

Name of the student	<u>Abhishekraddi G Muganur</u>
Roll No	<u>137</u>
USN	<u>01FE22BEI043</u>
Div	<u>G</u>

Part- A

Variables and Data Types:
1. What is the difference between a variable and a data type in C programming? Provide examples to illustrate. A. data type defines the kind of data a variable can hold. It acts like a blueprint, specifying the range of values, operations allowed, and the amount of memory allocated for the variable. Common data types in C include (integers), float (floating-point numbers), char (characters), and double (double-precision floating-point numbers). Ex.int age; Float pi;
2. Explain the concept of data types in C programming. Discuss the different types of data types available in C. A. Data types in C act like blueprints, defining the kind of information variables can store. They specify the value range, allowed operations, and memory allocation for each variable. Common types include int (integers), float (decimals), char (characters), and double (high-precision decimals). Choosing the right data type ensures efficient memory usage and prevents errors in calculations.
3. How are variables declared and initialized in C programming? Provide examples of variable declarations with different data types A. In C, you declare variables by specifying the data type and giving them a name. Here are some examples Int age; (stores integers), float pi = 3.14; (stores decimals with initial value),char initial = 'A'; (stores a single character),double distance; (stores high-precision decimals)
4. Discuss the scope and lifetime of variables in C programming. What are global and local variables? A. Scope and lifetime define where and how long variables exist in C. Scope: Refers to the program area where a variable is accessible. It's like a zone where the variable's name is recognized. Lifetime: Determines how long the memory allocated for the variable remains valid during program execution. There are two main types: Global: Declared outside functions, accessible throughout the program (broader scope, program lifetime).

Local: Declared within functions, accessible only inside that function (limited scope, function execution lifetime).

5. Explain the concept of type casting in C programming. When is type casting necessary, and how is it performed?

A. Type casting in C lets you explicitly convert a variable from one data type to another. Imagine it as forcing a value into a different sized box.

It's useful in a few cases:

Assigning values between incompatible types: Going from int (whole numbers) to float (decimals) for calculations.

Function compatibility: Using a variable in a function expecting a different data type.

Casting is done using the cast operator (data_type) before the variable you want to convert. For example, (float)age converts age (int) to a float. Be cautious though, as casting can lead to data loss if the target type can't hold the original value's range.

Operators:

1. Describe the purpose and usage of the ternary conditional operator in C programming. Provide an example demonstrating its usage.

A. The ternary conditional operator, also called the conditional operator, offers a concise way to make decisions in C code. It acts like a shorthand if-else statement within a single expression.

condition: The expression to be evaluated.

Expression if true: The value returned if the condition is true.

Expression if false: The value returned if the condition is false.

Example:

2. Discuss the bitwise operators available in C programming. Explain their usage with suitable examples.

A. C programming offers bitwise operators that perform operations on individual bits of data. These operators work on integer data types.

Bitwise AND (&): Sets a bit to 1 only if the corresponding bits in both operands are 1. x & y: 1010 & 0110 = 0010

Bitwise OR (|): Sets a bit to 1 if at least one corresponding bit in either operand is 1. $x | y$:
 $1010 | 0110 = 1110$

Bitwise XOR (^): Sets a bit to 1 if the corresponding bits in the operands are different. $x \wedge y$:
 $1010 \wedge 0110 = 1100$

Left shift (<<): Shifts bits of the first operand to the left by the number of bits specified by the second operand. $x \ll 2$: $1010 \ll 2 = 101000$

Right shift (>>): Shifts bits of the first operand to the right by the number of bits specified by the second operand. $x \gg 1$: $1010 \gg 1 = 0101$

3. Explain the difference between the postfix and prefix increment operators (++) in C programming. Provide examples to illustrate.

A. In C, the prefix (++) and postfix (++) increment operators both increase a variable's value by 1, but the key difference lies in when the increment happens:

- **Prefix (++):** Increments the variable first, then uses the new value.
 - Example: `int x = 5; ++x; // x becomes 6 (increment happens before assignment)`
- **Postfix (++):** Uses the variable's current value first, then increments it.
 - Example: `int x = 5; int y = x++; // y becomes 5 (original value), then x becomes 6 (assignment uses original value, then increment happens)`

4. What is the significance of the logical AND (&&) and logical OR (||) operators in C programming? How are they used in conditional expressions?

A. Logical AND (&&) and logical OR (||) operators are essential for combining conditions in C's conditional expressions. They determine the overall truth value based on the operands (conditions) being evaluated.

- **Logical AND (&&):** Represents "both conditions must be true". The expression returns true only if all operands evaluate to true. If the first operand is false, the second operand isn't even evaluated (short-circuit evaluation).
- **Logical OR (||):** Represents "at least one condition must be true". The expression returns true if any operand evaluates to true. Evaluation stops as soon as a true operand is encountered (short-circuit evaluation).

5. Discuss the concept of operator precedence and associativity in C programming. Provide examples to demonstrate how they affect expression evaluation.

A. Operator precedence and associativity are crucial concepts in C programming that dictate how expressions are evaluated.

- **Precedence:** Defines the order in which different operators are evaluated. Operators with higher precedence are evaluated first. Imagine it as a hierarchy, resolving high-priority operations before moving on.

- **Associativity:** Determines the order of evaluation for operators with the same precedence. It can be left-to-right or right-to-left. Think of it as a tiebreaker rule within the same precedence level.

Control Structures:

1. Describe the purpose and usage of the switch statement in C programming. How does it differ from the if-else statement?

A. The switch statement in C provides a multi-way branching mechanism for handling different cases based on a single variable's value. It acts like a more concise alternative to long chains of if-else statements.

1. **Expression:** The switch evaluates an expression (usually an integer or character).

2. **Case Labels:** Each case label represents a possible value the expression can hold.

3. **Code Block:** The code block associated with a matching case label is executed.

4. **break (Optional):** The break statement exits the switch after a matching case is executed, preventing fall-through to subsequent cases

Differences from if-else:

Clarity: switch can be more readable for handling multiple conditions based on a single variable.

Fall-through: By default, switch cases fall through to the next case if no break is present. if-else avoids this behaviour

Data Types: switch expressions are typically limited to integers or characters, while if-else can handle more complex conditions.

2. Explain the concept of nested control structures in C programming. Provide an example demonstrating nested if-else statements.

A. Nested control structures in C allow you to create complex decision-making logic by embedding one control structure within another. It's like creating layers of conditions.

Example (Nested if-else):

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

```
int main() {  
  
    int age = 20;
```

```
char initial = 'A';

if (age >= 18) {

    printf("You are an adult.\n");

    if (initial == 'A' || initial == 'E' || initial == 'T' || initial == 'O' || initial == 'U') {

        printf("Your initial '%c' is a vowel.\n", initial);

    } else {

        printf("Your initial '%c' is a consonant.\n", initial);

    }

} else {

    printf("You are not an adult.\n");

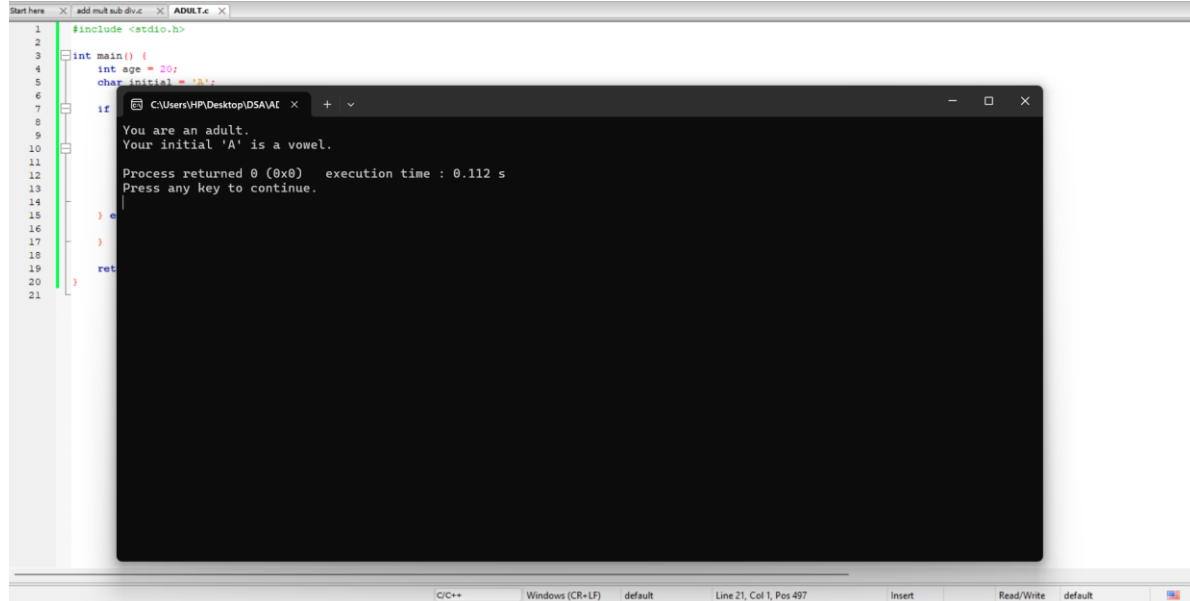
}

return 0;

}
```

Here, the outer if checks if age is 18 or more. If true, an inner if-else checks the vowel condition for the initial only if the outer condition is met. This demonstrates nesting if statements for a more intricate decision flow.

3. Discuss the role of the break and co



```
1 #include <stdio.h>
2
3 int main() {
4     int age = 20;
5     char initial = 'A';
6
7     if (age > 18) {
8         printf("You are an adult.\n");
9         if (initial == 'A' || initial == 'a') {
10             printf("Your initial 'A' is a vowel.\n");
11         }
12     }
13     printf("Process returned 0 (0x0)   execution time : 0.112 s\n");
14     printf("Press any key to continue.\n");
15     getch();
16 }
17
18
19
20
21
```

ntinue statements in loop control in C programming. Provide examples to illustrate their usage.

A. Purpose of break: Terminates the loop entirely, exiting the loop's body as soon as it's encountered.

```
}
```

Purpose of continue: Skips the current iteration of the loop and jumps to the beginning of the next iteration. The remaining code within the current iteration is not executed.

4. What are the advantages of using the for loop over the while loop in C programming? Provide examples comparing the two.

A. Advantages

Clarity and Readability:

- for loops explicitly combine initialization, condition check, and increment/decrement in a single line, making the loop's purpose clearer at a glance.

Conciseness:

- This combined structure often leads to more concise code compared to while loops, especially when all three components (initialization, condition, increment) are }

5. Explain the concept of short-circuit evaluation in C programming. How does it affect the evaluation of logical expressions in if statements?

A. Short-circuit evaluation is an optimization technique used in C programming for logical operators (&& - AND, || - OR). It ensures expressions are evaluated only as far as necessary to determine the final outcome.

Benefits:

1.Improves efficiency by avoiding unnecessary calculations, especially when dealing with functions that might have side effects.

2.Can enhance code readability by allowing you to write conditions where the outcome of the second operand depends on the first.

Functions:

1. Describe the purpose and structure of a function prototype in C programming. Why is it necessary to declare function prototypes?

A. In C programming, a function prototype acts like a blueprint or an announcement for a function. It provides essential information about the function to the compiler before the actual function definition is encountered.

Importance:

- **Type Checking:** The compiler uses the prototype to ensure the function call matches the declared parameters and return type. This helps catch errors like passing incorrect data types or missing arguments during compilation.
- **Code Organization:** Prototypes improve code readability and maintainability by documenting function details in a separate header file. This allows you to use functions across multiple source files without worrying about the definition order.

2. Explain the difference between call by value and call by reference in C programming. Provide examples to illustrate both concepts.

A. Call by Value:

- A copy of the actual argument's value is passed to the function.
- Changes made to the parameter inside the function do not affect the original variable in the calling code.
- This is the default mechanism in C for most data types (like int, float).

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

```
void swap(int *a, int *b) {
```

```
    int temp = *a;
```

```
    *a = *b;
```

```
    *b = temp;
```

```
}
```

```
int main() {
```

```
int x = 5, y = 10;

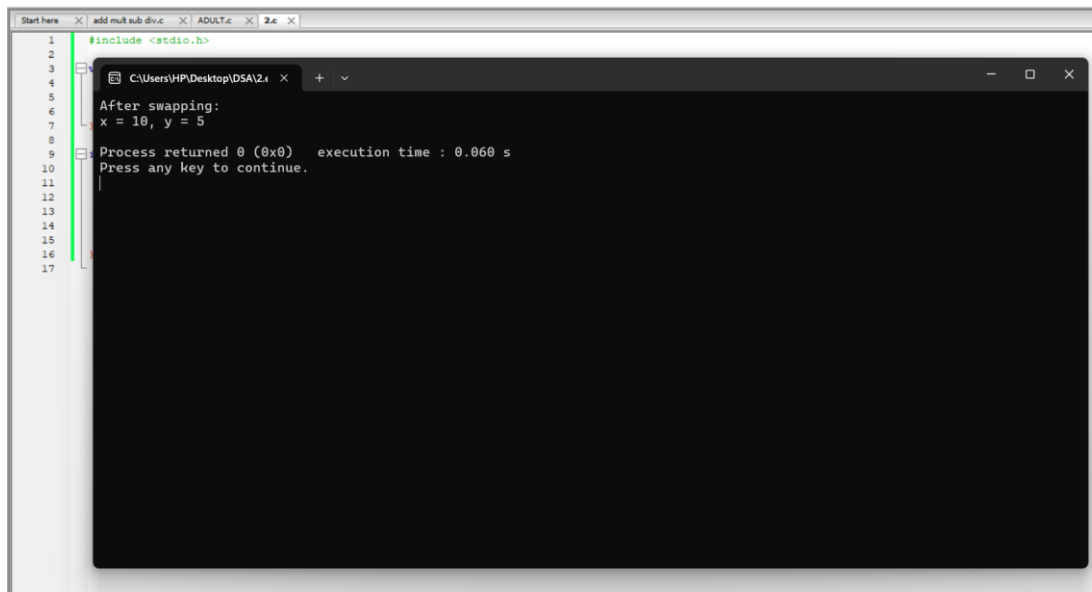
swap(&x, &y);

printf("After swapping:\n");

printf("x = %d, y = %d\n", x, y);

return 0;

}
```

The image shows a screenshot of a C program being executed. On the left, a code editor window displays the source code for a swap function. The code includes `<stdio.h>`, defines two integers `x` and `y` with values 5 and 10 respectively, calls `swap(&x, &y);`, prints "After swapping:", and then prints the values of `x` and `y` using `printf("x = %d, y = %d\n", x, y);`. The main function returns 0. On the right, a terminal window shows the output of the program. It displays "After swapping:" followed by "x = 10, y = 5" on the next line. Below this, it shows "Process returned 0 (0x0) execution time : 0.060 s" and "Press any key to continue." The terminal window has a dark background and white text.

call by Reference:

- The address (memory location) of the actual argument is passed to the function.
- Modifications made to the parameter through the address directly affect the original variable.
- In C, you need to use pointers (*) to achieve call by reference

3. Discuss the concept of recursion in C programming. Provide an example of a recursive function and explain how it works.

