3 MARK

SUBMODULE 1

1.What are the rules of pointer operations?(3mark)

Three main rules:

1. Addition/Subtraction: You can add or subtract an integer to/from a pointer (e.g., ptr + 1, ptr - 2).
2. Pointer Comparison: You can compare two pointers of the same type (e.g., ptr1 == ptr2).
3. Dereferencing: You can access the value at the memory location using the dereference operator * (e.g., *ptr)

2.Differentiate between char name[] and char *name in C.(3mark)

char name[]	char *name
Declares a character array	Declares a pointer to a character
Allocated at compile time (static memory)	Can point to any memory (heap, static, etc.)
Contents can be modified	If pointing to a string literal, it's read-only
Stores characters directly	Stores the address of the first character
Fixed at declaration	Not fixed; can point to different strings

3.What do you mean by a pointer variable? How is it initialized? (3mark)

A pointer variable stores the **address** of another variable.

It is declared using an asterisk *.
Example: int *ptr; (pointer to an integer)

It is initialized using the address-of operator &.

Example:

```
int a = 10;
ptr = &a;
```

SUBMODULE 2

4.How can an array be passed to a function using a pointer? Provide an example. (3mark)

An array can be passed to a function by passing the address of its first element (i.e., a pointer).

Eg:

```
void display(int *arr, int size) {
   for(int i = 0; i < size; i++)
      printf("%d ", *(arr + i));
}
int main() {
   int a[] = {1, 2, 3};
   display(a, 3);
   return 0;
}
```

5.differentiate between an array of pointers and a pointer to an array. (3mark)

1. Array of Pointers:
    - It is a collection of pointers.
    - Example: int *arr[5]; (5 pointers to int)
    - used for pointing to multiple variables or strings.
2. Pointer to an Array:
    - It is a pointer pointing to the entire array.
    - Example: int (*ptr)[5]; (pointer to an array of 5 ints)
    - used when handling whole arrays in functions.

6. Explain the concept of a function to return a pointer to the calling function with example. (3mark)

A function can return a pointer to give back the address of a variable or array to the calling function.

```
int* getMax(int *a, int *b) {
    return (*a > *b) ? a : b;
}
int main() {
    int x = 10, y = 20;
    int *max = getMax(&x, &y);
    printf("Max = %d", *max);
    return 0;
}
```

SUBMODULE 3

7.What is an array of pointers, and how is it useful in string manipulation? (3mark)

An array of pointers is a collection of pointers that can each point to a string or a character array.
Example:

```
char *names[] = {"John", "Mary", "Alex"};
```

Each element in names points to a different string. This is useful in string manipulation because:

- Strings can be of different lengths.
- We can easily sort, compare, or access strings using their pointers.
- Saves memory by storing only addresses.

8.How does pointer arithmetic help in traversing a string? (3mark)

Pointer arithmetic allows us to move through a string character by character using a pointer.

Example:

```
char *str = "Hello";
while (*str != '\0') {
   printf("%c ", *str);
   str++; // move to next character
}
```

Using str++, we move the pointer to the next character. This makes string traversal faster and memory efficient compared to using array indexing.

9.Explain the difference between *ptr++ and (*ptr)++ if ptr is pointing to the first element of an array. (3mark)

*ptr++ means:
First use the value pointed by ptr, then move the pointer to the next element.
It moves the pointer.

(*ptr)++ means:
Increase the value stored at the current pointer location.
It changes the value, not the pointer.

Example:
If ptr points to a[0] = 5:

- *ptr++ → pointer moves to a[1], value still 5
- (*ptr)++ → a[0] becomes 6, pointer stays at a[0]

SUBMODULE 4

10.Explain the difference between accessing structure members using a structure variable and a pointer to structure. Provide an example. (3mark)

When using a structure variable, we use the dot (.) operator to access members.

When using a pointer to a structure, we use the arrow (->) operator.

