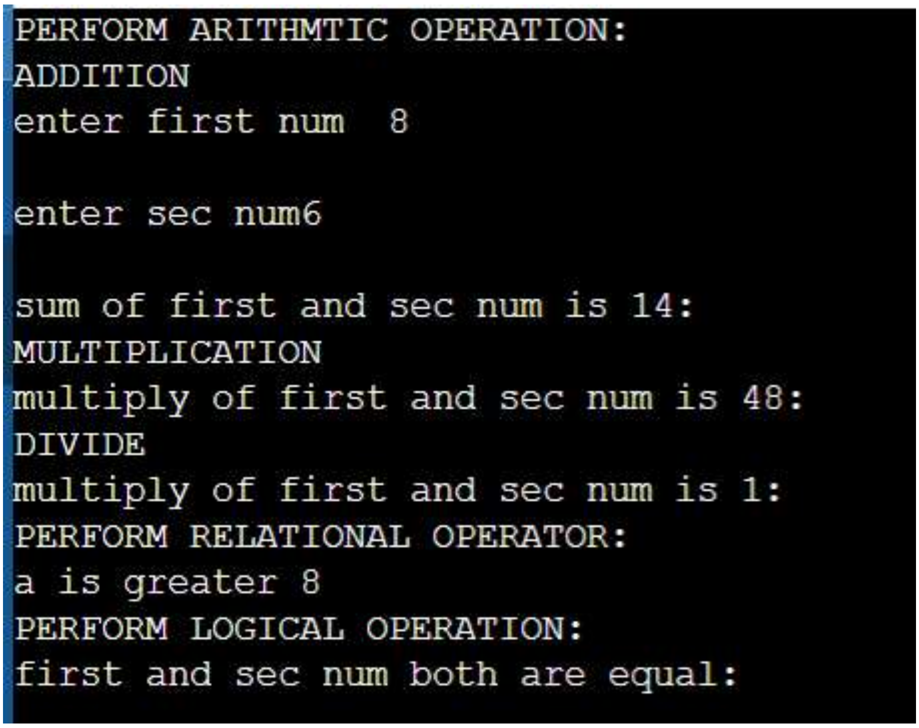
- true expression prove’s false

- **LAB EXERCISE:**

o Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.



```c
#include <stdio.h>

int main()

{
    int first, sec, third;
    int mul,divide;
    printf("PERFORM ARITHMTIC OPERATION:");
    printf("\nADDITION ");
    printf("\nenter first num  ");
    scanf("%d",&first);
     printf("\nenter sec num");
    scanf("%d",&sec);

    third=first+sec;
    printf("\nsum of first and sec num is %d: ",third);
    printf("\nMULTIPLICATION ");
    mul=first*sec;
    printf("\nmultiply of first and sec num is %d: ",mul);
    printf("\nDIVIDE ");
    divide=first/sec;
    printf("\nmultiply of first and sec num is %d: ",divide);
    printf("\nPERFORM RELATIONAL OPERATOR:");
    if(first>=sec)
    printf("\na is greater %d",first);
    else
    printf("\nb is greater %d",sec);
     printf("\nPERFORM LOGICAL OPERATION:");
     if(first && sec)
     printf("\nfirst and sec num both are equal:");
     else
     printf("\nnot both are equal :");
}
```



```
PERFORM ARITHMTIC OPERATION:
ADDITION
enter first num  8

enter sec num6

sum of first and sec num is 14:
MULTIPLICATION
multiply of first and sec num is 48:
DIVIDE
multiply of first and sec num is 1:
PERFORM RELATIONAL OPERATOR:
a is greater 8
PERFORM LOGICAL OPERATION:
first and sec num both are equal:
```

5.Control Flow Statements in C

- **THEORY EXERCISE:**

o Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

--> Conditional Statements in C

-Decision making statements

-Comparision Statements

----

1) if
2) if.. else
3) if ..else if.. else
4) nested if
5) switch.. case

1) if

-to execute one condition by checking.

```
syntax :  if(condition) //true
          {
                    block of code.
          }
```

2) if…else

-to evalate the condition by true or false

syntax :

```
if(condition) //true
{
        block of code for true condition.
}
else
{
        block of code for false condition
}
```

3) if ..else if.. else (evaluate high to low)

-else if ladder.

-to evaluate multiple conditions.

4) nested if

if inside if

syntax :

```
if(condition-1) //outer if
{
        if(condition-2)       //inner if
        {
                block of code condition-1 & 2
        }
        else
        {
                block of code for condition1
        }
}
else
{
}
```

5) switch ... case

-to make menu driven code.

-never use comparision operator

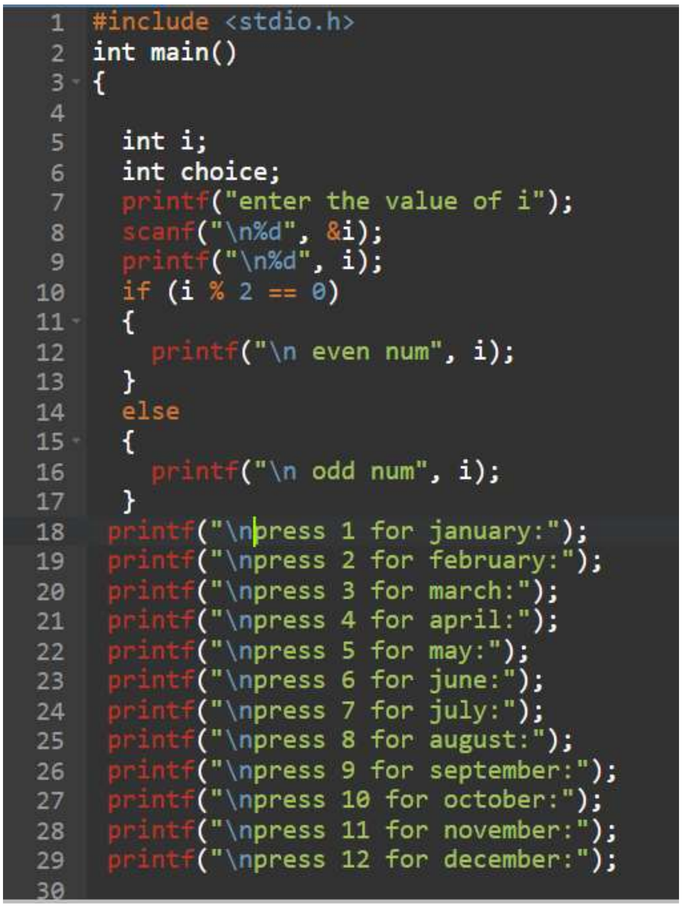
- keywords are used : switch, case, break, default

-switch case can be only applied on integer & character data types.