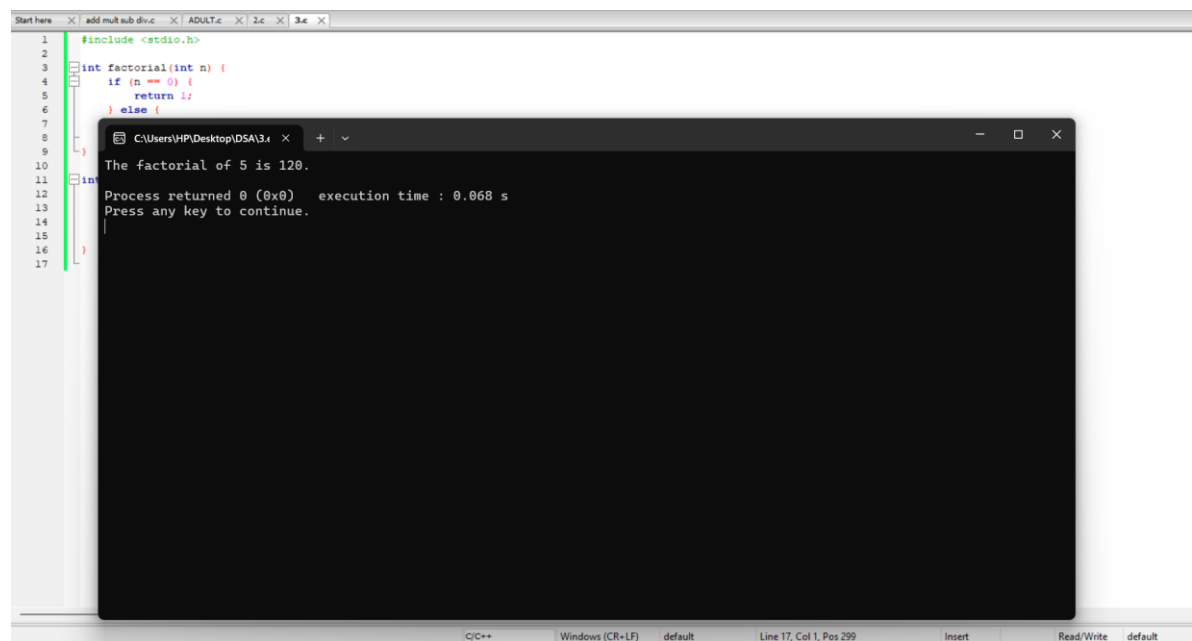
A. Recursion in C programming involves a function calling itself. It's a technique for solving problems by breaking them down into smaller, self-similar subproblems.

Example: Factorial Function

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



```
int factorial(int n) {  
  
    if (n == 0) {  
  
        return 1;  
  
    } else {  
  
        return n * factorial(n - 1);  
  
    }  
  
}  
  
int main() {  
  
    int number = 5;  
  
    int result = factorial(number);  
  
    printf("The factorial of %d is %d.\n", number, result);  
  
    return 0;  
  
}
```



The screenshot shows a C++ IDE with a file named 'mult sub div.c'. The code is as follows:

```
1 #include <stdio.h>  
2  
3 int factorial(int n) {  
4     if (n == 0) {  
5         return 1;  
6     } else {  
7  
8     }  
9 }  
10  
11 int main() {  
12  
13     int number = 5;  
14  
15     int result = factorial(number);  
16  
17     printf("The factorial of %d is %d.\n", number, result);  
18  
19     return 0;  
20 }
```

The output window shows the following text:

```
The factorial of 5 is 120.  
Process returned 0 (0x0)   execution time : 0.068 s  
Press any key to continue.
```

4. What is the significance of the return statement in C programming? How are values returned from functions?

A. The return statement in C programming plays a crucial role in function calls. It serves two key purposes:

1.Terminating Function Execution: It marks the end of the function's execution and instructs the program to return control back to the point where the function was called.

2.Returning Values (Optional): The return statement can optionally be used to send a value back from the function to the calling code. This value becomes the result of the function call.

Returning values

1.The return_type specifies the data type of the value being returned.

2.The value_to_return is the actual data you want to send back from the function.

5.Describe the role of function parameters and arguments in C programming. How are function arguments passed to parameters

A. In C programming, functions operate on data but often need input from the calling code. This is where function parameters and arguments come into play:

1.Parameters: These are variables declared within the function definition. They act like placeholders that receive the values passed during the function call. You specify their data types to indicate the kind of data the function expects.

Arguments: These are the actual values passed to the function when it's called. They are listed within parentheses after the function name in the call statement.

Passing Arguments:

1.During a function call, arguments are matched with parameters based on their position in the call statement (not by name).

2.The value of the argument is copied (call by value) to the parameter's memory location within the function.

Arrays:

1. Explain the concept of arrays in C programming. How are arrays declared and initialized?

A. Arrays in C act like collections of elements of the same data type. Imagine them as a fixed-size box where each slot stores a value. You declare them by specifying the data type, array name, and size enclosed in square brackets []. Initialization (assigning values) can be done during declaration or later.

2. Discuss the difference between a one-dimensional array and a multidimensional array in C programming. Provide examples of both

A. One-dimensional arrays hold a single list of elements. Imagine a row of boxes. You declare them with `data_type array_name[size]`. Example: `int numbers[5] = {1, 2, 3, 4, 5};`

Multi-dimensional arrays (like 2D arrays) represent a grid-like structure with rows and columns. Think of a table. Declaration: `data_type array_name[rows][columns]`. Example: `int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};`

3. Describe the process of accessing array elements in C programming. How are array indices used to access elements?

A. To access elements in a C array, you use the array name followed by the element's index enclosed in square brackets `[]`. The index starts from 0, so the first element has index 0, the second has index 1, and so on.

4. What is the significance of the null character (`'\0'`) in C strings? How is it used to determine the end of a string?

A. In C strings, the null character (`'\0'`) acts as a special sentinel marking the string's end. It's a non-printable character with an ASCII value of 0. String functions in C rely on this null terminator to identify the string's boundaries, allowing them to process characters until they encounter the `'\0'`. This is essential because C strings are essentially character arrays without a built-in mechanism to store their length.

5. Explain the concept of dynamic memory allocation for arrays in C programming. How are dynamic arrays allocated and deallocated?

A. In C, arrays are typically fixed-size during declaration. Dynamic memory allocation offers flexibility to create arrays with sizes determined at runtime. Here's a basic overview

1. Functions like `malloc` (allocate memory) and `calloc` (allocate and initialize) are used to request memory blocks of a desired size (in bytes) based on the number of elements needed.

2. A pointer variable is used to store the memory address returned by `malloc` or `calloc`. This pointer acts like a reference to the dynamically allocated array.

3. After use, the allocated memory must be explicitly released using `free` to prevent memory leaks. This frees the memory block back to the system.

Pointers:

1. Describe the purpose and usage of pointers in C programming. How are pointers declared and initialized

A. In C programming, pointers are variables that store memory addresses. They act like signposts pointing to locations in memory where other data resides. Here's a breakdown (within 4 lines):

Purpose: Pointers offer several functionalities:

Dynamic memory allocation (discussed earlier).

Passing arguments to functions by reference (modifying the original variable).

Working with complex data structures like linked lists and trees.

Declaration and Initialization: Pointers are declared by specifying an asterisk (*) before the data type they point to, followed by the pointer variable name. Initialization can involve assigning the address of a variable (using the & operator) or NULL (indicating no memory allocated yet).

2. Explain the concept of pointer arithmetic in C programming. Provide examples to illustrate addition and subtraction operations on pointers.

A. Pointer arithmetic in C lets you perform calculations on memory addresses stored in pointers. It considers the data type the pointer points to when performing these operations.

Addition

```
int arr[5] = {1, 2, 3, 4, 5};  
int *ptr = arr; // ptr points to the first element (arr[0])  
  
ptr++; // Moves ptr one element forward (points to arr[1])  
  
printf("Value at ptr: %d\n", *ptr); // Output: 2 (value at arr[1])
```

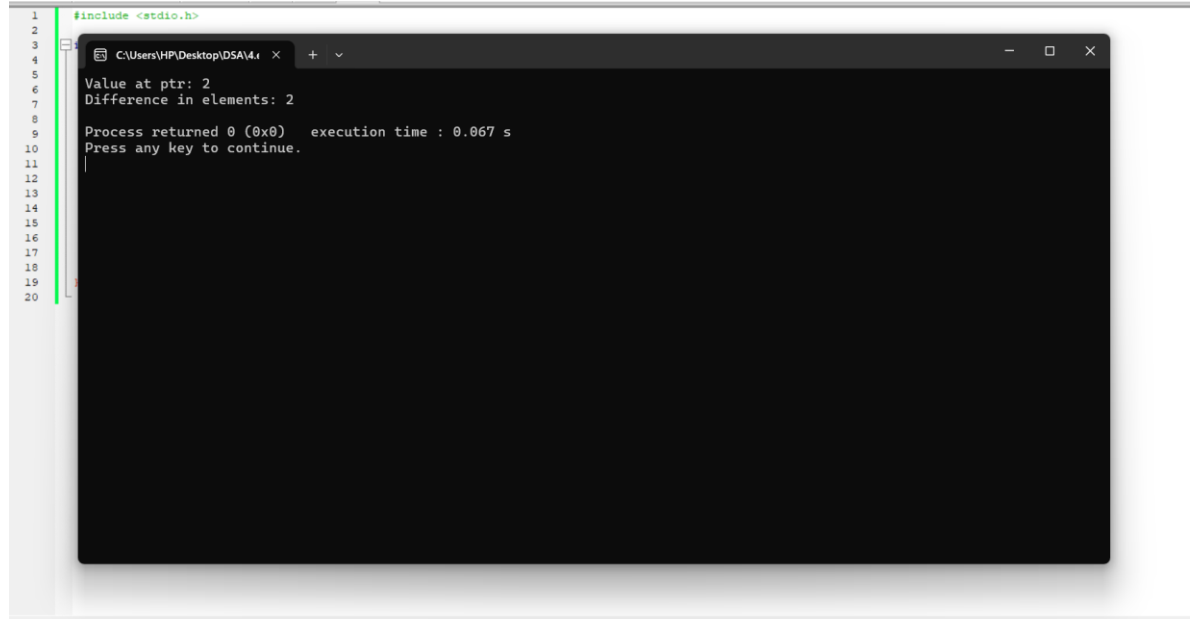
Here, ptr++ increments the pointer by the size of an integer (usually 4 bytes), effectively moving it to the next element in the array.

Subtraction

```
int numbers[4] = {10, 20, 30, 40};  
int *p1 = &numbers[2]; // p1 points to numbers[2]  
int *p2 = numbers; // p2 points to numbers[0]  
  
int difference = p1 - p2; // Calculates the number of elements between p1 and p2  
  
printf("Difference in elements: %d\n", difference); // Output: 2 (p1 points to the 3rd  
element, 2 elements ahead of p2)
```

Subtracting two pointers pointing to the same array results in the number of elements between them. This is because the difference considers the size of the data type they point

to.



The screenshot shows a C program being executed in a terminal window. The program includes `<stdio.h>` and prints the following output:

```
Value at ptr: 2
Difference in elements: 2
Process returned 0 (0x0)   execution time : 0.067 s
Press any key to continue.
```

3. Discuss the difference between pass by value and pass by reference in function arguments using pointers in C programming. Provide examples to illustrate both approaches

A. C primarily uses pass by value for function arguments. This means a copy of the argument's value is passed to the function. Changes made inside the function only affect the copy, not the original variable. For example, a function trying to increment an integer passed by value would only modify the copy. Pass by reference, achieved using pointers, allows functions to modify the original variable. In C, we pass the memory address of a variable (using the `&` operator) to the function. The function then works with the actual memory location of the variable. For instance, a function swapping two integers using pointers can directly change their values in memory.

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

```
void incrementByValue(int *x) {

    (*x)++;

    printf("Inside function: %d\n", *x);

}

int main() {

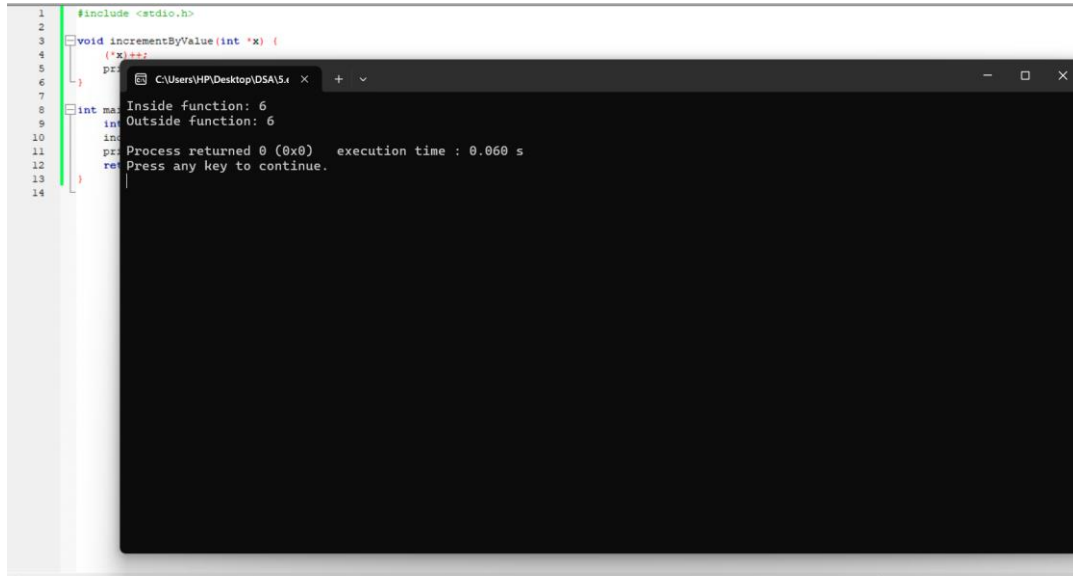
    int num = 5;

    incrementByValue(&num);

    printf("Outside function: %d\n", num);
```

```
return 0;
```

```
}
```



The screenshot shows a C program being executed. The code on the left includes `<stdio.h>`, defines a function `incrementByValue` that increments a pointer, and a `main` function that calls it. The output window on the right shows the execution results: 'Inside function: 6', 'Outside function: 6', 'Process returned 0 (0x0) execution time : 0.060 s', and 'Press any key to continue.'

4. Describe the concept of NULL pointers in C programming. How are NULL pointers used and checked for in programs?

A. A null pointer in C is a special pointer variable that doesn't hold a valid memory address. It essentially points to "nothing." We use the NULL macro to assign a null pointer. Null pointers are used to indicate uninitialized pointers or the end of a linked list. We check for null pointers using the `==` operator before dereferencing (accessing the data they point to) to avoid program crashes.

5. Explain the role of pointers in dynamic memory allocation in C programming. How are pointers used to allocate and deallocate memory dynamically?

A. In dynamic memory allocation, pointers act as handles. Functions like `malloc` and `calloc` return pointers to the allocated memory block. These pointers allow us to access and manipulate the dynamically allocated data. We use `free` with the pointer to release the memory when it's no longer needed, preventing memory leaks.

STRINGS:

1. Discuss the concept of strings in C programming. How are strings represented and manipulated in C?

A. In C, strings are simply character arrays terminated by a null character (`\0`). This null terminator acts as a flag to mark the string's end. String manipulation functions like `strcpy`, `strcat`, and `strlen` work with pointers to manage these character arrays.

2. Explain the difference between character arrays and string literals in C programming. Provide examples to illustrate both concept

A. In C, character arrays and string literals are both used for storing text, but with key differences. A character array is a variable that holds multiple characters. You can declare its size and even modify the characters within it. For instance, `char name[20] = "Alice";` creates a character array `name` that can hold up to 19 characters (excluding the null terminator). String literals, written in double quotes like `"Hello"`, are fixed text defined

directly in your code. They are stored in read-only memory and their content cannot be changed.

