

# Shakuntala Krishana Institute Of Technology



College Code : - KD64

Affiliated To CSJM University Kanpur

## Assignment

Assignment No. : ..... 1<sup>st</sup> .....

Subject : ..... C Programming .....

Assignment Topic

Submitted To :

Miss. Aarti Mam .....

Subject Teacher Of :

Miss. Aarti Mam .....

Department :

B.C.A .....

Submitted By :

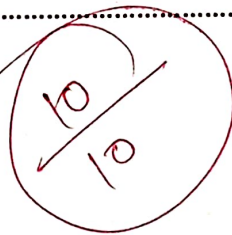
Devika .....

Course :

B.C.A .....

Semester :

1<sup>st</sup> .....



✓ good

19/09/23

Question-1 : Write a short note on the history of C language.

Why is it called a middle-level language?

List any four important features of C.

Also explain the advantage and disadvantage of C language.

Answer :

History of C language —

- The C language was developed in 1972 by Dennis Richie at Bell Laboratories (AT and T, USA)
- It was derived from an earlier language called B (developed by Ken Thompson).
- Initially, C was designed to implement the UNIX operating system.
- Later, it became one of the most popular programming language worldwide.



## Why C is called a Middle-level language

### Low-level features :

Direct memory access,  
pointer usage.

### High-level features :

Functions, loops, structures

Hence, it combines the power of assembly language (low-level) and simplicity of high level language.

## Four Important Features of C :

### 1. Portability —

Programs can run on different machines with little/no change.

### 2. Structured Language —

Programs are divided into functions and blocks.

3. Rich Library -

Provides built-in functions for mathematical, string, and input-output operations.

4. Efficiency and Speed -

Close to machine code, hence fast execution.

### Advantages of C language

- Portable and efficient.
- Strong foundation for learning other languages like C++, Java, Python
- Flexible : Can write both system software and application software
- Large user community and resources.



## Disadvantages of C Language

- No object oriented programming (OOP) features.
- No built-in support for modern concepts like graphics, networking.
- Manual memory management (risk of errors).
- Less secure (no data hiding).



Question-2 : Draw and explain the basic structure of C program with the help of a simple program that prints your name.

Answer -

Structure of C Program -

```
#include <stdio.h> // Header file for standard  
input/output functions
```

```
// main function - program execution starts here
```

```
int main() {
```

```
// printf is used to display text on  
the screen
```

```
printf ("Hello, World!");
```

```
return 0; // return 0 means program ended  
successfully
```

```
}
```

## Program Explanation

1. #include <stdio.h>

This line includes the `stdio.h` header file which contains functions like `printf` and `scanf`.

2. int main()

The execution of the program starts here.

3. printf("Hello, World!");

This line will print "Hello, World!" on the screen.

4. return 0;

This tells the compiler that the program ended successfully.

## Example Program

```
#include <stdio.h>
int main() {
    // Printing my name
    printf("My name is Devika\n");
    return 0;
}
```

Question - 3 : What are C conventions?  
Write rules for naming identifiers and explain the use of comments that prints your name.

Answer -

C conventions -

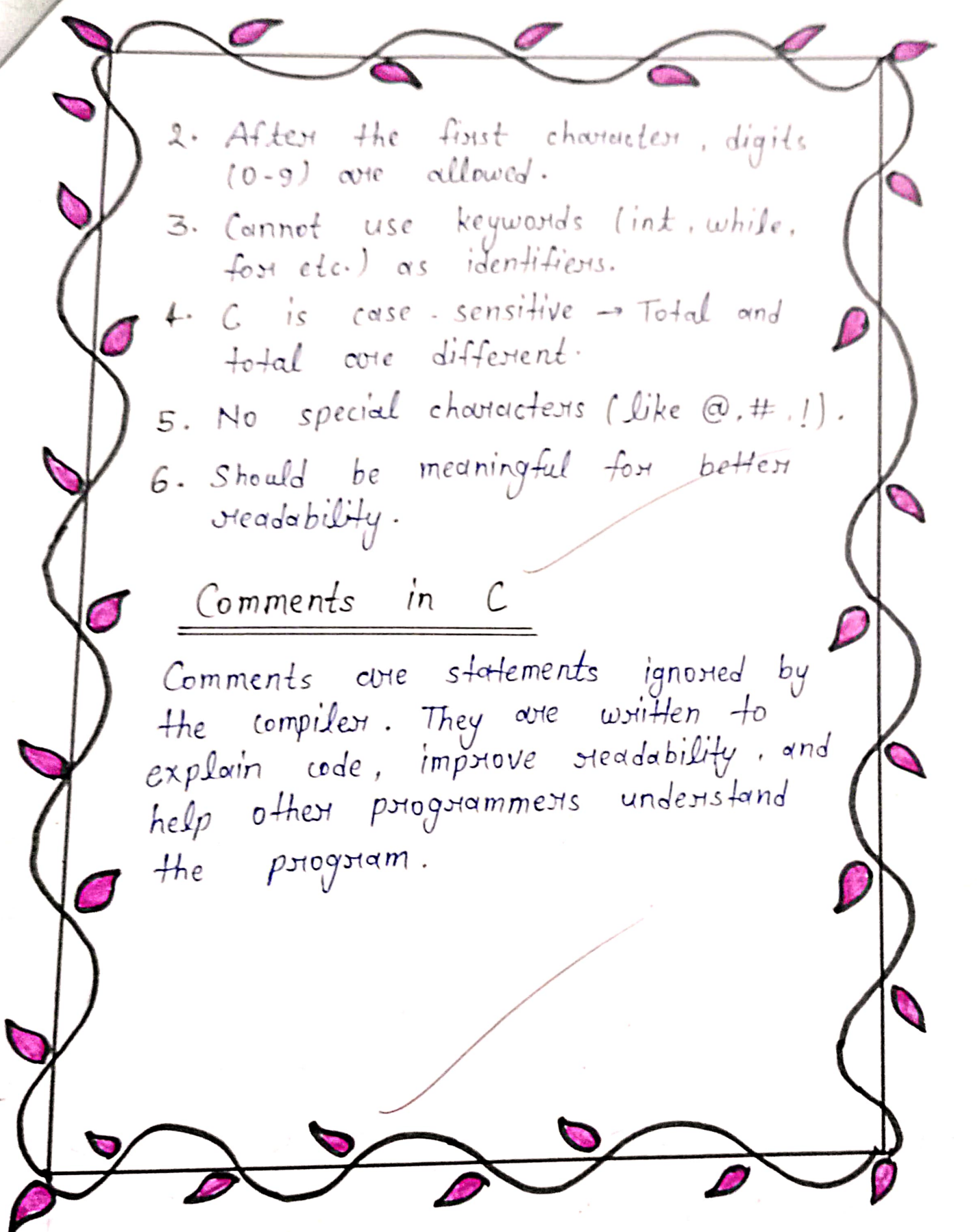
C conventions are standard rules and guidelines that every C programmer follows to write clean, readable, and error-free code.

