# *Lab Report*

**Course Code :** CIS122L

**Course Title :** Data Structure Lab

| **Submitted by** | **Submitted to** |
|---|---|
| Md. Samiul Islam | Mr. Md. Faruk Hosen |
| ID : 242-16-010 | Lecturer, |
| Section : 20_A | Department of CIS,DIU |

**Date of Submission : 15 March,2025**

# Recursion

**Experiment No: 1**

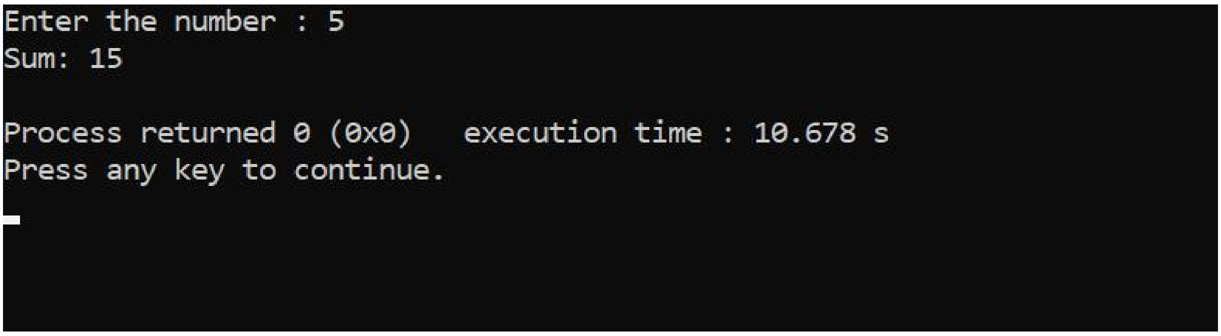
**Experiment name:** Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as a parameter.

## Solution:

## Corresponding code:

```
#include <stdio.h>
int sum(int n) {
        if (n == 0) return 0;
        return n + sum(n - 1);
}
int main() {
        int n;
        printf("Enter the number : ");
        scanf("%d", &n);
        printf("Sum: %d\n", sum(n));
        return 0;
}
```

## Output:



```
Enter the number : 5
Sum: 15

Process returned 0 (0x0)   execution time : 10.678 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Write a recursive function that finds and returns the minimum element in an array.where the array and its size are given as parameters.

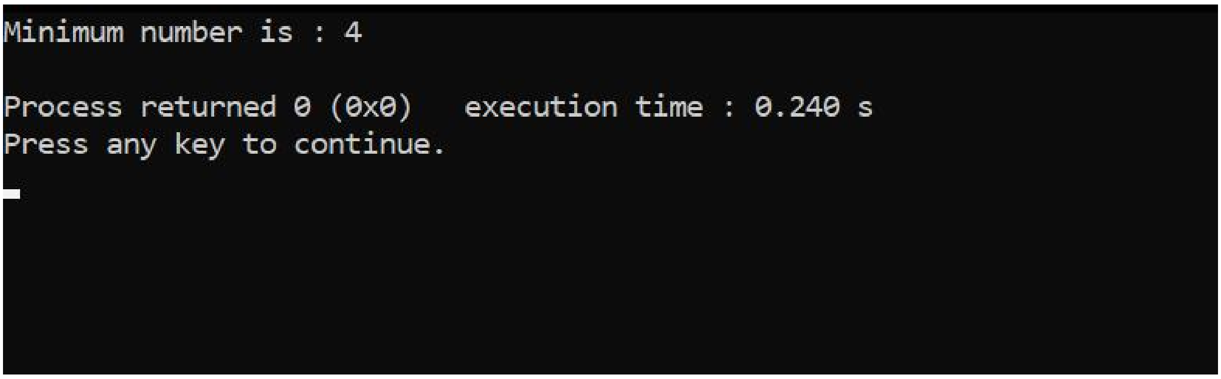
---

## Solution:

## Corresponding code:

