PAPER SOLUTION
WINTER 2023

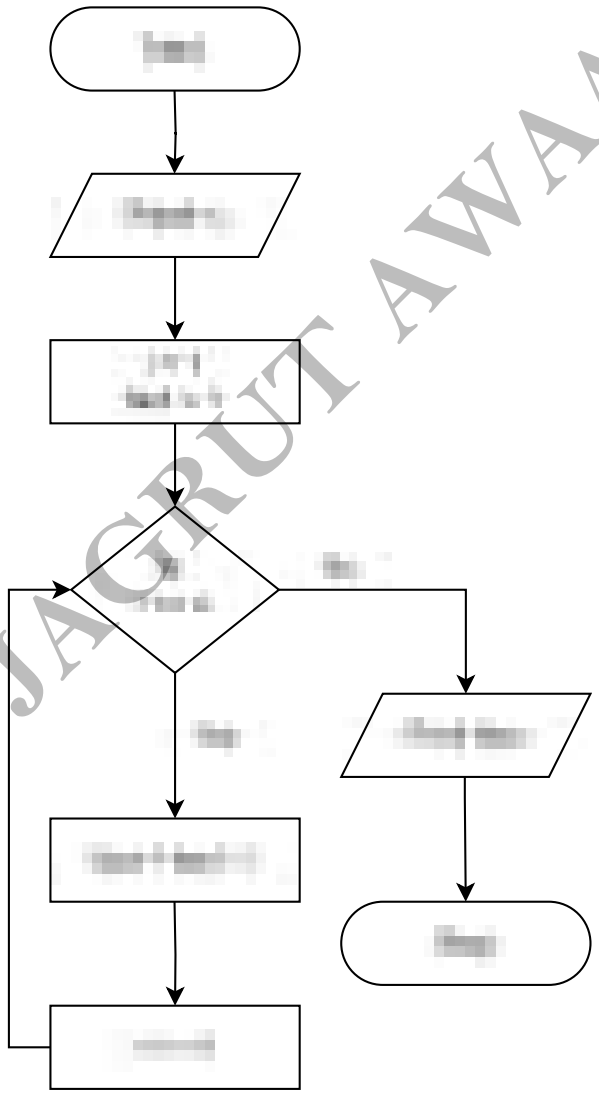
Subject code : 3110003
Subject Name : Programming for Problem Solving

Q-1 (a) Write an algorithm to check whether the entered number is , Even or odd.

Ans : Step 1 : Start
Step 2 : Read a number to N
Step 3 : Divide the number by 2 and store the remainder in R
Step 4 : if R = 0 then go to step 6
Step 5 : print "N is odd" go to step 7
Step 6 : print "N is even"
Step 7 : stop

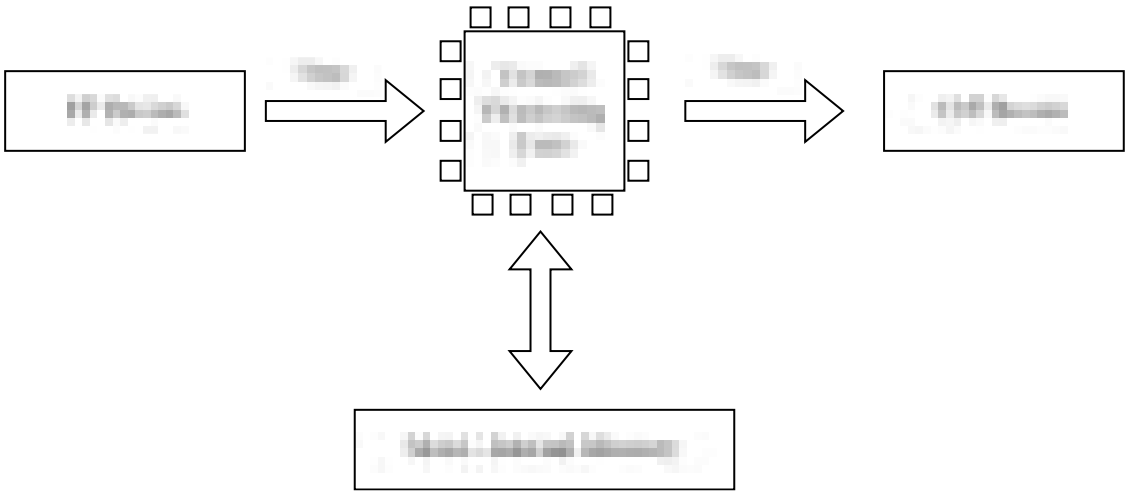
(b) Draw the flowchart to find the factorial of a number given by user .

