Introduction to C Programming

LAB # 04

Spring 2023

CSE-204L Operating Systems Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

"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. Madiha Sher

Date:

21st March 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

OBJECTIVES:

The aim of this laboratory is to learn and practice SHELL scripts by writing small SHELL programs.

To gain experience with:

- Writing simple C programs with more than one function
- Basic concepts of Pointers in C
- Using Arrays in C
- Using Structures in C
- Dynamic Memory Allocation
- Use of Linked List

PURPOSE:

To gain experience about C programming

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

Read:

```
int scanf ( const char * format, ... );
```

This function reads data from stdin and stores them according to the parameter format into the locations pointed by the additional arguments.

Display:

```
int printf ( const char * format, ... );
```

This function writes the C string pointed by format to the standard output (stdout). If format includes format specifiers (subsequences beginning with %), the additional arguments following format are formatted and inserted in the resulting string replacing their respective specifiers.

specifier	**Output**	**Example**
`d` *or* `i`	Signed decimal integer	`392`
`U`	Unsigned decimal integer	`7235`
`O`	Unsigned octal	`610`
`X`	Unsigned hexadecimal integer	`7fa`
`F`	Decimal floating point	`392.65`
`E`	Scientific notation (mantissa/exponent)	`3.9265e+2`
`A`	Hexadecimal floating point, lowercase	`-0xc.90fep-2`
`C`	Character	`a`
`S`	String of characters	`sample`
`P`	Pointer address	`b8000000`

Task # 1: Write a program reads a number from user and finds its factorial using function. Pass the argument to function by value.

2. Basic concepts of Pointers in C

Every variable in C has a name (variable name), a memory location (to store the data), and an address.

For the variable declaration

```
int  num;
```

num **-------------------------->** variable name

memory

8152 ------------------> address

A variable used to store the address value is called as the Pointer. It can be defined as `int *ptr;`

Task 2: The following program demonstrates about the pointer variable, * and & operators. Run and observe the output.

```
#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);
}
```

Task 3: Redo task number 1. The result should be passed by pointer.

3. Using Arrays in C

Arrays are important to C and should need a lot more attention. The following important concepts related to array should be clear to a C programmer.

S.N.	Concept & Description
1	Multi-dimensional arrays C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
2	Passing arrays to functions You can pass to the function a pointer to an array by specifying the array's name without an index.
3	Return array from a function C allows a function to return an array.
4	Pointer to an array You can generate a pointer to the first element of an array by simply specifying the array name, without any index.

Task 4: Write a function that calculates the dot product of two dimensional array. Call this function from main() function and display the product.

4. Using Structures in C

Arrays allow defining type of variables that can hold several data items of the same kind. Similarly, structure is a collection/group of different/same variables.

Task 5: Run the following program and observe the output.

Program

```
#include<stdio.h>
main(){
struct student {
      char name[20];
      int id;
};

struct student s1, s2, s3;

printf(“Please enter the student name, and id\n”);
scanf(“%s %d”, &s1.name, &s1.id);
scanf(“%s %d”, &s2.name, &s2.id);
scanf(“%s %d”, &s3.name, &s3.id);

printf(“\nThe student details”);
printf(“\n%s \t\t%d”,s1.name,s1.id);
printf(“\n%s \t\t%d”,s2.name,s2.id);
printf(“\n%s \t\t%d”,s3.name,s3.id);
}
Sample output
colonel$ ./structure
Please enter the student name, and  id
Ahamed 9876
Ali 9979
Yahya 9988


The student details
Ahamed          9876
Ali             9979
Yahya           9988
```

Task 6: Write a C code to declare "Time" structure that contains hour, minute and seconds as its data members. Write a function that adds two time instances and return the resultant time to the main function.

5. Dynamic Memory Allocation

Using malloc to obtain memory at run-time:

- Memory can be allocated dynamically (at run-time) using the function **malloc()** – accessible through **<stdlib.h>**

- The allocation is made from a special memory area called the **heap.**

- The function, malloc() returns a pointer (address) to the allocated storage.

- However, malloc() does not associate any type to the pointer it returns – it is said to be **void.**

- For the pointer to be useful, it must be associated with a type using casting .**e**.g.,

  ```
  int  *int_ptr;

  int_ptr=(int *) malloc(4);

  *int_ptr =17;
  ```

- The above statements reserve four bytes and return the address of first byte, cast it to **int** and assign it to integer pointer **int_ptr**.

- Since the bytes allocated to **int** is system-dependent, it is safer to use the function **sizeof ()** to get the actual number of bytes associated with the particular type being considered.

- sizeof() is system-independent and can be used even with user-defined types.

- Thus, the above statements are better represented as follows:

  ```
  int *int_ptr;
  int_ptr=(int *) malloc(sizeof(int));

  *int_ptr =17;
  ```

- Note that there is no name associated with the memory obtained by malloc. It can only be accessed as ***int_ptr**. It is sometimes called **anonymous** variable.

- Thus, should **int_ptr** be given another address, the location (returned by malloc) will be lost. It can neither be accessed by the program nor by the system. It is said to be a **lost** object.

- When we no longer need a dynamic variable, we can return the storage it occupies using the free() function.

  e.g.   free(int_ptr);

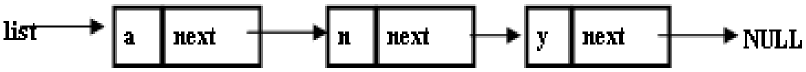
Task 7: Write a program that takes the size of the array as input from the user, create the array and then take the elements of array as input and sort in ascending order.

6. Use of Linked List in C

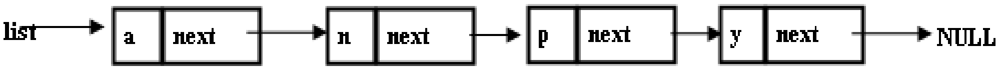
A linked list is a sequence of data structures, which are connected together via links. Linked list, like stack and queue is a homogeneous linear list consisting of nodes in which each node is linked to the next.

However, unlike stack and queue, an item can be deleted at any location in the list, and can be added (inserted) at any location provided the order of the items in the list is maintained.

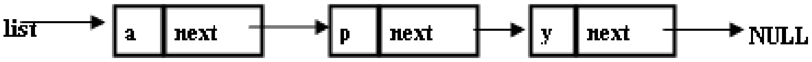
The following figure shows the format of a linked list and how it behaves on insertion and deletion:



If p is inserted, the list becomes:



If n is deleted, the list becomes:



Thus, a linked-list is very useful in applications that require data to be in some order at all times.

Task 8: Write a complete menu driven program to do the following:

- Build a linked list to save a list of names. Name will not exceed 50 characters.
- Write a function `add` to append a new name to the list. The function prototype is given as

```
void add (list *head, char *newname);
```

- Write a function search to look for a given name in the list. If that name is found in list then the function should return true, otherwise, return false.
- Write a main method to test your two functions.

In C language, the boolean type and the boolean literals (true, false) are not defined. We can define these in our program as follow:

```
typedef enum {false = 0, true} boolean;
```

The skeleton of your program should look like the following:

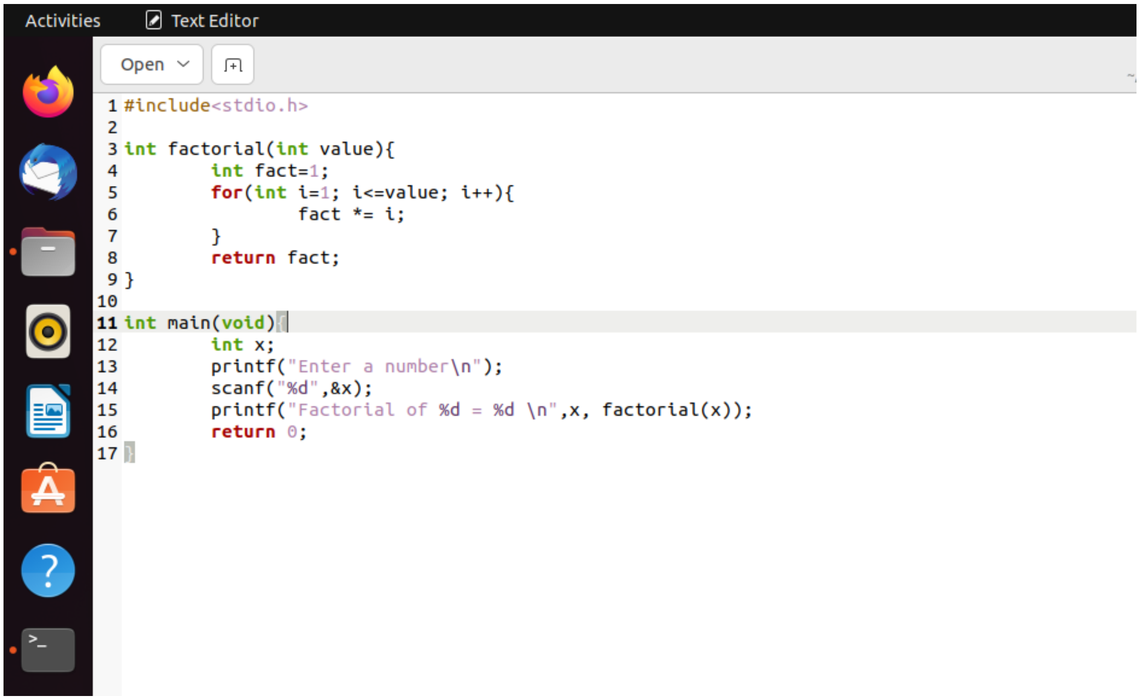
```
#include <stdio.h>
#include <stdlib.h>

typedef struct list {
      ...
      ...
} list;

typedef enum {false=0, true} boolean;
void add (list *, char *);
boolean search (list *, char *);

int main()
{
      ...
      ...
}

void add (list *head, char *newname)
```

```
{
        ...
}
boolean search (list *head, char *name)
{
        ...
}
```