3. Describe common string manipulation functions available in the C standard library.

Provide examples of functions like `strlen`, `strcpy`, `strcat`, and `strcmp`

A. C provides a set of powerful tools in the `<string.h>` header for manipulating strings.

These functions make working with text data more efficient. The `strlen` function calculates the length of a string, excluding the null terminator that marks the string's end. You can use `strcpy` to copy one string to another, but be cautious! `strcpy` doesn't check if the destination string has enough space, which can lead to program crashes. For safer copying, consider `strncpy` where you specify the maximum number of characters to copy. Similar caution applies to `strcat` which appends strings but can also cause buffer overflows. Safer alternatives include `strncat` with a limit on characters appended or pre-allocating enough space in the destination string. Finally, the `strcmp` function helps compare strings, returning 0 if they are identical.

4. Discuss the concept of string tokenization in C programming. How are strings split into tokens using delimiter characters?

A. String tokenization in C involves splitting a string into smaller substrings (tokens) based on separator characters (delimiters). The `strtok` function (from `<string.h>`) is commonly used for this purpose. It modifies the original string, keeping track of its position with subsequent calls.

5. Explain the importance of null-terminated strings in C programming. How does the null character signify the end of a string?

A. Null-terminated strings are fundamental in C for managing character arrays as strings. The null character (`\0`), with a value of zero, acts as a silent sentinel at the end of the string. Since arrays don't store their size, this null terminator allows functions to determine the string's length by iterating until they encounter the `\0`.

1. Describe the purpose and usage of structures in C programming. How are structures declared and accessed?

A. In C, structures group variables of potentially different data types under a single user-defined type. This allows us to represent real-world entities with various attributes. We declare structures using the `struct` keyword, specifying member names and data types. We access structure members using the dot (`.`) operator with the structure variable name.

2. Discuss the concept of structure members in C programming. How are individual members of a structure accessed and modified?

A. Structure members are individual variables of various data types bundled within a structure. They represent specific attributes of the entity the structure defines. We access and modify them using the dot (`.`) operator after the structure variable name followed by the member's name. For example, `struct_name.member_name`. This allows us to directly interact with specific data points within the structure.

3. Explain the difference between structures and unions in C programming. When would you choose one over the other?

A. In C programming, structures group different data types together under one name, allowing access to each member individually. Unions, on the other hand, share the same memory space for all members, allowing only one member to be active at a time. When you need to store and access multiple pieces of related data simultaneously. Choose unions when you need to conserve memory by storing different types of data in the same memory location, and only one type needs to be accessed at a given time.

4. Describe the concept of nested structures in C programming. How are structures within structures defined and accessed?

A. Nested structures allow you to create complex data hierarchies in C. One structure can contain another structure as a member, enabling you to model entities with composed attributes. You define nested structures by placing a structure declaration within another structure's member definition. To access members of a nested structure, you use the dot (.) operator chained with both the outer and inner structure member names.

5. Discuss the concept of typedef in C programming. How is typedef used to define custom data types, including structures and unions?

A. The typedef keyword in C lets you create aliases for existing data types, including structures and unions. This improves code readability by providing more meaningful names. You use typedef followed by the desired type name and the original data type. For structures and unions, you can typedef the entire structure/union declaration to create a new user-friendly type.

File handling:

1. Explain the concept of file handling in C programming. How are files opened, read from, and written to using standard file handling functions?

A. File handling in C allows programs to interact with files on the storage device. We use functions like fopen to open a file in a specific mode (read, write, append). Then, fread reads data from the file into a buffer, and fwrite writes data from a buffer to the file. Finally, fclose closes the file, releasing resources.

2. Describe the role of file pointers in C programming. How are file pointers used to navigate and manipulate files?

A. File pointers in C act as handles to open files. They store information about the file's state, including the current read/write position. We use these pointers with functions like fseek to move the pointer to specific locations within the file, enabling us to perform operations like random access reads or writes at desired positions.

3. Discuss the difference between text files and binary files in C programming. How are they opened and processed differently?

A. In C, text files store human-readable characters using ASCII encoding. Binary files store raw data (numbers, images, etc.) as sequences of bytes. (1).Text Files: Opened in text mode ("r", "w", "a") with functions like fgets (read) and fputs (write) that handle newline

characters appropriately. (2). Binary Files: Opened in binary mode ("rb", "wb", "ab") using fread and fwrite to read/write raw byte sequences without newline interpretation.

4. Explain the purpose of file modes in C programming. Provide examples of different file modes like "r", "w", "a", etc.

A. In C programming, file modes determine how files are opened and accessed. They dictate whether a file is opened for reading ("r"), writing ("w"), or appending ("a"). Additional modes include "r+" for reading and writing, "w+" for reading and writing, and "a+" for reading and appending

5. Describe error handling techniques in file operations in C programming. How are errors detected and handled when working with files?

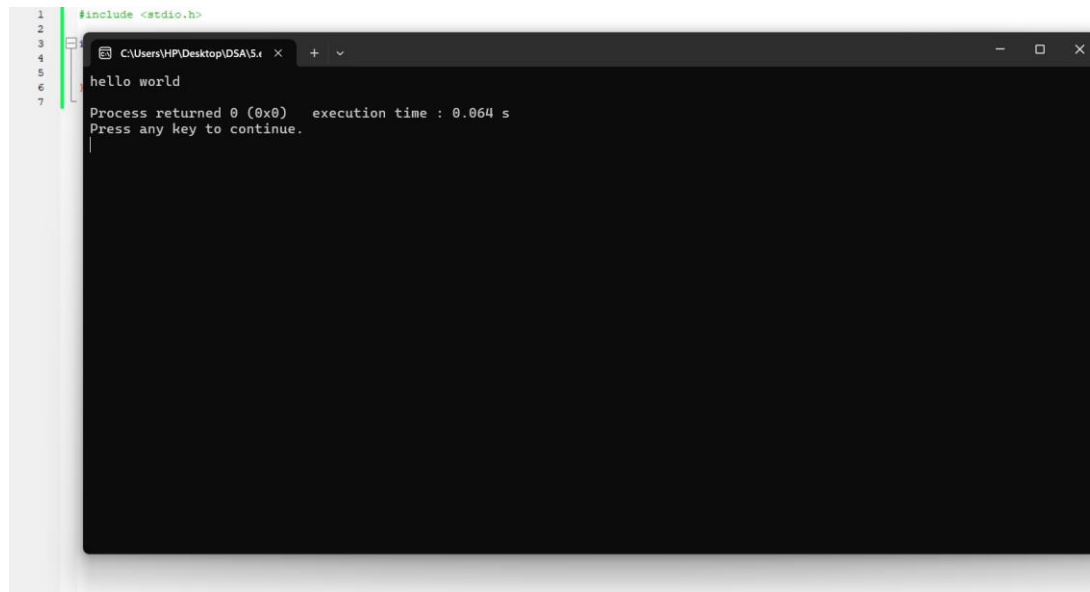
A. In C programming, errors in file operations are commonly detected through return values from functions like fopen(), fread(), fwrite(), etc. Error handling techniques include checking return codes for errors, using perror() to print error messages, and employing errno to identify specific error conditions. Additionally, file-related errors can be handled using techniques such as try-catch blocks or using macros for cleaner error handling code.

Part- B

1. Hello world

Code

```
#include<stdio.h>
main()
{
printf("hello world");
}
```



2. Factorial

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

#define MAX_NUMBER 100
#define NUM_FACTORS 5

unsigned long long calculate_factorial(int n) {
    if (n == 0) {
        return 1;
    }

    unsigned long long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

```
int main() {
    srand(time(NULL));

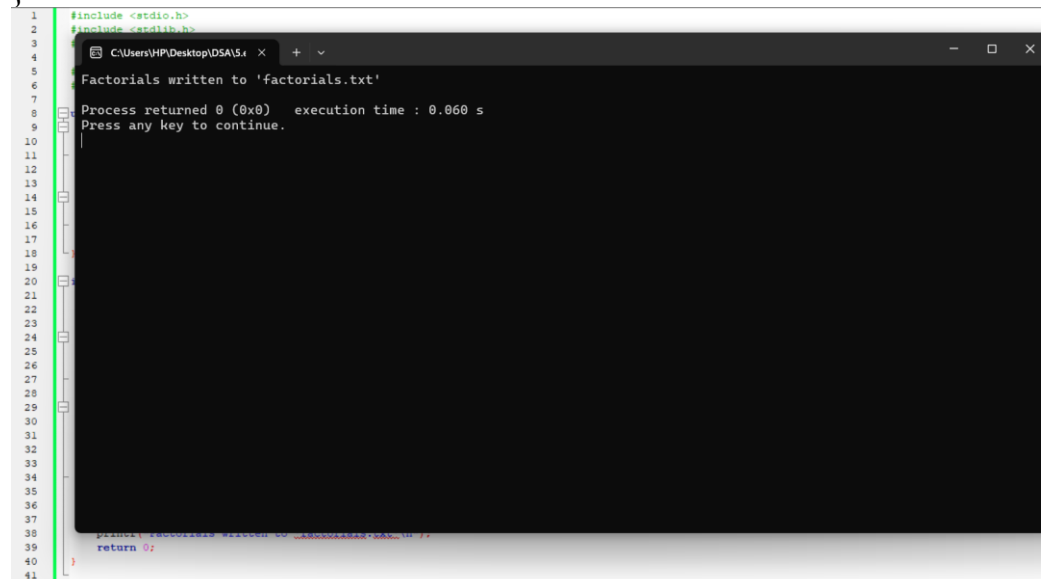
    FILE *fp = fopen("factorials.txt", "w");
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

    for (int i = 0; i < NUM_FACTORS; i++) {
        int random_number = rand() % (MAX_NUMBER + 1);
        unsigned long long factorial = calculate_factorial(random_number);

        fprintf(fp, "Factorial of %d is %llu\n", random_number, factorial);
    }

    fclose(fp);

    printf("Factorials written to 'factorials.txt'\n");
    return 0;
}
```



3.Prime number

Code

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

#define MAX_NUMBER 10000

int is_prime(int num) {
    if (num <= 1) return 0;
    if (num <= 3) return 1;
```

```
    if (num % 2 == 0 || num % 3 == 0) return 0;

    int i = 5;
    while (i * i <= num) {
        if (num % i == 0 || num % (i + 2) == 0) return 0;
        i += 6;
    }

    return 1;
}

int main() {
    srand(time(NULL));

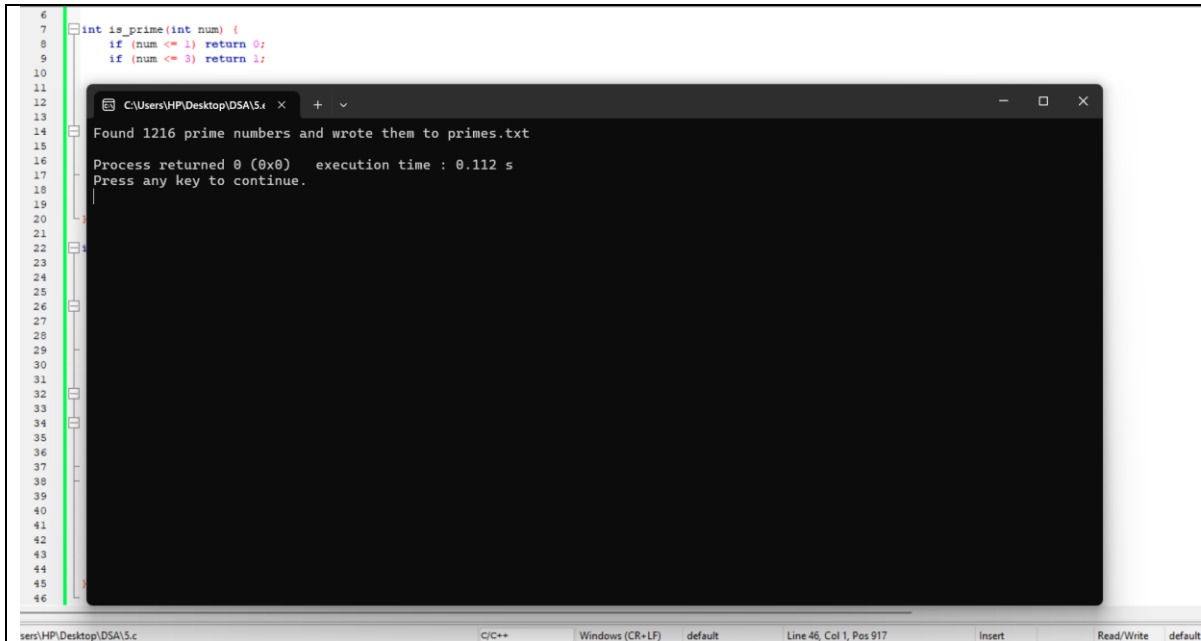
    FILE *fp = fopen("primes.txt", "w");
    if (fp == NULL) {
        perror("Error opening file");
        exit(1);
    }

    int num_primes = 0;
    for (int i = 0; i < MAX_NUMBER; i++) {
        int random_num = rand() % MAX_NUMBER + 1;
        if (is_prime(random_num)) {
            fprintf(fp, "%d\n", random_num);
            num_primes++;
        }
    }

    fclose(fp);

    printf("Found %d prime numbers and wrote them to primes.txt\n", num_primes);

    return 0;
}
```



```
6
7 int is_prime(int num) {
8     if (num <= 1) return 0;
9     if (num <= 3) return 1;
10
11
12
13
14
15 Found 1216 prime numbers and wrote them to primes.txt
16 Process returned 0 (0x0)   execution time : 0.112 s
17 Press any key to continue.
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

4.Fibonacci Series

Code

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

void generateFibonacci(int limit, const char *filename) {
    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    srand(time(NULL));
    int prev = 0, current = 1, next;
    while (current <= limit) {
        fprintf(file, "%d\n", current);
        next = prev + current;
        prev = current;
        current = next;
    }

    fclose(file);
}

int main() {
    int limit;
    const char *filename = "fibonacci.txt";

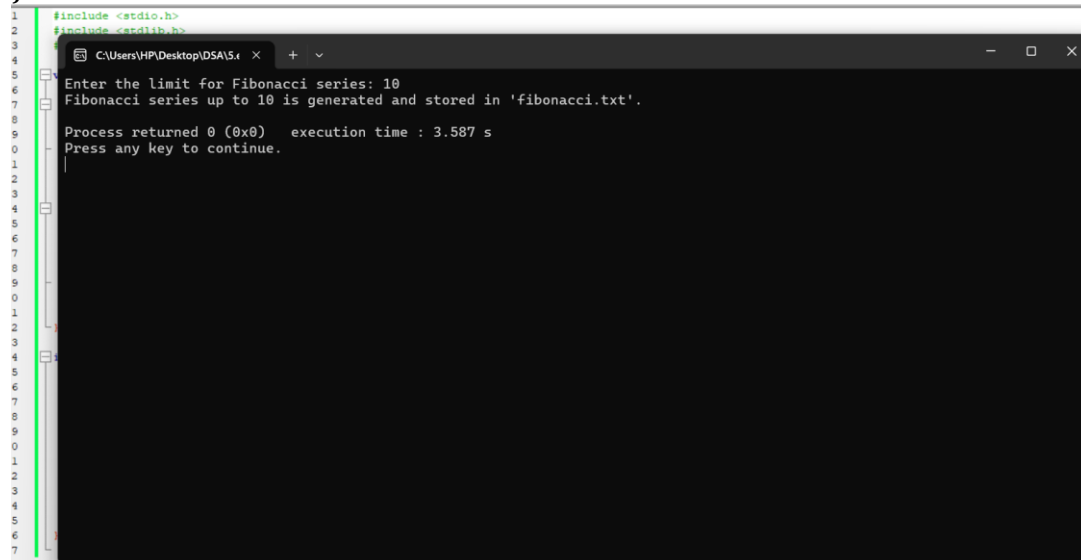
    printf("Enter the limit for Fibonacci series: ");
```