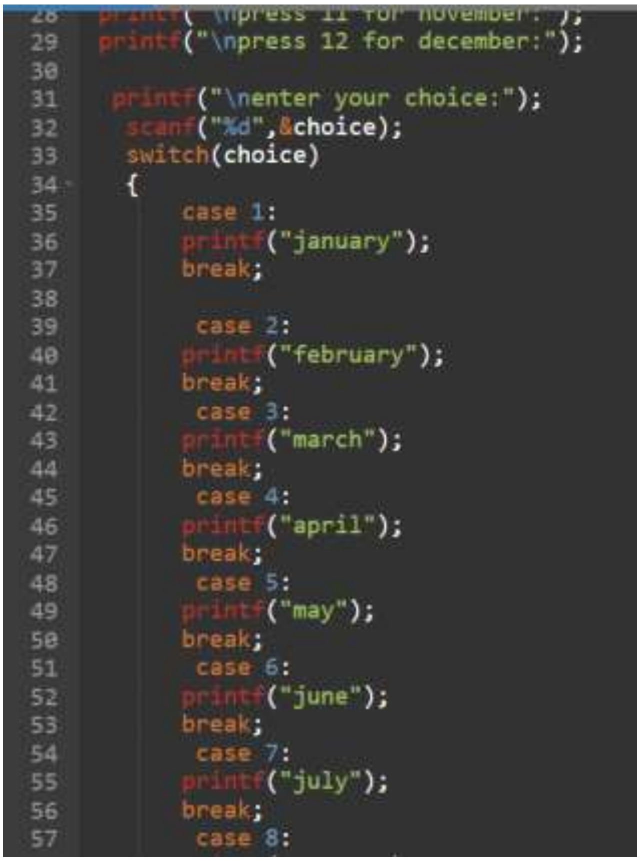
- **LAB EXERCISE:**

6 TOPS Technologies 2024

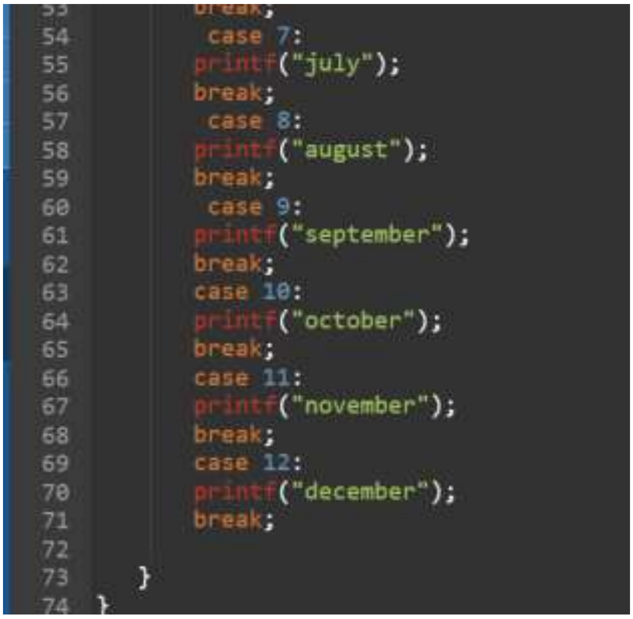
o Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user’s input (1 for January, 2 for February, etc.).



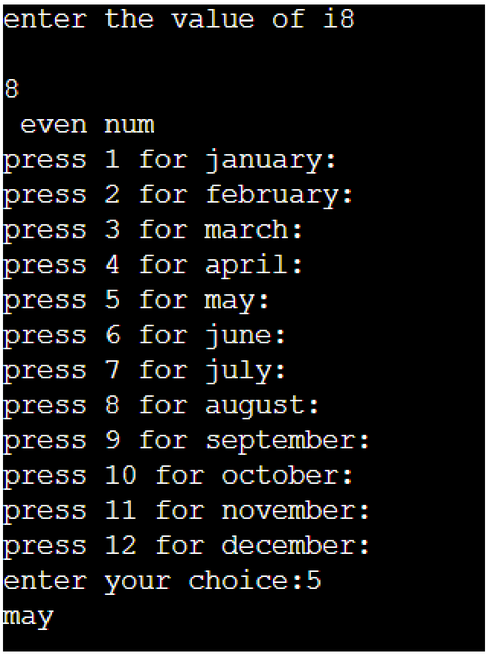
```
#include <stdio.h>
int main()
{

  int i;
  int choice;
  printf("enter the value of i");
  scanf("\n%d", &i);
  printf("\n%d", i);
  if (i % 2 == 0)
  {
    printf("\n even num", i);
  }
  else
  {
    printf("\n odd num", i);
  }
 printf("\npress 1 for january:");
 printf("\npress 2 for february:");
 printf("\npress 3 for march:");
 printf("\npress 4 for april:");
 printf("\npress 5 for may:");
 printf("\npress 6 for june:");
 printf("\npress 7 for july:");
 printf("\npress 8 for august:");
 printf("\npress 9 for september:");
 printf("\npress 10 for october:");
 printf("\npress 11 for november:");
 printf("\npress 12 for december:");
```



```
printf("\npress 11 for november:");
printf("\npress 12 for december:");

 printf("\nenter your choice:");
  scanf("%d",&choice);
  switch(choice)
  {
      case 1:
      printf("january");
      break;

       case 2:
      printf("february");
      break;
       case 3:
      printf("march");
      break;
       case 4:
      printf("april");
      break;
       case 5:
      printf("may");
      break;
       case 6:
      printf("june");
      break;
       case 7:
      printf("july");
      break;
       case 8:
```



```
      break;
       case 7:
      printf("july");
      break;
       case 8:
      printf("august");
      break;
       case 9:
      printf("september");
      break;
      case 10:
      printf("october");
      break;
      case 11:
      printf("november");
      break;
      case 12:
      printf("december");
      break;

  }
}
```



```
enter the value of i8

8
 even num
press 1 for january:
press 2 for february:
press 3 for march:
press 4 for april:
press 5 for may:
press 6 for june:
press 7 for july:
press 8 for august:
press 9 for september:
press 10 for october:
press 11 for november:
press 12 for december:
enter your choice:5
may
```

6.Looping in C

• THEORY EXERCISE:

o Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

1. While Loop

- Syntax:

```
while (condition) {
   // Code to execute
}
```

- Key Points:
    - Checks the condition before executing the loop.
    - May not run at all if the condition is false initially.
- Best For: Situations where the number of iterations is unknown and depends on a condition (e.g., reading user input until a specific value).

2. For Loop

- Syntax:

```
for (initialization; condition; increment)
{
   // Code to execute
}
```

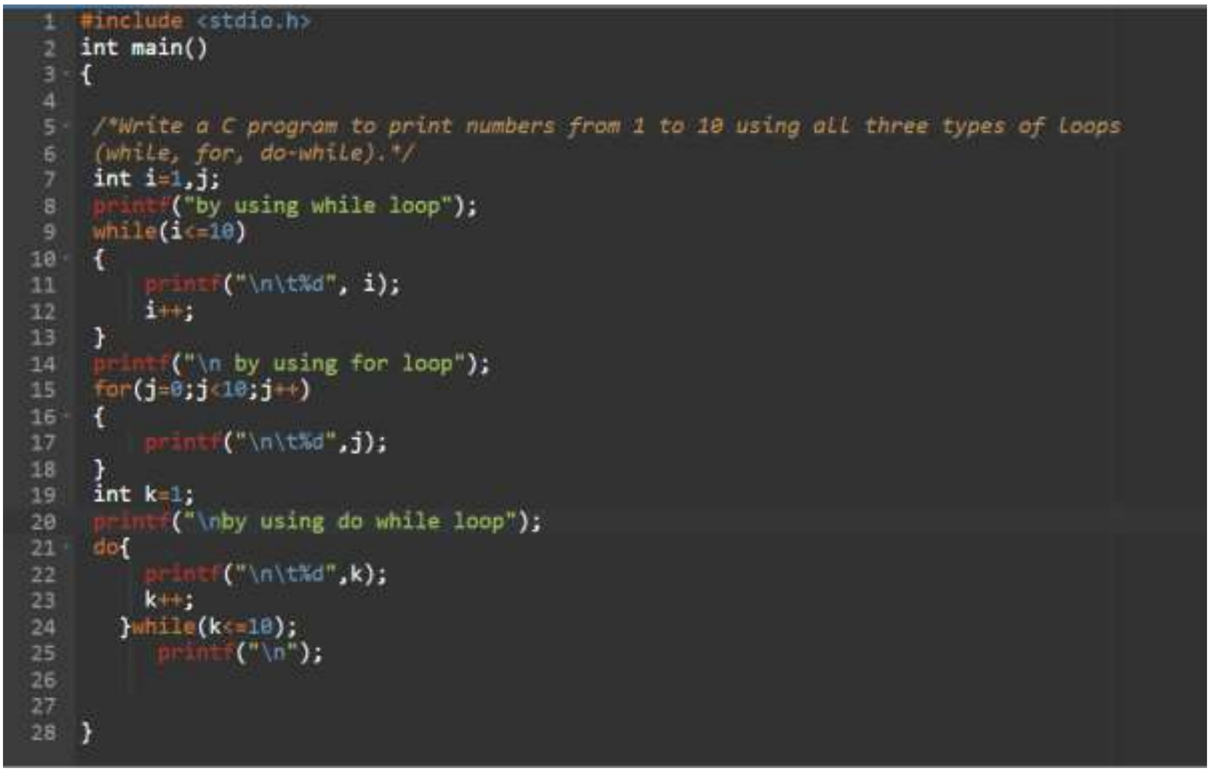
- Key Points:

    - Combines initialization, condition check, and increment in one line.
    - Runs a specific number of times based on the condition.
- Best For: Count-controlled iterations where the number of loops is known (e.g., iterating through an array).
  3. Do-While Loop
- Syntax:

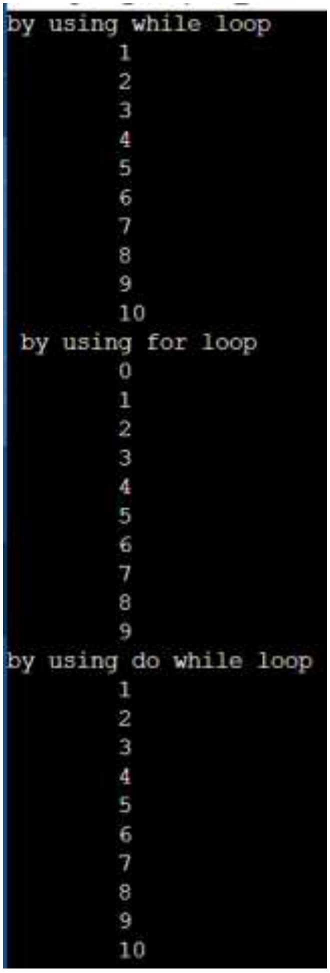
```
do {
   // Code to execute
 } while (condition);
```

• LAB EXERCISE:

o Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-wh



```c
#include <stdio.h>
int main()
{

 /*Write a C program to print numbers from 1 to 10 using all three types of loops
 (while, for, do-while).*/
 int i=1,j;
 printf("by using while loop");
 while(i<=10)
 {
     printf("\n\t%d", i);
     i++;
 }
 printf("\n by using for loop");
 for(j=0;j<10;j++)
 {
     printf("\n\t%d",j);
 }
 int k=1;
 printf("\nby using do while loop");
 do{
     printf("\n\t%d",k);
     k++;
   }while(k<=10);
      printf("\n");


}
```



```
by using while loop
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
 by using for loop
        0
        1
        2
        3
        4
        5
        6
        7
        8
        9
by using do while loop
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
```

7.Loop Control Statements

• THEORY EXERCISE:

o Explain the use of break, continue, and goto statements in C. Provide examples of each.

-- Control Statements :

-break
to terminate or break the code.
rest of the code will not be executed.

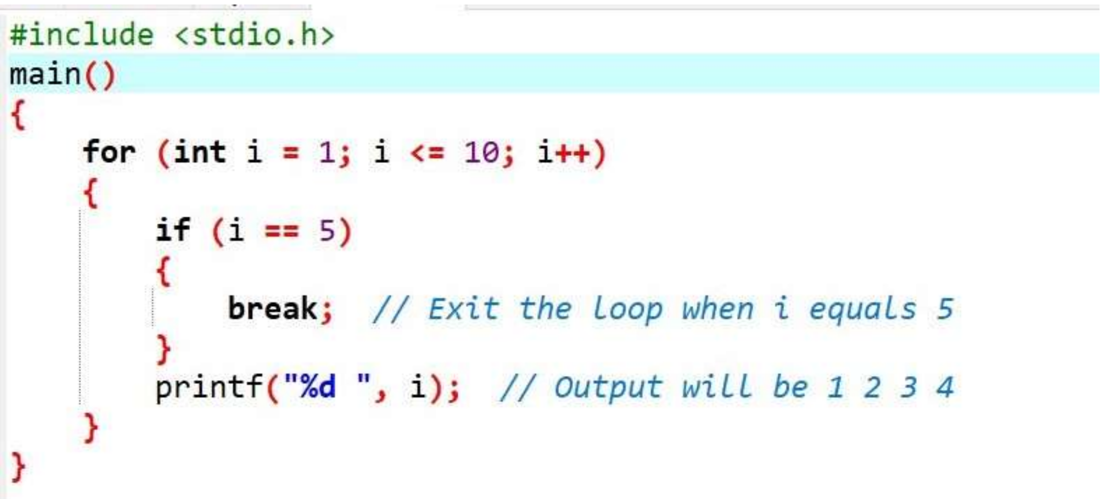
-continue
to skip the specific iteration.
rest of the code will be executed.

-goto (without loop)
to define one label & repeate the code by goto that label.

LAB EXERCISE:

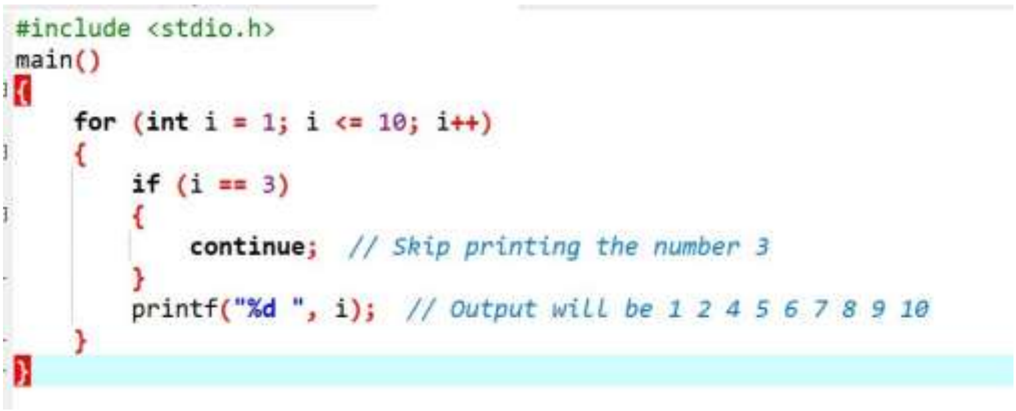
**o Write a C program that uses the break statement to stop printing numbers
when it reaches 5. Modify the program to skip printing the number 3 using the
continue statement.**

- **Program using break statement :**



```
#include <stdio.h>
main()
{
    for (int i = 1; i <= 10; i++)
    {
        if (i == 5)
        {
            break;   // Exit the loop when i equals 5
        }
        printf("%d ", i);   // Output will be 1 2 3 4
    }
}
```