Ans :



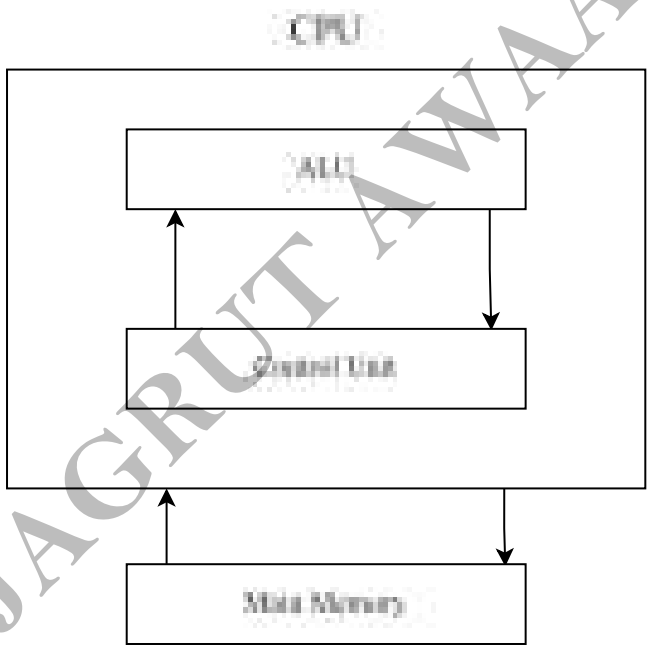
(c) Briefly explain different components of computer system.

Ans : Computer system consists of three components as shown in below image. Central processing unit , input devices and output devices. Input devices provide data input to processor which processes data and generate useful information that's displayed to the user through output devices . This is starred in computer's memory.



- Central Processing Unit:

  The Central Processing Unit (CPU) is called "the brain of computer" as it controls operation of all parts of computer. It consists of two components : Arithmetic Logic Unit (ALU) and Control unit.



- Control Unit :

  As name indicates , the part of CPU extracts instruction , perform execution , maintains and directs operations of entire system.

- Functions of control unit :

  Control unit perform following functions,
  - It controls all activities of computer
  - Supervises flow of data within CPU
  - Directs flow of data within CPU

- Memory Unit :

  This is unit in which data and instructions given to computer as well as results given by computer are stored unit of memory is "Byte"

  1 Byte = 8 Bits

Q-2 (a) Give the output of following C code.

```
int main(){
  printf("%d", 15<2);
  printf("%d", 15&&2);
  printf("%d", 7%10);
  return 0;
}
```

Ans : Output : 017

(b) Demonstrate the use of bitwise operators with an example.

Ans : The following 6 operators are bitwise operators. They are used to perform bitwise operations in C.

1. The & (bitwise AND) in C takes two numbers ae operands does and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. The 1 (Bitwise OR) in C takes two numbers as operands and does OR on every bit of two numbers. The results of OR is 1 if any of the two bits is 1
3. The (bitwise XOR) in c takes two numbers as operands And does XOR on every bit of two numbers. The results of XOR is 1 if the two bits are different.
4. The << (Left shift) takes two numbers the left shifts the bits of the first operand, and the second operand decides the number of places to shift.
5. The >> (Right shift) in C takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
6. The ~ (bitwise NOT) in C takes one number and inverts all bits of it.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

(c) Explain C token in detail.

