Spring 2024

Submitted by: **Hassan Zaib Jadoon**

Registration No: **22pwsce2144**

Class Section: **A**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Abdullah Hamid

15 April 2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Operating Systems Lab 4: Introduction to C Programming

Objectives:

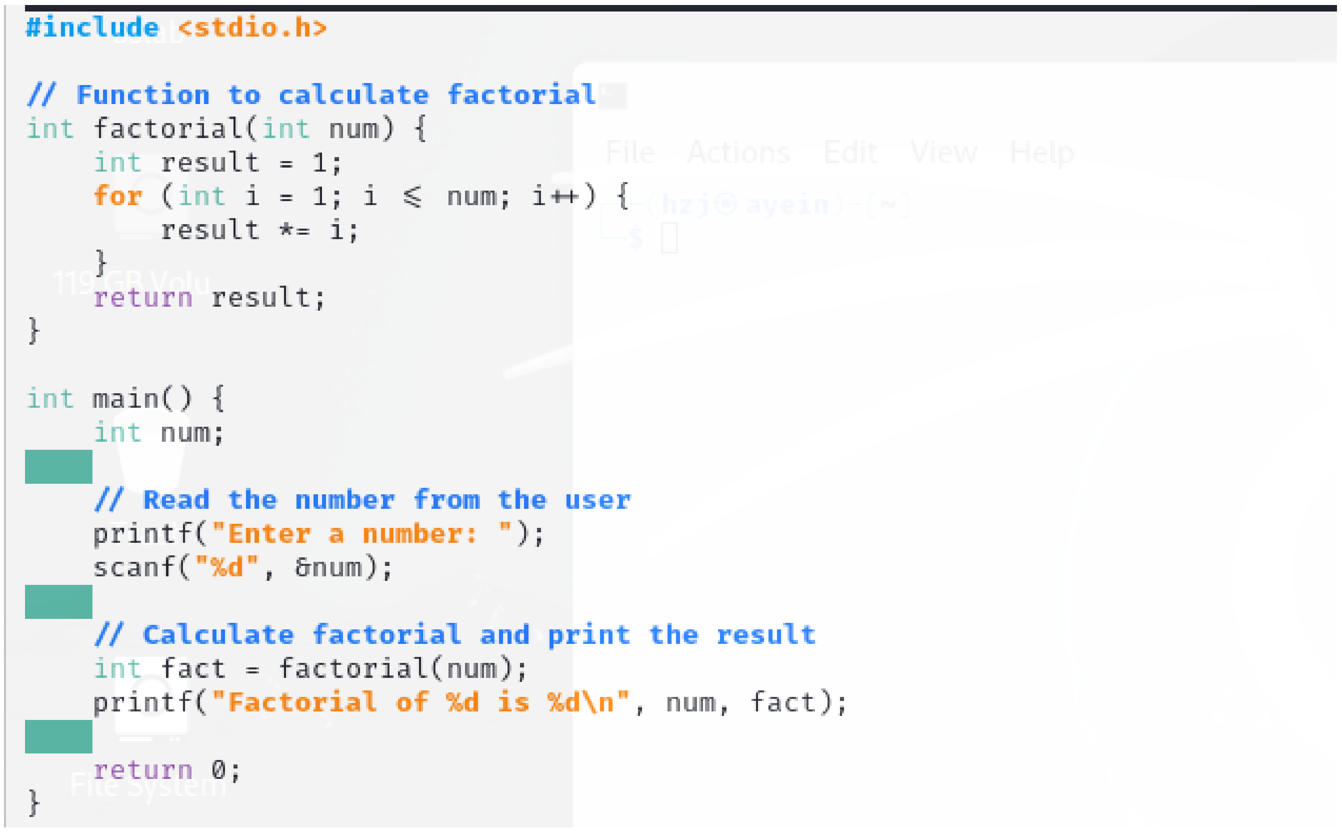
The primary objective of this lab session is to gain practical experience in C programming, focusing on various fundamental concepts such as functions, pointers, arrays, structures, dynamic memory allocation, and linked lists.

Tasks:

Task 1: A Simple C program with more than one function (Parameters passed by value)

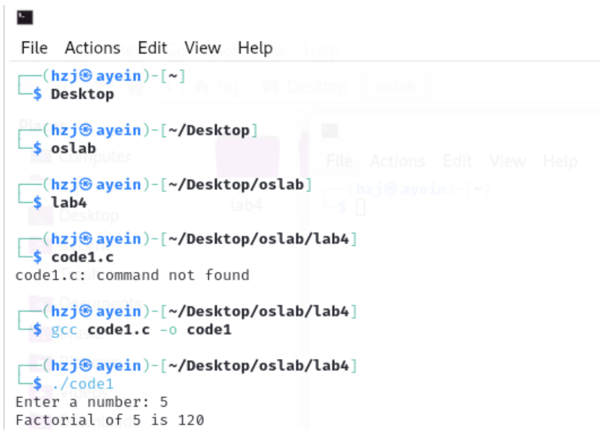
Write a program that reads a number from the user and finds its factorial using a function. The argument should be passed to the function by value.