- **Modify the program to skip printing the number 3 using continue statement :**



```
#include <stdio.h>
main()
{
    for (int i = 1; i <= 10; i++)
    {
        if (i == 3)
        {
            continue;  // Skip printing the number 3
        }
        printf("%d ", i);  // Output will be 1 2 4 5 6 7 8 9 10
    }
}
```

8. Functions in C •

THEORY EXERCISE:

o What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

-Function/method : block of code

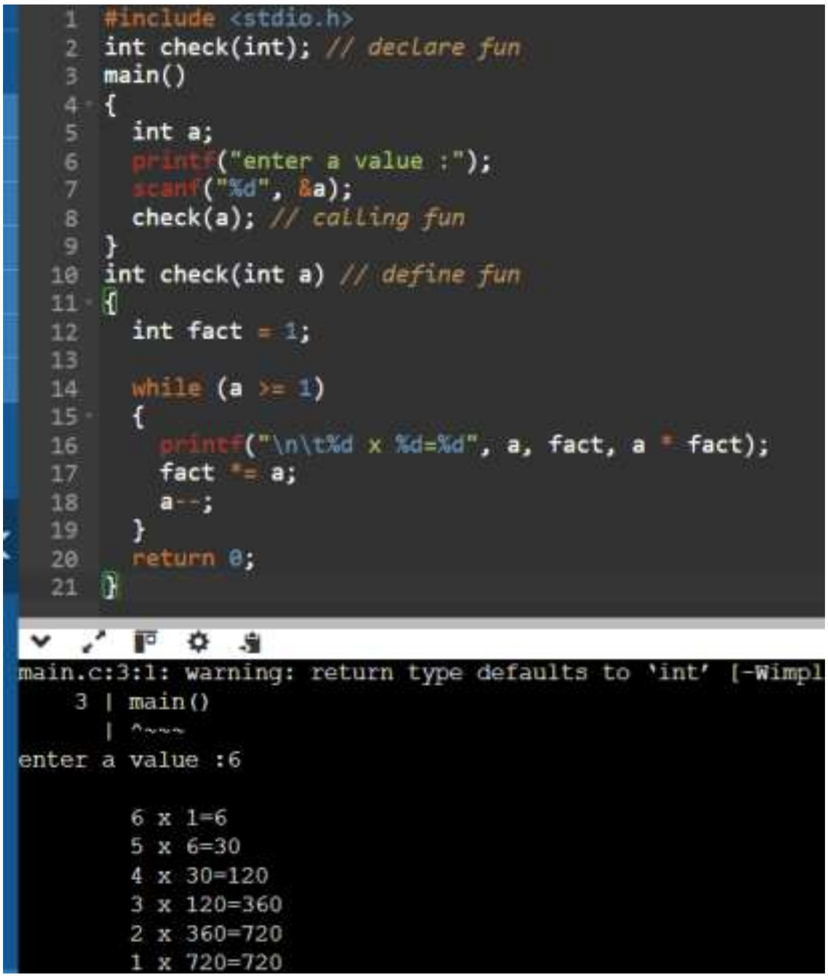
```
#include <stdio.h>

// Function Declaration
int add(int a, int b);

int main() {
   int num1 = 5, num2 = 10;
   int sum;

   // Function Call
   sum = add(num1, num2);

   printf("The sum of %d and %d is %d\n", num1, num2, sum);
   return 0;
}

// Function Definition
```

```
int add(int a, int b)
 {
   return a + b; // Returns the sum of a and b
}
```

LAB EXERCISE:

o Write a C program that calculates the factorial of a number using a function.

Include function declaration, definition, and call.



```
#include <stdio.h>
int check(int); // declare fun
main()
{
  int a;
  printf("enter a value :");
  scanf("%d", &a);
  check(a); // calling fun
}
int check(int a) // define fun
{
  int fact = 1;

  while (a >= 1)
  {
    printf("\n\t%d x %d=%d", a, fact, a * fact);
    fact *= a;
    a--;
  }
  return 0;
}
```

```
main.c:3:1: warning: return type defaults to 'int' [-Wimpl
    3 | main()
      | ^~~~
enter a value :6

        6 x 1=6
        5 x 6=30
        4 x 30=120
        3 x 120=360
        2 x 360=720
        1 x 720=720
```

9. Arrays in C

• THEORY EXERCISE:

o Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

-An array in C is a way to store multiple values of the same type in a single variable. Think of it as a row of boxes, where each box can hold a value, and you can easily access these values using their position (index).

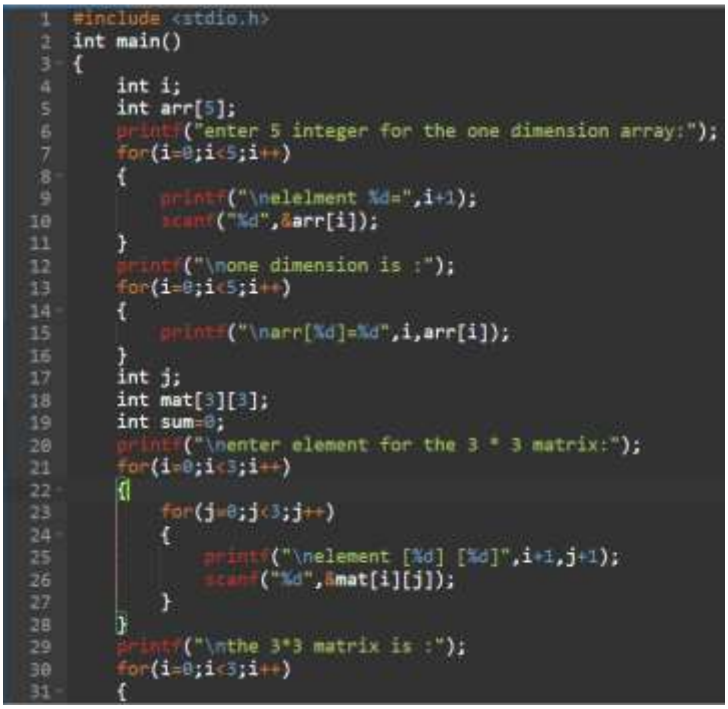
Key Points About Arrays:

- Fixed Size: You need to decide how many items you want to store when you create the array.
- Same Type: All items in an array must be of the same type (like all integers or all floats).
- Indexing: You access items in an array using an index, which starts at 0.