Rules for Naming Identifiers

Identifiers are names given to variables, functions, arrays etc. The rules are:

1. Must begin with a letter (A-Z, a-z) or underscore (\_) only.



- 
2. After the first character, digits (0-9) are allowed.
  3. Cannot use keywords (int, while, for etc.) as identifiers.
  4. C is case-sensitive → Total and total are different.
  5. No special characters (like @, #, !).
  6. Should be meaningful for better readability.

### Comments in C

Comments are statements ignored by the compiler. They are written to explain code, improve readability, and help other programmers understand the program.

## 1. Single - line Comment :

// This is a single line comment

## 2. Multi - line Comment :

/\*

This is a  
multi - line comment

\*/

## Example program with identifiers & Comments

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

```
int main() {
```

```
    int age = 18; // 'age' is an identifier
```

```
    char grade = 'A'; /* 'grade' is another  
                        identifier
```

This stores student's  
grade \*/

```
    printf ("Age : %d\n", age);
```

```
    printf ("Grade : %d\n", grade);
```

```
    return 0 ;
```

```
}
```

Question-4 : Define identifiers and keywords in

C. Write the rules for identifiers and give ten examples of C with keywords.

Answer :

Identifiers in C →

Identifiers are user-defined names given to program elements such as variables, functions, arrays, structures etc.

They help us identify and use data in the program.

Rules for Identifiers

1. Must begin with a letter or underscore (\_).
2. Can contain letters, digits, and underscore.



3. No spaces or special characters allowed.

4. Case - sensitive (Ex - Age and age are different).

5. Cannot be a keyword.

### Examples of Identifiers

Valid - marks, roll-no, total1

Invalid - 1age, float, total@

### Keywords in C

Keywords are reserved words in C that have a predefined meaning for the compiler.

They cannot be used as identifiers.

Total 32 keywords defined in C

## Examples of keywords

int, float, char, double, if, else, for, while, switch, case, break, return etc.

## Difference Between Identifiers & Keywords

| Identifiers                     | Keywords                        |
|---------------------------------|---------------------------------|
| 1. User - defined names.        | 1. Predefined reserved words.   |
| 2. Created by the programmer.   | 2. Fixed by the C language.     |
| 3. <u>Ex</u> - marks, sum, age. | 3. <u>Ex</u> - int, for, return |
| 4. Unlimited                    | 4. Only 32 keywords in C        |

Question-5: What are Simple data types in C? Differentiate between int, float, char and double with examples. Write a program to display value of all these data types.

Answer:

### Simple data Types in C

In C language, data types define the type of data a variable can store. The Simple data types are:

#### 1. int (Integer)

- Used to store whole numbers (Positive or negative)
- Size: Usually 2 or 4 bytes (depends on system).
- Format: %d



## 2. Float (Floating Point):

- Used to store decimal numbers (Single precision).
- Size: 4 bytes
- Format specifier: %f

## 3. double (Double Precision Floating point)

- Similar to float but stores numbers with more accuracy.
- Size: 8 bytes
- Format specifier: %lf

## 4. Char (Character)

- Used to store a single character
- Size: 1 byte
- Format specifier: %c

## Example Program Using All Simple Data Types :

```
#include <stdio.h>

int main() {
    int age = 18;
    float price = 99.50;
    double pi = 3.1415926535;
    char grade = 'A';

    printf("Age : %d\n", age);
    printf("Price : %f\n", price);
    printf("Value of pi : %.1f\n", pi);
    printf("Grade : %c\n", grade);

    return 0;
}
```

# Shakuntala Krishana Institute Of Technology



College Code : - KD64

Affiliated To CSJM University Kanpur

## Assignment

Assignment No. : ..... 2 .....

Subject : ..... C Programming .....

Assignment Topic

Submitted To :

..... Miss Aarti mam .....

Submitted By :

..... Devika .....

Subject Teacher Of :

..... Miss Aarti Mam .....

Course :

..... B.C.A .....

Department :

..... BCA .....

Semester :

..... 1<sup>st</sup> .....

10  
10  
29/09/23



Question-1 : Explain variables, constants, and modifiers in C.

1. Define variables and rules for naming them.
2. Explain constants with examples (`#define` and `const`).
3. Explain data type modifiers (short, long, signed, unsigned) with examples.

Answer :

Variables in C :

A variable is a named memory location used to store data. The value of a variable can be changed during the execution of a program.

Rules for Naming Variables -

1. A variable name can contain letters, digits, and underscored (`_`).
2. The first character must be a letter or an underscore.

3. No spaces or special characters are allowed.
4. C is case sensitive (age and Age are different).
5. Keywords (like int, float, return) cannot be used as variable names.
6. Variable names should be meaningful for readability.

### Constants in C -

A constant is a value that does not change during the execution of a program. Constants make program easier to read and maintain.

### Types of Constants

#### 1. Literal Constants -

Directly used values.

Ex-

int days = 7 ; // 7 is an integer constant  
float pi = 3.14 ; // 3.14 is a float constant  
char grade = 'A' ; // 'A' is a character constant.

## 2. Symbolic Constants (Using #define) -

Defined at compile-time using preprocessor directives.

Ex - `#define Pi 3.14`  
`#define MAX 100`

Pi and MAX cannot be changed during execution.

## 3. Constant Variables (Using const) -

Declared using the const keyword.

Ex - `const int maxmarks = 100;`  
`const float GST = 18.0;`

Once initialized, their value cannot be modified.

## Data Types Modifiers in C -

C provides modifiers to change the storage size or sign representation of basic data types.

Modifiers: short, long, signed, unsigned.



Example —

short int x; // smallest range integer  
long int y; // longest range integer  
unsigned int z; // only positive values  
signed int a; // positive and negative values

Example Program —

```
#include <stdio.h>
int main() {
    short int s = 32767;
    long int l = 123456789;
    unsigned int u = 4000;
    signed int si = -200;

    printf("Short: %d\n", s);
    printf("Long: %ld\n", l);
    printf("Unsigned: %u\n", u);
    printf("Signed: %d\n", si);
    return 0;
}
```

Question-2 : What are operators in C? Explain different types of operators and operator precedence with examples.

1. Arithmetic, Relational, Logical, Assignment, Increment / Decrement, Conditional, Bitwise operators.
2. Explain operator precedence and associativity with a precedence table.

Answer —

Operators — Operators are symbols in C that perform operations on data/operands.