RESULTS AND EXPLANATION:

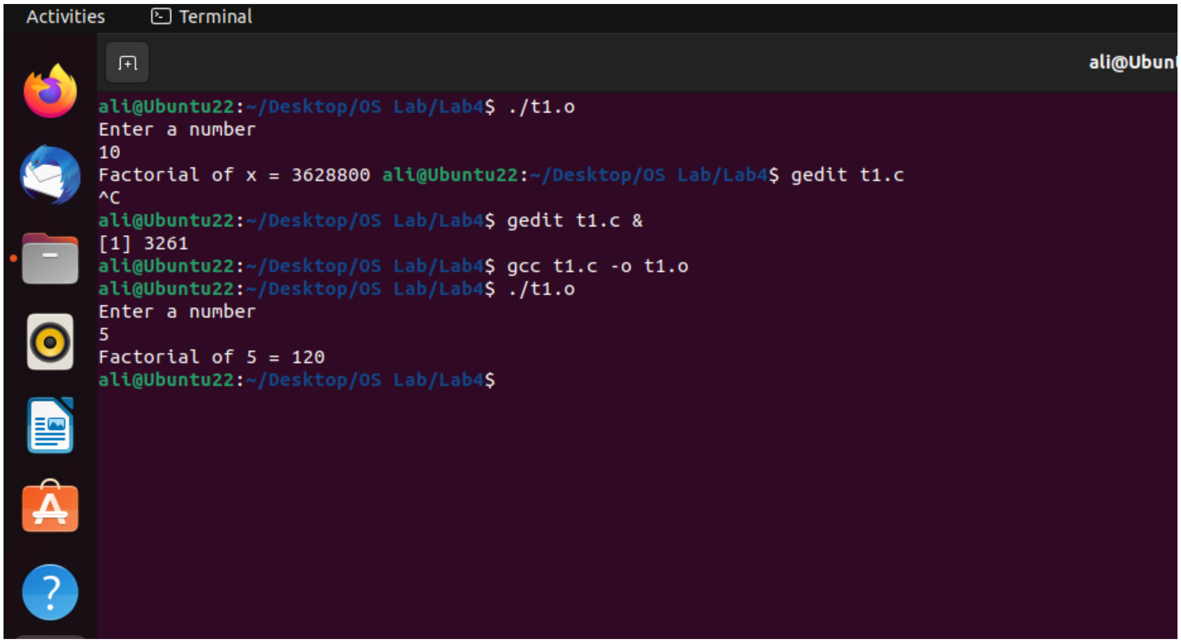
Task # 1: Write a program reads a number from user and finds its factorial using function.

Pass the argument to function by value.



```c
#include<stdio.h>

int factorial(int value){
        int fact=1;
        for(int i=1; i<=value; i++){
                fact *= i;
        }
        return fact;
}

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

Figure 1-1:Code of Task 1

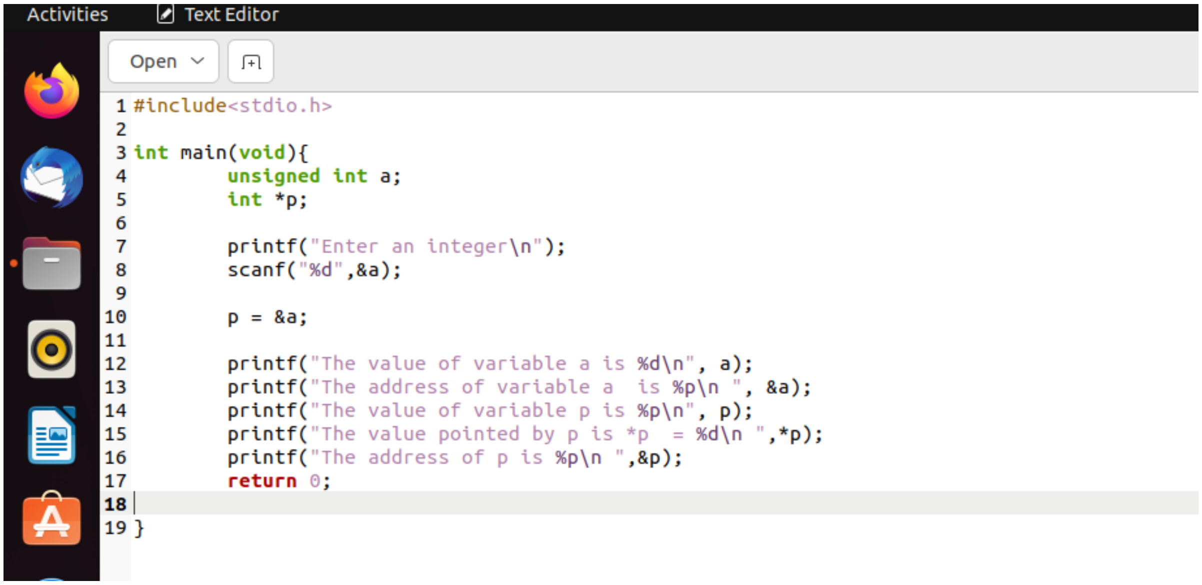


```
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t1.o
Enter a number
10
Factorial of x = 3628800 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c
^C
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c &
[1] 3261
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t1.c -o t1.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t1.o
Enter a number
5
Factorial of 5 = 120
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$
```

Figure 1-2:Output of Task 1

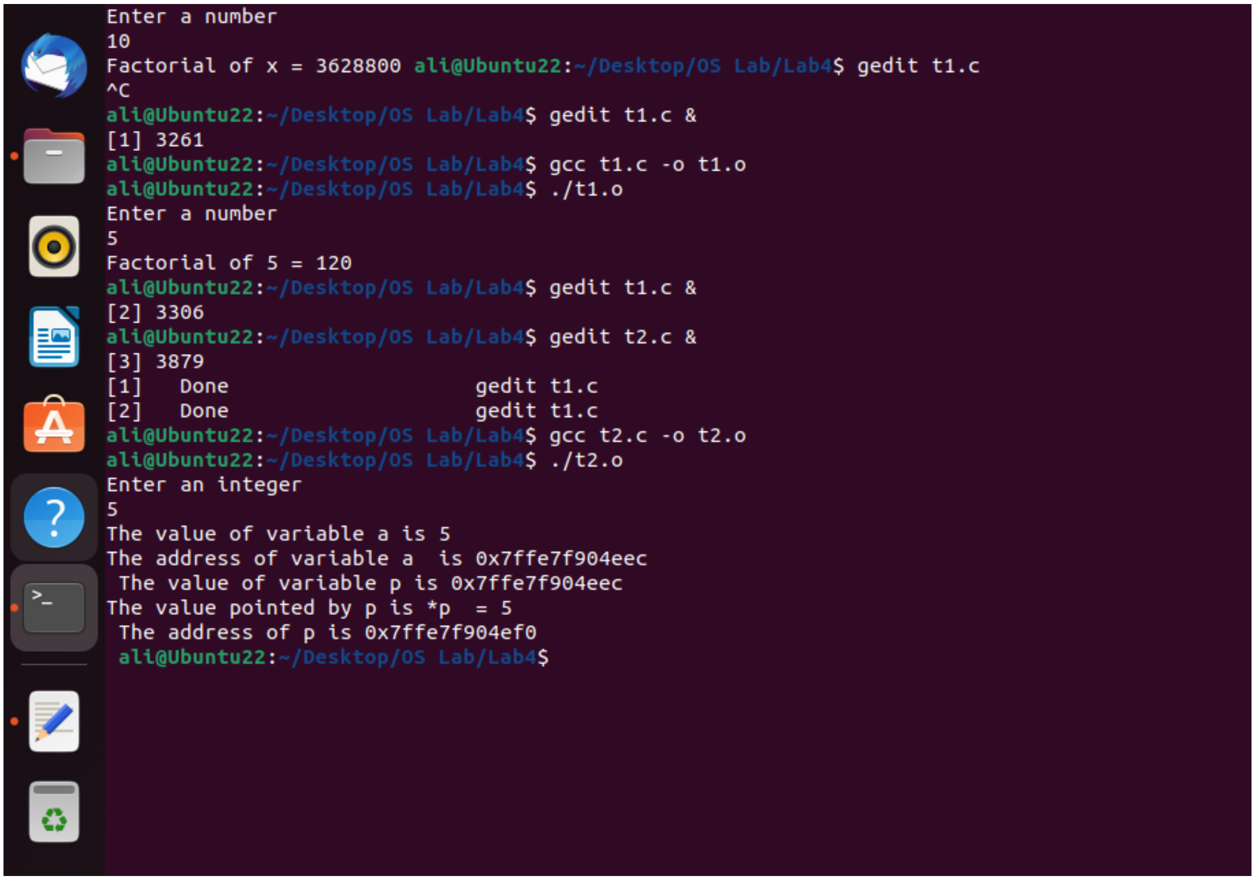
Task 2: The following program demonstrates about the pointer variable, * and & operators.

Run and observe the output.