Ans : A token in C can be defined as the smallest individual element of the C programming language that is meaningful to the complier. It is the basic components of a C program. The tokens of C language can be classified into 6 types based on the functions they are used to perform the types of C tokens are as follow .

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

1) Keywords

The keywords are pre-defined or reserved words in a programming language each keyword is meant to perform a specific function in a program C language supports 32 keywords which are given below .

Auto	double	int	struct
Break	else	long	switch
Case	enum	register	typedef

Char	extern	return	union
Const	float	short	unsigned
Continue	for	signed	void
Default	goto	sizeof	volatile
Do	if	static	while

2) Identifiers

These are used as the general terminology for the naming of variables , functions, and arrays .

- They must begin with a letter or underscore ( )
- They must consist of only letters digits or underscore no other special character is allowed
- It should not be a keyword
- It must not contain white space
- It should be up to 31 characters long as only the first 31 characters are significant

3) Constant

The constants referred to the variables with fixed values

Examples :

Const int c-var = 20;

Const int • Const ptr = & c-var;

4) Strings

Strings are nothing but an array of characters ended with a null character ('\0'). Strings are always enclosed in double quoter

Example :

Char string [20] = "geeksforgeeks";

5) Special Symbols

The following special symbols are used in C having some special meaning and thus , cannot use for any other purpose

- brackets [ ]
- Parentheses ( )
- Braces { }
- Comma (,)
- colon (:)
- Semicolon (;)
- Asterisk *
- Assignment operator (=)

6) Operators

The operators in C are the special symbols that triggers on action when applied to C variables and other objects. The data items on which operators act are called operands .

- Unary Operator
- Binary Operator
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Bitwise Operators
- Ternary Operator

(c) Briefly explain different storage classes used in C with appropriate example.

Ans : C Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility, and lifetime which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes .

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	Global multiple files	Till end of program
Static	Data segment	Zero	Within block	Till end of program
Register	CPU register	Garbage	Within block	End of block

1. Auto :
   This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared.
2. Extern :
   Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block.
3. Statics :
   This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have the property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared.
4. Register :
   This storage class declares register variables that have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.

Q-3 (a) Demonstrate the use of ternary operator with an example.

Ans : The conditional operator in C is kind of similar to the if-else statement as it follows the same algorithm as of if- Else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the ternary operator in C as it operates on three operands.

- Syntax

  The conditional operator can be in the form
  Variable = Expression1 ? Expression2 : Expression3;

It can be visualized into an if-else statement as:

```
if (Expression1)
 {
   Variable = Expression2;
}
else {
   Variable = Expression3;
}
```

- Working :
  Step 1: Expression1 is the condition to be evaluated.
  Step 2A: If the condition(Expression1) is True then Expression2 will be executed.
  Step 2B: If the condition(Expression1) is false then Expression3 will be executed.
  Step 3: Results will be returned.

(b) Give the output of following C codes.

1.
```
int main( ){
  int i=10;
 while(i<10){
  printf("%d", i);
  i++;
  }
  return 0;
  }
```

2.
```
int main( ){
 int i=10;
  do{
  printf("%d", i);
 i++;
}while(i<10);
  return 0;
}
```

Ans : First Code : No Output

Second Code : 10

(c) Write a C program to print following pattern using loop.

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```

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

Q-3 (a) Differentiate between break and continue.

Ans :

Sr. No.	Break	Continue
1.	Break is used to break loop or iteration	Continue continues the loop or iteration.
2.	Used with switch case loop	Not used with switch case.
3.	Keyword used is "break".	Keyword used is "continue".
4.	Control is transferred outside the loop.	Control remains in the same loop

(b) Demonstrate the use of forward jump and backward jump with an example.