Ex - +, -, \*, /, =

Types of Operators in C

- (i) Arithmetic Operators
- (ii) Relational Operators
- (iii) Logical Operators

- (iv) Assignment Operators
- (v) Increment/Decrement Operators
- (vi) Conditional (Ternary) Operator
- (vii) Bitwise Operators

### (i) Arithmetic Operators -

- Used for basic mathematical operations.

| Operator | Meaning        | Example      |
|----------|----------------|--------------|
| +        | Addition       | $5 + 3 = 8$  |
| -        | Subtraction    | $5 - 3 = 2$  |
| *        | Multiplication | $5 * 3 = 15$ |
| /        | Division       | $5 / 2 = 2$  |
| %        | Modulus        | $5 \% 2 = 1$ |

Example -

```
int a = 10, b = 3;
printf("%d\n", a+b);
printf("%d\n", a%b);
```



## (ii) Relational Operators —

- Used to compare two values, results are true(1) or false(0).

| Operator           | Meaning                  |
|--------------------|--------------------------|
| <code>==</code>    | Equal to                 |
| <code>!=</code>    | Not equal to             |
| <code>&gt;</code>  | Greater than             |
| <code>&lt;</code>  | Less than                |
| <code>&gt;=</code> | greater than or equal to |
| <code>&lt;=</code> | less than or equal to    |

Example -

```
int x = 5, y = 8;  
printf("%d", x < y);
```

## (iii) Logical Operators —

- Combine multiple conditions.

| Operator                | Meaning | Example                                     |
|-------------------------|---------|---------------------------------------------|
| <code>&amp;&amp;</code> | AND     | <code>(x &gt; 0 &amp;&amp; y &gt; 0)</code> |
| <code>!</code>          | NOT     | <code>!(x &gt; 0)</code>                    |

Example -

```
int x = 5, y = -3;  
printf("%d", (x > 0 && y > 0));
```

(iv) Assignment Operators —

- Assign values to variables.

| Operator | Example  | Equivalent  |
|----------|----------|-------------|
| =        | $x = 10$ | assign 10   |
| +=       | $x += 5$ | $x = x + 5$ |
| -=       | $x -= 3$ | $x = x - 3$ |
| *=       | $x *= 2$ | $x = x * 2$ |
| /=       | $x /= 2$ | $x = x / 2$ |

(v) Increment and Decrement Operators —

- Increase or decrease value by 1.

| Operator | Example        | Meaning                   |
|----------|----------------|---------------------------|
| ++a      | Pre increment  | Increase first, then use  |
| a++      | Post increment | Use first, then increase  |
| --a      | Pre decrement  | Decrease first, then use  |
| a--      | Post decrement | Use first, then decrease. |

### (vi) Conditional Operator -

- Short form of if-else
- Syntax: (Condition) ? value\_if\_true : value\_if\_false;

Example -

```
int a = 10, b = 20;  
int max = (a > b) ? a : b;  
printf("Max = %d", max); // output : 20
```

### (vii) Bitwise Operators

- Work at binary (bit) level.

| Operator | Meaning     |
|----------|-------------|
| &        | AND         |
| ^        | OR          |
| ~        | NOT         |
| <<       | Left shift  |
| >>       | Right shift |

Example - `int a = 5, b = 3; // a = 0101, b = 0011`  
`printf("%d", a & b); // 1 (0001)`



## Operator Precedence And Associativity -

When an expression has multiple operators, precedence decides which operator is executed first.

If operators have the same precedence, associativity decides the direction (left to right or right to left).

### Precedence Table -

| Precedence Level | Operators    | Associativity |
|------------------|--------------|---------------|
| 1                | ++, --, !, ~ | Right to left |
| 2                | *, /, %      | Left to right |
| 3                | +, -         | Left to right |
| 4                | <, <=, >, >= | Left to right |
| 5                | ==, !=       | Left to right |
| 6                | &&           | Left to right |
| 7                |              |               |
| 8                | =, +=, -=    | Right to left |

### Example Demonstrating Precedence

```
#include <stdio.h>
int main() {
    int a = 10, b = 5, c = 2;
    int result = a - b * c;
    printf("Result = %d", result);
    return 0;
}
```

Output : 0

### Question - 3: Explain Input and Output operations in C.

1. Single character input/output : `getchar()`, `putchar()`.
2. Formatted I/O : `scanf()`, `printf()`.
3. Examples of reading/writing integers, floats, characters, and strings.

### Answer -

Input Operations - Taking data from the user into the program.

## Output Operations -

Displaying data / results from the program to the user.

### Single Character Input / Output -

#### (a) getchar() -

- Reads a single character from the keyboard.

Syntax: `char c = getchar()`

Example:

```
#include <stdio.h>
int main () {
    char ch;
    printf("Enter a character:");
    ch = getchar(); // user types 'A'
    printf("You entered:");
    putchar(ch); // Output: A
    return 0;
}
```

#### (b) putchar()

- Prints a single character on the screen.

Syntax: `putchar(c);`



### 3. Formatted Input/Output

#### (a) scanf() —

- Used to read formatted input (integers, floats, characters, strings).

Syntax —

`scanf("format_specifier", &variable);`

Note — & (address - of operator) is required before variable names (except strings).

Example —

```
int age;  
printf("Enter age:");  
scanf("%d", &age);
```

#### (b) printf() —

- Used to display output with formatting.

Syntax: `printf("format_specifier", variable);`

Example —

```
float salary = 25000.50;  
printf("salary = %.2f", salary);
```

Output: salary = 25000.50

## Examples of Reading/Writing Different Data Types :

### (a) Integers -

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer;");
    scanf("%d", &num);
    printf("You entered: %d", num);
    return 0;
}
```

### (b) Floats -

```
#include <stdio.h>
int main() {
    float num;
    printf("Enter a float number;");
    scanf("%f", &num);
    printf("You entered: %.2f", num);
    return 0;
}
```

(c) Characteris -

```
#include <stdio.h>
int main() {
    char ch;
    printf("Enter a character:");
    scanf("%c", &ch);
    printf("Character entered : %c", ch);
    return 0;
}
```

(d) Strings -

```
#include <stdio.h>
int main() {
    char name[20];
    printf("Enter your name:");
    scanf("%s", name);
    printf("Hello, %s", name);
    return 0;
}
```



## Difference Between getch() / putchar() and scanf() / printf()

| Function  | Purpose               | Works on |
|-----------|-----------------------|----------|
| getchar() | Reads 1 character     | Keyboard |
| putchar() | Print 1 character     | Screen   |
| scanf()   | Reads formatted data  | Keyboard |
| printf()  | Prints formatted data | Screen   |

Question - 4 : Explain control structures in C with examples.

1. Conditional: if, if-else, nested if, switch.
2. Iterative: for, while, do-while.
3. Jump: break, continue, goto.

Answer -

Control Structures in C -

- Control structures in C are statements that control the flow of execution of a program.
- Without control structures, the program would run line by line from top to bottom.

- Control structures make programs flexible, logical, and powerful.

There are three main types of control structures in C.

1. Decision - making (selection)
2. Looping (Iteration)
3. Jumping (Branching / Transfer of control)

### Decision - Making Structures -

- Used to execute different statements based on conditions.

#### (a) if statement -

Executes a block if condition is true.

Example -

```
int age = 18;  
if (age >= 18) {  
    printf("Eligible to vote");  
}
```

#### (b) if - else statement

provides two way decision making.

Example -

```
int marks = 40;  
if (marks >= 33) {  
    print("Pass");  
} else {  
    print("Fail");  
}
```

(c) nested if -

if inside another if.

Example -

```
if (condition1) {  
    if (condition2) {  
        // executes only if both are true  
    }  
}
```

(d) else-if ladder -

For multiple conditions.

Example -