Example:

```
struct Student {
   int id;
   char name[20];
};
struct Student s = {101, "Alanta"};
struct Student *ptr = &s;
printf("%d", s.id);
printf("%d", ptr->id);
```

11.Explain the effects of the following statements:

a) int a, *b=&a;

b) int p, *p;

c) char *s;   (3mark)

int a, *b = &a;

- Declares an integer a and a pointer b that stores the address of a.

int p, *p;

- This is incorrect because p is declared twice: once as an int and once as a pointer.
- This causes a redefinition error.

char *s;

- Declares a pointer s that can point to a character or string.
- Commonly used in string manipulation.

12.What is a pointer to a structure in C? Explain with an example program how to access structure members using a pointer. (3mark)

A pointer to a structure stores the address of a structure variable. It allows indirect access to structure members using the -> operator.

Example:

```
#include <stdio.h>
struct Student {
   int id;
   char name[20];
};
int main() {
   struct Student s = {101, "Alanta"};
   struct Student *ptr = &s;
   printf("ID: %d\n", ptr->id);
   printf("Name: %s\n", ptr->name);
   return 0;
}
```

SUBMODULE 5

13.What are the consequences of not using free() after dynamic memory allocation in C? (3mark)

If we don’t use free(), the memory stays reserved and is not released.

This causes memory leaks, which means wasted memory.

Over time, the program may slow down or crash due to less available memory.

14.What are the differences between malloc() and calloc() in C? (3mark)

- malloc() allocates memory but does not set it to zero.
- calloc() allocates memory and initializes all bytes to zero.
- malloc() takes 1 argument (total size), while calloc() takes 2 arguments (number of blocks and size of each).

Example:

int *a = (int*) malloc(5 * sizeof(int));

int *b = (int*) calloc(5, sizeof(int));

15.What is dynamic memory allocation? Why is it needed? (3mark)

- Dynamic memory allocation means assigning memory during program execution using functions like malloc(), calloc(), etc.
- It is needed when we don't know in advance how much memory is required.
- It helps to use memory efficiently and is useful for things like arrays, linked lists, and large data.

Example:

int *arr = (int*) malloc(10 * sizeof(int));

SUBMODULE 6

16.How can you perform read and write operations on an unformatted data file in C? (3mark)

- We use fread() to read and fwrite() to write unformatted (binary) data.
- These functions deal with raw bytes, not text.

Example:

fwrite(&num, sizeof(num), 1, fp);   // Write to file

fread(&num, sizeof(num), 1, fp);

17.When a program is terminated, all the files used by it are automatically closed. Why is it then necessary to close a file during execution of the program? (3mark)

Closing a file using fclose() saves data and clears the file buffer.

It frees system resources used by the file.

Prevents data loss or file corruption, especially during write operations.

18.Write any three file-handling functions in C. (3mark)

fopen() – Used to open a file.
Example: fp = fopen("data.txt", "r");

fclose() – Closes the opened file.
Example: fclose(fp);

fprintf() / fscanf() – For writing to and reading from a file (formatted).
Example: fprintf(fp, "%d", num);

SUBMODULE 7

19.Explain the use of fseek() and ftell() in file handling. (3mark)

fseek(fp, offset, origin) – Moves the file pointer to a specific position.
Example: fseek(fp, 0, SEEK_END); moves to the end of the file.

ftell(fp) – Returns the current position of the file pointer (in bytes).
Example: long pos = ftell(fp);

20.What is the difference between fgets() and fscanf() for reading a file? (3mark)

fgets() reads a full line (including spaces) as a string.

fscanf() reads formatted data (like integers, strings) and stops at spaces.

fgets() is safer for reading strings; fscanf() is good for reading structured input.

21.What is the purpose of the FILE pointer in C? (3mark)

It is used to handle files in C.

It points to a file and keeps track of reading or writing position.

Needed for functions like fopen(), fclose(), fread(), etc.

SUBMODULE 8

22.Is it possible to read from and write to the same file without resetting the file pointer? Justify your answer. (3mark)

Yes, using modes like "r+", "w+", or "a+".

But use fseek() or fflush() between reading and writing.

Otherwise, the behavior may be incorrect.

23.Explain the purpose of ftell(fp) in C. If ftell(fp) returns 50, what does this indicate? (3mark)

ftell(fp) gives the current position in the file (in bytes).