```
#include <stdio.h>
int min(int a[ ], int b) {
        if (b == 1) {
                return a[0];
        }
        int c = min(a + 1, b - 1);
        if (a[0] < c) {
                return a[0];
        } else {
                return c;
        }
}

int main() {
        int arr[ ] = {7, 10, , 11,4, 25, 9};
        int s = 6;

        int result = min(arr, s);
        printf("Minimum number is : %d\n", result);

        return 0;
}
```

## Output:



```
Minimum number is : 4

Process returned 0 (0x0)   execution time : 0.240 s
Press any key to continue.
```

# Array

**Experiment No: 1**

**Experiment name:** Write a C program that reads an array and displays the sum of the elements of that array.

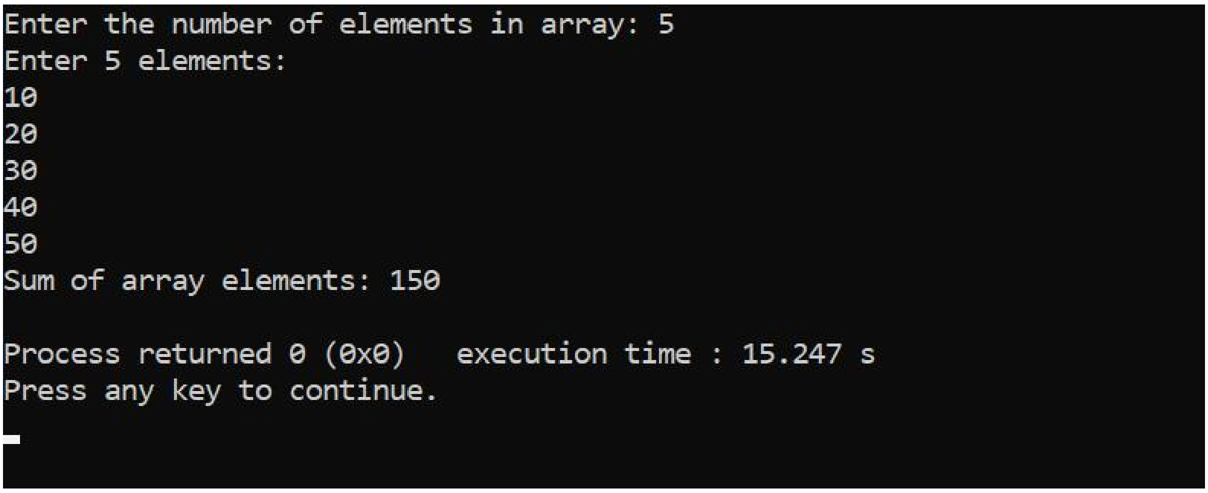
---

## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {

        int n, i, a = 0;
        printf("Enter the number of elements in array: ");
        scanf("%d", &n);

        int arr[n];
        printf("Enter %d elements:\n", n);
        for(i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
                a += arr[i];
        }
        printf("Sum of array elements: %d\n", a);

        return 0;
}
```

## Output:



```
Enter the number of elements in array: 5
Enter 5 elements:
10
20
30
40
50
Sum of array elements: 150

Process returned 0 (0x0)   execution time : 15.247 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Write a C program to insert an element at a specific position in an array.

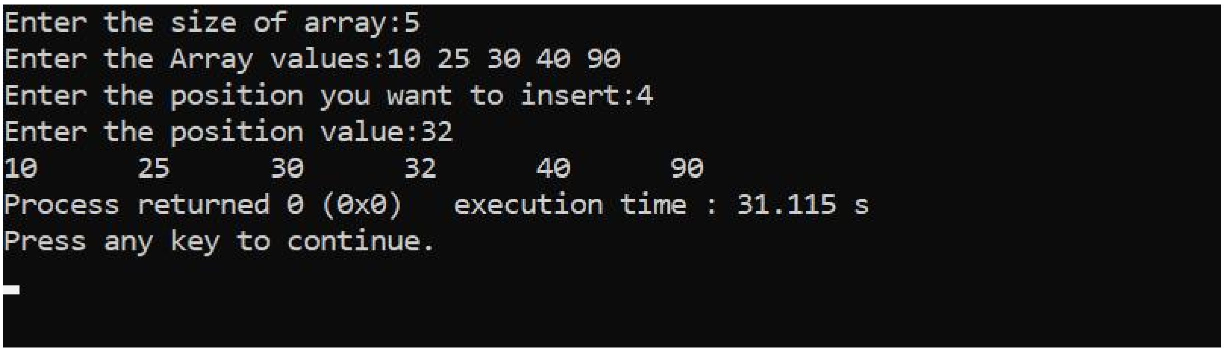
---

## Solution:

## Corresponding code:

```
#include<stdio.h>
int main() {

      int n,i,a[100],pos,val;
      printf("Enter the size of array:");
      scanf("%d",&n);
      printf("Enter the Array values:");

      for(i=1;i<=n;i++)
      {
        scanf("%d", &a[i]);
      }
        printf("Enter the position you want to insert:");
        scanf("%d",&pos);
        printf("Enter the position value:");
        scanf("%d",&val);

      for(i=n; i>=pos; i--)
      {
        a[i+1]=a[i];
      }
        a[pos]=val;
      for(i=1; i<=n+1; i++)
      {
        printf("%d\t",a[i]);
      }
}
```

## Output:



```
Enter the size of array:5
Enter the Array values:10 25 30 40 90
Enter the position you want to insert:4
Enter the position value:32
10      25      30      32      40      90
Process returned 0 (0x0)   execution time : 31.115 s
Press any key to continue.
```

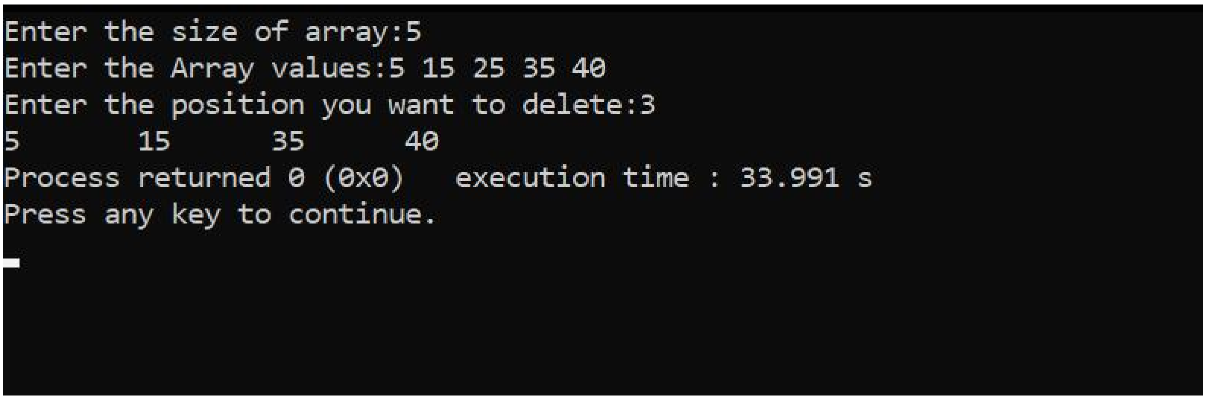
**Experiment No: 3**
**Experiment name:** Write a C program to delete an element from a specific position in an array.

---

## <u>Solution:</u>
## <u>Corresponding code:</u>

```
#include<stdio.h>
int main() {

int n,i,a[100],pos,val;
    printf("Enter the size of array:");
    scanf("%d",&n);
    printf("Enter the Array values:");

    for(i=1;i<=n;i++)
    {
      scanf("%d", &a[i]);
    }
    printf("Enter the position you want to delete:");
      scanf("%d",&pos);
    for(i=pos; i<=n; i++)
    {
      a[i]=a[i+1];
    }
    for(i=1; i<=n-1; i++)
    {
    printf("%d\t",a[i]);
    }
}
```

## <u>Output:</u>