```c
#include<stdio.h>

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

        printf("Enter an integer\n");
        scanf("%d",&a);

        p = &a;

        printf("The value of variable a is %d\n", a);
        printf("The address of variable a  is %p\n ", &a);
        printf("The value of variable p is %p\n", p);
        printf("The value pointed by p is *p  = %d\n ",*p);
        printf("The address of p is %p\n ",&p);
        return 0;

}
```

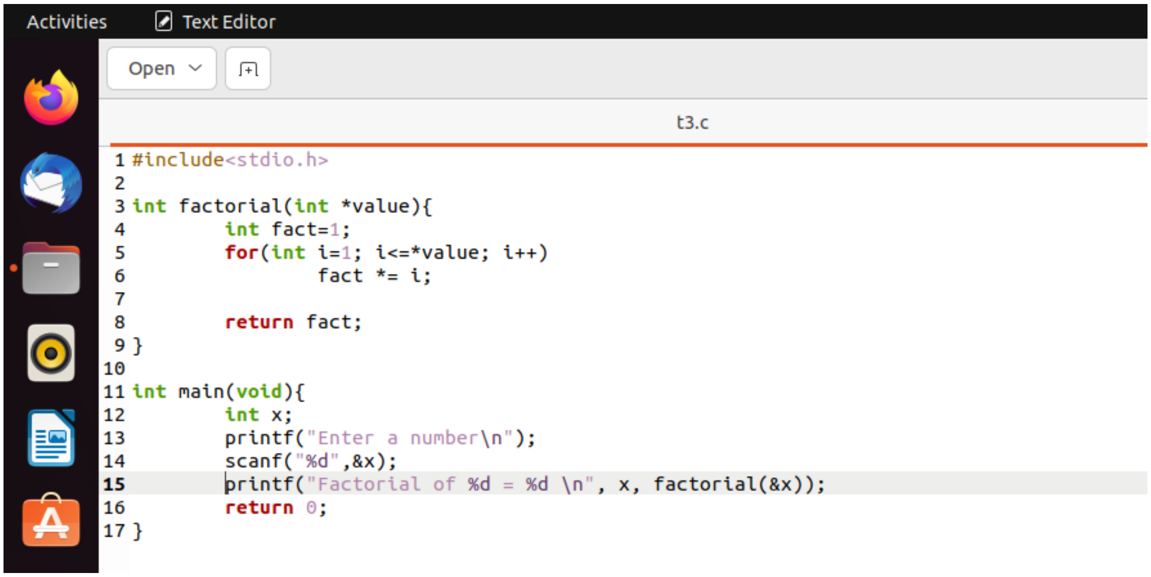
Figure 2-1:Code of Task 2



```
Enter a number
10
Factorial of x = 3628800 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c
^C
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c &
[1] 3261
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t1.c -o t1.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t1.o
Enter a number
5
Factorial of 5 = 120
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c &
[2] 3306
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t2.c &
[3] 3879
[1]   Done                    gedit t1.c
[2]   Done                    gedit t1.c
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t2.c -o t2.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t2.o
Enter an integer
5
The value of variable a is 5
The address of variable a  is 0x7ffe7f904eec
 The value of variable p is 0x7ffe7f904eec
The value pointed by p is *p  = 5
 The address of p is 0x7ffe7f904ef0
 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$
```

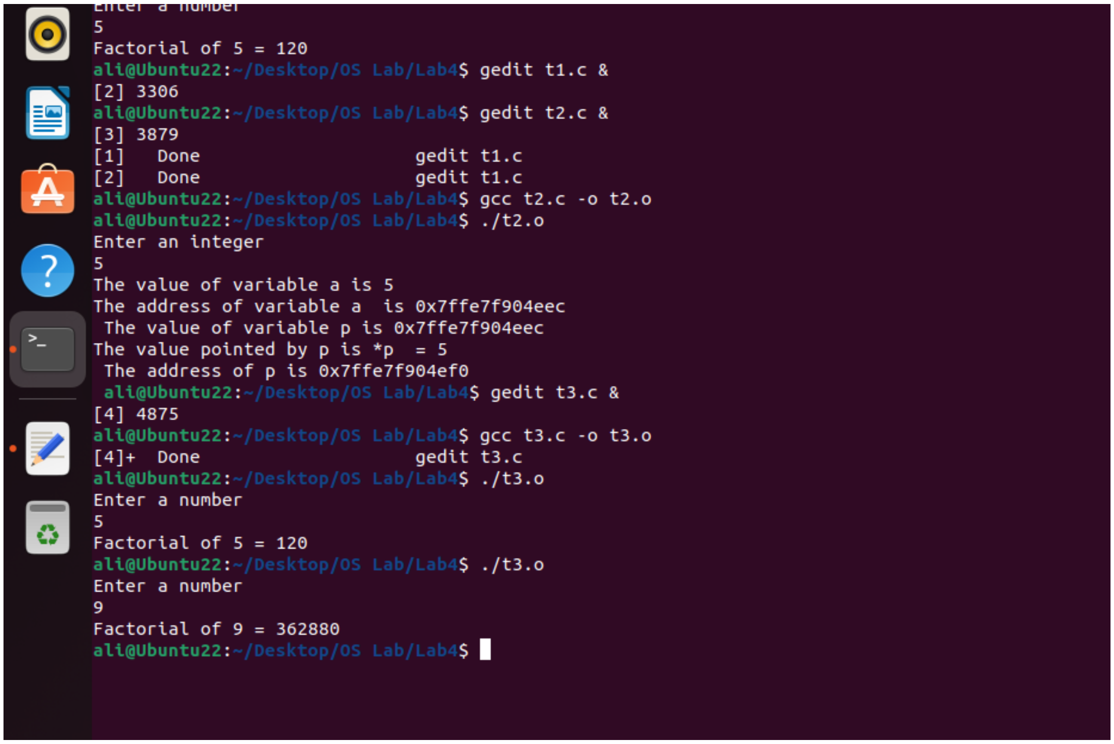
Figure 2-2:Output of Task 2

Task 3: Redo task number 1. The result should be passed by pointer.



```
#include<stdio.h>

int factorial(int *value){
        int fact=1;
        for(int i=1; i<=*value; i++)
                fact *= i;

        return fact;
}

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

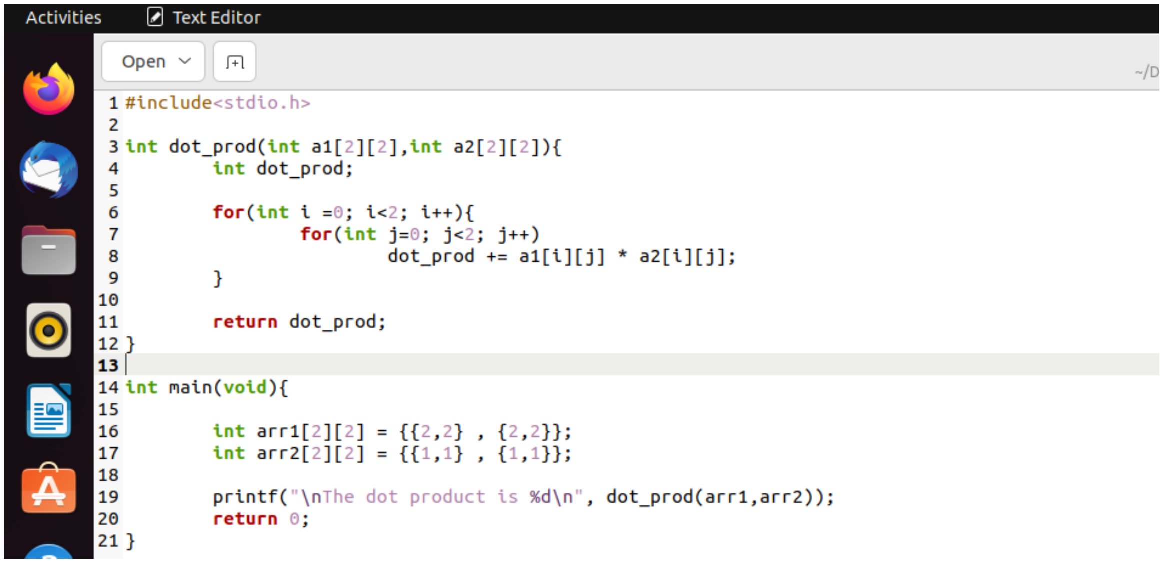
Figure 3-1:Code of Task 3



```
Enter a number
5
Factorial of 5 = 120
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t1.c &
[2] 3306
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t2.c &
[3] 3879
[1]   Done                    gedit t1.c
[2]   Done                    gedit t1.c
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t2.c -o t2.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t2.o
Enter an integer
5
The value of variable a is 5
The address of variable a  is 0x7ffe7f904eec
 The value of variable p is 0x7ffe7f904eec
The value pointed by p is *p  = 5
 The address of p is 0x7ffe7f904ef0
 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t3.c &
[4] 4875
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t3.c -o t3.o
[4]+  Done                    gedit t3.c
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t3.o
Enter a number
5
Factorial of 5 = 120
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t3.o
Enter a number
9
Factorial of 9 = 362880
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ 
```