If it returns 50, the file pointer is at the 51st byte.

Useful for knowing or storing file positions.

24. Explain the purpose and usage of the fseek() and fwrite() functions in file handling. (3mark)

fseek() moves the file pointer to a specific position.

fwrite() writes data from memory to a file.

Together, they help write data at the desired location in a file.

9 MARK

SUBMODULE 1

25.Write a function (using pointer parameters) that compares two integer arrays to see whether they are identical. The function returns 1 if they are identical, 0 otherwise. (9 mark)

```
#include <stdio.h>
int Iden(int *arr1, int *arr2, int size) {
    for (int i = 0; i < size; i++) {
        if (*(arr1 + i) != *(arr2 + i)) {
            return 0;
        }
    }
    return 1;
}
int main() {
    int a[] = {1, 2, 3, 4};
    int b[] = {1, 2, 3, 4};
    int c[] = {1, 2, 0, 4};
    int size = sizeof(a) / sizeof(a[0]);
   if (Iden(a, b, size))
        printf("a and b are identical.\n");
    else
        printf("a and b are not identical.\n");
   if (Iden(a, c, size))
        printf("a and c are identical.\n");
    else
        printf("a and c are not identical.\n");
    return 0;
}
```

26. What is the difference between a pointer and a normal variable? Provide examples to support your answer. (9mark)

A normal variable stores a value directly.

A pointer stores the address of another variable.

Pointers allow indirect access to values.

Pointers are declared using the * symbol (e.g., int *ptr;).

Useful in dynamic memory allocation, arrays, strings, and functions.

Can be used to access or modify data stored in another variable.

Help in efficient memory usage and faster execution in many cases.

Allows passing variables by reference to functions.

Pointer dereferencing (*ptr) is used to access the value at the stored address.

Example:

```
#include <stdio.h>
int main() {
   int a = 10;
   int *ptr = &a;
   printf("Value of a: %d\n", a);
   printf("Address of a: %p\n", &a);
   printf("Value of ptr: %p\n", ptr);
   printf("Value at ptr: %d\n", *ptr);
   return 0;
}
```

27. Explain pointer arithmetic in C with an example. Write a C program to demonstrate pointer arithmetic (increment, decrement, addition, subtraction, and pointer difference) using an integer array. Provide a detailed explanation of the output. (9mark)

Pointer arithmetic means performing operations on pointer variables.

Common operations:

- ptr++ → Move to next memory location
- ptr-- → Move to previous location
- ptr + n → Jump forward by n elements
- ptr - n → Jump backward by n elements
- ptr2 - ptr1 → Difference (number of elements)

Example:

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   int *p = arr;
   int *q = &arr[3];
   printf("Initial: %d\n", *p);
   p++;
   printf("p++: %d\n", *p);
   p--;
   printf("p--: %d\n", *p);
   p = p + 2;
   printf("p + 2: %d\n", *p);
   p = p - 1;
   printf("p - 1: %d\n", *p);
   printf("q - p: %d\n", q - p);
   return 0;
```

Output:

```
Initial: 10

p++: 20

p--: 10

p + 2: 30

p - 1: 20

q - p: 2
```

```
}
```

SUBMODULE 2

28. Write a function using pointers to add two matrices and to return the resultant matrix to the calling function. (9mark)

```
#include <stdio.h>
void addMatrices(int *a, int *b, int *res, int rows, int cols) {
   for (int i = 0; i < rows * cols; i++) {
      *(res + i) = *(a + i) + *(b + i);
   }
}
int main() {
   int a[2][2] = {{1, 2}, {3, 4}};
   int b[2][2] = {{5, 6}, {7, 8}};
   int res[2][2];
   addMatrices((int *)a, (int *)b, (int *)res, 2, 2);
   printf("Resultant Matrix:\n");
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         printf("%d ", res[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

Output:

```
Resultant Matrix:
6 8
10 12
```

29. Write a program to calculate the sum of two numbers which are passed as arguments using the call by reference method. (9mark)

```
#include <stdio.h>
void add(int *a, int *b, int *sum) {
   *sum = *a + *b;
}
int main() {
   int num1, num2, result;
   printf("Enter first number: ");
   scanf("%d", &num1);
   printf("Enter second number: ");
   scanf("%d", &num2);
```

Output:

```
Enter first number: 12
Enter second number: 22
Sum = 34
```

```
    add(&num1, &num2, &result);
    printf("Sum = %d\n", result);
    return 0;
}
```

30. Explain how pointers can be passed to functions with an example. (9mark)

Pointers can be passed to functions to provide indirect access to variables.