Code:



```c
#include <stdio.h>

// Function to calculate factorial
int factorial(int num) {
    int result = 1;
    for (int i = 1; i <= num; i++) {
        result *= i;
    }
    return result;
}

int main() {
    int num;

    // Read the number from the user
    printf("Enter a number: ");
    scanf("%d", &num);

    // Calculate factorial and print the result
    int fact = factorial(num);
    printf("Factorial of %d is %d\n", num, fact);

    return 0;
}
```

Output:

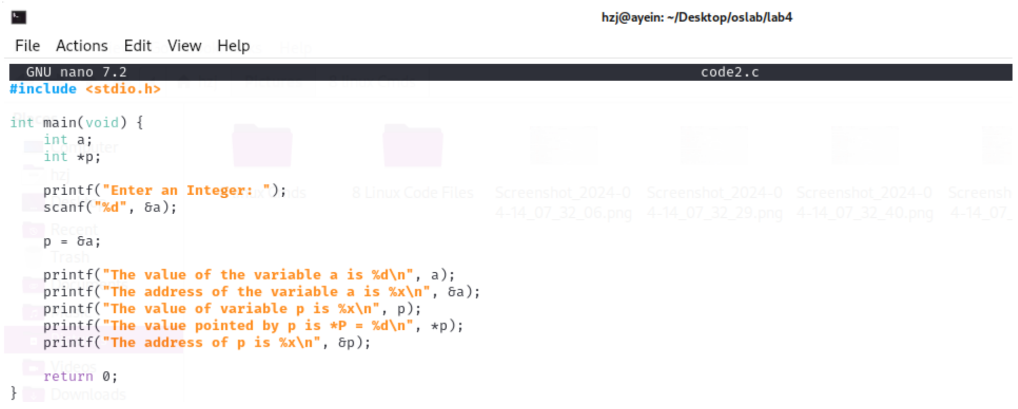


```
┌──(hzj㉿ayein)-[~]
└─$ Desktop

┌──(hzj㉿ayein)-[~/Desktop]
└─$ oslab

┌──(hzj㉿ayein)-[~/Desktop/oslab]
└─$ lab4

┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ code1.c
code1.c: command not found

┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ gcc code1.c -o code1

┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code1
Enter a number: 5
Factorial of 5 is 120
```

Task 2: Basic concepts of Pointers in C

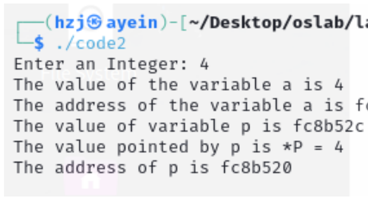
Explore pointer variables, and the * and & operators by running and observing a given program.

Code:



```c
#include <stdio.h>

int main(void) {
    int a;
    int *p;

    printf("Enter an Integer: ");
    scanf("%d", &a);

    p = &a;

    printf("The value of the variable a is %d\n", a);
    printf("The address of the variable a is %x\n", &a);
    printf("The value of variable p is %x\n", p);
    printf("The value pointed by p is *P = %d\n", *p);
    printf("The address of p is %x\n", &p);

    return 0;
}
```

Output:



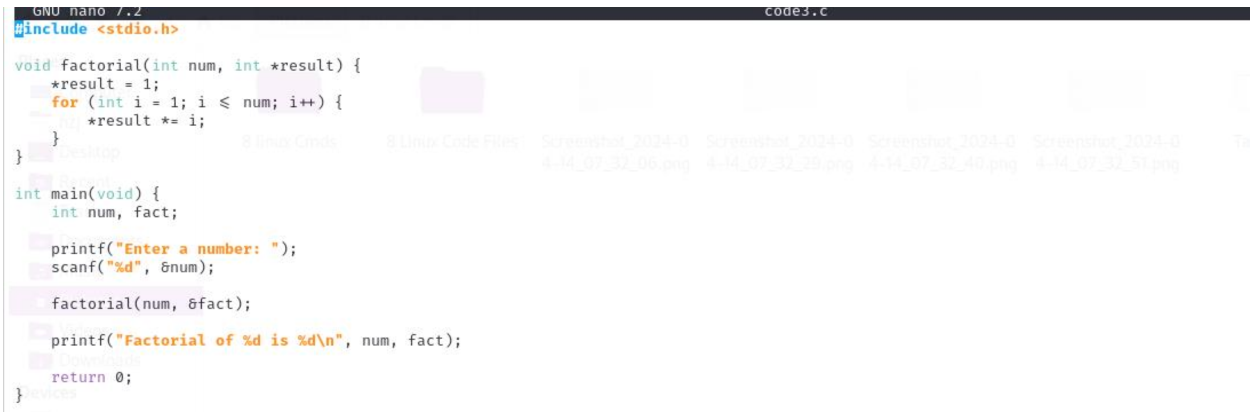
```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code2
Enter an Integer: 4
The value of the variable a is 4
The address of the variable a is fc8b52c
The value of variable p is fc8b52c
The value pointed by p is *P = 4
The address of p is fc8b520
```

Observation:

The program effectively illustrates the basic concepts of pointers in C by prompting the user for an integer input, displaying its value and memory address, assigning the address to a pointer variable, and finally, displaying the value pointed to by the pointer. It demonstrates the fundamental operations of pointer assignment and dereferencing, offering a clear insight into memory management in C programming.