```
if (marks >= 90)  
    printf("Grade A");  
else if (marks >= 75)  
    print("Grade B");  
else if (marks >= 50)  
    print("Grade C");
```



### (e) switch statement

Used when many conditions are based on the same variable.

Example -

```
switch(expression) {  
    case 1: printf("One"); break;  
    case 2: printf("Two"); break;  
    default: printf("other");  
}
```

### Looping Structures -

used when we need to repeat a block of code.

(a) for loop - Best when we need to number of iterations is known.

Example -

```
for (int i = 1; i <= 5; i++) {  
    printf("%d", i);  
}
```

// output: 1 2 3 4 5

(b) while loop -

Executes as long as condition is true.

Example -

```
int i = 1;
while (i <= 5) {
    printf("%d", i);
    i++;
}
```

(c) do - while loop -

Executes at least once, even if condition is false.

Example -

```
int i = 1;
do {
    printf("%d", i);
    i++;
} while (i <= 5);
```

Jumping Statements

Used to alter normal flow.

| Statement | Purpose                               |
|-----------|---------------------------------------|
| break     | Exits from loop or switch immediately |

|          |                                          |
|----------|------------------------------------------|
| continue | skips current iteration and goes to next |
| goto     | Transfers control to a labeled statement |
| return   | Exits from function and returns a value. |

Example -

```

for (int i = 1; i <= 5; i++) {
    if (i == 3) continue; // skip 3
    if (i == 5) break; // stop at 4
    printf("%d", i);
}
// output 1 2 4

```

Question-5: What is a looping statement?

Explain different types of loops in C with examples.

1. for loop, while loop, do while-loop.
2. Use of break and continue.
3. Example program: factorial using for, sum of numbers using while, menu-driven program using do-while.



## Answer:

### Looping Statement in C -

used to execute a block of code repeatedly until a specific condition is true.

It reduced code repetition and makes programs shorter and efficient.

### Types of loops in C -

#### 1. for loop -

Syntax: `for (initialization; condition; increment/decrement)`  
`{`  
`// body of loop`  
`}`

#### Example:

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

#### 2. while loop -

Condition is checked before execution of loop body.

Syntax: while (condition) {  
    // body of loop  
}

Example:

```
#include <stdio.h>
int main() {
    int i = 1, sum = 0;
    while (i <= 5) {
        sum += i;
        i++;
    }
    printf("Sum = %d", sum);
    return 0;
}
```

3. do - while loop - Body executes at least once (condition checked later).

Syntax: do {  
    // body  
} while (condition);

Example :

```
#include <stdio.h>
int main() {
    int choice;
    do {
        printf("1. Hello\n2. Exit\n");
        scanf("%d", & choice);
        if (choice == 1)
            printf("Hello World\n");
    } while (choice != 2);
    return 0;
}
```

Use of break and continue -

break → immediately exits from the loop.

continue → skips current iteration, jumps to next.

Example -

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue; // skip 3
    if (i == 5) break; // stop loop
    printf("%d", i);
}
```



## Example Programs -

### 1. Factorial using for loop -

```
#include <stdio.h>
int main() {
    int n = 5, fact = 1;
    for (int i = 1; i <= n; i++) {
        fact * = i;
    }
    printf("Factorial = %d", fact);
    return 0;
}
```

### 2. Sum of numbers using while loop

```
#include <stdio.h>
int main() {
    int n = 5, i = 1, sum = 0;
    while (i <= n) {
        sum + = i;
        i ++;
    }
    printf("Sum = %d", sum);
    return 0;
}
```



College Code : - KD64

Affiliated To CSJM University Kanpur

## Assignment

Assignment No. : ..... 3rd .....

Subject : ..... C Programming .....

Assignment Topic

Submitted To :

..... Miss Aarti Mam .....

Submitted By :

..... Devika .....

Subject Teacher Of :

..... Miss Aarti Mam .....

Course :

..... BCA .....

Department :

..... BCA .....

Semester :

..... 1st .....

09/10/23

Question-1: Define arrays, Differentiate between one dimensional and two-dimensional arrays with suitable examples in C.

Answer —

Definition of Array —

An array in C is a collection of elements of the same data type stored in contiguous memory locations.

- It allows storing multiple values using a single variable name.
- Each element of an array can be accessed using an index.
- The index of an array always starts from 0.

Syntax —

datatype array-name[size];

One - Dimensional Array —

- Also known as a linear array.
- Stores elements in a single row (like a list).
- Elements are accessed using a single subscript.



### Example -

```
#include <stdio.h>
int main() {
    int marks[5] = {90, 85, 88, 92, 76};
    printf("First element = %d\n", marks[0]);
    printf("Third element = %d\n", marks[2]);
    return 0;
}
```

Output - First element = 90  
Third element = 88

### Two - Dimensional Array -

- known as a matrix or table - like array.
- Stores elements in rows and columns.
- Elements are accessed using two subscripts (row index and column index).

### Example -

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

```
printf("Element at [0][1] = %d\n", matrix[0][1]);  
printf("Element at [1][2] = %d\n", matrix[1][2]);  
return 0;  
}
```

Output: Element at [0][1] = 2  
Element at [1][2] = 6

Question - 2: Write a C program to initialize an array of 10 integers, input values from the user, and then display the largest and smallest element.

Answer -

1. Array Initialization - We create an integer array of size 10 using:  
`int arr[10];`

2. User Input - We use a for loop to take 10 integer values from the user.

3. Finding Largest and Smallest -

Initialize `largest = arr[0]` and `smallest = arr[0]`.

Traverse the array:

if `arr[i] > largest`, update largest.

if `arr[i] < smallest`, update smallest

4. Output - print the largest and smallest value.

Program in C -

```
#include <stdio.h>
int main() {
    int arr[10], i;
    int largest, smallest;

    // Input values from the user
    printf("Enter 10 integers:\n");
    for (i = 0; i < 10; i++) {
        scanf("%d", &arr[i]);
    }

    // Initialize largest and smallest
    largest = arr[0];
    smallest = arr[0];

    // Find largest and smallest
    for (i = 1; i < 10; i++) {
        if (arr[i] > largest)
            largest = arr[i];
        if (arr[i] < smallest)
            smallest = arr[i];
    }
}
```



// Output result

```
printf("Largest element = %d\n", largest);
```

```
printf("smallest element = %d\n", smallest);
```

```
return 0;
```

```
}
```

Sample Input : 12 45 3 67 23 89 5

Output : Largest element = 89

smallest element = 3

Question - 3: Explain the concept of string manipulation in C. Write a program to count the number of vowels and consonants in a given string.

Answer -

Concept of String Manipulation in C -

- In C, string is a sequence of characters terminated by a null character ('\0').
- Example - "Hello" is stored as H e l l o \0
- Strings are stored in character arrays.
- String manipulation means performing operations on strings such as -
  - Finding length (strlen)
  - Copying (strcpy)

- Concatenation (strcat)
- Comparison (strcmp)
- Searching (strchr, strstr)
- Counting vowels, consonants, words etc.