```
Enter the size of array:5
Enter the Array values:5 15 25 35 40
Enter the position you want to delete:3
5       15      35      40
Process returned 0 (0x0)   execution time : 33.991 s
Press any key to continue.
```

**Experiment No: 4**

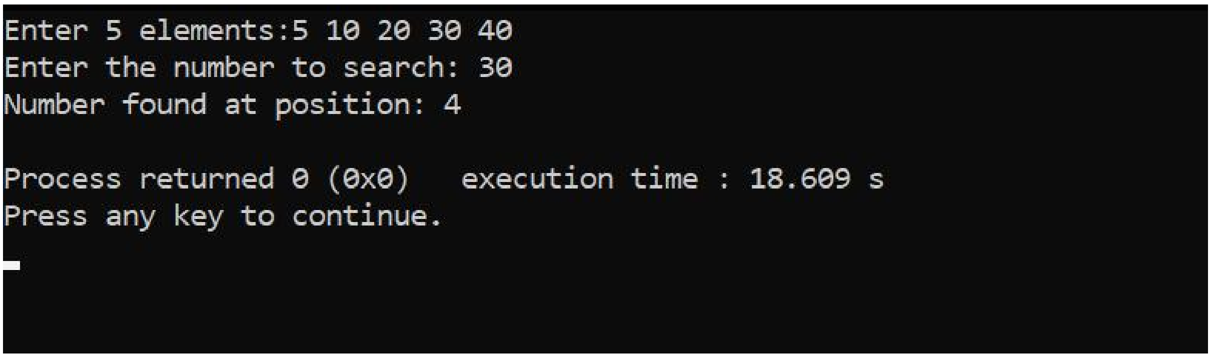
**Experiment name:** Write a C program to search for a given element in an array and display its position.

## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {
  int a[5], i, n, found = 0;
  printf("Enter 5 elements:");
  for (i = 0; i < 5; i++) {
  scanf("%d", &a[i]);
  }
  printf("Enter the number to search: ");
  scanf("%d", &n);
  for (i = 0; i < 5; i++) {
    if (a[i] == n) {
      found = 1;
  printf("Number found at position: %d\n", i + 1);
  break;
  }
  }
  if (!found) {
    printf("Number not found in the array.\n");
  }
return 0;
}
```

## Output:



```
Enter 5 elements:5 10 20 30 40
Enter the number to search: 30
Number found at position: 4

Process returned 0 (0x0)   execution time : 18.609 s
Press any key to continue.
```

## Experiment No: 5

**Experiment name:** Write a C program to update an element at a specific index in an array.

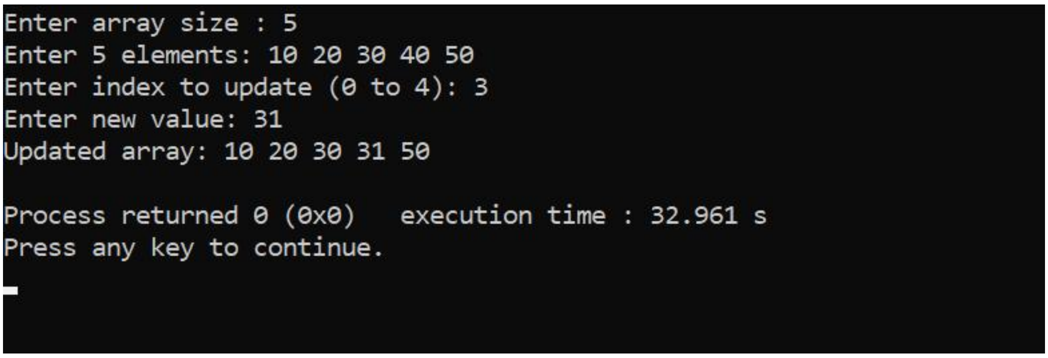
---

**Solution:**

**Corresponding code:**

```
#include <stdio.h>
int main() {
	int arr[100], size, index, newValue, i;
	printf("Enter array size : ");
	scanf("%d", &size);
	printf("Enter %d elements: ", size);
	for(i = 0; i < size; i++) {
		scanf("%d", &arr[i]);
	}
	printf("Enter index to update (0 to %d): ", size - 1);
	scanf("%d", &index);
	printf("Enter new value: ");
	scanf("%d", &newValue);

	if(index < 0 || index >= size) {
		printf("Invalid index! Please enter between 0 and %d\n", size - 1);
		return 1;
	}
	arr[index] = newValue;
	printf("Updated array: ");
	for(i = 0; i < size; i++) {
		printf("%d ", arr[i]);
	}
	printf("\n");

	return 0;
}
```