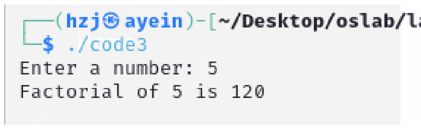
Task 3: Redo Task 1 with the result passed by pointer.

Code:



```c
#include <stdio.h>

void factorial(int num, int *result) {
    *result = 1;
    for (int i = 1; i <= num; i++) {
        *result *= i;
    }
}
int main(void) {
    int num, fact;

    printf("Enter a number: ");
    scanf("%d", &num);

    factorial(num, &fact);

    printf("Factorial of %d is %d\n", num, fact);

    return 0;
}
```

Output:

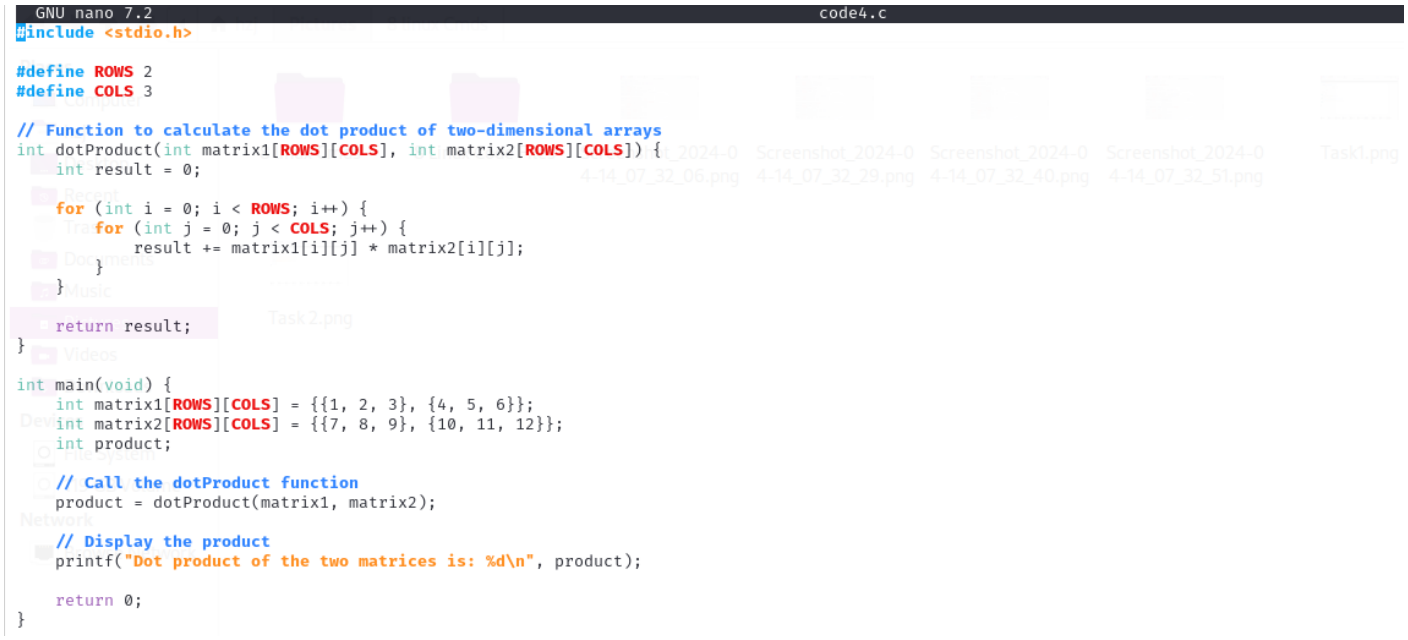


```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code3
Enter a number: 5
Factorial of 5 is 120
```

Task 4: Using Arrays in C

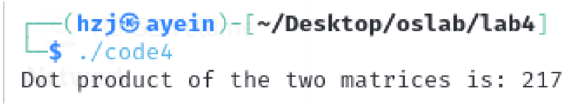
Write a function that calculates the dot product of a two-dimensional array. Call this function from the **main()** function and display the product.