### Program —

```
#include <stdio.h>
#include <ctype.h> // for tolower()
int main() {
    char str[100];
    int i, vowels = 0, consonants = 0;

    // Input string
    printf("Enter a string:");
    fgets(str, sizeof(str), stdin);

    // Traverse each character
    for (i = 0; str[i] != '\0'; i++) {
        char ch = tolower(str[i]);
        // convert to lowercase for easy checking
        if (ch >= 'a' && ch <= 'z') {
            if (ch == 'a' || ch == 'e' || ch == 'i' ||
                ch == 'o' || ch == 'u')
```

```
vowels++;  
else {  
    consonants++;  
}  
}
```

// Output

```
printf("Number of vowels = %d\n", vowels);  
printf("Number of consonants = %d\n", consonants);  
return 0;  
}
```

Sample Input -

Enter a string: Programming in C

Output -

Number of vowels = 4

Number of consonants = 10

Question - 4 : What are functions in C?

Explain the advantage of using functions. Write the syntax of function definition, function call, and function prototype with an example.

Answer -



## What are Functions in C ?

A Function in C is a block of code that performs a specific task, grouped under a name, and can be reused multiple times.

- It improves modularity (program divided into smaller parts).
- Functions can be library functions (printf(), scanf(), sqrt()) or user-defined functions (created by the program).

## Advantages of Using Functions —

1. Reusability - Once written, a function can be used multiple times.
2. Modularity - Program is divided into smaller manageable parts.
3. Readability - Code becomes easier to read and understand.
4. Debugging - Errors are easier to find in smaller functions.
5. Collaboration - Different programmers can work on different functions of the same project.

## Syntax of Functions -

### 1. Function Prototype (Declaration) :

return\_type function\_name (parameter\_list);

Ex - int add (int, int);

### 2. Function Definition -

```
return_type function_name (parameter_list) {  
    // body of the function  
}
```

Ex - int add (inta, intb) {  
 return a+b;  
}

### 3. Function Call -

function\_name (arguments);

Ex . sum = add(10, 20);

### Program Example -

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

```
// Function prototype
```

```
int add (int, int);
```

```
int main() {
```

```
    int x, y, result;
```

```
printf ("Enter two numbers:");  
scanf ("%d %d", &n, &y);  
  
// Function call  
result = add (n,y);  
printf ("Sum = %d\n", result);  
return 0;  
}
```

```
// Function Definition  
int add (inta, intb) {  
    return a+b;  
}
```

Sample Input - Enter two numbers : 12 8

Output - 20

Question-5: Write a C program to calculate the factorial of a number using recursive function. Also explain the difference between recursive and non-recursive functions.

Answer -



## Concept of Recursion -

- Recursion in C means a function calling itself either directly or indirectly until a base condition is met.
- Every recursive function must have:
  - (a) Base Case - condition that stops the recursion.
  - (b) Recursive Case - function calls itself with modified arguments.
- Recursion is often used in problems like factorial, Fibonacci series, Towers of Hanoi, searching, and sorting.

## How Recursion Works ?

- Each recursive call is stored in the stack memory.
- When the base case is reached, the function returns and the stack starts unwinding.

## Example Program : Factorial using Recursion -

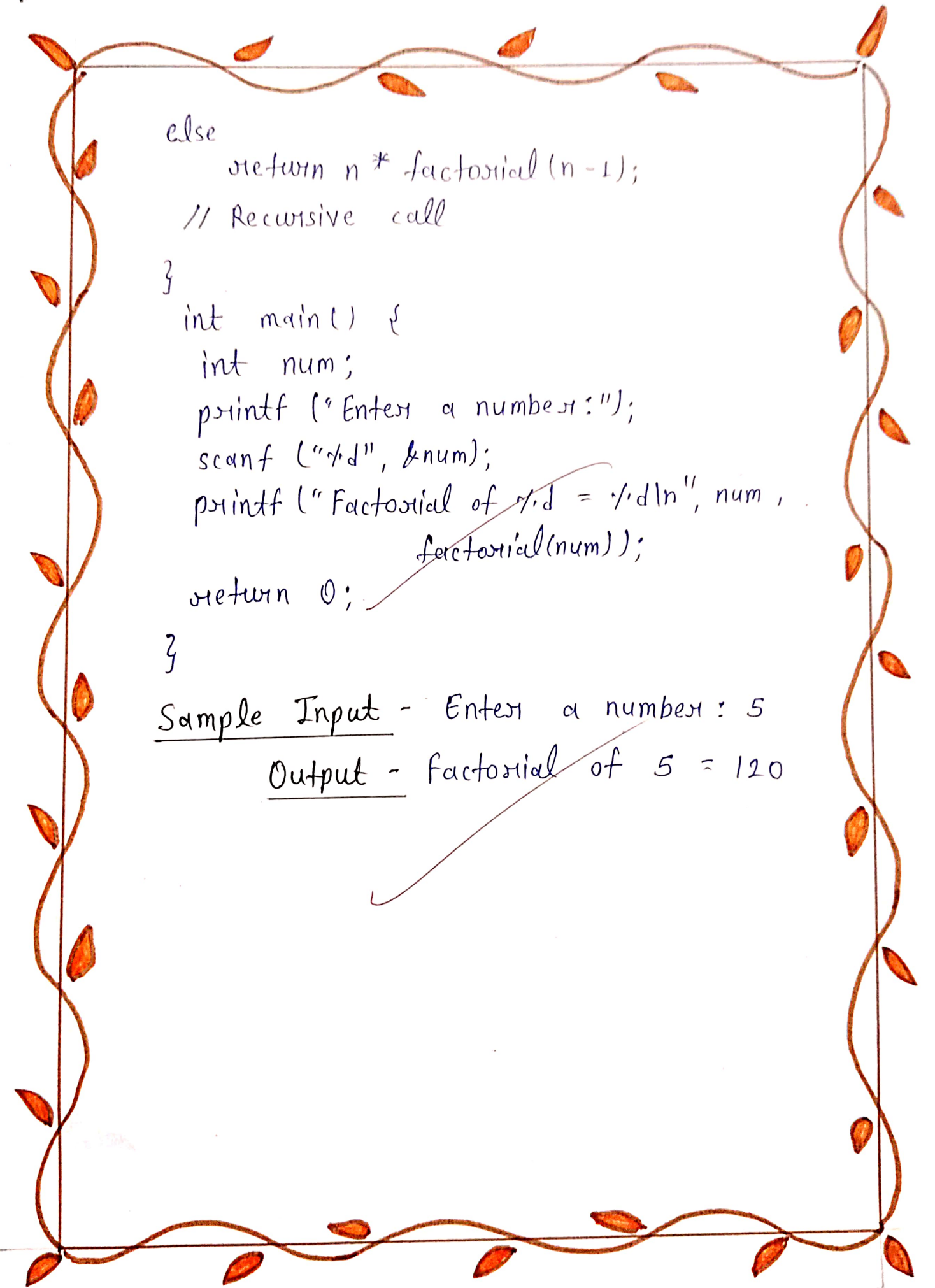
```
#include <stdio.h>
```

```
// Recursive function to calculate factorial
```

```
int factorial(int n) {
```

```
    int if (n == 0 || n == 1) // Base case
```

```
        return 1;
```



```
else
    return n * factorial(n-1);
// Recursive call
}

int main() {
    int num;
    printf("Enter a number:");
    scanf("%d", &num);
    printf("Factorial of %d = %d\n", num,
        factorial(num));
    return 0;
}
```

Sample Input - Enter a number: 5

Output - Factorial of 5 = 120

# Shakuntala Krishana Institute Of Technology



College Code : - KD64

Affiliated To CSJM University Kanpur

## Assignment

Assignment No. : ..... 4<sup>th</sup> .....