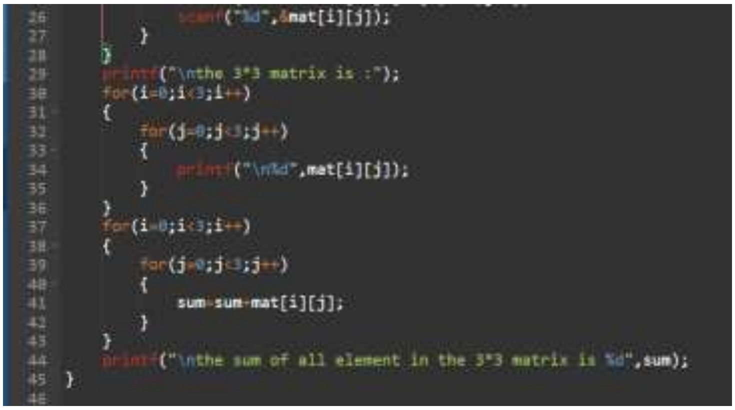
Feature	**One-Dimensional Array**	**Multi-Dimensional Array**
Shape	A single row of values.	Multiple rows and columns (2D), or even higher dimensions.
Accessing elements	Accessed using a single index.	Accessed using multiple indices (row, column, etc.).
Declaration	data_type array_name[size];	data_type array_name[rows][columns];
Example	int arr[5] = {1, 2, 3, 4, 5};	int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

• LAB EXERCISE:

o Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

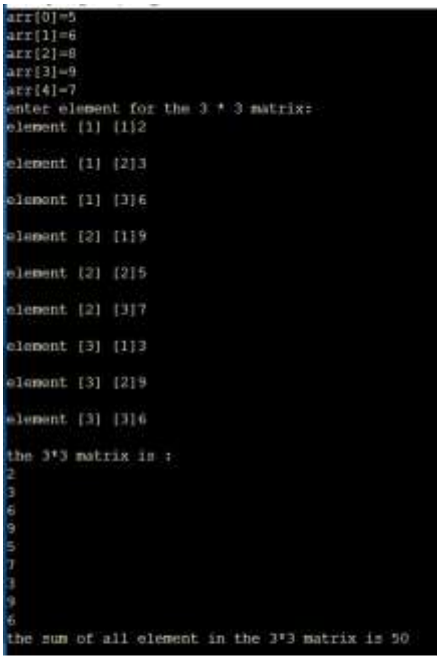


```c
#include <stdio.h>
int main()
{
    int i;
    int arr[5];
    printf("enter 5 integer for the one dimension array:");
    for(i=0;i<5;i++)
    {
        printf("\nelelment %d=",i+1);
        scanf("%d",&arr[i]);
    }
    printf("\none dimension is :");
    for(i=0;i<5;i++)
    {
        printf("\narr[%d]=%d",i,arr[i]);
    }
    int j;
    int mat[3][3];
    int sum=0;
    printf("\nenter element for the 3 * 3 matrix:");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\nelement [%d] [%d]",i+1,j+1);
            scanf("%d",&mat[i][j]);
        }
    }
    printf("\nthe 3*3 matrix is :");
    for(i=0;i<3;i++)
    {
```



```c
            scanf("%d",&mat[i][j]);
        }
    }
    printf("\nthe 3*3 matrix is :");
    for(i=0;i<3;i++)
    {
```

```c
        for(j=0;j<3;j++)
        {
            printf("\n%d",mat[i][j]);
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            sum=sum+mat[i][j];
        }
    }
    printf("\nthe sum of all element in the 3*3 matrix is %d",sum);
}
```



```
arr[0]=5
arr[1]=6
arr[2]=8
arr[3]=9
arr[4]=7
enter element for the 3 * 3 matrix:
element [1] [1]2

element [1] [2]3

element [1] [3]6

element [2] [1]9

element [2] [2]5

element [2] [3]7

element [3] [1]3

element [3] [2]9

element [3] [3]6

the 3*3 matrix is :
2
3
6
9
5
7
3
9
6
the sum of all element in the 3*3 matrix is 50
```

10. Pointers in C

• THEORY EXERCISE:

o Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

What are Pointers?

- Definition: Pointers are variables that store memory addresses of other variables.
- Purpose: They allow direct memory manipulation and efficient data handling.

Declaration

- Syntax: data_type *pointer_name;
    - Example: int *ptr; (declares a pointer to an integer)

Initialization

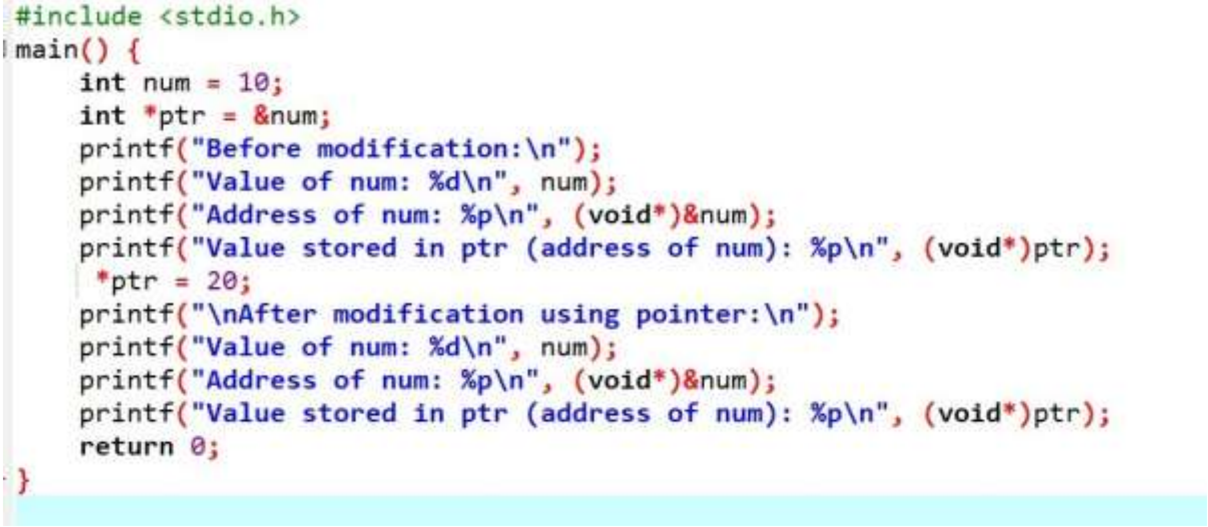
- Using Address-of Operator: Assign the address of a variable using **&**
- Example:

int var = 10;

int *ptr = &var;

• LAB EXERCISE:

o Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.



```
#include <stdio.h>
main() {
    int num = 10;
    int *ptr = &num;
    printf("Before modification:\n");
    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", (void*)&num);
    printf("Value stored in ptr (address of num): %p\n", (void*)ptr);
     *ptr = 20;
    printf("\nAfter modification using pointer:\n");
    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", (void*)&num);
    printf("Value stored in ptr (address of num): %p\n", (void*)ptr);
    return 0;
}
```

11. Strings in C

• THEORY EXERCISE:

o Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

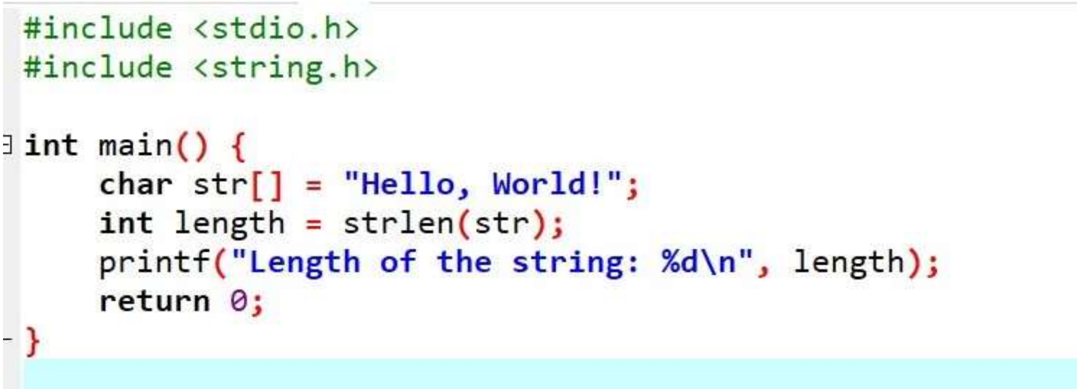
- strlen(): Get the length of a string

  ✓ The strlen() function returns the number of characters in a string, excluding the null character ('\0').