```
scanf("%d", &limit);

generateFibonacci(limit, filename);

printf("Fibonacci series up to %d is generated and stored in '%s'.\n", limit,
filename);

return 0;
}
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 Enter the limit for Fibonacci series: 10
6 Fibonacci series up to 10 is generated and stored in 'fibonacci.txt'.
7
8 Process returned 0 (0x0)   execution time : 3.587 s
9 Press any key to continue.
10
11
12
13
14
15
16
17
```

5.Sum of digits

Code

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

#define NUM_RANDOM_NUMS 10

int main() {
    srand(time(NULL));

    FILE *fp = fopen("random_numbers.bin", "wb");
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

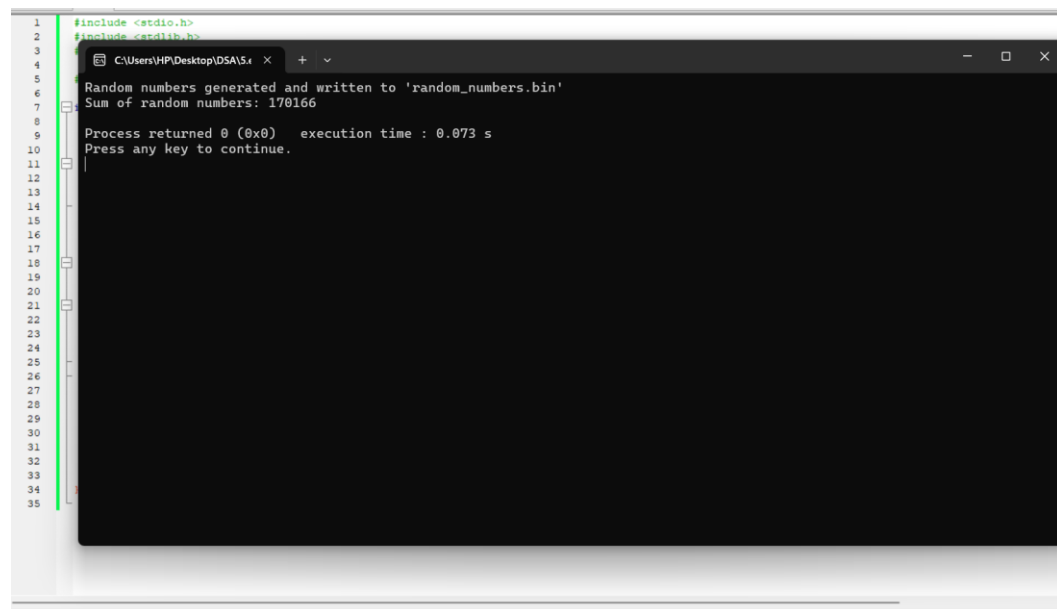
    int sum = 0;
    int num;
    for (int i = 0; i < NUM_RANDOM_NUMS; i++) {
        num = rand();
```

```
sum += num;
if (fwrite(&num, sizeof(int), 1, fp) != 1) {
    perror("Error writing to file");
    fclose(fp);
    return 1;
}
}

fclose(fp);

printf("Random numbers generated and written to 'random_numbers.bin'\n");
printf("Sum of random numbers: %d\n", sum);

return 0;
}
```

A screenshot of a C program being executed in a Windows command prompt. The program generates random numbers and writes them to a file named 'random_numbers.bin'. It then calculates the sum of these numbers and displays it. The output shows 'Random numbers generated and written to 'random_numbers.bin'', 'Sum of random numbers: 170166', and 'Process returned 0 (0x0) execution time : 0.073 s'. The prompt 'Press any key to continue.' is visible at the bottom.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5
6 Random numbers generated and written to 'random_numbers.bin'
7 Sum of random numbers: 170166
8
9 Process returned 0 (0x0) execution time : 0.073 s
10 Press any key to continue.
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

6. Reverse a Number

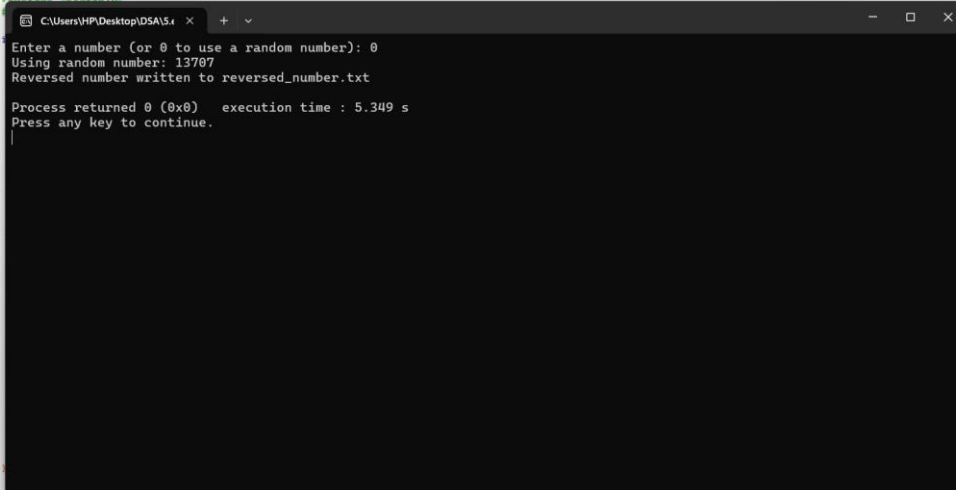
Code

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

int main() {
    int num, reversed = 0, digit;
    FILE *fp;
    srand(time(NULL));

    printf("Enter a number (or 0 to use a random number): ");
    scanf("%d", &num);
```

```
if (num == 0) {  
    num = rand() % 100000 + 1;  
    printf("Using random number: %d\n", num);  
}  
  
fp = fopen("reversed_number.txt", "w");  
if (fp == NULL) {  
    perror("Error opening file");  
    return 1;  
}  
  
while (num != 0) {  
    digit = num % 10;  
    reversed = reversed * 10 + digit;  
    num /= 10;  
}  
  
fprintf(fp, "%d\n", reversed);  
fclose(fp);  
  
printf("Reversed number written to reversed_number.txt\n");  
  
return 0;  
}
```



```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  
4  
5  Enter a number (or 0 to use a random number): 0  
6  Using random number: 13707  
7  Reversed number written to reversed_number.txt  
8  
9  
10 Process returned 0 (0x0)   execution time : 5.349 s  
11 Press any key to continue.  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37
```

7. Palindrome Check

Code

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



```
#include <string.h>

bool isPalindrome(const char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return false;
        }
    }
    return true;
}

int main() {
    // Generate a random number
    int num = rand() % 10000;

    // Convert the number to a string
    char num_str[20];
    sprintf(num_str, "%d", num);

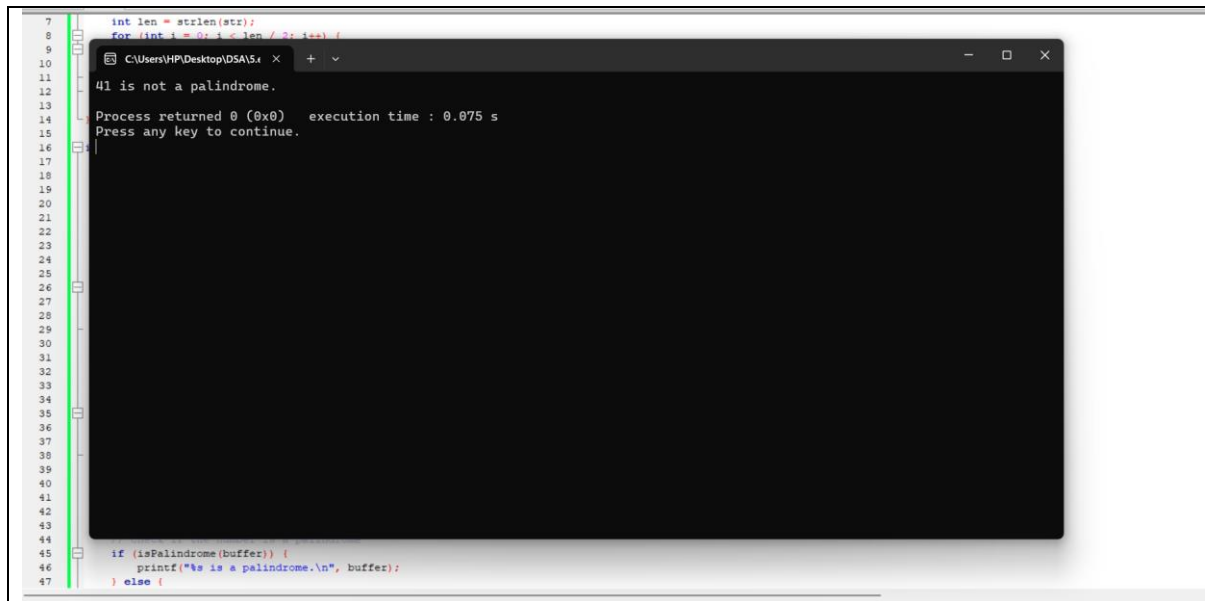
    // Write the number to a file
    FILE *file = fopen("number.txt", "w");
    if (file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }
    fprintf(file, "%s", num_str);
    fclose(file);

    // Read the number from the file
    file = fopen("number.txt", "r");
    if (file == NULL) {
        perror("Error opening file for reading");
        return 1;
    }

    char buffer[20];
    fscanf(file, "%s", buffer);
    fclose(file);

    // Check if the number is a palindrome
    if (isPalindrome(buffer)) {
        printf("%s is a palindrome.\n", buffer);
    } else {
        printf("%s is not a palindrome.\n", buffer);
    }

    return 0;
}
```



```
7 int len = strlen(str);
8 for (int i = 0; i < len / 2; i++) {
9
10
11
12 41 is not a palindrome.
13
14 Process returned 0 (0x0) execution time : 0.075 s
15 Press any key to continue.
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 if (isPalindrome(buffer)) {
46     printf("%s is a palindrome.\n", buffer);
47 } else {
```

8. Area of Shapes

Code

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

// Function to calculate the area of a triangle given base and height
float calculateArea(float base, float height) {
    return 0.5 * base * height;
}

int main() {
    FILE *outputFile;
    float base, height, area;

    // Seed the random number generator with the current time
    srand(time(NULL));

    // Open the output file for writing
    outputFile = fopen("triangle_area.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }

    // Generate 10 random triangles and calculate their areas
    for (int i = 0; i < 10; i++) {
        // Generate random values for base and height (up to 99)
        base = (float)(rand() % 100);
        height = (float)(rand() % 100);
```

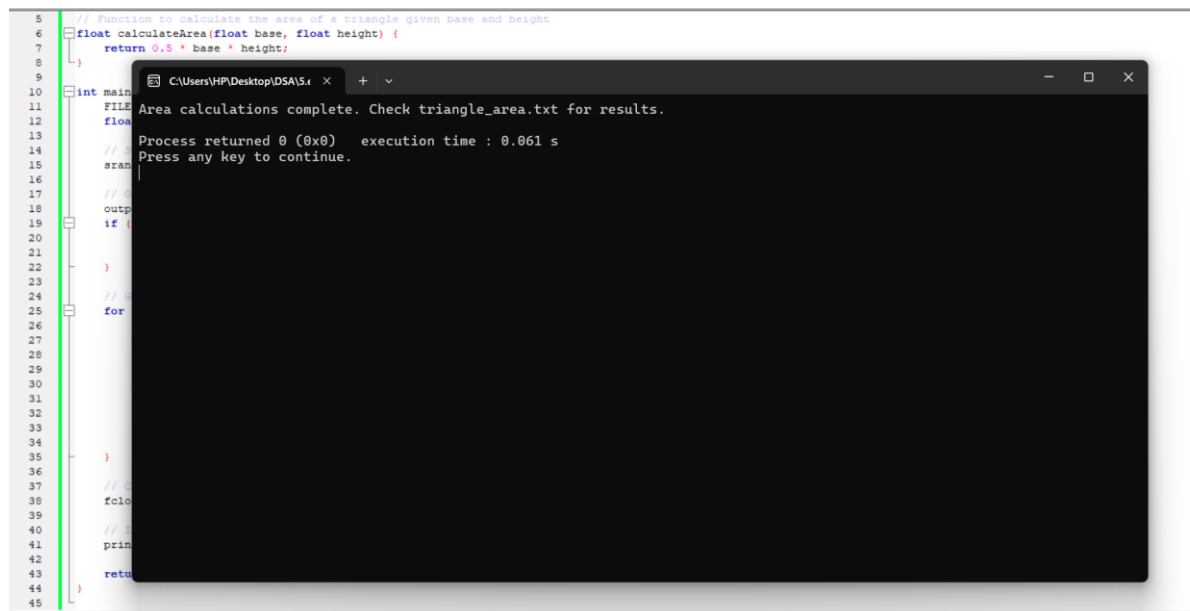
```
// Calculate the area of the triangle
area = calculateArea(base, height);

// Write the triangle's information to the output file
fprintf(outputFile, "Base: %.2f, Height: %.2f, Area: %.2f\n", base, height, area);
}

// Close the output file
fclose(outputFile);

// Inform the user that the area calculations are complete
printf("Area calculations complete. Check triangle_area.txt for results.\n");

return 0;
}
```

The image shows a screenshot of a code editor with a C++ program and its execution output. The code defines a function `calculateArea` that takes base and height as arguments and returns the area. The `main` function calls this function, prints the results to a file named `triangle_area.txt`, and then prints a message to the console. The execution output shows the message "Area calculations complete. Check triangle_area.txt for results." and "Process returned 0 (0x0) execution time : 0.061 s".

```
5 // Function to calculate the area of a triangle given base and height
6 float calculateArea(float base, float height) {
7     return 0.5 * base * height;
8 }
9
10 int main() {
11     FILE *outputFile;
12     fopen("triangle_area.txt", "w");
13     float base, height, area;
14     printf("Enter base and height: ");
15     scanf("%f %f", &base, &height);
16     area = calculateArea(base, height);
17     fprintf(outputFile, "Base: %.2f, Height: %.2f, Area: %.2f\n", base, height, area);
18     fclose(outputFile);
19     printf("Area calculations complete. Check triangle_area.txt for results.\n");
20     return 0;
21 }
```

9. . Simple Calculator:

Code

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

int main() {
    FILE *outputFile;
    char operations[] = {'+', '-', '*', '/'};
    int num1, num2, result;
    char operation;

    // Seed the random number generator with the current time
```

```
srand(time(NULL));

// Open the output file for writing
outputFile = fopen("calculator_output.txt", "w");
if (outputFile == NULL) {
    perror("Error opening output file");
    exit(EXIT_FAILURE);
}

// Generate two random numbers
num1 = rand() % 100;
num2 = rand() % 100;