Subject : ..... C Programming .....

Assignment Topic

Submitted To :

..... Miss Aarti Maam .....

Submitted By :

..... Devika .....

Subject Teacher Of :

..... Miss Aarti Maam .....

Course :

..... BCA .....

Department :

..... BCA .....

Semester :

..... 1<sup>st</sup> .....

10/09/2023



Question - 1: Explain Bubble sort ,  
Selection Sort , and  
Insertion Sort with algorithms and write  
a C program to implement any  
one of them.

Answer —

Bubble Sort —

Bubble sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process is repeated until the array becomes sorted.

Algorithm (Ascending Order) —

1. Start with the first element.
2. Compare it with the next element.
3. If the first is greater , swap them.
4. Repeat this for the entire array (one pass)

5. Repeat steps until no swaps are needed.

### Time Complexity -

- Best case :  $O(n)$  (already sorted)
- Worst case :  $O(n^2)$

## 2. Selection Sort -

Selection sort finds the minimum element in the unsorted part of the array and places it at the beginning.

### Algorithm (Ascending Order) -

1. Starts from the first index.
2. Find the minimum element from the unsorted part.
3. Swap it with the current index element.
4. Move the boundary of sorted part one step forward.
5. Repeat until array is sorted.

### Time Complexity -

- Always  $O(n^2)$  (independent of input order).

### 3. Insertion Sort -

Insertion sort builds the sorted array one element at a time, inserting each new element into its correct position.

#### Algorithm (Ascending Order) -

1. Assume first element is sorted.
2. Take the next element and insert it into the correct position in the sorted part.
3. Shift elements if needed.
4. Repeat for all elements.

#### Time Complexity -

- Best case :  $O(n)$  (already sorted)
- Worst case :  $O(n^2)$

### C Program : Bubble Sort Implementation

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

```
void bubble sort (int arr[], int n) {
```



```

int i, j, temp;
for (i = 0; i < n - 1; i++) {
    for (j = 0; j < n - 1; j++) {
        if (arr[j] > arr[j+1]) {
            // swap
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}

int main() {
    int n, i;
    printf("Enter number of elements:");
    scanf("%d", &n);
    int arr[n];
    printf("Enter elements: \n");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    bubblesort(arr, n);
    printf("Sorted array:");
    for (i = 0; i < n; i++) {
        printf("%d", arr[i]);
    }
    return 0;
}

```

Question - 2: Explain Quick Sort and Merge Sort algorithms with examples.

Answer —

1. Quick Sort —

Quick Sort is a divide and conquer algorithm. It picks a pivot element, partitions the array into two sub-arrays.

- Left sub-array - elements smaller than pivot.
- Right sub-array - elements greater than pivot.

Then recursively sorts the sub-arrays.

Algorithm —

1. If array has 0 or 1 element  $\rightarrow$  ~~or already~~ sorted.
2. Select a pivot (commonly first / last / middle element).
3. Rearrange elements so that:

- All smaller elements go before pivot.
- All greater elements go after pivot.
- 4. Recursively apply Quick Sort on left and right sub-arrays.

Example —

Array = [20, 10, 30, 5]

- Choose pivot = 20
- Partition = [10, 5] | 20 | [30]
- Recursively sort = [5, 10] | 20 | [30]
- Final = [5, 10, 20, 30]

Time Complexity

- Best / Average case :  $O(n \log n)$
- Worst case :  $O(n^2)$  (if array is already sorted and bad pivot is chosen)



## 2. Merge Sort -

Merge sort is also a divide and conquer algorithm. It repeatedly divides the array into two halves, sorts them recursively, and merges them.

### Algorithm -

1. If array has 1 element  $\rightarrow$  return
2. Divide array into two halves.
3. Recursively sort both halves.
4. Merge the two sorted halves.

### Example:

Array = [20, 10, 30, 5]

- Divide  $\rightarrow$  [20, 10] and [30, 5]
- Sort individually  $\rightarrow$  [10, 20], [5, 30]
- Merge  $\rightarrow$  [5, 10, 20, 30]

### Time Complexity:

- Always  $O(n \log n)$
- Needs extra space for merging

## Program Example -

```
#include <stdio.h>
void quickSort (int a[], int low, int high) {
    int i = low, j = high, pivot = a[(low + high)/2], temp;
    while (i <= j) {
        while (a[i] < pivot) i++;
        while (a[j] > pivot) j--;
        if (i <= j) {
            temp = a[j]; a[j] = a[i]; a[i] = temp;
            i++; j--;
        }
    }
    if (low < j) quickSort(a, low, j);
    if (i < high) quickSort(a, i, high);
}

int main() {
    int n, i, a[50];
    printf("Enter n:"); scanf("%d", &n);
    for (i = 0; i < n; i++) scanf("%d", &a[i]);
    quickSort(a, 0, n-1);
    for (i = 0; i < n; i++) printf("%d", a[i]);
}
```

Question - 3: Differentiate between Linear Search and Binary Search with algorithms and write a program for Binary Search.

Answer -

Linear Search -

- Check each element of the array one by one until we find the key or reach the end.
- Works on unsorted arrays also.

Algorithm -

1. Start from first element.
2. Compare element with key.
3. If found  $\rightarrow$  return index.
4. Else  $\rightarrow$  continue till end.
5. If not found  $\rightarrow$  return -1.



### Time Complexity -

Best :  $O(1)$

Worst :  $O(n)$

### Binary Search -

- Works only on sorted arrays.
- Repeatedly divides the array into halves to search the key.

### Algorithm -

1. Set  $low = 0$ ,  $high = n - 1$ .
2. Find  $mid = (low + high) / 2$
3. If  $key = a[mid] \rightarrow$  found
4. If  $key < a[mid] \rightarrow$  search left half.
5. Else  $\rightarrow$  search right half.
6. Repeat until  $low > high$ .