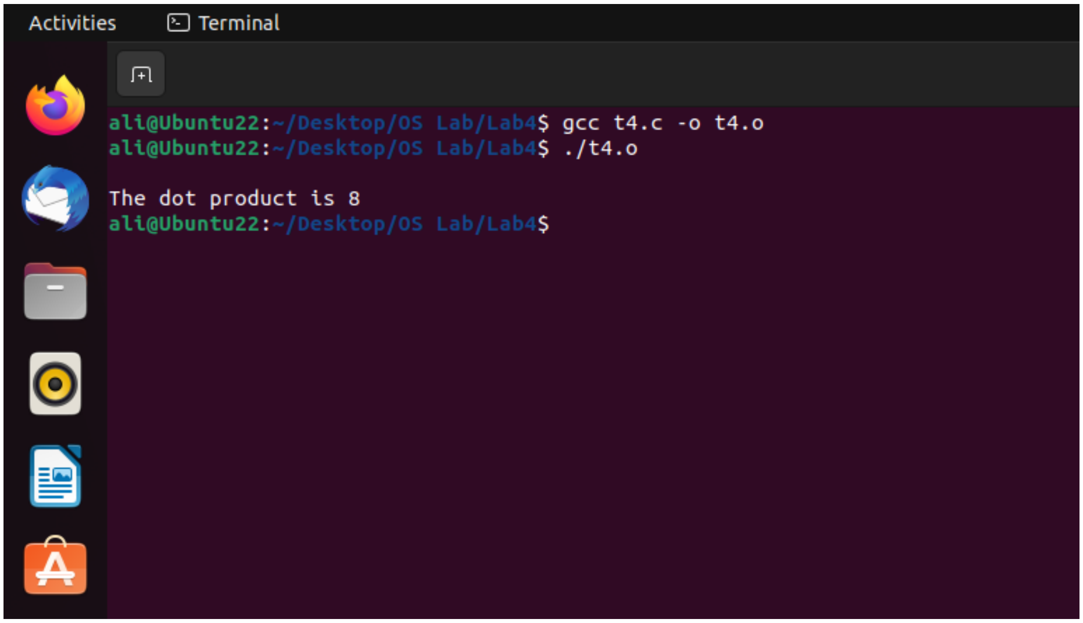
Figure 3-2:Output of Task 3

Task 4: Write a function that calculates the dot product of two dimensional array. Call this function from main() function and display the product.



```c
#include<stdio.h>

int dot_prod(int a1[2][2],int a2[2][2]){
        int dot_prod;

        for(int i =0; i<2; i++){
                for(int j=0; j<2; j++)
                        dot_prod += a1[i][j] * a2[i][j];
        }

        return dot_prod;
}

int main(void){

        int arr1[2][2] = {{2,2} , {2,2}};
        int arr2[2][2] = {{1,1} , {1,1}};

        printf("\nThe dot product is %d\n", dot_prod(arr1,arr2));
        return 0;
}
```

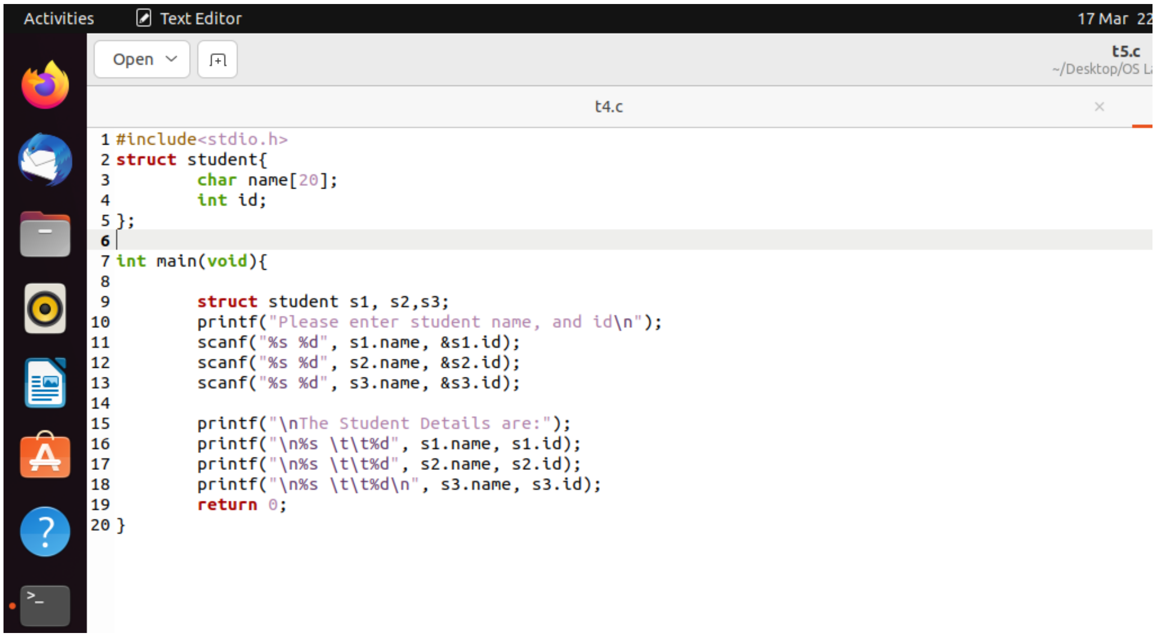
Figure 4-1:Code of Task 4



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t4.c -o t4.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t4.o

The dot product is 8
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$
```

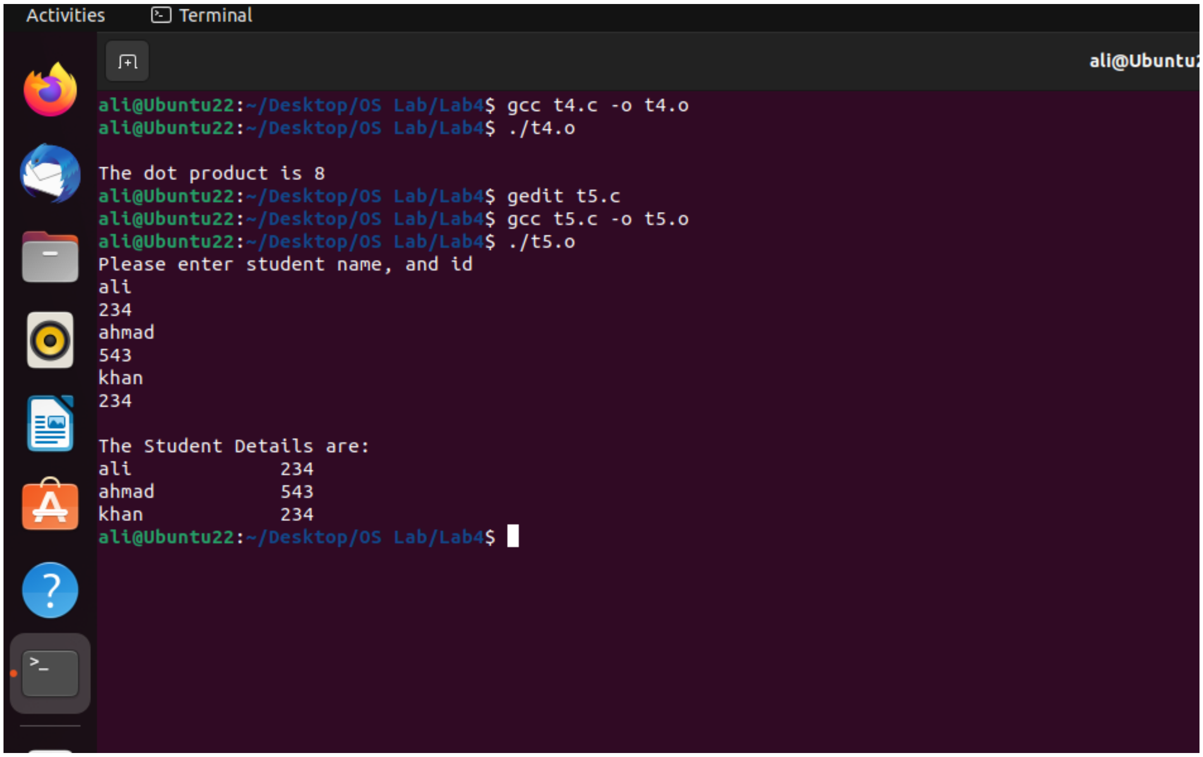
Figure 4-2:Output of Task 4

Task 5: Run the following program and observe the output.