// Select a random operation
operation = operations[rand() % 4];

// Perform the selected operation
switch (operation) {
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case '*':
        result = num1 * num2;
        break;
    case '/':
        if (num2 == 0) {
            fprintf(outputFile, "Error: Division by zero\n");
            fclose(outputFile);
            exit(EXIT_FAILURE);
        }
        result = num1 / num2;
        break;
}

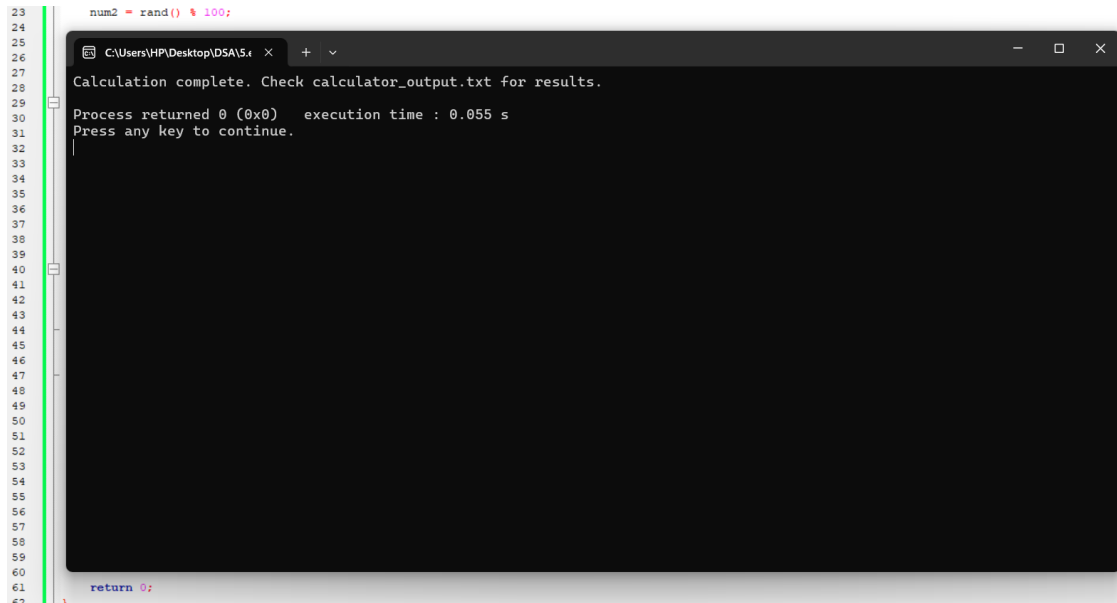
// Write the numbers, operation, and result to the output file
fprintf(outputFile, "Number 1: %d\n", num1);
fprintf(outputFile, "Number 2: %d\n", num2);
fprintf(outputFile, "Operation: %c\n", operation);
fprintf(outputFile, "Result: %d\n", result);

// Close the output file
fclose(outputFile);

// Inform the user that the calculation is complete
printf("Calculation complete. Check calculator_output.txt for results.\n");

return 0;
```

```
}
```



```
23 num2 = rand() % 100;
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61 return 0;
```

10. Array Operations

Code

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

#define ARRAY_SIZE 10

// Function to generate random numbers and fill an array
void generateRandomNumbers(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }
}

// Function to find the largest element in an array
int findLargest(int arr[], int size) {
    int largest = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > largest) {
            largest = arr[i];
        }
    }
    return largest;
}
```

```
// Function to find the smallest element in an array
int findSmallest(int arr[], int size) {
    int smallest = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < smallest) {
            smallest = arr[i];
        }
    }
    return smallest;
}

// Function to find the sum of elements in an array
int findSum(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}

// Function to calculate the average of elements in an array
double findAverage(int sum, int size) {
    return (double) sum / size;
}

int main() {
    int numbers[ARRAY_SIZE];
    int largest, smallest, sum;
    double average;
    FILE *outputFile;

    srand(time(NULL)); // Seed the random number generator

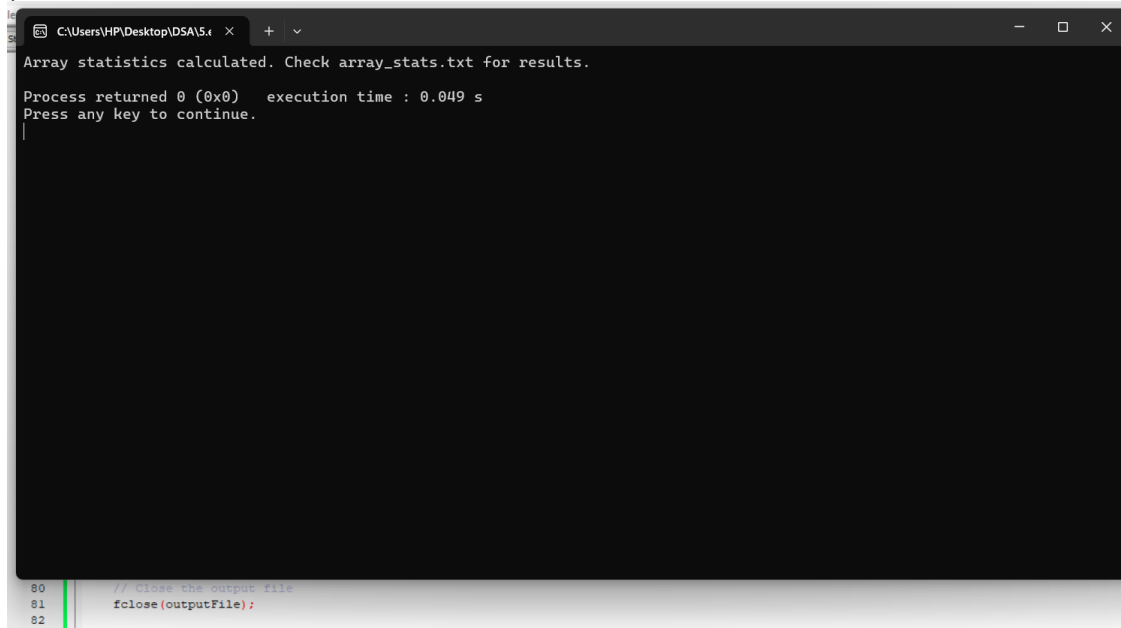
    // Generate random numbers and fill the array
    generateRandomNumbers(numbers, ARRAY_SIZE);

    // Calculate array statistics
    largest = findLargest(numbers, ARRAY_SIZE);
    smallest = findSmallest(numbers, ARRAY_SIZE);
    sum = findSum(numbers, ARRAY_SIZE);
    average = findAverage(sum, ARRAY_SIZE);

    // Open the output file for writing
    outputFile = fopen("array_stats.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }

    // Write array statistics to the output file
    fprintf(outputFile, "Largest: %d\n", largest);
    fprintf(outputFile, "Smallest: %d\n", smallest);
}
```

```
fprintf(outputFile, "Sum: %d\n", sum);  
fprintf(outputFile, "Average: %.2f\n", average);  
  
// Close the output file  
fclose(outputFile);  
  
printf("Array statistics calculated. Check array_stats.txt for results.\n");  
  
return 0;  
}
```



11. String Operations

Code

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
  
#define MAX_STRING_LENGTH 100  
  
// Function to generate a random string  
void generateRandomString(char *str, int length) {  
    const char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  
  
    for (int i = 0; i < length - 1; i++) {  
        int index = rand() % (sizeof(charset) - 1);  
        str[i] = charset[index];  
    }  
    str[length - 1] = '\0'; // Null-terminate the string
```

```
}

int main() {
    FILE *outputFile;
    char    string1[MAX_STRING_LENGTH],    string2[MAX_STRING_LENGTH],
    concatenated[MAX_STRING_LENGTH * 2];

    // Seed the random number generator
    srand(time(NULL));

    // Open the output file for writing
    outputFile = fopen("string_operations.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }

    // Generate two random strings
    generateRandomString(string1, MAX_STRING_LENGTH);
    generateRandomString(string2, MAX_STRING_LENGTH);

    // Concatenate the two strings
    strcpy(concatenated, string1);
    strcat(concatenated, string2);
    fprintf(outputFile, "Concatenated string: %s\n", concatenated);

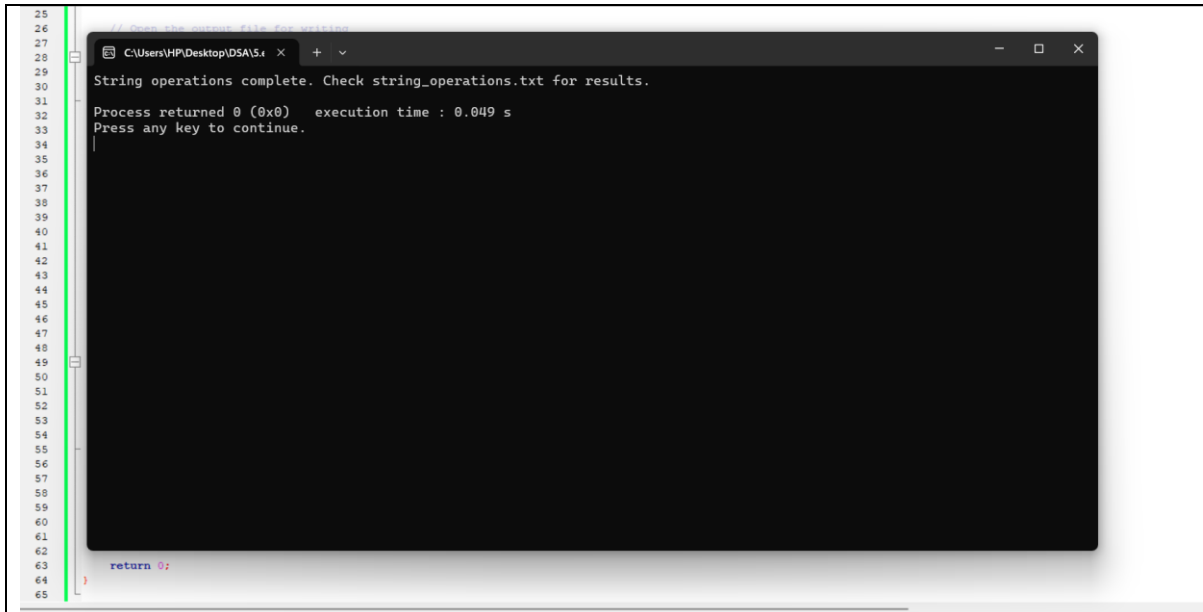
    // Copy the first string
    char copied[MAX_STRING_LENGTH];
    strcpy(copied, string1);
    fprintf(outputFile, "Copied string: %s\n", copied);

    // Compare the two strings
    int comparisonResult = strcmp(string1, string2);
    if (comparisonResult < 0) {
        fprintf(outputFile, "%s is lexicographically smaller than %s\n", string1, string2);
    } else if (comparisonResult > 0) {
        fprintf(outputFile, "%s is lexicographically greater than %s\n", string1, string2);
    } else {
        fprintf(outputFile, "%s is lexicographically equal to %s\n", string1, string2);
    }

    // Close the output file
    fclose(outputFile);

    // Inform the user that string operations are complete
    printf("String operations complete. Check string_operations.txt for results.\n");

    return 0;
}
```

12.Linear search

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

#define ARRAY_SIZE 10

// Function to perform linear search in an array
int linearSearch(int arr[], int size, int key) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            return i; // Return the index if key is found
        }
    }
    return -1; // Return -1 if key is not found
}

int main() {
    FILE *outputFile;
    int arr[ARRAY_SIZE];
    int searchKey, result;
    srand(time(NULL)); // Seed the random number generator

    // Generate random numbers and fill the array
    for (int i = 0; i < ARRAY_SIZE; i++) {
        arr[i] = rand() % 100; // Generate random numbers between 0 and 99
    }

    // Open the output file for writing
```

```
outputFile = fopen("array_elements.txt", "w");
if (outputFile == NULL) {
    perror("Error opening output file");
    exit(EXIT_FAILURE);
}

// Write the array elements to the output file
fprintf(outputFile, "Array elements:\n");
for (int i = 0; i < ARRAY_SIZE; i++) {
    fprintf(outputFile, "%d ", arr[i]);
}
fprintf(outputFile, "\n");

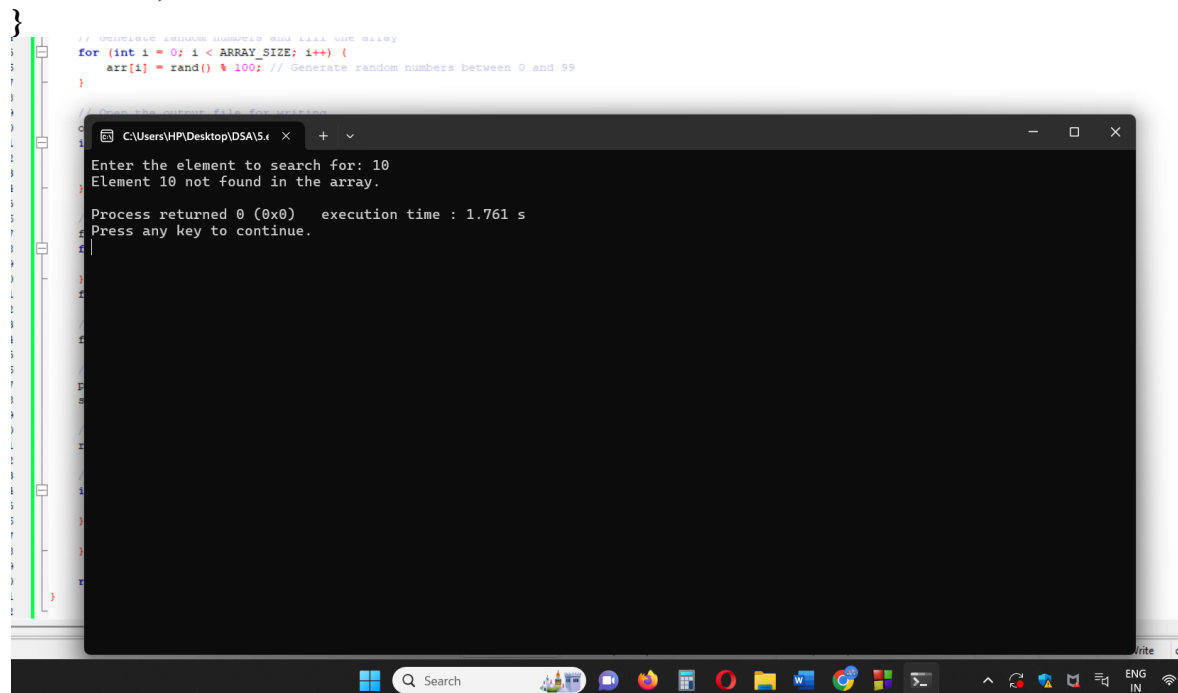
// Close the output file
fclose(outputFile);

// Prompt the user to enter the element to search for
printf("Enter the element to search for: ");
scanf("%d", &searchKey);

// Perform linear search
result = linearSearch(arr, ARRAY_SIZE, searchKey);

// Print the result
if (result != -1) {
    printf("Element %d found at index %d.\n", searchKey, result);
} else {
    printf("Element %d not found in the array.\n", searchKey);
}

return 0;
}
```



```
// UTILITY: RANDOM NUMBER: FOR THE ARRAY
for (int i = 0; i < ARRAY_SIZE; i++) {
    arr[i] = rand() % 100; // Generate random numbers between 0 and 99
}
```

C:\Users\HP\Desktop\DSA\5.4

Enter the element to search for: 10
Element 10 not found in the array.