### Time Complexity -

- Best :  $O(1)$
- Worst :  $O(\log n)$

## Difference Table -

| Feature          | Linear Search                    | Binary Search      |
|------------------|----------------------------------|--------------------|
| Requirement      | Works on unsorted / sorted array | Needs sorted array |
| Method           | Sequential check                 | Divide and conquer |
| Time Complexity  | $O(n)$                           | $O(\log n)$        |
| Space Complexity | $O(1)$                           | $O(1)$             |
| Efficiency       | Less efficient                   | More efficient     |

## Program -

```
#include <stdio.h>
int main() {
    int n, i, key, low, high, mid, a[50];
    printf("Enter n:"); scanf("%d", &n);
    printf("Enter sorted elements: \n");
    for (i = 0; i < n; i++) scanf("%d", &a[i]);
    printf("Enter key:"); scanf("%d", &key);
```

```
low = 0; high = n - 1;
while (low <= high) {
    mid = (low + high) / 2;
    if (a[mid] == key) {
        printf ("Found at index %d", mid);
        return 0;
    }
    else if (a[mid] < key) low = mid + 1;
    else high = mid - 1;
}
printf ("Not Found");
return 0;
}
```

Question - 4: What are Structures in C? Explain their advantages with examples. Write a program using Array of Structures to store and display student details.

Answer -



## What are Structures in C?

A Structure in C is a user-defined data type that groups different types of variables under one name.

It is used when we want to store multiple related data items together (like student details: roll number, name, marks).

### Syntax —

```
struct structure_name {  
    data type member 1;  
    data type member 2;  
    ...  
};
```

### Advantages of Structures —

1. Group different data types (int, char, float, etc.) together.
2. Better data organization (Ex - Student info, Employee info).
3. Code reusability (can be used in arrays, functions, pointers).
4. Memory management (related data is stored together).

Example -

```
struct student {  
    int roll;  
    char name[20];  
    float marks;  
};
```

Program -

```
#include <stdio.h>  
struct student {  
    int roll;  
    char name[20];  
    float marks;  
};  
int main () {  
    int n, i;  
    printf ("Enter number of students:");  
    scanf ("%d", &n);  
    struct Student s[n];  
    for (i=0; i<n; i++) {  
        printf ("\n Enter roll, name, marks:");  
        scanf ("%d %s %f", &s[i].roll, s[i].name,  
                &s[i].marks);  
    }
```

```

    }
    printf("\n... Student Details ... \n");
    for (i=0; i < n; i++) {
        printf("Roll: %d Name: %s Marks: %.2f\n",
            s[i].roll, s[i].name, s[i].marks);
    }
    return 0;
}

```

Question-5: What are Pointers in C? Explain with example.

Write a program to demonstrate pointers with arrays.

Answer —

Pointers —

- A pointer is a variable which stores the address of another variable.
- Declared using \* operator.
- Used for dynamic memory, arrays, strings, and functions.



Syntax -

```
data_type *ptr :  
ptr = &variable;
```

### Advantages of Pointers —

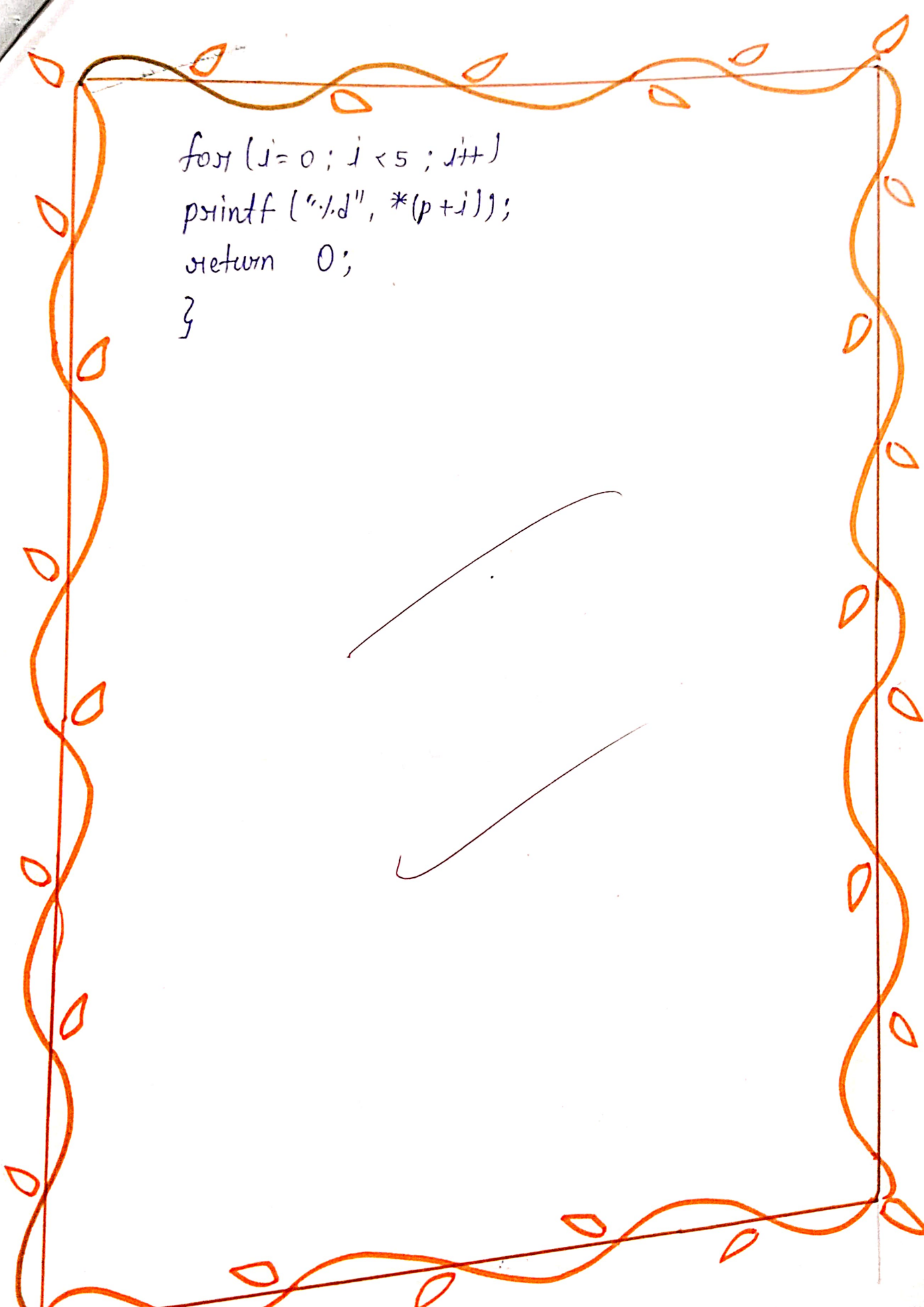
1. Efficient array handling.
2. Dynamic memory allocation (malloc, calloc).
3. Call by reference in functions.
4. Used for data structures (linked list, trees).

Example —

```
int a = 10;  
int *p = &a;  
printf("%d", *p);
```

Program —

```
#include <stdio.h>  
int main() {  
    int a[5], i, *p;  
    printf("Enter 5 numbers:");  
    for (i=0; i<5; i++)  
        scanf("%d", &a[i]);  
    p = a;  
    printf("In Array elements using pointer:\n");
```



```
for (j = 0; j < 5; j++)  
    printf("%.d", *(p+j));  
return 0;  
}
```

# Shakuntala Krishana Institute Of Technology



College Code : - KD64

Affiliated To CSJM University Kanpur

## Assignment

Assignment No. : ..... 5<sup>th</sup> .....