This allows the function to modify the original variables in the caller function.

Call by Reference:

When a pointer is passed, the function works with the memory address of the variable.

Changes made inside the function will reflect in the original variable.

Syntax:

Function declaration: void functionName(int *ptr);

Function call: functionName(&variable);

Example:

```
#include <stdio.h>
void doubleValue(int *n) {
    *n = *n * 2;
}
int main() {
    int num;
    scanf("%d", &num);
    doubleValue(&num);
    printf("%d\n", num);
    return 0;
}
```

SUBMODULE 3

31.How do you declare and initialize a pointer to a pointer? Write a program to demonstrate this concept. (9mark)

A pointer to a pointer (also called double pointer) stores the address of another pointer.

Syntax:

```
int a = 10;
int *p = &a;
int **pp = &p;
```

program:

```
#include <stdio.h>
int main() {
    int a = 10;
    int *p = &a;
    int **pp = &p;
    printf("Value of a = %d\n", a);
    printf("Value using *p = %d\n", *p);
    printf("Value using **pp = %d\n", **pp);
    return 0;
}
```

32. Write a C program to reverse a string using pointers. (9mark)

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];
    char *start, *end, temp;
    scanf("%s", str);
    start = str;
    end = str + strlen(str) - 1;
    while (start < end) {
        temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
    printf("Reversed string: %s\n", str);
    return 0;
}
```

33. Write a C program to print the elements of an array in reverse order using pointers. (9mark)

```
#include <stdio.h>
int main() {
    int arr[5], *ptr;
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }
    ptr = arr + 4;
    for (int i = 0; i < 5; i++) {
        printf("%d ", *ptr);
        ptr--;
    }
    return 0;
}
```

SUBMODULE 4

34. Write a program using pointers to compute the sum of all elements stored in an array. (9mark)

```
#include <stdio.h>
int main() {
    int arr[5], *ptr, sum = 0;
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }
    ptr = arr;
    for (int i = 0; i < 5; i++) {
        sum += *(ptr + i);
    }
    printf("Sum = %d\n", sum);
    return 0;
}
```

35. Write a function using pointers to exchange the values stored in two locations in the memory. (9mark)

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 5, y = 10;
    swap(&x, &y);
    printf("After swap: x = %d, y = %d\n", x, y);
    return 0;
}
```

36. What are the advantages of using function pointer in C? Write a program that uses a function pointer as a function argument. (9mark)

advantages:

Enables dynamic function calls.

Allows callbacks (passing functions as arguments).

Helps in implementing menus, plugins, or event-driven programs.

Provides flexibility and reduces code duplication.

Program:

```
#include <stdio.h>
void add(int a, int b) {
   printf("Sum = %d\n", a + b);
}
void operate(void (*func)(int, int), int x, int y) {
   func(x, y);
}
int main() {
   operate(add, 4, 5);
   return 0;
}
```

SUBMODULE 5

37.Explain the purpose and working of malloc(), calloc(), realloc(), and free() functions in C with suitable examples. (9mark)

malloc() (Memory Allocation):

- Allocates memory block of given size (in bytes).
- Returns void pointer.
- Memory is not initialized.

```
int *ptr = (int *)malloc(5 * sizeof(int));
```

calloc() (Contiguous Allocation):

- Allocates memory for multiple blocks.
- Initializes memory to zero.
- Returns void pointer.

```
int *ptr = (int *)calloc(5, sizeof(int));
```

realloc() (Re-allocation):