Code:



```
  GNU nano 7.2                                   code4.c
#include <stdio.h>

#define ROWS 2
#define COLS 3

// Function to calculate the dot product of two-dimensional arrays
int dotProduct(int matrix1[ROWS][COLS], int matrix2[ROWS][COLS]) {
    int result = 0;

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            result += matrix1[i][j] * matrix2[i][j];
        }
    }

    return result;
}

int main(void) {
    int matrix1[ROWS][COLS] = {{1, 2, 3}, {4, 5, 6}};
    int matrix2[ROWS][COLS] = {{7, 8, 9}, {10, 11, 12}};
    int product;

    // Call the dotProduct function
    product = dotProduct(matrix1, matrix2);

    // Display the product
    printf("Dot product of the two matrices is: %d\n", product);

    return 0;
}
```

Output:

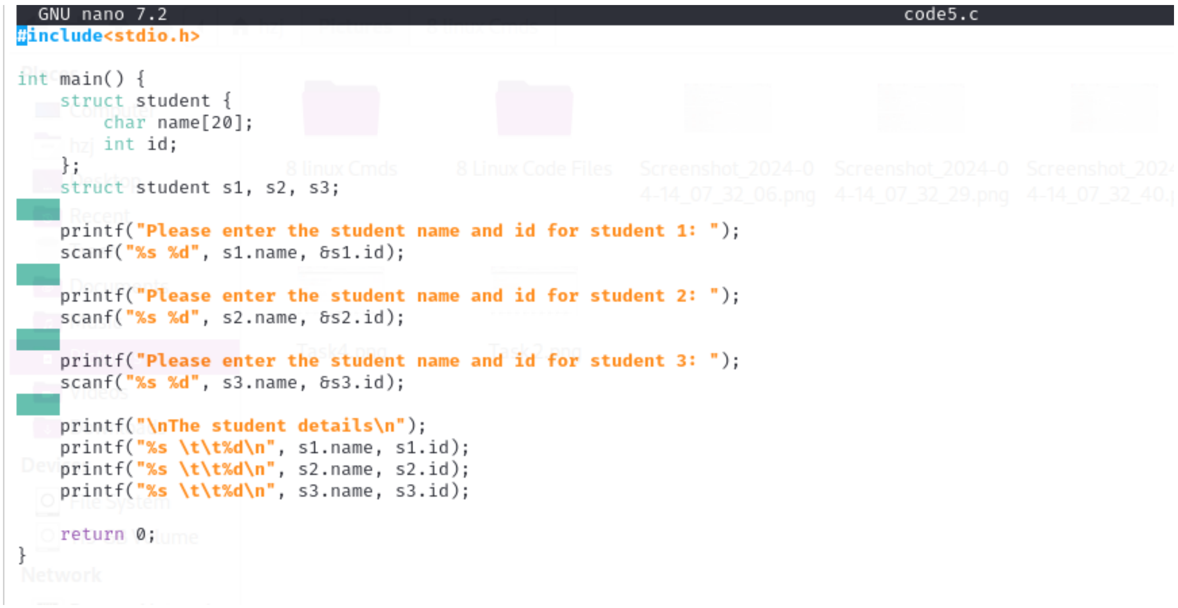


```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code4
Dot product of the two matrices is: 217
```

Task 5: Using Structures in C

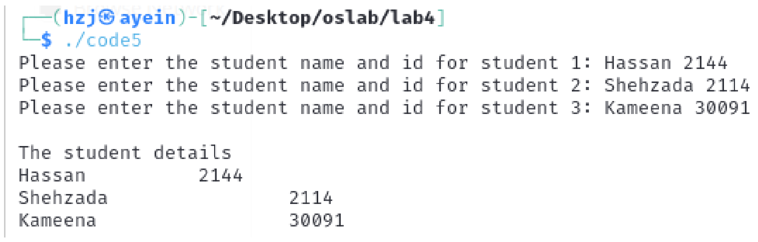
Explore the use of structures in C by running a provided program that collects student details and displays them.

Code:



```
  GNU nano 7.2                                   code5.c
#include<stdio.h>

int main() {
    struct student {
        char name[20];
        int id;
    };
    struct student s1, s2, s3;

    printf("Please enter the student name and id for student 1: ");
    scanf("%s %d", s1.name, &s1.id);

    printf("Please enter the student name and id for student 2: ");
    scanf("%s %d", s2.name, &s2.id);

    printf("Please enter the student name and id for student 3: ");
    scanf("%s %d", s3.name, &s3.id);

    printf("\nThe student details\n");
    printf("%s \t\t%d\n", s1.name, s1.id);
    printf("%s \t\t%d\n", s2.name, s2.id);
    printf("%s \t\t%d\n", s3.name, s3.id);

    return 0;
}
```

Output:

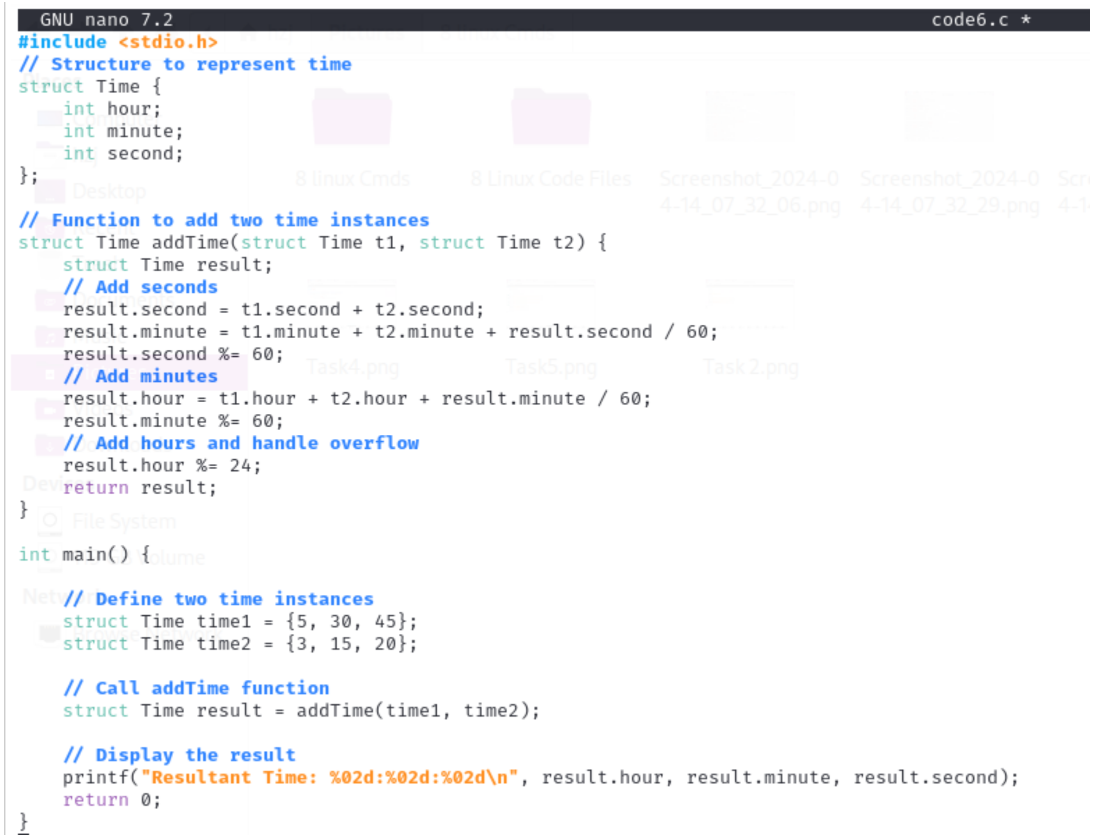


```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code5
Please enter the student name and id for student 1: Hassan 2144
Please enter the student name and id for student 2: Shehzada 2114
Please enter the student name and id for student 3: Kameena 30091

The student details
Hassan          2144
Shehzada                2114
Kameena                 30091
```

Task 6:

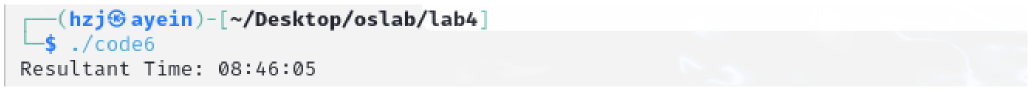
Write a C code to declare a "Time" structure that contains hour, minute, and seconds as its data members. Write a function that adds two time instances and returns the resultant time to the main function.

Code:



```
  GNU nano 7.2                                                              code6.c *
#include <stdio.h>
// Structure to represent time
struct Time {
    int hour;
    int minute;
    int second;
};

// Function to add two time instances
struct Time addTime(struct Time t1, struct Time t2) {
    struct Time result;
    // Add seconds
    result.second = t1.second + t2.second;
    result.minute = t1.minute + t2.minute + result.second / 60;
    result.second %= 60;
    // Add minutes
    result.hour = t1.hour + t2.hour + result.minute / 60;
    result.minute %= 60;
    // Add hours and handle overflow
    result.hour %= 24;
    return result;
}

int main() {

    // Define two time instances
    struct Time time1 = {5, 30, 45};
    struct Time time2 = {3, 15, 20};

    // Call addTime function
    struct Time result = addTime(time1, time2);

    // Display the result
    printf("Resultant Time: %02d:%02d:%02d\n", result.hour, result.minute, result.second);
    return 0;
}
```

Output:

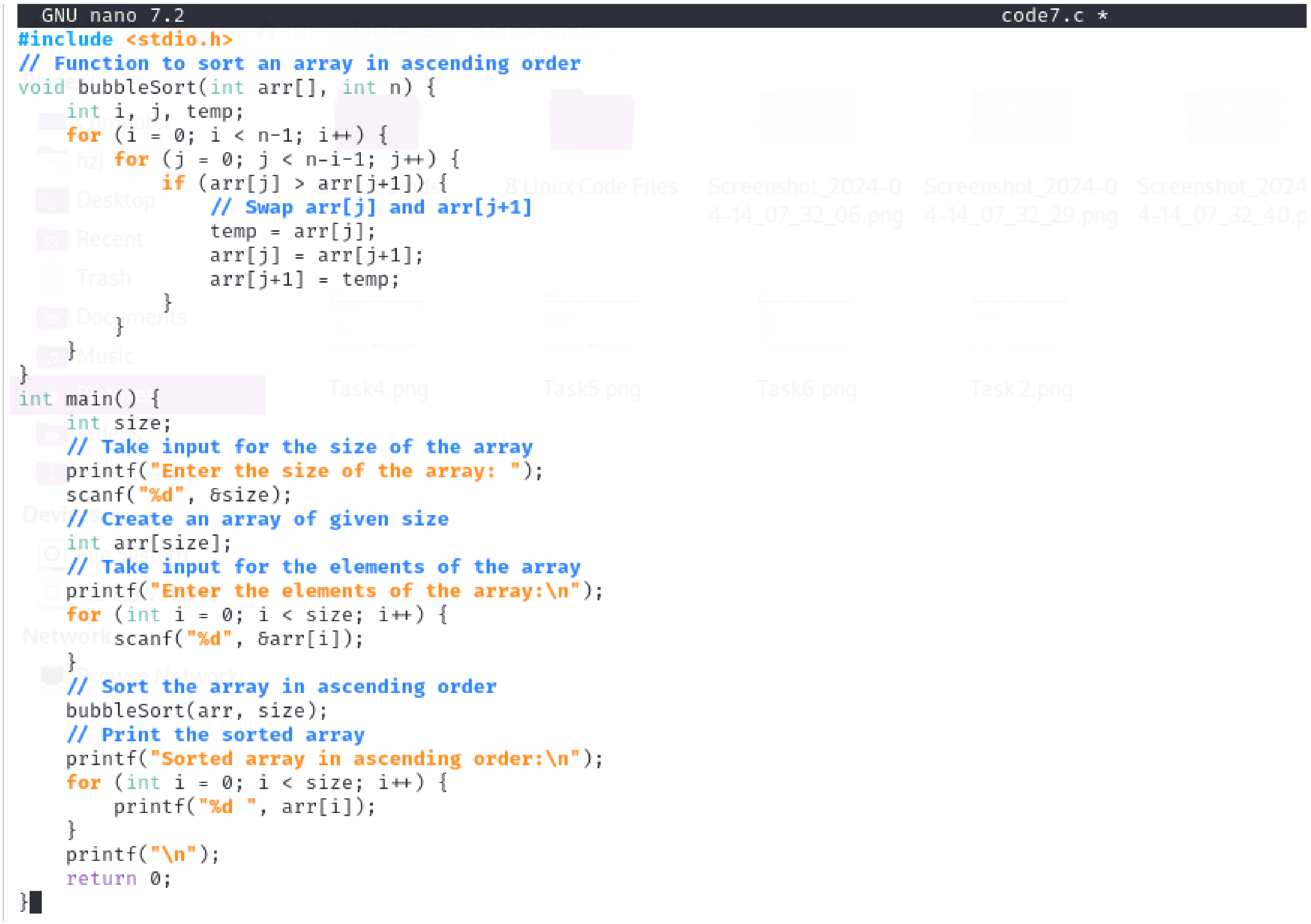


```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code6
Resultant Time: 08:46:05
```

Task 7: Dynamic Memory Allocation

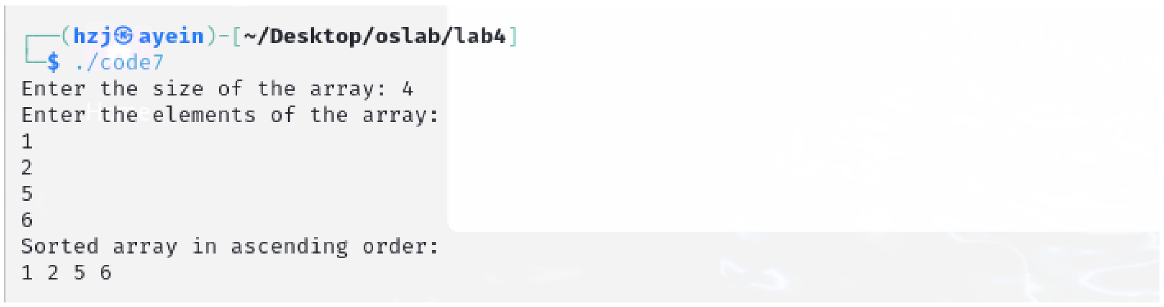
Write a program that takes the size of an array as input from the user, creates the array, takes the elements of the array as input, and sorts them in ascending order using dynamic memory allocation.