Subject : ..... C Programming .....

Assignment Topic

Submitted To :

..... Miss Aarti Mam .....

Submitted By :

..... Devika .....

Subject Teacher Of :

..... Miss Aarti Mam .....

Course :

..... BCA .....

Department :

..... BCA .....

Semester :

..... 1<sup>st</sup> .....



Question - 1 Explain File handling in C. Write a program to read and write data using file function (fopen, fclose, fprintf, fscanf).

Answer —

File Handling

File handling in C allows data to be stored permanently in files on the disk.

It is done using functions from `<stdio.h>`.

A file must be opened before use and close after operations are done.

C uses a file pointer of type `FILE*` to access file.

Common Functions —

- `fopen("filename", "mode")` → Open file.
- `fprintf()` → writes formatted data to file.

- `fscanf()` → Reads formatted data from file.
- `fclose()` → Closes the file.

### File Modes —

- "r" — Read
- "w" — Write (creates new file or overwrites)
- "a" — Append

### Program —

```
#include <stdio.h>
int main() {
    FILE *fp;
    char name[30];
    int age;

    fp = fopen("data.txt", "w");
    if (fp == NULL) {
        printf("File not found!");
        return 0;
    }
    fprintf(fp, "Name: %s\nAge: %d", "Riya", 20);
    fclose(fp);
}
```

```
fp = fopen("data.txt", "r");  
fscanf(fp, "Name: %s\nAge: %d", name, &age);  
printf("Name: %s\nAge: %d", name, age);  
fclose(fp);  
return 0;  
}
```

Output —

~~Name : Riya  
Age : 20~~



Question - 2: What are file functions in C? Explain the use of `getc`, `putc`, `getw`, `putw`, `fread`, `fwrite`, `fgets`, `fputs`, `feof` with examples.

Answer -

File Functions — C provides several functions to perform input/output operations on files. These functions are declared in the `<stdio.h>` header file.

| Function                 | Purpose                               | Example                     |
|--------------------------|---------------------------------------|-----------------------------|
| <code>getc(fp)</code>    | Reads a single character from a file. | <code>ch = getc(fp);</code> |
| <code>putc(ch,fp)</code> | Writes a single character to a file.  | <code>putc(ch,fp);</code>   |

|                   |                                          |                               |
|-------------------|------------------------------------------|-------------------------------|
| getw(fp)          | Reads an integer from a file.            | n = getw(fp);                 |
| putw(n,fp)        | Writes an integer to a file.             | putw(n,fp);                   |
| fread()           | Reads a block of data file (binary).     | fread(&a, sizeof(a), 1, fp);  |
| fwrite()          | Writes a block of data to file (binary). | fwrite(&a, sizeof(a), 1, fp); |
| fgets(str, n, fp) | Reads a string from a file.              | fgets(name, 20, fp);          |
| fputs(str, fp)    | Writes a string to a file.               | fputs("Hello", fp);           |
| feof(fp)          | Checks end of file condition.            | while(!feof(fp))<br>{ - - - } |

## Program —

```
#include <stdio.h>
int main() {
    FILE *fp;
    char ch;
    fp = fopen("sample.txt", "w");
    fputs("Hello C File", fp);
    fclose(fp);
    fp = fopen("sample.txt", "r");
    while ((ch = getc(fp)) != EOF)
        putchar(ch);
    fclose(fp);
    return 0;
}
```

Output —

Hello C File



Question - 3: What is the Preprocessor in C? Explain #define, #include, #undef, and conditional compilation directives with examples.

Answer —

Preprocessor in C —

The C preprocessor is a program that processes source code before compilation. It handles macros, file inclusion, and conditional compilation.

All preprocessor commands begin with the symbol #.

1. #define —

Used to define constants or macros.

Example -

```
#define PI 3.14  
#define AREA(x) (PI * x * x)  
printf("Area = %.2f", AREA(5));
```

Output -

Area = 78.50

2. #include —

Used to include header files or user-defined files.

Example -

```
#include <stdio.h>  
#include "myfile.h"
```

3. #undef —

Removes a previously defined macro.

Example -

```
#define VALUE 10  
#undef VALUE
```

#### 4. Conditional Compilation Directives -

Used to compile parts of a program conditionally.

Example -

```
#define DEBUG
#ifdef DEBUG
printf("Debug mode ON\n");
#else
printf("Debug mode OFF\n");
#endif
```

Output -

Debug mode ON



Question - 4 : Write short notes on Standard Library and Header Files in C. Explain the importance of `<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<math.h>`, and `<time.h>`.

Answer —

Standard Library and Header Files —

In C, header files contain predefined functions, macros, and definitions that make programming easier.

They are included at the beginning of a program using the `#include` directive.

Example - `#include <stdio.h>`

## Important Header Files -

| Header File | Purpose                                                         | Common Functions                                         |
|-------------|-----------------------------------------------------------------|----------------------------------------------------------|
| <stdio.h>   | Standard Input/Output functions                                 | printf(), scanf(),<br>getc(),putc(),<br>fopen(),fclose() |
| <stdlib.h>  | General utilities like memory, process control, and conversion. | malloc(), free(),<br>exit(), atoi(),<br>rand()           |
| <string.h>  | String handling functions                                       | strlen(), strcpy(),<br>strcmp(), strcat()                |
| <math.h>    | Mathematical operations                                         | sqrt(), pow(),<br>sin(), cos(), abs()                    |
| <time.h>    | Time and data handling                                          | time(), ctime(),<br>difftime(), local<br>time()          |

### Example -

```
#include <stdio.h>
#include <math.h>
int main() {
    double n = 25;
    printf("Square root = %.2f", sqrt(n))
    return 0;
```

Question - 5 : Explain commonly used String, Mathematical, and Time functions in C with examples.

Answer —

1. String Functions — Used to perform operations on strings such as copying, comparing and finding length.

- `strlen(str)` — Returns length of string
- `strcpy(d, s)` — Copies one string to another
- `strcmp(s1, s2)` — Compares two strings.
- `strcat(s1, s2)` — Concatenates strings.

Example —

```
#include <stdio.h>
#include <string.h>
int main() {
    char s1[20] = "Hello", s2[20] = "World";
    strcat(s1, s2);
    printf("%s", s1);
    return 0;
}
```



## 2. Math Function

Used for performing mathematical calculations.

`sqrt(x)` → square root

`pow(x,y)` → Power Function

`abs(x)` → Absolute Value

`ceil(x)` → Rounds up

`floor(x)` → Rounds down

### Example -

```
#include <stdio.h>
#include <math.h>
int main() {
    printf("%.2f", pow(3,2));
    return 0;
}
```

Output - 9.00

### 3. Time Function —

Used to work with data and time.

`time()` → Returns current time in seconds

`ctime()` → Converts time to readable string

`localtime()` → Converts time to structure format

`difftime()` → Finds difference between two times.

#### Example —

```
#include <stdio.h>
#include <time.h>
int main() {
    time_t t;
    time(&t);
    printf("Current time: %s", ctime(&t));
    return 0;
}
```