Ans : forward jump and backward jump are concepts of offer use in loop constructs in C language these concepts are typically implemented using 'continue' and 'break' statement

Example :

```
# include<stdio.h>
    int main(){
        printf("Froward jump(using Continue):\n");
        for(int i=1 ; i<=5 ; i++){
            if (i==3){
                printf("Skipping iteration %d\n",i);
                continue;
            }
            printf("Iteration %d\n",i);
        }

        printf("\n Backward Jump(using break):\n");
        for(int j=1 ; j<=5 ; j++){
            if(j==3){
                printf("stopping loop at iteration %d\n",j);
                break;
            }
            printf("Iteration %d\n",j);
        }
    return 0;
    }
```


Output :

```
Forward Jump ( using continue) :
Iteration 1
Iteration 2
Skipping iteration 3
Iteration 4
Iteration 5

Backward Jump ( using break)
Iteration 1
Iteration 2
Stopping loop at iteration 3
```

(c) Explain else if ladder with an example.

Ans : if else if ladder in C programming is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false. If any of the conditional expressions evaluate to be true,the appropriate code block will be executed, and the entire if-else ladder will be terminated.

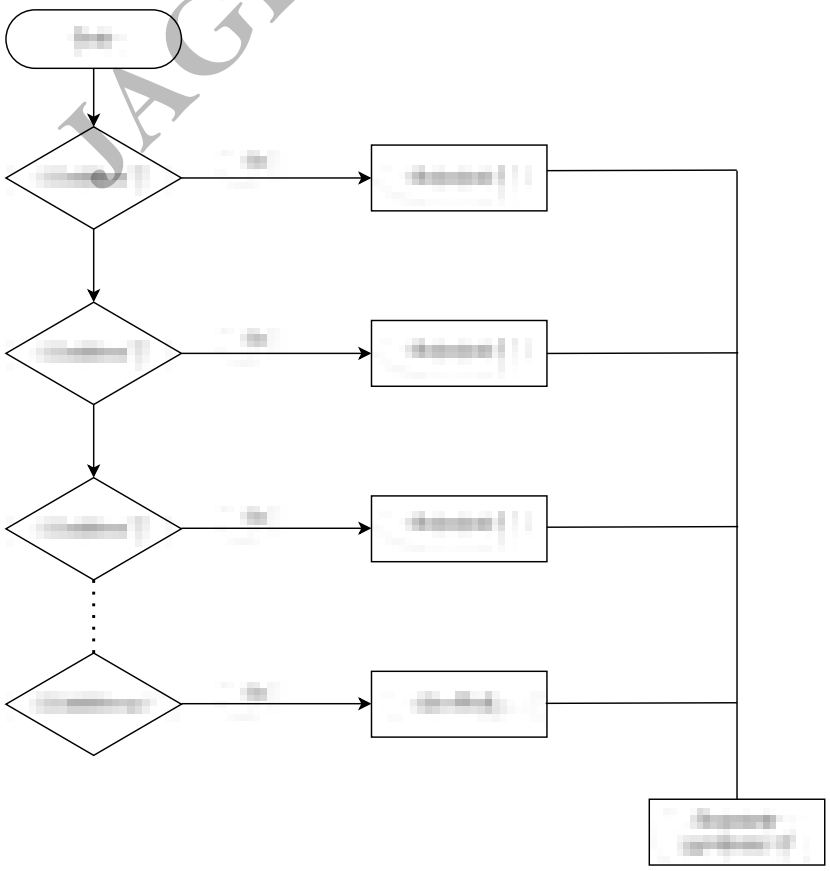
- **Syntax**:

```
// any if-else ladder starts with an if statement only
if(condition) {
}
else if(condition) {
 // this else if will be executed when condition in if is false and
 // the condition of this else if is true
}
…. // once if-else ladder can have multiple else if
else { // at the end we put else
}
```

- **Working Flow of the if-else-if ladder:**

1. The flow of the program falls into the if block.
2. The flow jumps to 1st Condition
3. 1st Condition is tested respectively:
   - If the following Condition yields true, go to Step 4.
   - If the following Condition yields false, go to Step 5.
4. The present block is executed. Goto Step 7.
5. The flow jumps to Condition 2.
   - If the following Condition yields true, go to step 4.
   - If the following Condition yields false, go to Step 6.
6. The flow jumps to Condition 3.
   - the following Condition yields true, go to step 4.
   - If the following Condition yields false, execute the else block. Goto Step 7.
7. Exits the if-else-if ladder.