Process returned 0 (0x0) execution time : 1.761 s
Press any key to continue.

13.Binary search

code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

#define ARRAY_SIZE 10

// Comparison function for qsort
int cmpfunc(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

// Binary search function
int binarySearch(int arr[], int size, int target) {
    int left = 0, right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid; // Return the index if target is found
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Return -1 if target is not found
}

int main() {
    int arr[ARRAY_SIZE];
    int target, index;
    FILE *outputFile;

    // Seed the random number generator
    srand(time(NULL));

    // Generate random numbers and fill the array
    for (int i = 0; i < ARRAY_SIZE; i++) {
        arr[i] = rand() % 100; // Generate random numbers between 0 and 99
    }

    // Sort the array
    qsort(arr, ARRAY_SIZE, sizeof(int), cmpfunc);
```

```
// Open the output file for writing
outputFile = fopen("binary_search_results.txt", "w");
if (outputFile == NULL) {
    perror("Error opening output file");
    exit(EXIT_FAILURE);
}

// Choose a random target to search for
target = rand() % 100; // Generate a random target between 0 and 99
printf("Searching for target: %d\n", target);

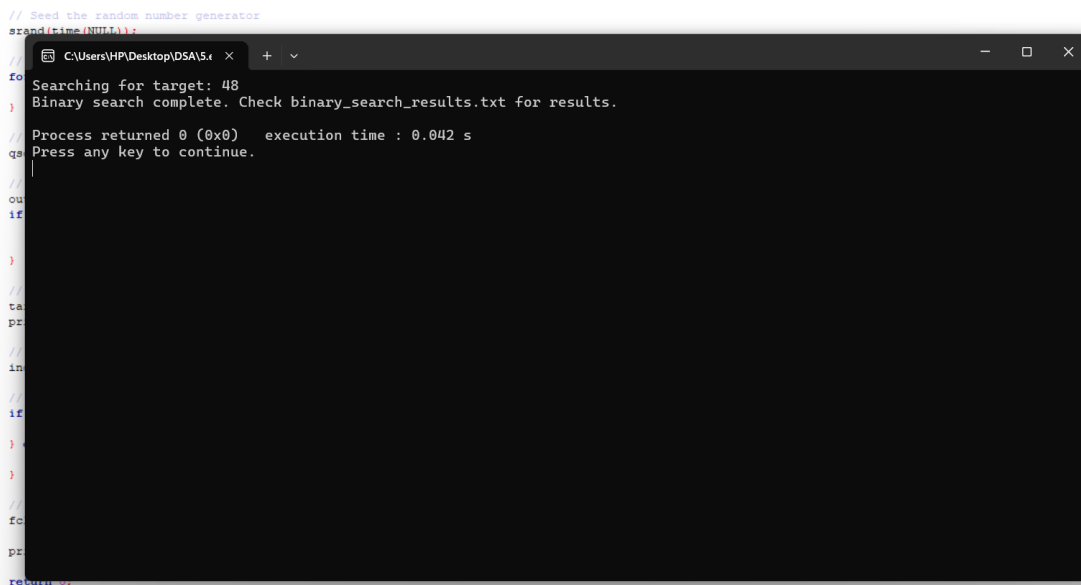
// Perform binary search
index = binarySearch(arr, ARRAY_SIZE, target);

// Write the search result to the output file
if (index != -1) {
    fprintf(outputFile, "Target %d found at index %d in the sorted array.\n", target, index);
} else {
    fprintf(outputFile, "Target %d not found in the sorted array.\n", target);
}

// Close the output file
fclose(outputFile);

printf("Binary search complete. Check binary_search_results.txt for results.\n");

return 0;
}
```



The screenshot shows a Windows command prompt window with the following output:

```
C:\Users\HP\Desktop\DSA\5.4 > .\program.exe
Searching for target: 48
Binary search complete. Check binary_search_results.txt for results.
Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

The background shows the source code of the program, which includes a seed for the random number generator and the binary search function.

14. Selection Sort

Code

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

#define ARRAY_SIZE 10

// Function to perform selection sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_index]) {
                min_index = j;
            }
        }

        // Swap arr[i] with the smallest element
        int temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
    }
}

int main() {
    FILE *outputFile;
    int arr[ARRAY_SIZE];
    srand(time(NULL));

    // Generate random numbers and fill the array
    for (int i = 0; i < ARRAY_SIZE; i++) {
        arr[i] = rand() % 100;
    }

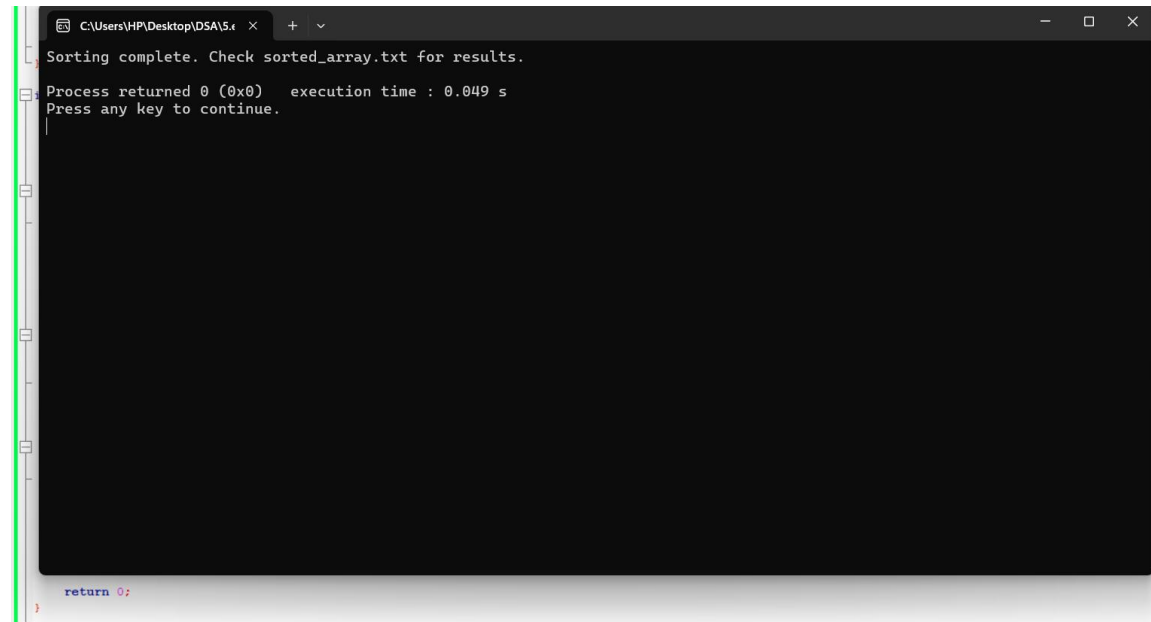
    // Perform selection sort
    selectionSort(arr, ARRAY_SIZE);

    // Open the output file for writing
    outputFile = fopen("sorted_array.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }

    // Write the sorted array to the output file
    fprintf(outputFile, "Sorted Array:\n");
    for (int i = 0; i < ARRAY_SIZE; i++) {
        fprintf(outputFile, "%d\n", arr[i]);
    }

    // Close the output file
    fclose(outputFile);
}
```

```
printf("Sorting complete. Check sorted_array.txt for results.\n");  
  
return 0;  
  
}
```



```
return 0;  
}
```

15.Bubble sort

Code

```
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
#define ARRAY_SIZE 10  
  
// Function to perform bubble sort  
void bubbleSort(int arr[], int n) {  
    for (int i = 0; i < n - 1; i++) {  
        for (int j = 0; j < n - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                // Swap arr[j] with arr[j + 1]  
                int temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
            }  
        }  
    }  
}
```

```
// Function to print an array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    FILE *outputFile;
    int n = ARRAY_SIZE; // Number of elements in the array
    int arr[n];

    srand(time(NULL)); // Seed the random number generator
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100; // Generate random numbers between 0 and 99
    }

    printf("Unsorted array: ");
    printArray(arr, n);

    // Perform bubble sort
    bubbleSort(arr, n);

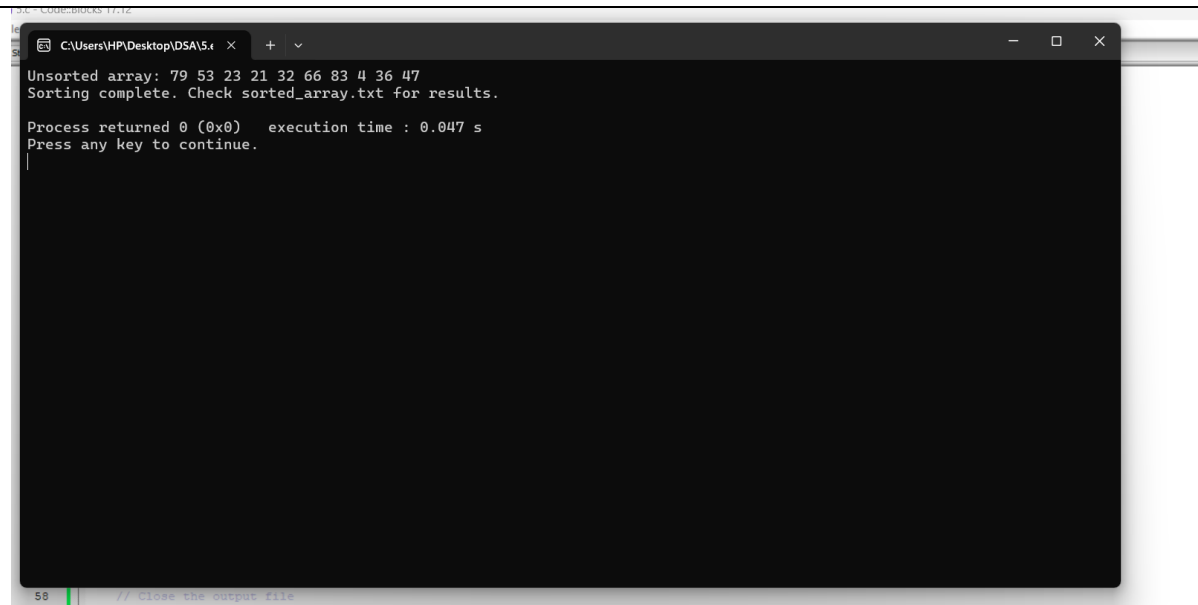
    // Open the output file for writing
    outputFile = fopen("sorted_array.txt", "w");
    if (outputFile == NULL) {
        perror("Error opening output file");
        exit(EXIT_FAILURE);
    }

    // Write the sorted array to the output file
    fprintf(outputFile, "Sorted array: ");
    for (int i = 0; i < n; i++) {
        fprintf(outputFile, "%d ", arr[i]);
    }

    // Close the output file
    fclose(outputFile);

    printf("Sorting complete. Check sorted_array.txt for results.\n");

    return 0;
}
```



```
C:\Users\HP\Desktop\DSA\5.r x + v
Unsorted array: 79 53 23 21 32 66 83 4 36 47
Sorting complete. Check sorted_array.txt for results.

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
|
```

16.Insertion sort

Code

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

#define ARRAY_SIZE 10

// Function to perform insertion sort
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    FILE *outputFile;
    int arr[ARRAY_SIZE]; // Array to store random numbers
    srand(time(NULL)); // Seed the random number generator

    // Generate random numbers and fill the array
    for (int i = 0; i < ARRAY_SIZE; i++) {
        arr[i] = rand() % 100; // Generate random numbers between 0 and 99
    }
}
```



```
}

// Perform insertion sort
insertionSort(arr, ARRAY_SIZE);

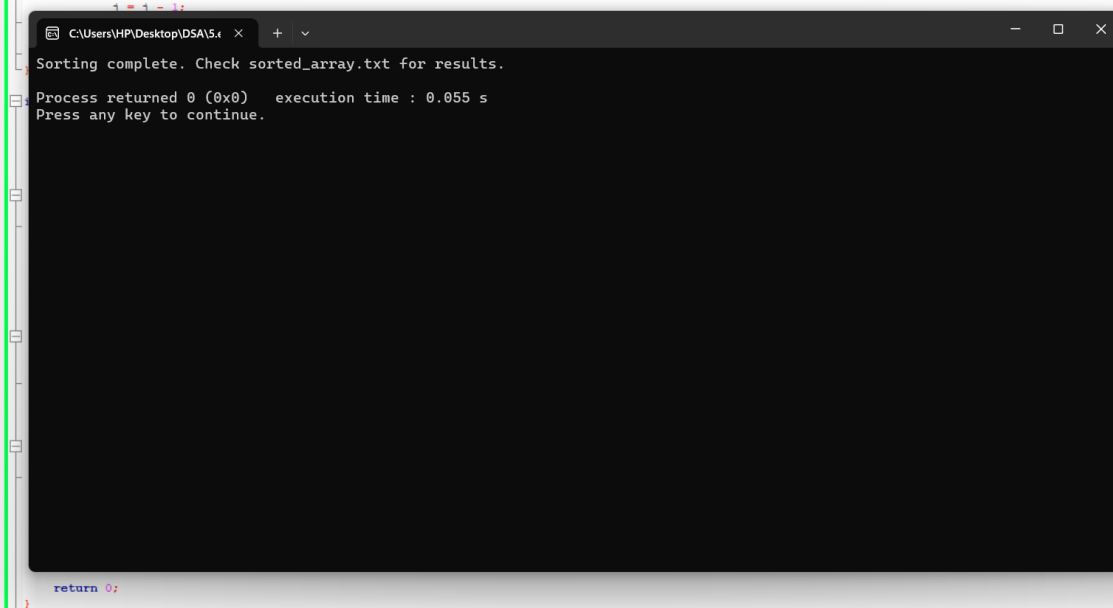
// Open the output file for writing
outputFile = fopen("sorted_array.txt", "w");
if (outputFile == NULL) {
    perror("Error opening output file");
    exit(EXIT_FAILURE);
}

// Write the sorted array to the output file
fprintf(outputFile, "Sorted Array:\n");
for (int i = 0; i < ARRAY_SIZE; i++) {
    fprintf(outputFile, "%d ", arr[i]);
}

// Close the output file
fclose(outputFile);

printf("Sorting complete. Check sorted_array.txt for results.\n");

return 0;
}
```