Code:



```
  GNU nano 7.2                                                          code7.c *
#include <stdio.h>
// Function to sort an array in ascending order
void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                // Swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
int main() {
    int size;
    // Take input for the size of the array
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    // Create an array of given size
    int arr[size];
    // Take input for the elements of the array
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    // Sort the array in ascending order
    bubbleSort(arr, size);
    // Print the sorted array
    printf("Sorted array in ascending order:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output:



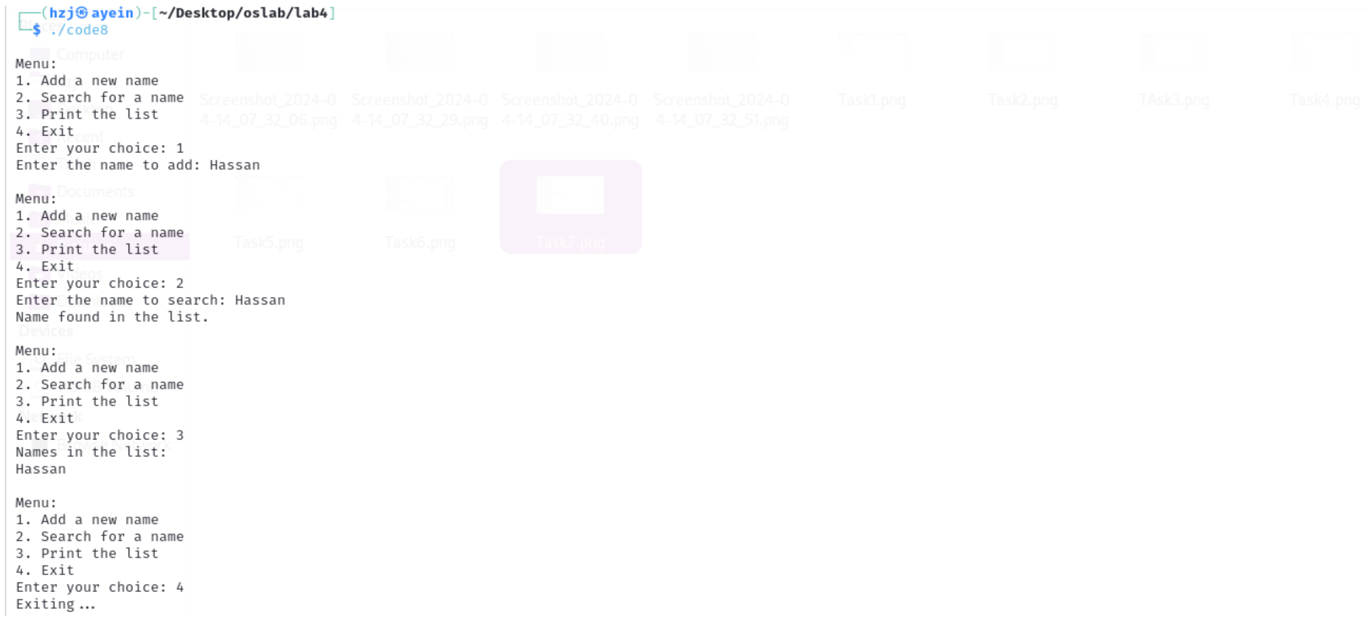
```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code7
Enter the size of the array: 4
Enter the elements of the array:
1
2
5
6
Sorted array in ascending order:
1 2 5 6
```

Task 8:Use of Linked List in C

Develop a complete menu-driven program to:

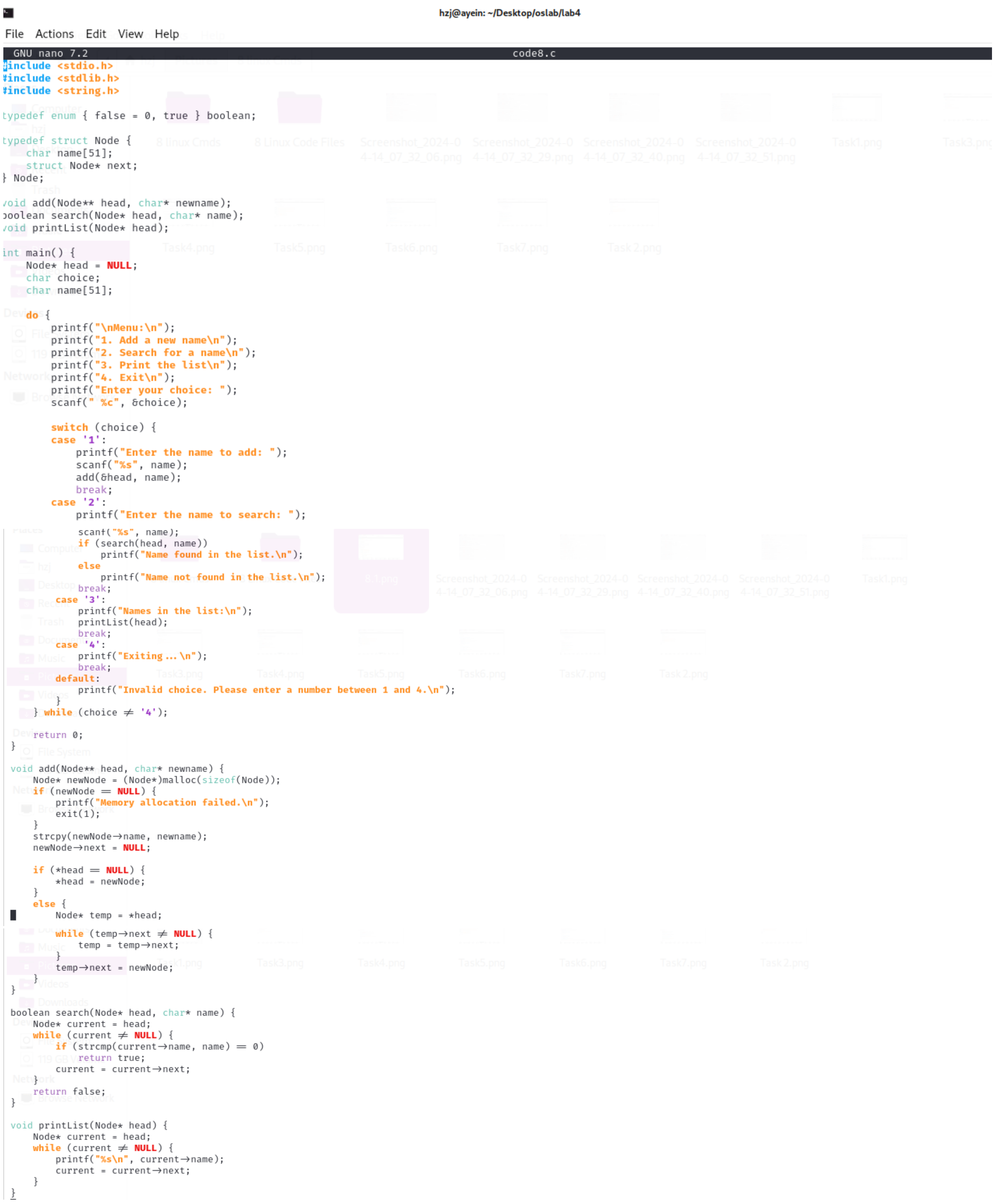
- Build a linked list to save a list of names (each name not exceeding 50 characters).
- Write a function to append a new name to the list.
- Write a function to search for a given name in the list.

Output:



```
┌──(hzj㉿ayein)-[~/Desktop/oslab/lab4]
└─$ ./code8
Menu:
1. Add a new name
2. Search for a name
3. Print the list
4. Exit
Enter your choice: 1
Enter the name to add: Hassan

Menu:
1. Add a new name
2. Search for a name
3. Print the list
4. Exit
Enter your choice: 2
Enter the name to search: Hassan
Name found in the list.

Menu:
1. Add a new name
2. Search for a name
3. Print the list
4. Exit
Enter your choice: 3
Names in the list:
Hassan

Menu:
1. Add a new name
2. Search for a name
3. Print the list
4. Exit
Enter your choice: 4
Exiting...
```

Code:



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

typedef enum { false = 0, true } boolean;

typedef struct Node {
    char name[51];
    struct Node* next;
} Node;

void add(Node** head, char* newname);
boolean search(Node* head, char* name);
void printList(Node* head);

int main() {
    Node* head = NULL;
    char choice;
    char name[51];

    do {
        printf("\nMenu:\n");
        printf("1. Add a new name\n");
        printf("2. Search for a name\n");
        printf("3. Print the list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf(" %c", &choice);

        switch (choice) {
        case '1':
            printf("Enter the name to add: ");
            scanf("%s", name);
            add(&head, name);
            break;
        case '2':
            printf("Enter the name to search: ");
            scanf("%s", name);
            if (search(head, name))
                printf("Name found in the list.\n");
            else
                printf("Name not found in the list.\n");
            break;
        case '3':
            printf("Names in the list:\n");
            printList(head);
            break;
        case '4':
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please enter a number between 1 and 4.\n");
        }
    } while (choice ≠ '4');

    return 0;
}

void add(Node** head, char* newname) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    strcpy(newNode→name, newname);
    newNode→next = NULL;

    if (*head == NULL) {
        *head = newNode;
    }
    else {
        Node* temp = *head;
        while (temp→next ≠ NULL) {
            temp = temp→next;
        }
        temp→next = newNode;
    }
}

boolean search(Node* head, char* name) {
    Node* current = head;
    while (current ≠ NULL) {
        if (strcmp(current→name, name) == 0)
            return true;
        current = current→next;
    }
    return false;
}

void printList(Node* head) {
    Node* current = head;
    while (current ≠ NULL) {
        printf("%s\n", current→name);
        current = current→next;
    }
}
```

CSE 302L: Operating Systems Lab

LAB ASSESSMENT RUBRICS

Marking Criteria	**Exceeds expectation (2.5)**	**Meets expectation (1.5)**	**Does not meet expectation (0)**	**Score**
1. Correctness	Program compiles (no errors and no warnings). Program always works correctly and meets the specification(s). Completed between 81-100% of the requirements.	Program compiles (no errors and some warnings). Some details of the program specification are violated, program functions incorrectly for some inputs. Completed between 41-80% of the requirements.	Program fails to or compile with lots of warnings. Program only functions correctly in very limited cases or not at all. Completed less than 40% of the requirements.	
2. Delivery	Delivered on time, and in correct format (disk, email, hard copy etc.)	Not delivered on time, or slightly incorrect format.	Not delivered on time or not in correct format.	
3. Coding Standards	Proper indentation, whitespace, line length, wrapping, comments and references.	Missing some of whitespace, line length, wrapping, comments or references.	Poor use of whitespace, line length, wrapping, comments and references.	
4. Presentation of document	Includes name, date, and assignment title. Task titles, objectives, output screenshots included and good formatting and excellently organized.	Includes name, date, and assignment title. Task titles, objectives, output screenshots included and good formatting.	No name, date, or assignment title included. No task titles, no objectives, no output screenshots, poor formatting.	

Instructor:

Name: Engr. Abdullah Hamid Signature: ______________________