Flowchart :



Example :

```
#include <stdio.h>
Int main()
{
  int marks = 91;
  if (marks <= 100 && marks >= 90)
    printf("A+ Grade");
  else if (marks < 90 && marks >= 80)
    printf("A Grade");
  else if (marks < 80 && marks >= 70)
    printf("B Grade");
  else if (marks < 70 && marks >= 60)
    printf("C Grade");
  else if (marks < 60 && marks >= 50)
    printf("D Grade");
  else
    printf("E Failed");
  return 0;
}
```

Output : A+ Grade

Q-4 (a) Give the significance of puts(), getchar(), getch().

Ans : 1. Puts() :

The 'Puts()' function is used to write a string to the standard output stream. It automatically appends a newline character ('\n') at the end of the string.

2. getchar () :

The 'getchar ()' function is used to read a single character from the standard input stream. It waits for the user to input a character and then returns the ASCII value of that character.

3. getch () :

The 'getch()' function is used to read a single character from the standard without waiting for the user to press Enter. It commonly used in scenarios where input needs to read without displaying it on the screen , Such as in password entry or Manu navigation.

(b) Differentiate between call by value and call by reference.

Ans :

Sr. No.	Call by value	Call by Reference
1.	Calling function sends copies to data.	Calling function sends address of data.
2.	The formal parameters are ordinary variables.	The formal parameters are pointer variable
3.	Atmost only one values can be sent back to the calling function.	Several results can be sent back to the calling function.
4.	Actual parameters are affected by changes made within the functions	Direct changes are made to the actual parameters

(c) Write a C program to check whether two strings are same or not.

Ans : Input :

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

int main( ) {
        char str1[100],str2[100];
        printf("Enter the first String :");
        scanf("%s",str1);

        printf("Enter the second String :");
        scanf("%s",str2);

        if(stremp(str1,str2)==0){
                printf("The two string are the same.\n");
        }
        else {
          printf("The two string are different \n");
        }
          return 0;
}
```

Q-4 (a) Give the output of following C code

```
int main( ){
int val=20, *p;
p = &val;
printf("%d %d %d", val, *p, sizeof(p));
return 0;
}
```

Ans : Output : 20 20 8

(b) Demonstrate the use of recursion with an example.

Ans : Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem .Here's a simple example demonstrating the use of recursion to calculate the factorial of a number.

```
#include<stdio.h>
int factorial(int n){
        if(n==0||n==1){
                return 1;
        }
        else {
                return n*factorial(n-1);
        }
}

int main(){
        int num;
        printf("Enter a number to calculate its factorial:");
```

```
        scanf("%d",&num);
        int result=factorial(num);
        printf("Factorial of %d is %d \n",num.result);
        return 0;
}
```

(c) Write a C program to find sum of digits for a given number using the concept Of User Defined Function (UDF).
(Hint: For number 3278, sum of digits is 3+2+7+8 = 20)
Ans :

```
#include <stdio.h>
int sumofDigits(int sum){
        int sum=0;
        while(num!=0);
        sum+=num%10;
        num/=10;
}
return sum;
int main(){
        int number , sum;
        printf("Enter a number :");
        scanf("%d",&number);
        sum= sumofDigits(number);
        print("sum of digits for number %d is : %d \n ",number,sum);
        return 0;
}
```

Q-5 (a) Differentiate between structure and union
Ans :

Sr. No.	union	Structure
1.	Union keyword is used to define the union type	Structure keyword is used to define the struct type
2.	Memory is allocated as per largest member	Memory is allocated to each member
3.	Example : union Data { int a; char b; } Data	Example : Struct Data { int a; char b; } Data

(b) Briefly explain any two file handling functions with an example.
Ans :

1. fopen ()

The 'fopen()' function is used to open a file and associate it with a stream. It takes two arguments : the name Of the file and the mode in which the file should be opened

Example :