- Changes size of previously allocated memory.
- Used to expand or shrink memory.

```
ptr = (int *)realloc(ptr, 10 * sizeof(int));
```

free():

- Releases memory allocated by malloc(), calloc(), or realloc().
- Prevents memory leaks.

```
free(ptr);
```

38. What is dynamic memory allocation in C? How does it differ from static memory allocation? Explain with an example. (9mark)

Dynamic Memory Allocation:

- Memory is allocated at runtime using functions like malloc(), calloc(), realloc().
- Size can be changed during program execution.
- Example:

```
int *ptr = (int *)malloc(5 * sizeof(int));
```

Static Memory Allocation:

- Memory is allocated at compile time.
- Fixed size, cannot change during execution.
- Example:

```
int arr[5];
```

difference:

Allocation Time:

- Static: At compile time
- Dynamic: At runtime

Size Flexibility:

- Static: Fixed size
- Dynamic: Can be resized

Memory Control:

- Static: Controlled by system
- Dynamic: Controlled by programmer

Functions Used:

- Static: No special functions needed
- Dynamic: Uses malloc(), calloc(), etc.

Memory Release:

- Static: Done automatically
- Dynamic: Must use free() manually

Example:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int a[3] = {1, 2, 3};
```

```
    printf("Static Array: %d %d %d\n", a[0], a[1], a[2]);
    int *b = (int *)malloc(3 * sizeof(int));
    b[0] = 4;
    b[1] = 5;
    b[2] = 6;
    printf("Dynamic Array: %d %d %d\n", b[0], b[1], b[2]);
   free(b);
    return 0;
}
```

39. Write a C program to dynamically allocate memory for an array of integers, take input values from the user, and print the array. Use malloc() for allocation. (9mark)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *arr, n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Array elements are:\n");
    for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
    }
    free(arr);
    return 0;
}
```

SUBMODULE 6

40.Explain the different modes of opening a file with an example. (9mark)

"r"	Open for reading. File must exist.
"w"	Open for writing. Creates a new file or overwrites if file exists.
"a"	Open for appending. Writes at end. Creates file if not exists.
"r+"	Open for both reading and writing. File must exist.
"w+"	Open for both reading and writing. Overwrites existing file.
"a+"	Open for reading and appending. Creates file if not exists.

Example:

```
FILE *fp;
fp = fopen("example.txt", "w");
if (fp == NULL) {
    printf("File cannot be opened.\n");
}
```

41. Explain any 5 file handling functions in C. (9mark)

1. fopen()

- Opens a file in a given mode.

```
FILE *fp = fopen("data.txt", "r");
```

2. fprintf()

- Writes formatted output to a file.

```
fprintf(fp, "Hello, File!");
```

3. fscanf()

- Reads formatted input from a file.

```
fscanf(fp, "%d", &num);
```

4. fgetc()

- Reads a single character from a file.

```
char ch = fgetc(fp);
```

5. fclose()

- Closes the opened file.

```
fclose(fp);
```

42. Write a program to read data from the keyboard, write it to a file called INPUT, again read the same data from the INPUT file, and display it on the screen. (9mark)

```
#include <stdio.h>
int main() {
    FILE *fp;
    char data[100];
    printf("Enter some text: ");
    fgets(data, sizeof(data), stdin);
    fp = fopen("INPUT.txt", "w");
    if (fp == NULL) {
        printf("File cannot be opened.\n");
        return 1;
    }
    fputs(data, fp);
    fclose(fp);
    fp = fopen("INPUT.txt", "r");
    if (fp == NULL) {
        printf("File cannot be opened.\n");
        return 1;
    }
    printf("Data read from file:\n");
    while (fgets(data, sizeof(data), fp) != NULL) {
        printf("%s", data);
    }
   fclose(fp);
    return 0;
}
```

SUBMODULE 7

43. Write a program in C to copy the contents of one file into another. (9mark)

```
#include <stdio.h>
int main() {
    FILE *sourceFile, *destFile;
    char ch;
    sourceFile = fopen("source.txt", "r");
    destFile = fopen("destination.txt", "w");
    if (sourceFile == NULL || destFile == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    while ((ch = fgetc(sourceFile)) != EOF) {
        fputc(ch, destFile);
    }
    printf("File copied successfully.\n");
    fclose(sourceFile);
    fclose(destFile);
    return 0;
}
```