  ✓ Syntax :

  size_t strlen(const char *str);

  ✓ Example :



```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello, World!";
    int length = strlen(str);
    printf("Length of the string: %d\n", length);
    return 0;
}
```

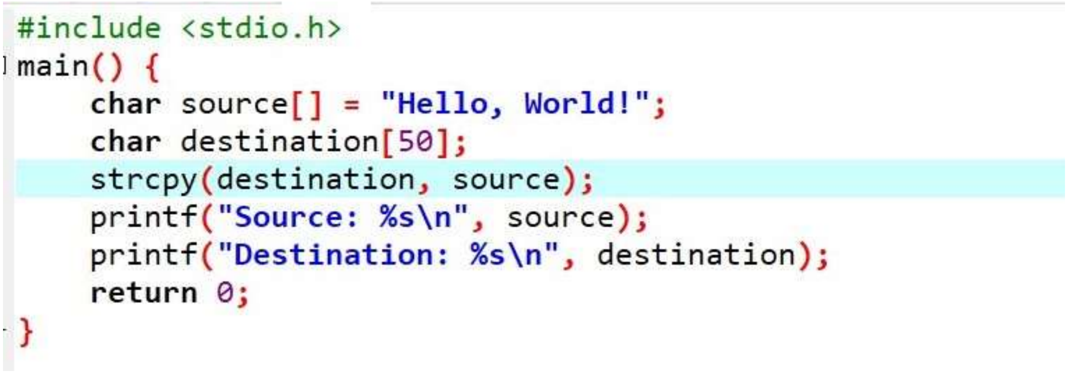
- strcpy(): Copy a string

  ✓ The strcpy() function copies the content of one string into another.

  ✓ Syantax :

  char *strcpy(char *dest, const char *src);

  ✓ Example :



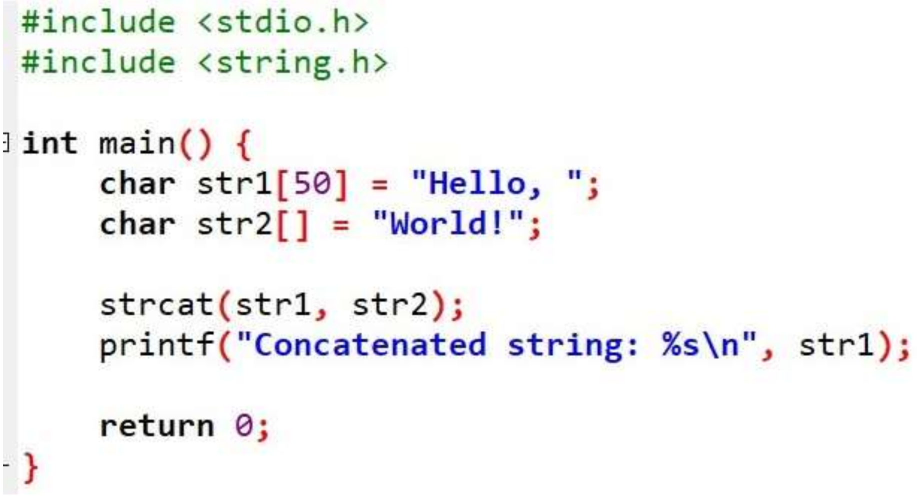
```
#include <stdio.h>
main() {
    char source[] = "Hello, World!";
    char destination[50];
    strcpy(destination, source);
    printf("Source: %s\n", source);
    printf("Destination: %s\n", destination);
    return 0;
}
```

- strcat(): Concatenate two strings

    ✓ The strcat() function appends one string to the end of another string.

    ✓ Syntax :
    char *strcat(char *dest, const char *src);

    ✓ Example :



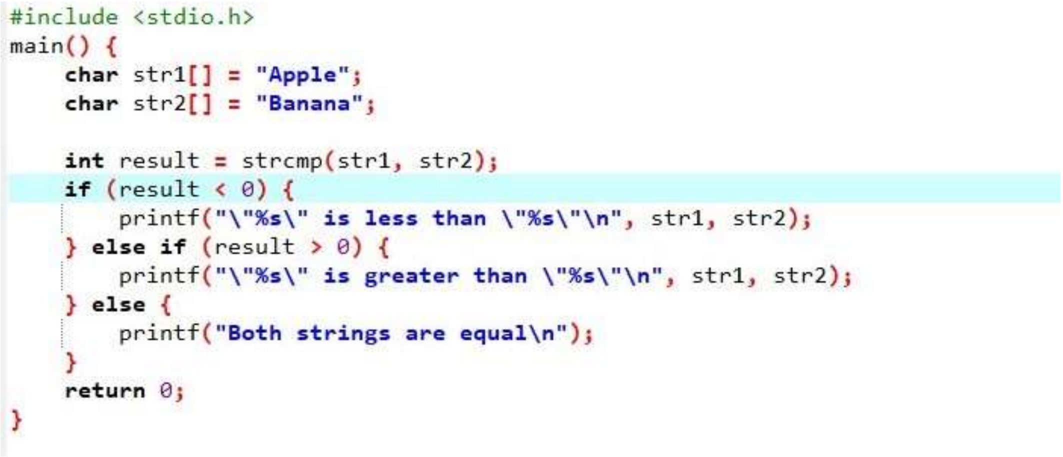
```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[50] = "Hello, ";
    char str2[] = "World!";

    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);

    return 0;
}
```

- strcmp(): Compare two strings

    ✓ The strcmp() function compares two strings lexicographically (character by character).

    ✓ Syntax :
    int strcmp(const char *str1, const char *str2);

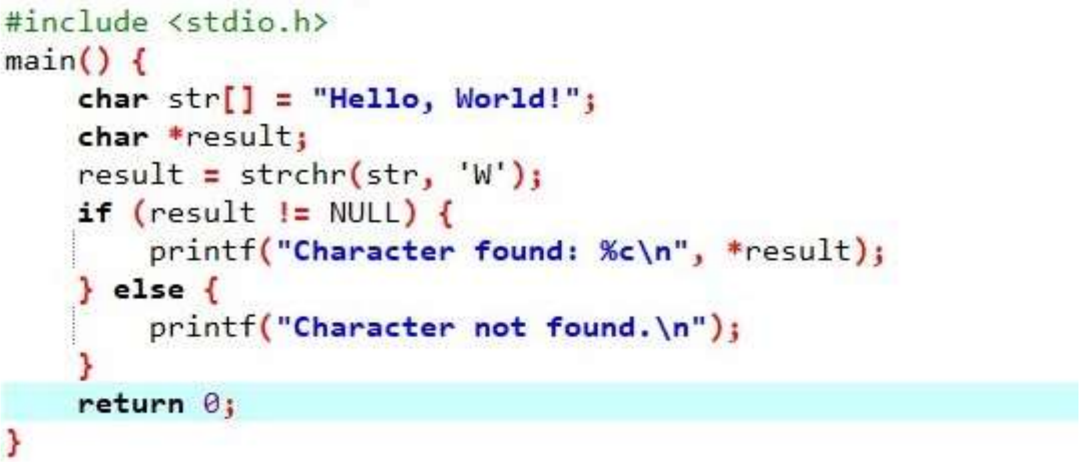
    ✓ Example :



```
#include <stdio.h>
main() {
    char str1[] = "Apple";
    char str2[] = "Banana";

    int result = strcmp(str1, str2);
    if (result < 0) {
        printf("\"%s\" is less than \"%s\"\n", str1, str2);
    } else if (result > 0) {
        printf("\"%s\" is greater than \"%s\"\n", str1, str2);
    } else {
        printf("Both strings are equal\n");
    }
    return 0;
}
```