```c
#include<stdio.h>
struct student{
        char name[20];
        int id;
};

int main(void){

        struct student s1, s2,s3;
        printf("Please enter student name, and id\n");
        scanf("%s %d", s1.name, &s1.id);
        scanf("%s %d", s2.name, &s2.id);
        scanf("%s %d", s3.name, &s3.id);

        printf("\nThe Student Details are:");
        printf("\n%s \t\t%d", s1.name, s1.id);
        printf("\n%s \t\t%d", s2.name, s2.id);
        printf("\n%s \t\t%d\n", s3.name, s3.id);
        return 0;
}
```

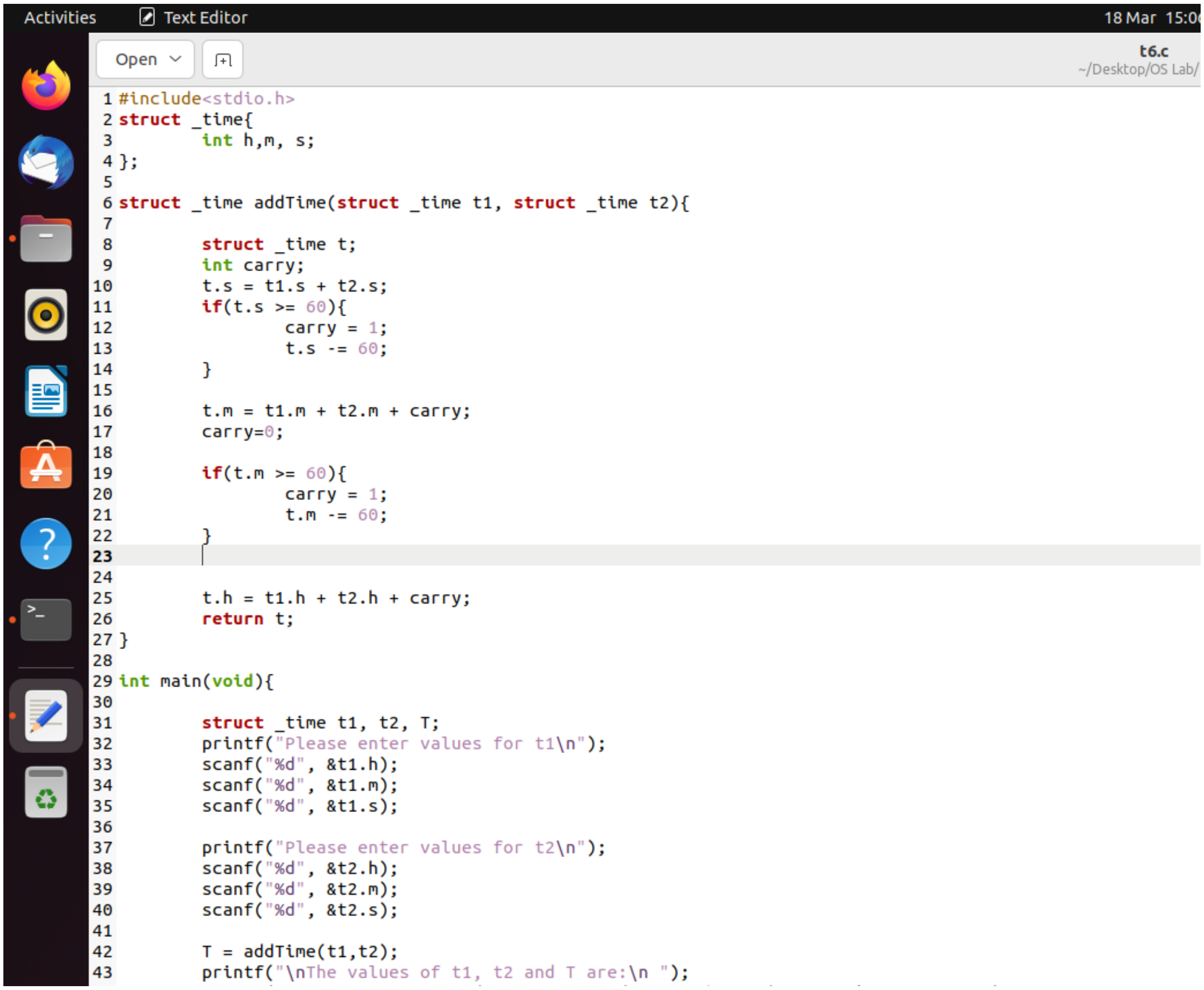
Figure 5-1:Code of Task 5



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t4.c -o t4.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t4.o

The dot product is 8
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t5.c
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t5.c -o t5.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t5.o
Please enter student name, and id
ali
234
ahmad
543
khan
234

The Student Details are:
ali             234
ahmad           543
khan            234
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ 
```

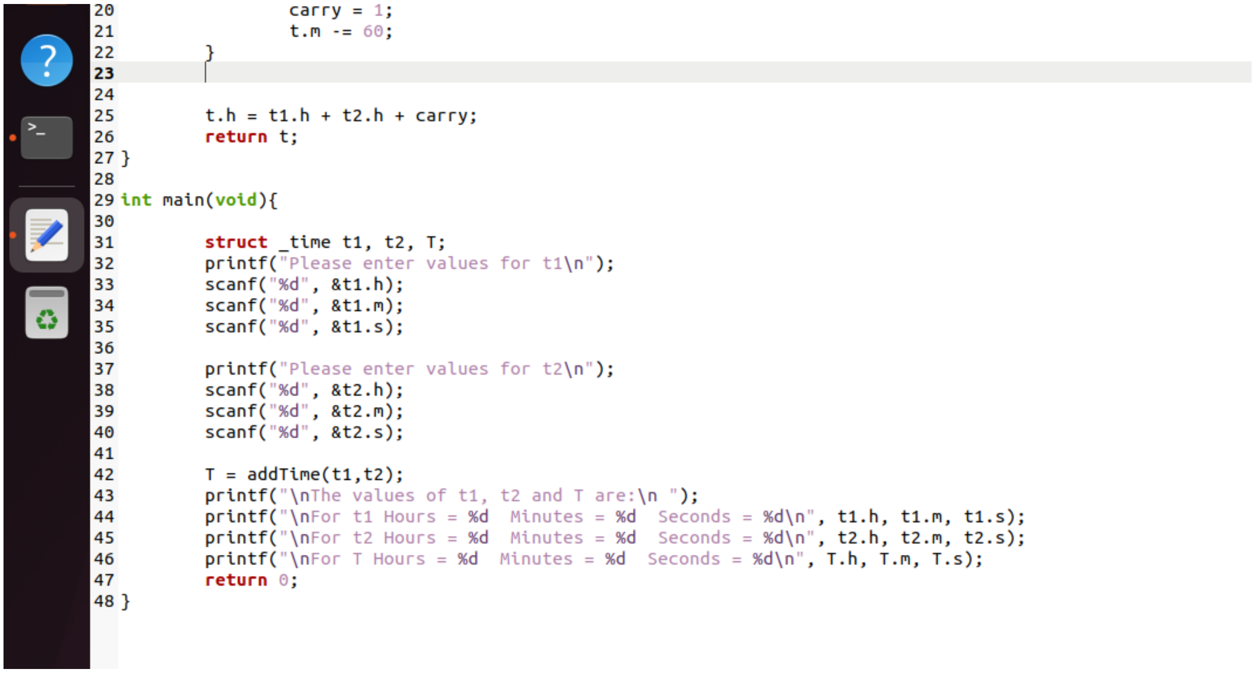
Figure 5-2:Output of Task 5

Task 6: Write a C code to declare “Time” structure that contains hour, minute and seconds as its data members. Write a function that adds two time instances and return the resultant time to the main function.