44. Write a C program to count the number of lines in a text file. The program should: Open a file in read mode using fopen(), read characters using fgetc(), and count newline (\n) characters; Display the total number of lines and close the file using fclose().(9mark)

```
#include <stdio.h>
int main() {
    FILE *file;
    char ch;
    int lines = 0;
    file = fopen("text.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    while ((ch = fgetc(file)) != EOF) {
        if (ch == '\n') {
            lines++;
        }
    }
    printf("Total number of lines: %d\n", lines);
    fclose(file);
    return 0;
}
```

45. Write a C program to replace vowels in a text file with the character ‘x’. (9mark)

```
#include <stdio.h>
#include <ctype.h>
int isVowel(char c) {
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
int main() {
    FILE *inputFile, *outputFile;
    char ch;
    inputFile = fopen("input.txt", "r");
    outputFile = fopen("output.txt", "w");
    if (inputFile == NULL || outputFile == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    while ((ch = fgetc(inputFile)) != EOF) {
        if (isVowel(ch)) {
            fputc('x', outputFile);
        } else {
            fputc(ch, outputFile);
        }
    }
    printf("Vowels replaced with 'x' successfully.\n");
    fclose(inputFile);
    fclose(outputFile);
    return 0;
}
```

SUBMODULE 8

46. Write a C program to open a text file, move the file pointer to a specific position using fseek(), and display the content from that position. (9mark)

```
#include <stdio.h>
int main() {
    FILE *file;
    char ch;
    int position;
    file = fopen("sample.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    printf("Enter the position to start reading from: ");
    scanf("%d", &position);
    fseek(file, position, SEEK_SET);
    printf("Content from position %d:\n", position);
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);
    }
    fclose(file);
    return 0;
}
```

47. Write a program to open a file named INVENTORY and store in it the following data:

Item name	Number	Price	Quantity
AAA-1	111	17.50	115
BBB-2	125	36.00	75
C-3	247	31.75	104

Extend the program to read this data from the file INVENTORY and display the inventory table with the value of each item. (9mark)

```
#include <stdio.h>
struct Item {
    char name[10];
    int number;
    float price;
    int quantity;
};
int main() {
    FILE *file;
    struct Item items[3] = {
        {"AAA-1", 111, 17.50, 115},
        {"BBB-2", 125, 36.00, 75},
        {"C-3",   247, 31.75, 104}
    };
    file = fopen("INVENTORY.txt", "w");
    if (file == NULL) {
        printf("Error creating file.\n");
        return 1;
    }
    for (int i = 0; i < 3; i++) {
        fprintf(file, "%s %d %.2f %d\n", items[i].name, items[i].number, items[i].price, items[i].quantity);
    }
    fclose(file);
    file = fopen("INVENTORY.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    printf("Item Name\tNumber\tPrice\tQuantity\tValue\n");
    printf("----------------------------------------------------------\n");
    while (fscanf(file, "%s %d %f %d", items[0].name, &items[0].number, &items[0].price, &items[0].quantity) !=
EOF) {
        float value = items[0].price * items[0].quantity;
        printf("%s\t\t%d\t%.2f\t%d\t\t%.2f\n", items[0].name, items[0].number, items[0].price, items[0].quantity,
value);
    }
    fclose(file);
    return 0;
}
```

48. Write a C program which computes the factorial of a given number and write the result to a file named factorial. (9mark)

```
#include <stdio.h>
long long factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}
int main() {
    int num;
    FILE *file;
    long long result;
    printf("Enter a number to find its factorial: ");
    scanf("%d", &num);
    result = factorial(num);
    file = fopen("factorial.txt", "w");
    if (file == NULL) {
        printf("Error creating file.\n");
        return 1;
    }
    fprintf(file, "Factorial of %d is %lld\n", num, result);
    printf("Result written to file successfully.\n");
    fclose(file);
    return 0;
}
```