- strchr(): Find the first occurrence of a character in a string
  - ✓ The strchr() function searches for the first occurrence of a specified character in a string.
  - ✓ Syntax :

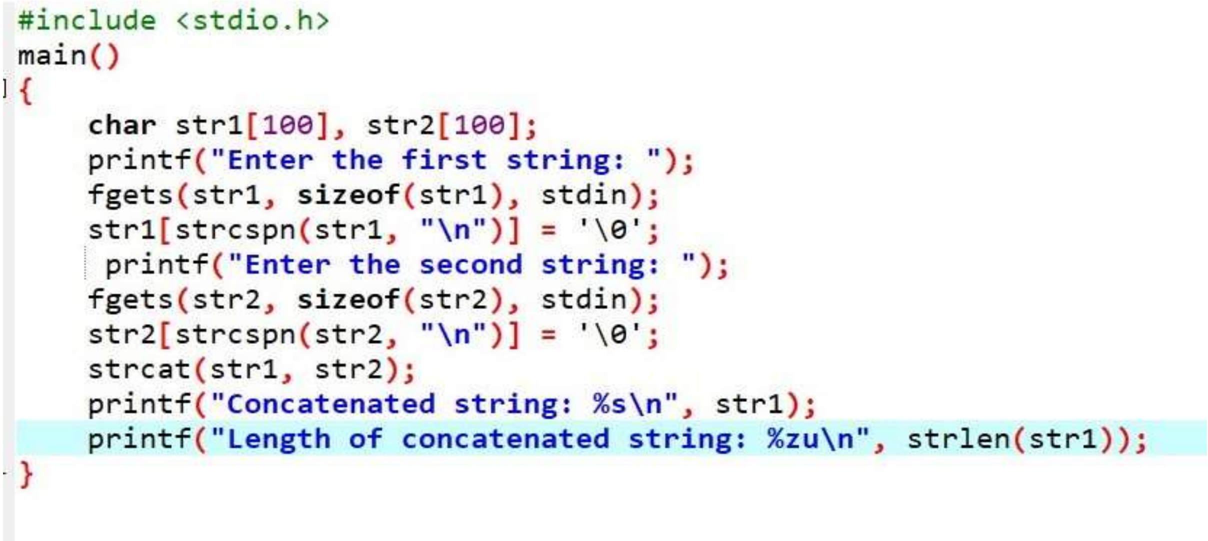
    char *strchr(const char *str, int c);
  - ✓ Example :



```
#include <stdio.h>
main() {
    char str[] = "Hello, World!";
    char *result;
    result = strchr(str, 'W');
    if (result != NULL) {
        printf("Character found: %c\n", *result);
    } else {
        printf("Character not found.\n");
    }
    return 0;
}
```

• LAB EXERCISE:

o Write a C program that takes two strings from the user and concatenates them using strcat(). Display the concatenated string and its length using strlen().



```c
#include <stdio.h>
main()
{
    char str1[100], str2[100];
    printf("Enter the first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = '\0';
     printf("Enter the second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = '\0';
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);
    printf("Length of concatenated string: %zu\n", strlen(str1));
}
```

12. Structures in C

- **THEORY EXERCISE:**

o Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

- Concept of Structures in C : A structure in C is a user-defined data type that allows grouping of different types of variables (called members or fields) under a single name. Each member of a structure can have a different data type (e.g., integers, floats, arrays, or even other structures).

- Declaring a Structure : To define a structure, you use the struct keyword followed by the structure name and its members inside curly braces.

Here is the syntax to declare a structure:

```
 struct structure_name
{
          data_type member1;
          data_type member2;
          // More members
};
```

- Initializing a Structure : There are two ways to initialize a structure :
  - ✓ At the time of declaration (Static Initialization): You can initialize the structure members when you declare a structure variable:

struct Student student1 = {"John Doe", 20, 85.5};

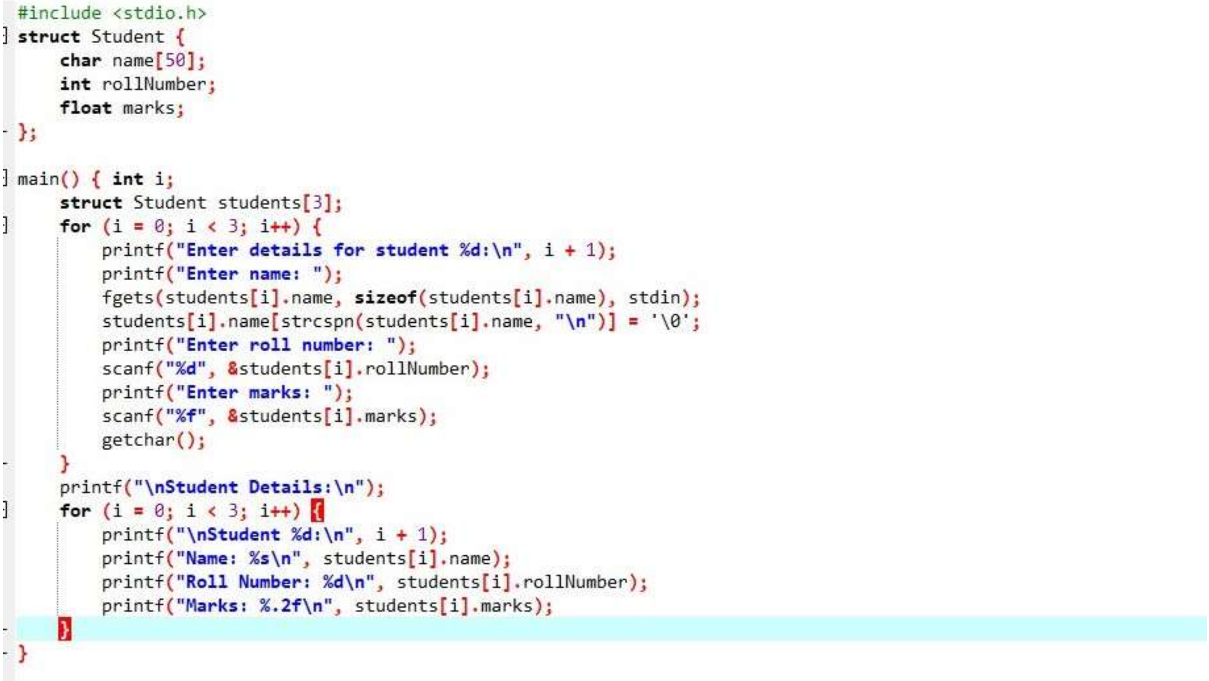
  - ✓ After Declaration (Dynamic Initialization): You can declare the structure first and then assign values to its members individually:

struct Student student1;
strcpy(student1.name, "John Doe"); // Using strcpy for string assignment
student1.age = 20;
student1.marks = 85.5;

- Accessing Structure Members : Structure members can be accessed using the dot operator (.) for individual structure variables. If the structure variable is a pointer, you use the arrow operator (->) to access the members.