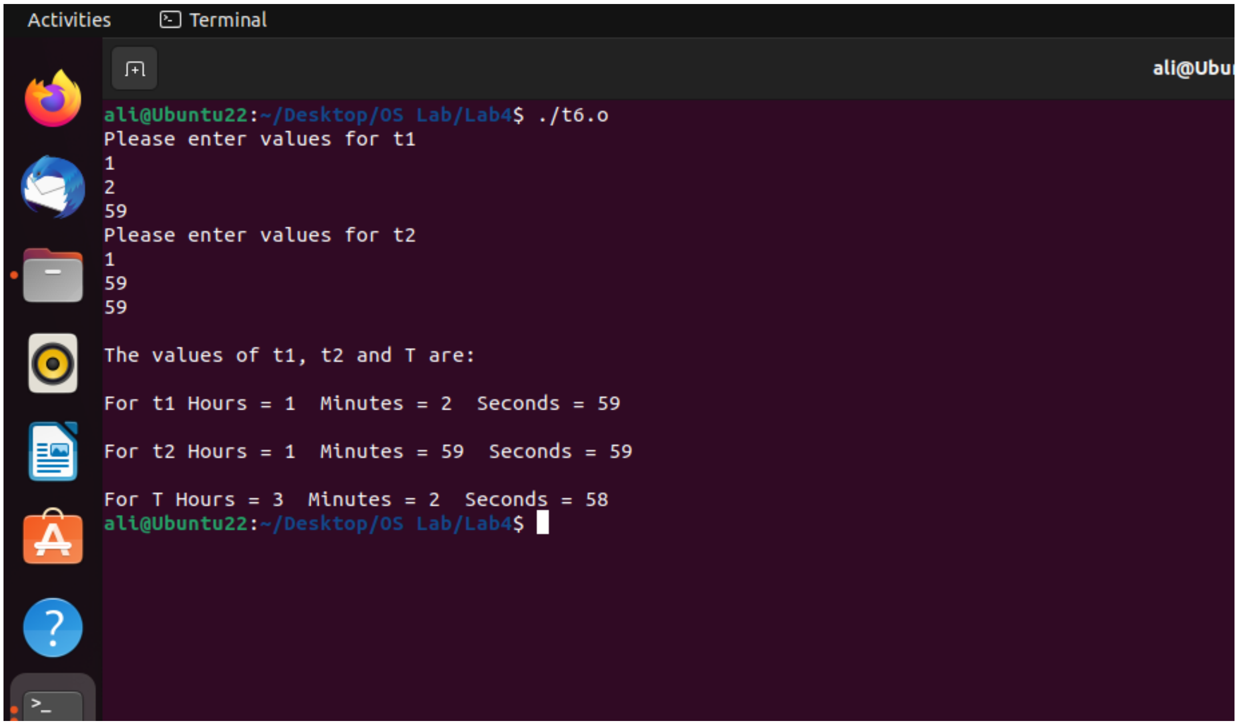
```c
#include<stdio.h>
struct _time{
        int h,m, s;
};

struct _time addTime(struct _time t1, struct _time t2){

        struct _time t;
        int carry;
        t.s = t1.s + t2.s;
        if(t.s >= 60){
                carry = 1;
                t.s -= 60;
        }

        t.m = t1.m + t2.m + carry;
        carry=0;

        if(t.m >= 60){
                carry = 1;
                t.m -= 60;
        }


        t.h = t1.h + t2.h + carry;
        return t;
}

int main(void){

        struct _time t1, t2, T;
        printf("Please enter values for t1\n");
        scanf("%d", &t1.h);
        scanf("%d", &t1.m);
        scanf("%d", &t1.s);

        printf("Please enter values for t2\n");
        scanf("%d", &t2.h);
        scanf("%d", &t2.m);
        scanf("%d", &t2.s);

        T = addTime(t1,t2);
        printf("\nThe values of t1, t2 and T are:\n ");
```

Figure 6-1:Code of Task 6



```
                carry = 1;
                t.m -= 60;
        }


        t.h = t1.h + t2.h + carry;
        return t;
}

int main(void){

        struct _time t1, t2, T;
        printf("Please enter values for t1\n");
        scanf("%d", &t1.h);
        scanf("%d", &t1.m);
        scanf("%d", &t1.s);

        printf("Please enter values for t2\n");
        scanf("%d", &t2.h);
        scanf("%d", &t2.m);
        scanf("%d", &t2.s);

        T = addTime(t1,t2);
        printf("\nThe values of t1, t2 and T are:\n ");
        printf("\nFor t1 Hours = %d  Minutes = %d  Seconds = %d\n", t1.h, t1.m, t1.s);
        printf("\nFor t2 Hours = %d  Minutes = %d  Seconds = %d\n", t2.h, t2.m, t2.s);
        printf("\nFor T Hours = %d  Minutes = %d  Seconds = %d\n", T.h, T.m, T.s);
        return 0;
}
```

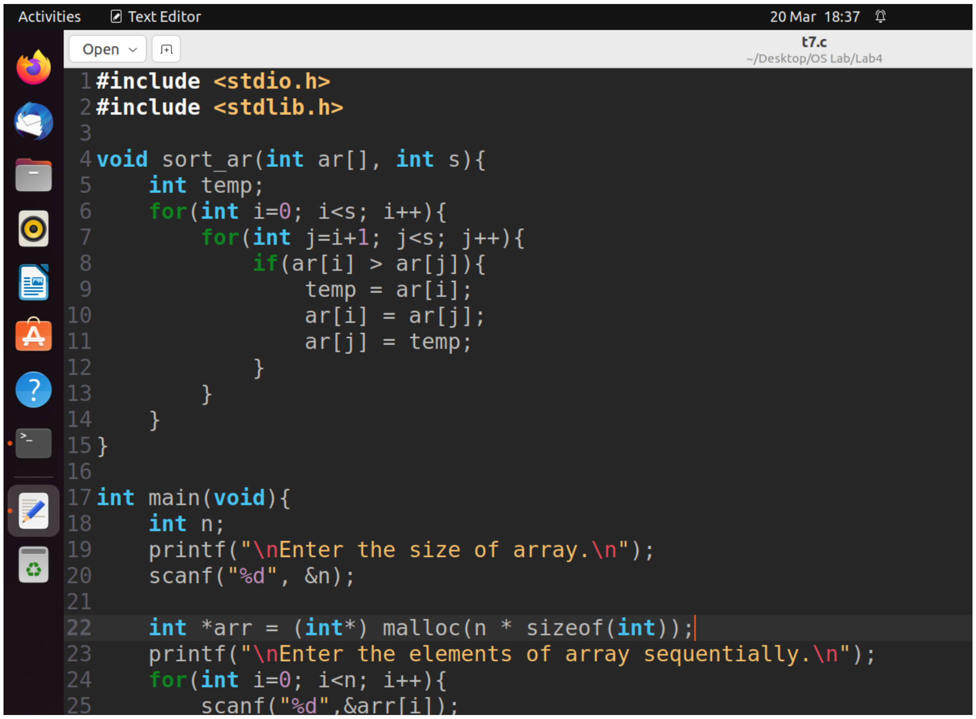
Figure 6-2:Code of Task 6



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t6.o
Please enter values for t1
1
2
59
Please enter values for t2
1
59
59

The values of t1, t2 and T are:

For t1 Hours = 1  Minutes = 2  Seconds = 59

For t2 Hours = 1  Minutes = 59  Seconds = 59

For T Hours = 3  Minutes = 2  Seconds = 58
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ 
```

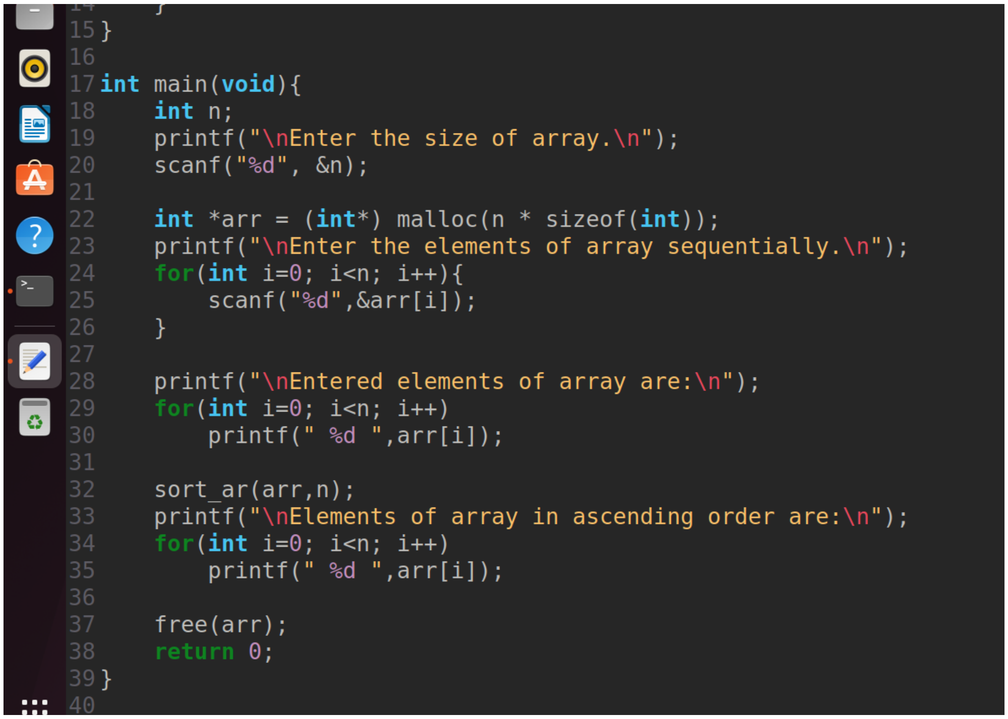
Figure 6-3:Output of Task 6

Task 7: Write a program that takes the size of the array as input from the user, create the array and then take the elements of array as input and sort in ascending order.



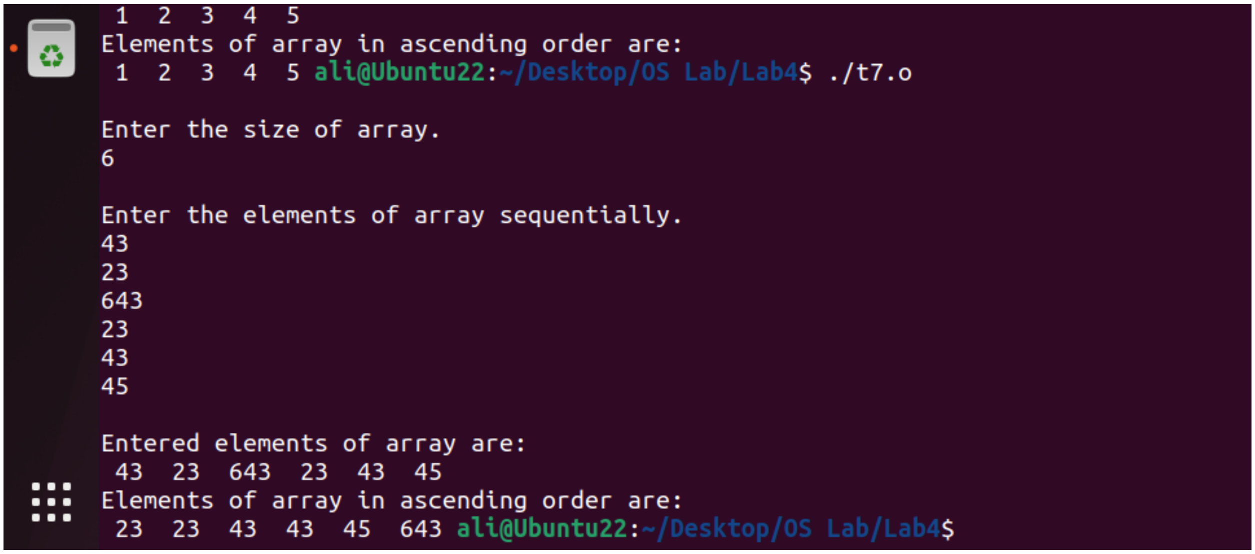
```c
#include <stdio.h>
#include <stdlib.h>

void sort_ar(int ar[], int s){
    int temp;
    for(int i=0; i<s; i++){
        for(int j=i+1; j<s; j++){
            if(ar[i] > ar[j]){
                temp = ar[i];
                ar[i] = ar[j];
                ar[j] = temp;
            }
        }
    }
}

int main(void){
    int n;
    printf("\nEnter the size of array.\n");
    scanf("%d", &n);

    int *arr = (int*) malloc(n * sizeof(int));
    printf("\nEnter the elements of array sequentially.\n");
    for(int i=0; i<n; i++){
        scanf("%d",&arr[i]);
```

Figure 7-1:Code of Task 7



```c
    }
}

int main(void){
    int n;
    printf("\nEnter the size of array.\n");
    scanf("%d", &n);

    int *arr = (int*) malloc(n * sizeof(int));
    printf("\nEnter the elements of array sequentially.\n");
    for(int i=0; i<n; i++){
        scanf("%d",&arr[i]);
    }

    printf("\nEntered elements of array are:\n");
    for(int i=0; i<n; i++)
        printf(" %d ",arr[i]);

    sort_ar(arr,n);
    printf("\nElements of array in ascending order are:\n");
    for(int i=0; i<n; i++)
        printf(" %d ",arr[i]);

    free(arr);
    return 0;
}
```

Figure 7-2:Code of Task 7

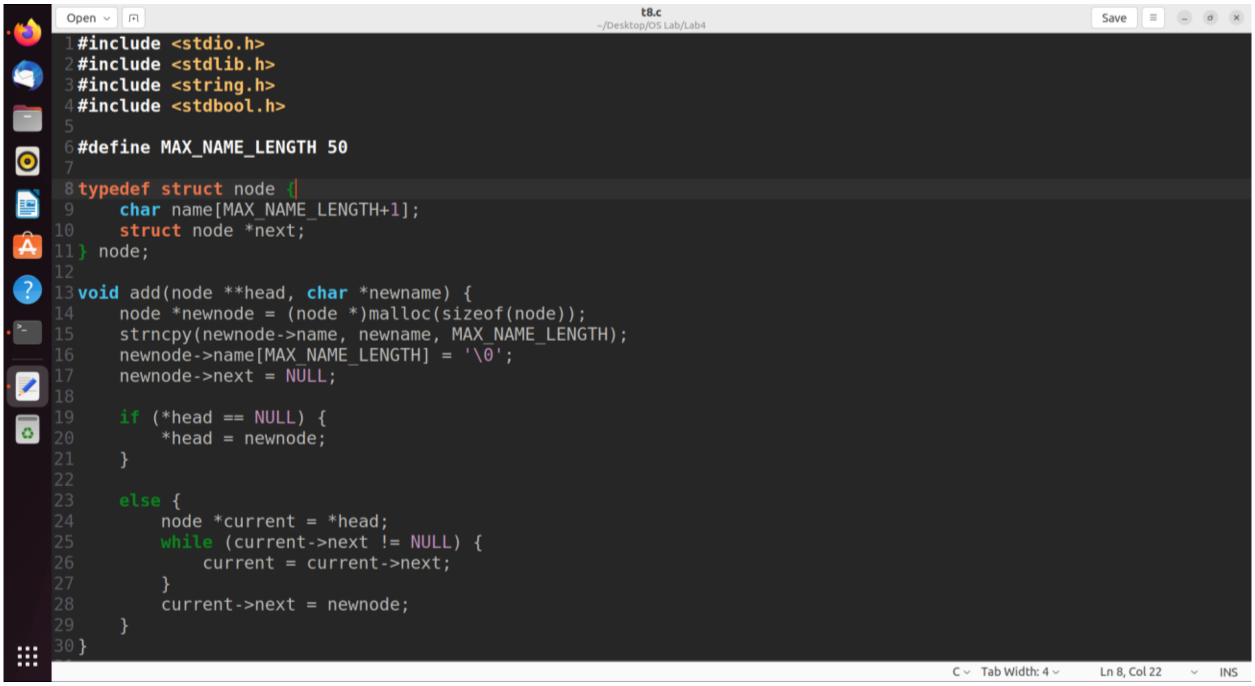


```
 1  2  3  4  5
Elements of array in ascending order are:
 1  2  3  4  5 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t7.o