17. Matrix Operations

Code

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

```
#include <time.h>

#define SIZE 3

void generateRandomMatrix(int matrix[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            matrix[i][j] = rand() % 10;
        }
    }
}

void printMatrix(int matrix[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

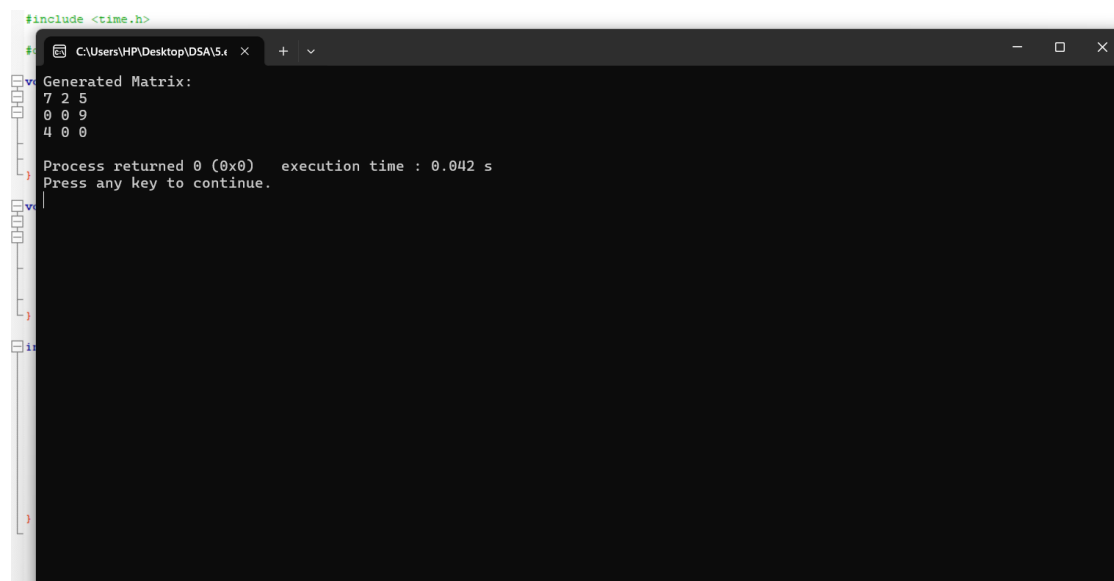
int main() {
    srand(time(NULL));

    int matrix[SIZE][SIZE];

    generateRandomMatrix(matrix);

    printf("Generated Matrix:\n");
    printMatrix(matrix);

    return 0;
}
```



The screenshot shows a Windows command prompt window with a dark background. The title bar indicates the file path is C:\Users\HP\Desktop\DSA\5.x. The output of the program is as follows:

```
#include <time.h>
Generated Matrix:
7 2 5
0 0 9
4 0 0

Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

Part- C

• **Basic operations**

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head, int newData);
void insertAtEnd(struct Node** head, int newData);
void deleteNode(struct Node** head, int key);
void printList(struct Node* head);

int main() {
    struct Node* head = NULL;

    insertAtBeginning(&head, 9);
    insertAtBeginning(&head, 7);
    insertAtBeginning(&head, 3);

    printf("Linked list after inserting at the beginning: ");
    printList(head);
    printf("\n");

    insertAtEnd(&head, 5);

    printf("Linked list after inserting at the end: ");
    printList(head);
    printf("\n");

    deleteNode(&head, 7);

    printf("Linked list after deleting node with value 7: ");
    printList(head);
    printf("\n");

    return 0;
}

void insertAtBeginning(struct Node** head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
```

```
newNode->next = *head;
*head = newNode;
}

void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head;
    newNode->data = newData;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

    while (last->next != NULL) {
        last = last->next;
    }

    last->next = newNode;
}

void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    struct Node* prev = NULL;

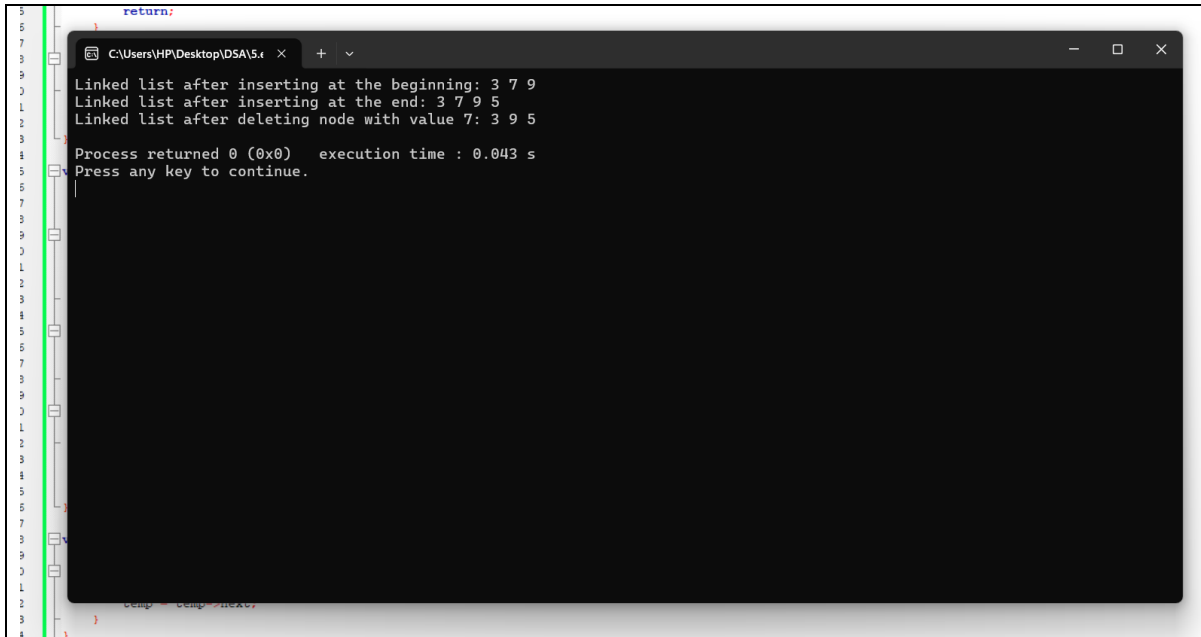
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }

    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        return;
    }

    prev->next = temp->next;
    free(temp);
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
```



The screenshot shows a C program execution window with the following output:

```
return;
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
26
```

```
int middle = findMiddle(head);
if (middle != -1) {
    printf("Middle element of the linked list: %d\n", middle);
} else {
    printf("Linked list is empty.\n");
}

return 0;
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

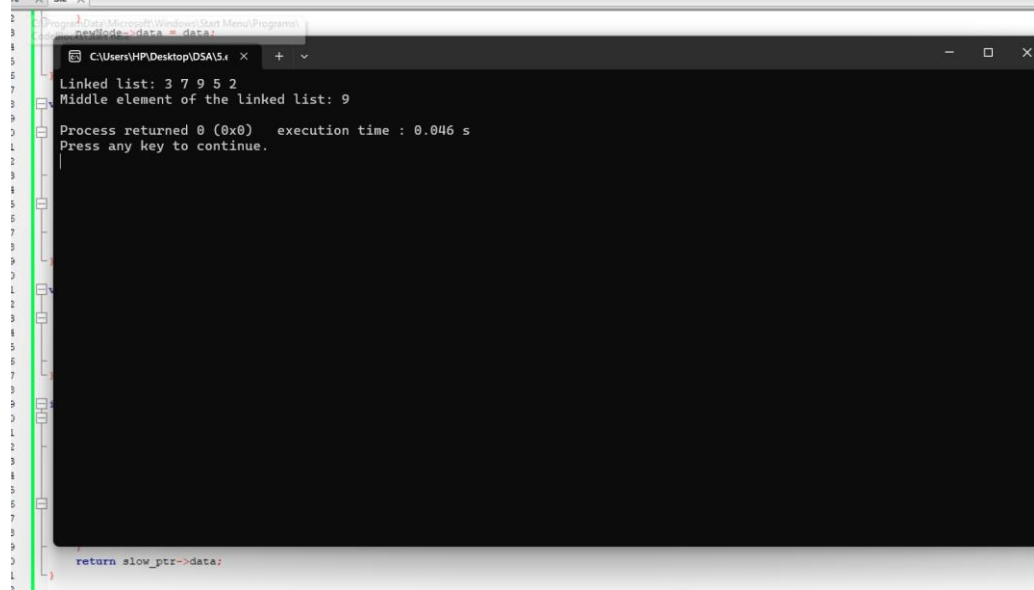
void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

int findMiddle(struct Node* head) {
    if (head == NULL) {
        return -1;
    }
    struct Node *slow_ptr = head;
    struct Node *fast_ptr = head;

    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next->next;
```

```
    slow_ptr = slow_ptr->next;  
}  
return slow_ptr->data;  
}
```

A screenshot of a C++ program execution. The program is titled 'C:\Users\HP\Desktop\DSA\...' and shows the output: 'Linked list: 3 7 9 5 2', 'Middle element of the linked list: 9', 'Process returned 0 (0x0) execution time : 0.046 s', and 'Press any key to continue.' The code snippet visible at the bottom of the window is 'return slow_ptr->data;'.

```
2 C:\ProgramData\Microsoft\Windows\Start Menu\Programs\...  
3 C:\Users\HP\Desktop\DSA\...  
4  
5  
6  
7 Linked list: 3 7 9 5 2  
8 Middle element of the linked list: 9  
9  
10 Process returned 0 (0x0) execution time : 0.046 s  
11 Press any key to continue.  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  

```

```
printf("Reversed linked list: ");
printList(head);
printf("\n");

return 0;
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

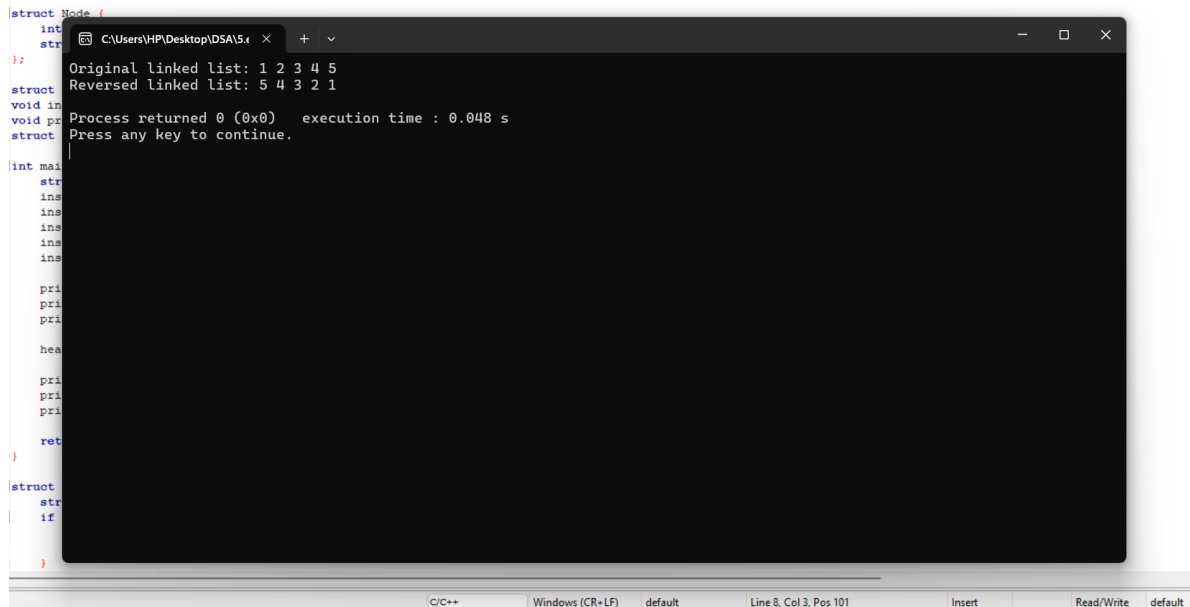
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
}
```



```
return prev;
}

struct Node {
    int data;
    struct Node* next;
};

void main() {
    struct Node* head = NULL;
    struct Node* temp = head;
    int data;
    while(1) {
        printf("Enter data: ");
        scanf("%d", &data);
        if(data == -1) break;
        insertAtEnd(&head, data);
    }
    printf("Original linked list: ");
    printList(head);
    printf("\n");
    reverseList(&head);
    printf("Reversed linked list: ");
    printList(head);
    printf("\n");
    return 0;
}
```



- **Reverse doubly link list**

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

```
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
```

```
struct Node* createNode(int data);
void insertAtEnd(struct Node** head, int newData);
void printList(struct Node* head);
void reverseList(struct Node** head);
```

```
int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 7);
    insertAtEnd(&head, 9);

    printf("Original doubly linked list: ");
    printList(head);
    printf("\n");

    reverseList(&head);

    printf("Reversed doubly linked list: ");
```

```
    printList(head);
    printf("\n");

    return 0;
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
    newNode->prev = last;
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

void reverseList(struct Node** head) {
    struct Node* current = *head;
    struct Node* temp = NULL;

    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
        if (current != NULL && current->prev == NULL) {
            *head = current;
        }
    }
}
```

```
}  
}  
  
t.here x 5.c x  
42  
43  
44  
45 Original doubly linked list: 3 7 9  
46 Reversed doubly linked list: 3  
47  
48 Process returned 0 (0x0) execution time : 0.041 s  
49 Press any key to continue.  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78 *head = current;  
79  
80
```

- **Rotate the link list**

Code

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

```
struct Node {  
    int data;  
    struct Node* next;  
};
```

```
struct Node* createNode(int data);  
void insertAtEnd(struct Node** head, int newData);  
void deleteMiddleNode(struct Node** head);  
void printList(struct Node* head);
```

```
int main() {  
    struct Node* head = NULL;  
    insertAtEnd(&head, 1);  
    insertAtEnd(&head, 2);  
    insertAtEnd(&head, 3);  
    insertAtEnd(&head, 4);  
    insertAtEnd(&head, 5);  
  
    printf("List before deletion: ");  
    printList(head);  
    printf("\n");
```

```
deleteMiddleNode(&head);

printf("List after deletion: ");
printList(head);
printf("\n");

return 0;
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

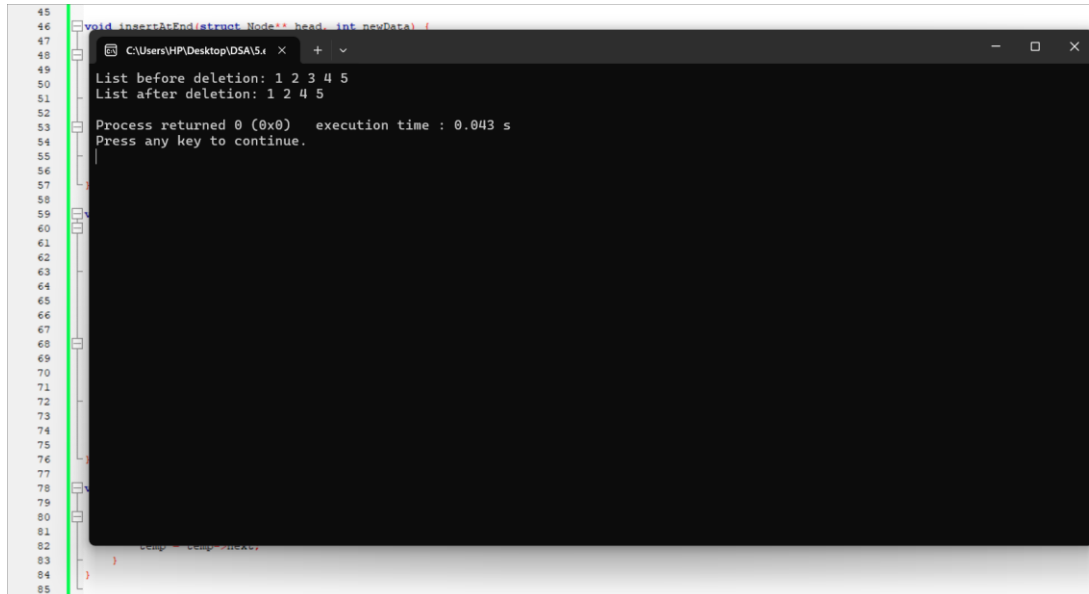
void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

void deleteMiddleNode(struct Node** head) {
    if (*head == NULL || (*head)->next == NULL) {
        printf("List is empty or contains only one node.\n");
        return;
    }
    struct Node* slowPtr = *head;
    struct Node* fastPtr = *head;
    struct Node* prev = NULL;

    while (fastPtr != NULL && fastPtr->next != NULL) {
        fastPtr = fastPtr->next->next;
        prev = slowPtr;
        slowPtr = slowPtr->next;
    }

    prev->next = slowPtr->next;
    free(slowPtr);
}
```

```
void printList(struct Node* head) {  
    struct Node* temp = head;  
    while (temp != NULL) {  
        printf("%d ", temp->data);  
        temp = temp->next;  
    }  
}
```