• LAB EXERCISE:

o Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.



```
#include <stdio.h>
struct Student {
    char name[50];
    int rollNumber;
    float marks;
};

main() { int i;
    struct Student students[3];
    for (i = 0; i < 3; i++) {
        printf("Enter details for student %d:\n", i + 1);
        printf("Enter name: ");
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = '\0';
        printf("Enter roll number: ");
        scanf("%d", &students[i].rollNumber);
        printf("Enter marks: ");
        scanf("%f", &students[i].marks);
        getchar();
    }
    printf("\nStudent Details:\n");
    for (i = 0; i < 3; i++) {
        printf("\nStudent %d:\n", i + 1);
        printf("Name: %s\n", students[i].name);
        printf("Roll Number: %d\n", students[i].rollNumber);
        printf("Marks: %.2f\n", students[i].marks);
    }
}
```

13. File Handling in C

• THEORY EXERCISE:

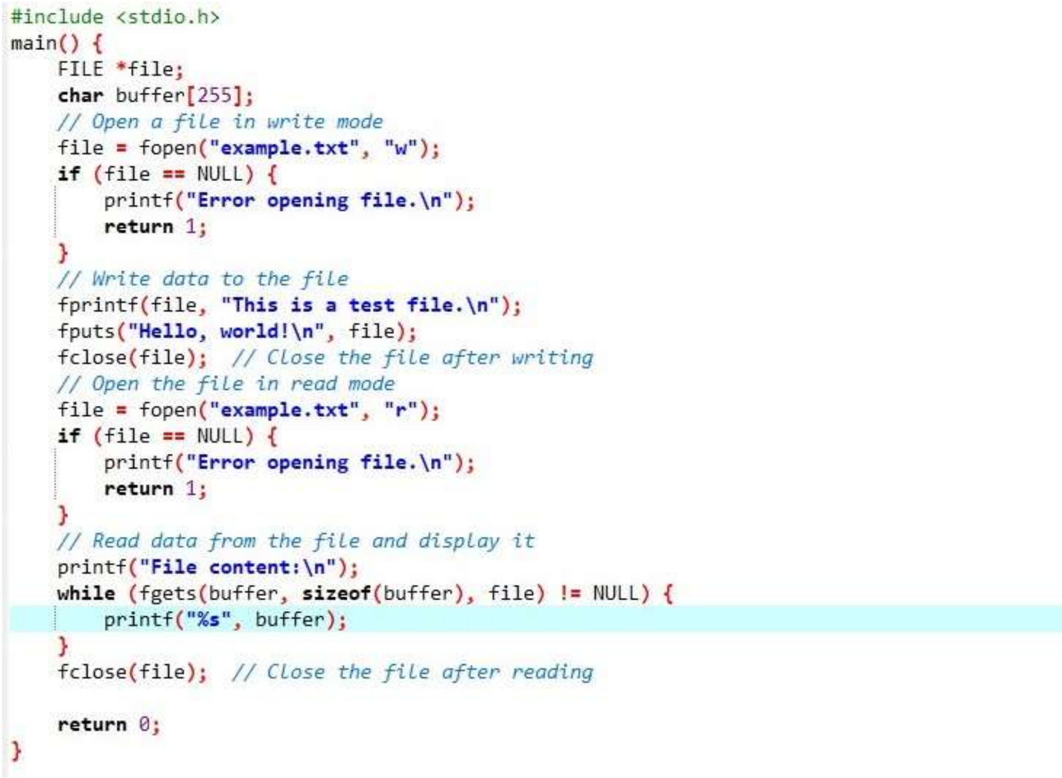
o Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

- Importance of File Handling in C :
  - ✓ Persistent Data Storage:
  - ✓ Data Sharing:
  - ✓ Efficient Data Management:
  - ✓ Flexible Data Operations:

- Basic File Operations in C :
  - ✓ Opening a File : To perform any operation on a file, you first need to open the file using the fopen() function. This function requires two arguments 1=The name of the file. 2=The mode in which you want to open the file.
    - ➢ Syntax :

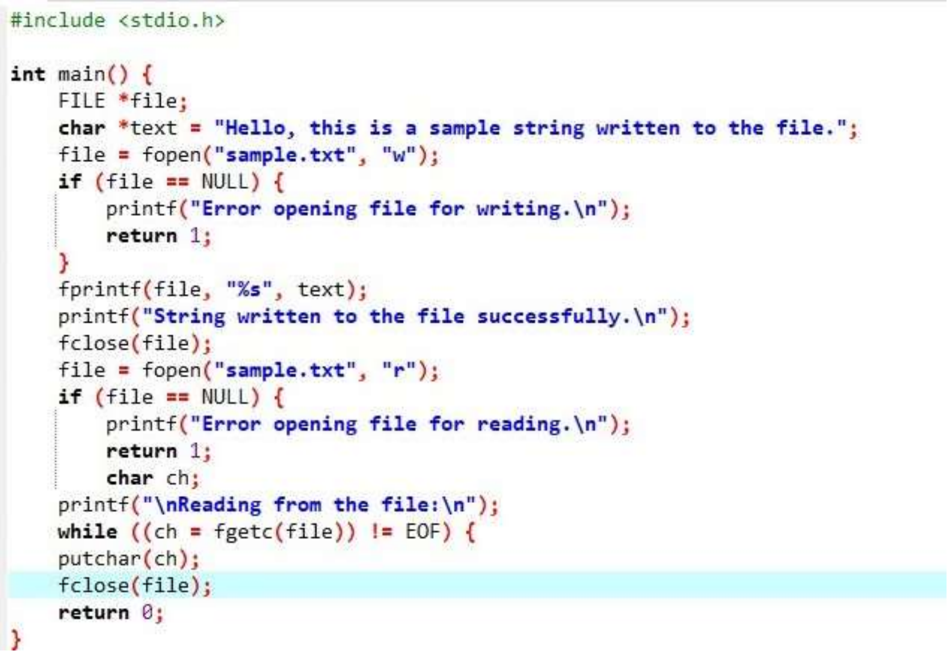
```
FILE *fopen(const char *filename, const char *mode);
```

  - ✓ Reading from a File : Once the file is opened in read mode, you can read from it using functions like fgetc(), fgets(), or fread().
  - ✓ Writing to a File : You can write data to a file using fputc(), fputs(), or fwrite().
  - ✓ Closing a File : After performing the required operations (read or write), it's essential to close the file using fclose() to release resources and ensure data integrity



```c
#include <stdio.h>
main() {
    FILE *file;
    char buffer[255];
    // Open a file in write mode
    file = fopen("example.txt", "w");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // Write data to the file
    fprintf(file, "This is a test file.\n");
    fputs("Hello, world!\n", file);
    fclose(file);  // Close the file after writing
    // Open the file in read mode
    file = fopen("example.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // Read data from the file and display it
    printf("File content:\n");
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("%s", buffer);
    }
    fclose(file);  // Close the file after reading

    return 0;
}
```

• LAB EXERCISE: o Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.



```c
#include <stdio.h>

int main() {
    FILE *file;
    char *text = "Hello, this is a sample string written to the file.";
    file = fopen("sample.txt", "w");
    if (file == NULL) {
        printf("Error opening file for writing.\n");
        return 1;
    }
    fprintf(file, "%s", text);
    printf("String written to the file successfully.\n");
    fclose(file);
    file = fopen("sample.txt", "r");
    if (file == NULL) {
        printf("Error opening file for reading.\n");
        return 1;
        char ch;
    printf("\nReading from the file:\n");
    while ((ch = fgetc(file)) != EOF) {
    putchar(ch);
    fclose(file);
    return 0;
}
```