Enter the size of array.
6

Enter the elements of array sequentially.
43
23
643
23
43
45

Entered elements of array are:
 43  23  643  23  43  45
Elements of array in ascending order are:
 23  23  43  43  45  643 ali@Ubuntu22:~/Desktop/OS Lab/Lab4$
```

Figure 7-3:Output of Task 7

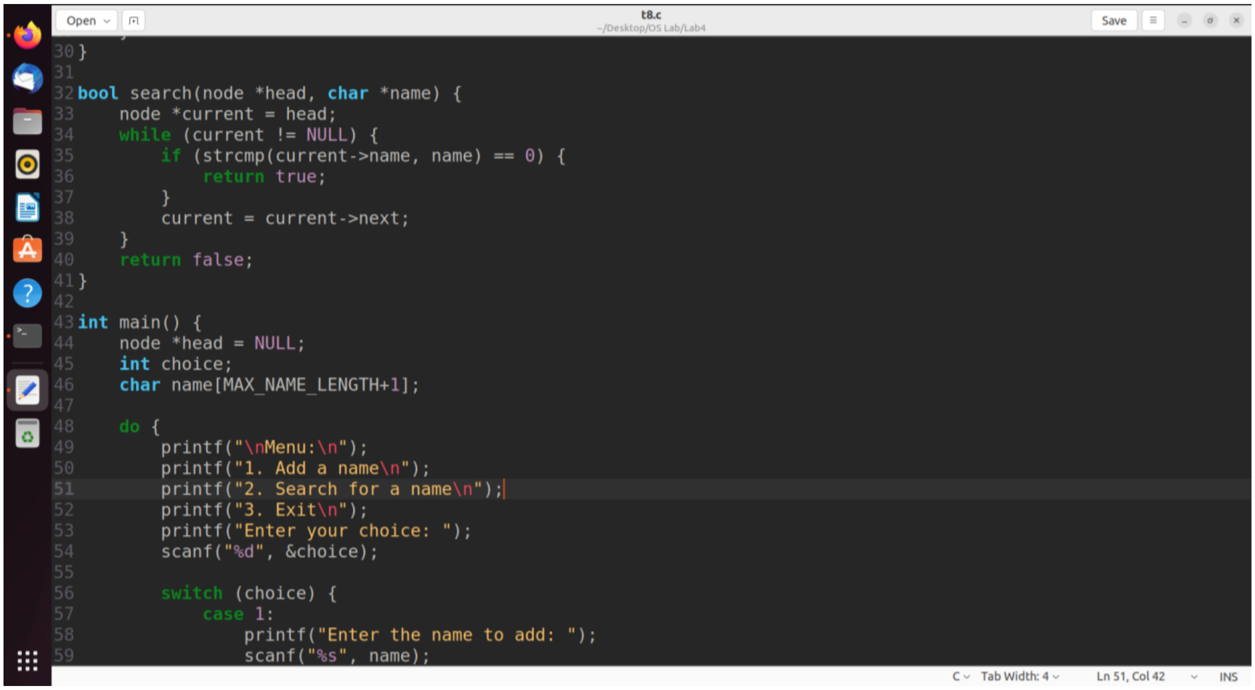
Task 8: Write a complete menu driven program to do the following:

- **Build a linked list to save a list of names. Name will not exceed 50 characters.**
- **Write a function add to append a new name to the list. The function prototype is given as**
  void add (list *head, char *newname);
- **Write a function search to look for a given name in the list. If that name is found in list then the function should return true, otherwise, return false.**
- **Write a main method to test your two functions.**



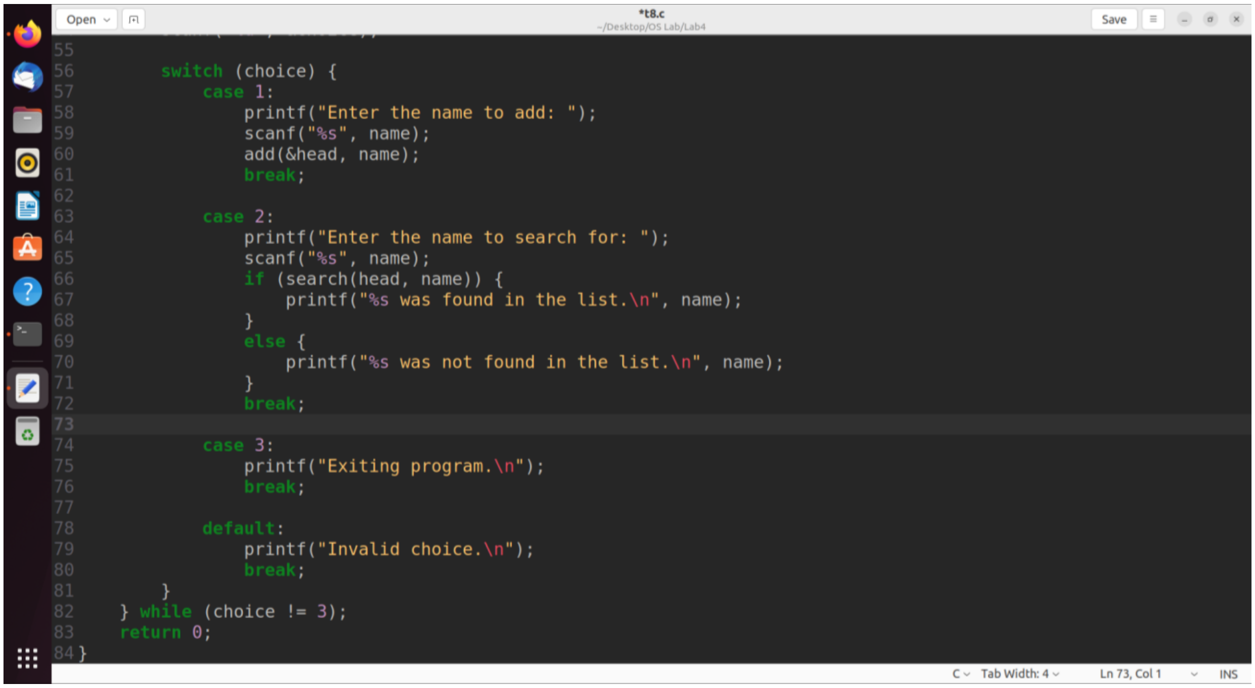
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_NAME_LENGTH 50

typedef struct node {
    char name[MAX_NAME_LENGTH+1];
    struct node *next;
} node;

void add(node **head, char *newname) {
    node *newnode = (node *)malloc(sizeof(node));
    strncpy(newnode->name, newname, MAX_NAME_LENGTH);
    newnode->name[MAX_NAME_LENGTH] = '\0';
    newnode->next = NULL;

    if (*head == NULL) {
        *head = newnode;
    }

    else {
        node *current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newnode;
    }
}
```

Figure 8-1:Code of Task 8



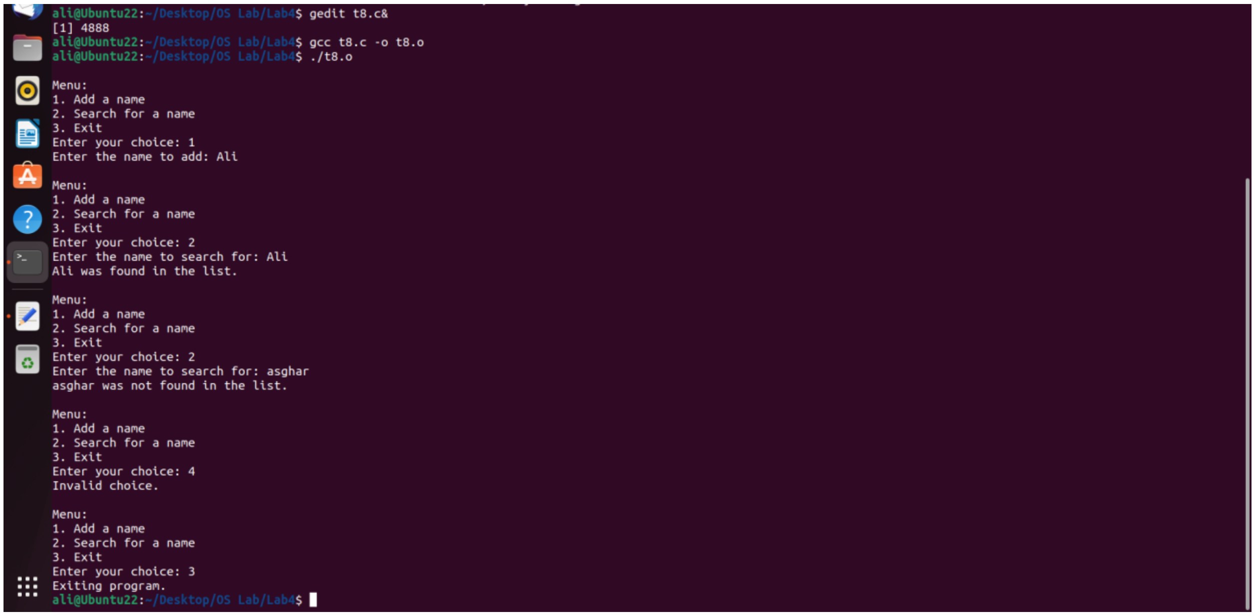
```c
}

bool search(node *head, char *name) {
    node *current = head;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            return true;
        }
        current = current->next;
    }
    return false;
}

int main() {
    node *head = NULL;
    int choice;
    char name[MAX_NAME_LENGTH+1];

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

        switch (choice) {
            case 1:
                printf("Enter the name to add: ");
                scanf("%s", name);
```

Figure 8-2:Code of Task 8



```c

        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 for: ");
                scanf("%s", name);
                if (search(head, name)) {
                    printf("%s was found in the list.\n", name);
                }
                else {
                    printf("%s was not found in the list.\n", name);
                }
                break;

            case 3:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice.\n");
                break;
        }
    } while (choice != 3);
    return 0;
}
```

Figure 8-3:Code of Task 8



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gedit t8.c&
[1] 4888
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ gcc t8.c -o t8.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ ./t8.o

Menu:
1. Add a name
2. Search for a name
3. Exit
Enter your choice: 1
Enter the name to add: Ali

Menu:
1. Add a name
2. Search for a name
3. Exit
Enter your choice: 2
Enter the name to search for: Ali
Ali was found in the list.

Menu:
1. Add a name
2. Search for a name
3. Exit
Enter your choice: 2
Enter the name to search for: asghar
asghar was not found in the list.

Menu:
1. Add a name
2. Search for a name
3. Exit
Enter your choice: 4
Invalid choice.

Menu:
1. Add a name
2. Search for a name
3. Exit
Enter your choice: 3
Exiting program.
ali@Ubuntu22:~/Desktop/OS Lab/Lab4$ 
```

Figure 8-4:Output of Task 8

CONCLUSION:

I concluded that I can compile and run C programs using GCC Compiler. I wrote some basic C programs and then I observed their outputs.