```
# include<stdio.h>
int main(){
        File *fp;
        char filename[]="example.txt";
        char mode[]="w";
```

```
        fp = fopen (filename , mode);
        if(fp==Null){
                printf("Error opening file! \n");
                return 1;
          }
        fprint(fp,"Hello, world!\n");
        fprint(fp,"This is a file handling example \n");

        fclose(fp);
        printf("File written successfully!\n");
        return 0;
      }
```

2. fclose ( ):

The 'fclose ()' function is used to close a file stream that was previously open using 'fopen()'
It takes a single argument : The file pointer associated with the stream to be closed

Example :

```
# include<stdio.h>
int main(){
        File *fp;
        char filename[]="example.txt";
        char mode[]="r";
        char ch;
        fp = fopen (filename , mode);
        if(fp==Null){
                printf("Error opening file! \n");
                return 1;
          }
        printf("Contents of the file:\n ");
        while((ch==fget(fp))!=EOF){
                printf("%c",ch);
        }
        fclose(fp);
        return 0;
      }
```

(c) Write a C program to find maximum and minimum from an array of 10 Elements.

Ans :

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

    printf("Enter 10 elements:\n");
    for (i = 0; i < 10; i++) {
       printf("Enter element %d: ", i + 1);
       scanf("%d", &arr[i]);
    }

    // Assume first element as maximum and minimum
```

```
    max = min = arr[0];

    // Traverse array to find maximum and minimum
    for (i = 1; i < 10; i++) {
      if (arr[i] > max){
         max = arr[i];
        }
      if (arr[i] < min){
         min = arr[i];
        }
    }

    // Print maximum and minimum
    printf("Maximum element: %d\n", max);
    printf("Minimum element: %d\n", min);

    return 0;
  }
```

Q-5 (a) Give the output of following C code.

```
int main(){
 int arr[3][2]={1,2,3,4,5,6};
 printf("%d %d %d", arr[0][1], arr[1][0], arr[2][1]);
 return 0;
}
```

Ans : Output : 2 3 6

(b) Briefly explain memory management functions.

Ans : memory management function in C are used to dynamically allocate and delicate memory during program execution. These function provide flexibility in managing memory efficiently. Here are some commonly used Memory management functions.

1. malloc ()
   Allocate memory dynamically from the heap
   Syntax : void * malloc (size of (int));
   Example :
   ```
   int * ptr = (int*)malloc(10*size of (int));
   ```

2. calloc ()
   Allocate and initialize memory dynamically from the heap .
   Syntax : void * calloc (size _t num , size_t size)
   Example :
   ```
   int * ptr = (int*)calloc(10,size of (int));
   ```

3. relloc ()
   Rellocate memory dynamically from the heap
   Syntax : void * relloc (void * ptr , size_t size);
   Example :
   ```
   ptr = (int*) realloc (ptr , 10 * size of (int));
   ```

(c) Write a C program to copy one file to other.

Ans :

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   FILE *sourceFile, *destinationFile;
   char sourceFileName[100], destinationFileName[100];
   char ch;

   // Input source file name
   printf("Enter the name of the source file: ");
   scanf("%s", sourceFileName);

   // Input destination file name
   printf("Enter the name of the destination file: ");
   scanf("%s", destinationFileName);

   // Open source file in read mode
   sourceFile = fopen(sourceFileName, "r");
   if (sourceFile == NULL) {
      printf("Unable to open source file.\n");
      return 1;
   }

   // Open destination file in write mode
   destinationFile = fopen(destinationFileName, "w");
   if (destinationFile == NULL) {
      printf("Unable to create destination file.\n");
      fclose(sourceFile);
      return 1;
   }

   // Copy content from source file to destination file
   while ((ch = fgetc(sourceFile)) != EOF) {
      fputc(ch, destinationFile);
   }

   // Close files
   fclose(sourceFile);
   fclose(destinationFile);

   printf("File copied successfully.\n");

   return 0;
}
```


*****