A screenshot of a C program execution in a Windows command prompt. The program is located at C:\Users\HP\Desktop\DSA\5.1. The output shows the list before deletion as 1 2 3 4 5 and the list after deletion as 1 2 4 5. The process returned 0 (0x0) and the execution time was 0.043 s. The prompt asks to press any key to continue.

```
45  
46  
47  
48  
49  
50 List before deletion: 1 2 3 4 5  
51 List after deletion: 1 2 4 5  
52  
53 Process returned 0 (0x0)   execution time : 0.043 s  
54 Press any key to continue.  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85
```

- **Delete middle of link list**

Code

```
#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
    int data;  
    struct Node* next;  
};  
  
struct Node* createNode(int data);  
void insertAtEnd(struct Node** head, int newData);  
void deleteMiddleNode(struct Node** head);  
void printList(struct Node* head);  
  
int main() {  
    struct Node* head = NULL;  
    insertAtEnd(&head, 1);  
    insertAtEnd(&head, 2);  
    insertAtEnd(&head, 3);  
    insertAtEnd(&head, 4);
```

```
insertAtEnd(&head, 5);

printf("List before deletion: ");
printList(head);
printf("\n");

deleteMiddleNode(&head);

printf("List after deletion: ");
printList(head);
printf("\n");

return 0;
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

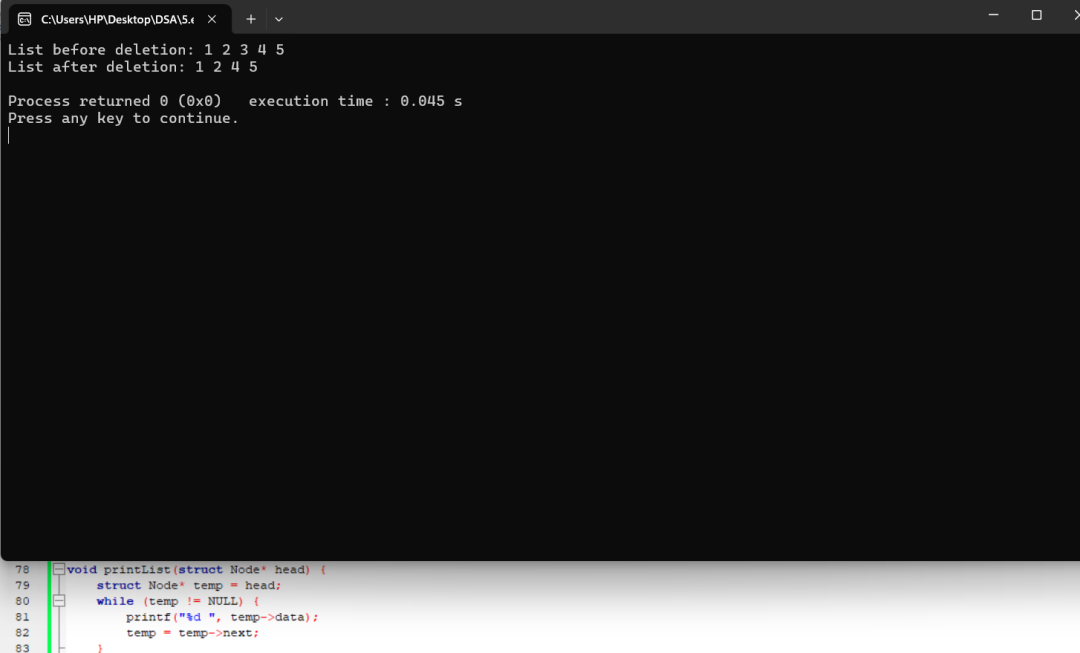
void deleteMiddleNode(struct Node** head) {
    if (*head == NULL || (*head)->next == NULL) {
        printf("List is empty or contains only one node.\n");
        return;
    }
    struct Node* slowPtr = *head;
    struct Node* fastPtr = *head;
    struct Node* prev = NULL;

    while (fastPtr != NULL && fastPtr->next != NULL) {
        fastPtr = fastPtr->next->next;
        prev = slowPtr;
        slowPtr = slowPtr->next;
    }
```

```
}

prev->next = slowPtr->next;
free(slowPtr);
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\HP\Desktop\DSA\5.i". The output of the program is as follows:

```
List before deletion: 1 2 3 4 5
List after deletion: 1 2 4 5

Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

Below the command prompt, a portion of the C code is visible, showing the `printList` function:

```
78 void printList(struct Node* head) {
79     struct Node* temp = head;
80     while (temp != NULL) {
81         printf("%d ", temp->data);
82         temp = temp->next;
83     }
84 }
```

- **Remove duplicate element from sorted Linked List**

Code

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

struct Node {
    int data;
    struct Node* next;
};
```

```
void removeDuplicates(struct Node* head);
void printList(struct Node* head);

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 2;
    head->next->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->next->data = 3;
    head->next->next->next->next = NULL;

    printf("Original list: ");
    printList(head);
    printf("\n");

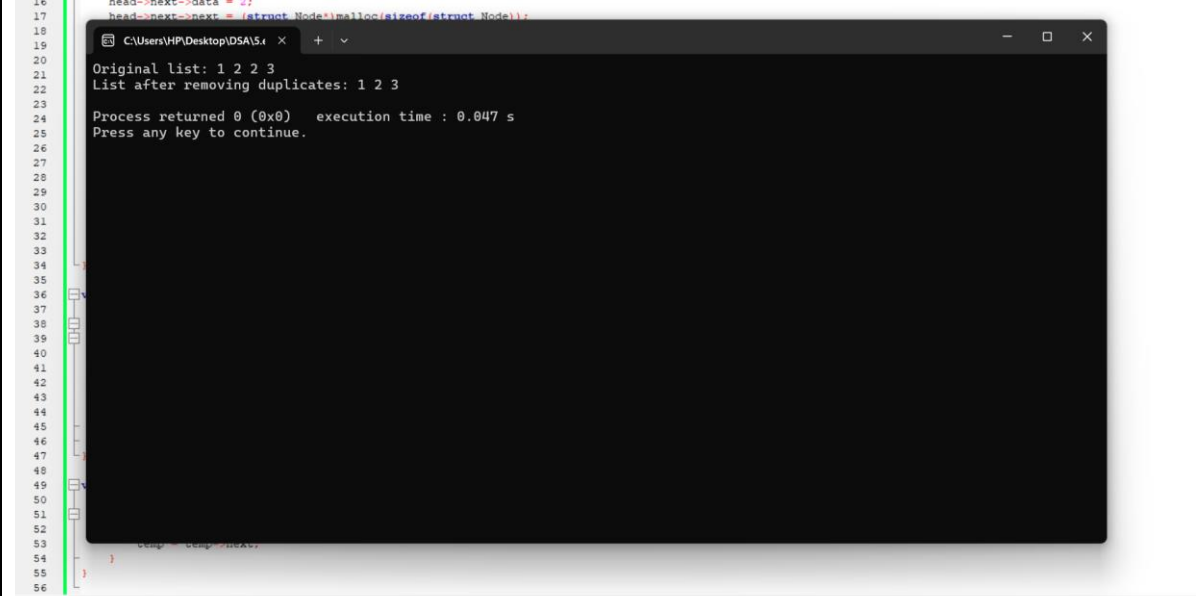
    removeDuplicates(head);

    printf("List after removing duplicates: ");
    printList(head);
    printf("\n");

    return 0;
}

void removeDuplicates(struct Node* head) {
    struct Node* current = head;
    while (current != NULL && current->next != NULL) {
        if (current->data == current->next->data) {
            struct Node* temp = current->next;
            current->next = current->next->next;
            free(temp);
        } else {
            current = current->next;
        }
    }
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
```

```
16 head->next->data = 2;
17 head->next->next = (struct Node*)malloc(sizeof(struct Node));
18
19
20
21 Original list: 1 2 2 3
22 List after removing duplicates: 1 2 3
23
24 Process returned 0 (0x0)   execution time : 0.047 s
25 Press any key to continue.
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

- Detect loop or cycle in a linked list

Code

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data);
void insertAtEnd(struct Node** head, int data);
bool detectLoop(struct Node* head);

int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    head->next->next->next->next = head->next;

    if (detectLoop(head)) {
        printf("Loop detected in the linked list.\n");
    } else {
        printf("No loop detected in the linked list.\n");
    }

    return 0;
}
```

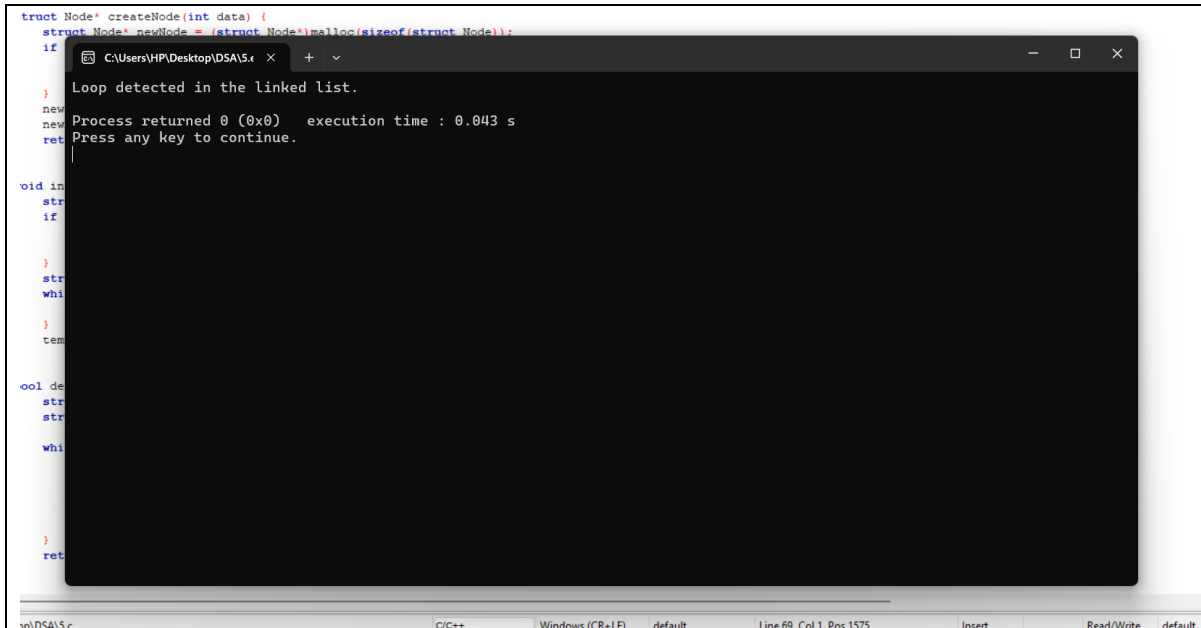
```
}

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

bool detectLoop(struct Node* head) {
    struct Node* slow = head;
    struct Node* fast = head;

    while (slow != NULL && fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```



The screenshot shows a C++ program running in a terminal window. The program is titled "C:\Users\HP\Desktop\DSA\5.r". The output of the program is as follows:

```
Loop detected in the linked list.
Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

- **Traversal of Circular Linked List**

Code

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data);
void insertAtEnd(struct Node** head, int newData);
void traverseCircularLinkedList(struct Node* head);

int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 3);
    insertAtEnd(&head, 7);
    insertAtEnd(&head, 9);

    printf("Circular linked list: ");
    traverseCircularLinkedList(head);

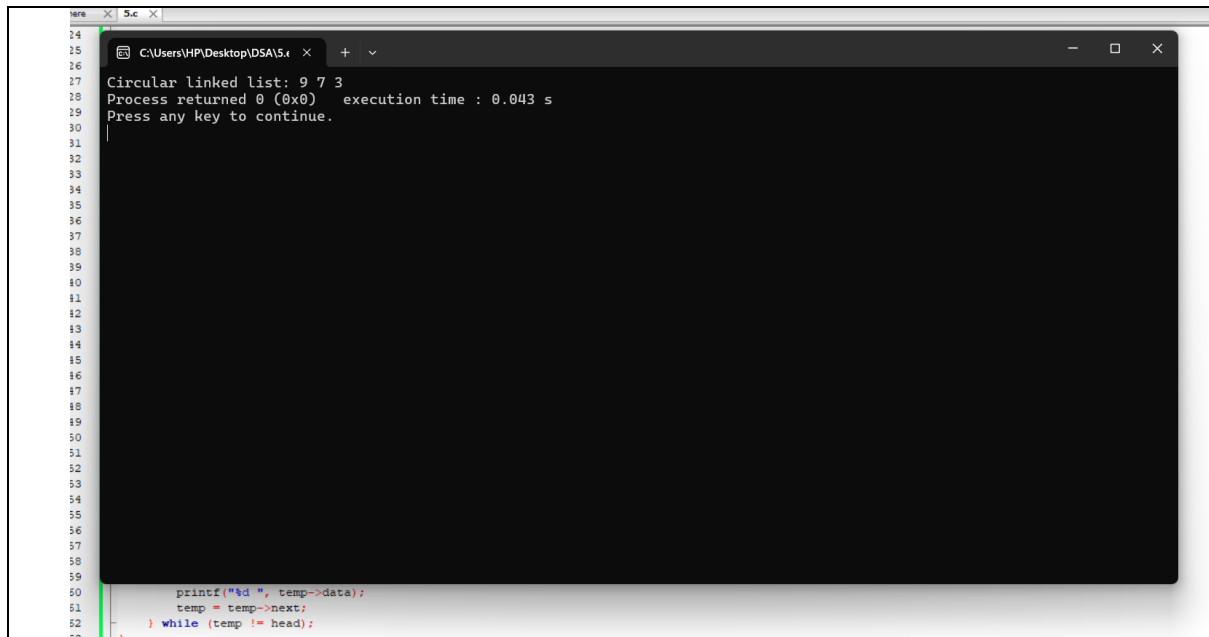
    return 0;
}

struct Node* createNode(int data) {
```

```
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
    printf("Memory allocation failed!\n");
    exit(EXIT_FAILURE);
}
newNode->data = data;
newNode->next = NULL;
return newNode;
}

void insertAtEnd(struct Node** head, int newData) {
    struct Node* newNode = createNode(newData);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head;
    } else {
        struct Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
        *head = newNode; // Update head to point to the new node
    }
}

void traverseCircularLinkedList(struct Node* head) {
    if (head == NULL) {
        printf("Circular linked list is empty\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
}
```



The screenshot shows a C program execution window. The output displays the circular linked list values: 9 7 3. The process returned 0 (0x0) and the execution time was 0.043 s. The program prompts the user to press any key to continue. The source code visible at the bottom of the window is as follows:

```
24  
25  
26  
27 Circular linked list: 9 7 3  
28 Process returned 0 (0x0)   execution time : 0.043 s  
29 Press any key to continue.  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60 printf("%d ", temp->data);  
61 temp = temp->next;  
62 } while (temp != head);  
63
```