**Output:**



```
Enter array size : 5
Enter 5 elements: 10 20 30 40 50
Enter index to update (0 to 4): 3
Enter new value: 31
Updated array: 10 20 30 31 50

Process returned 0 (0x0)   execution time : 32.961 s
Press any key to continue.
```

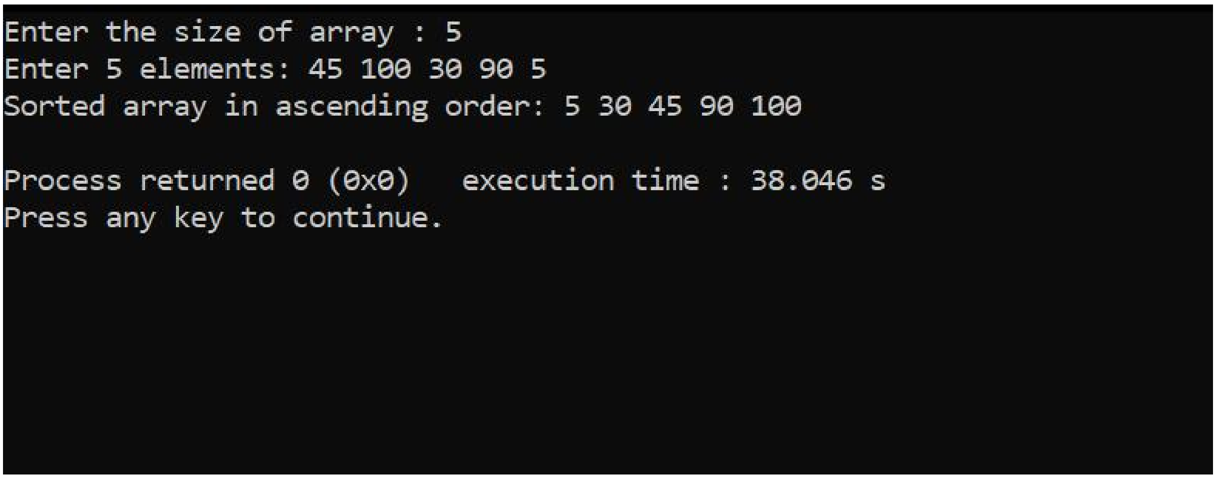
**Experiment No: 6**
**Experiment name:** Write a C program to sort an array in ascending order using bubble sorting algorithm.

---

## Solution:
## Corresponding code:

```
#include <stdio.h>
void bubble_Sort(int arr[], int size) {
        for (int i = 0; i < size - 1; i++) {
                for (int j = 0; j < size - i - 1; j++) {
                        if (arr[j] > arr[j + 1]) {

                                // Swap elements
                                int temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
                }
        }
}
int main() {
        int arr[100], size;

        printf("Enter the size of array : ");
        scanf("%d", &size);

        printf("Enter %d elements: ", size);
        for (int i = 0; i < size; i++) {
                scanf("%d", &arr[i]);
        }

        Bubble_Sort(arr, size);

        printf("Sorted array in ascending order: ");
        for (int i = 0; i < size; i++) {
                printf("%d ", arr[i]);
        }
        printf("\n");

        return 0;
}
```

## Output:



```
Enter the size of array : 5
Enter 5 elements: 45 100 30 90 5
Sorted array in ascending order: 5 30 45 90 100

Process returned 0 (0x0)   execution time : 38.046 s
Press any key to continue.
```

**Experiment No: 7**

**Experiment name:** Write a C program to merge two sorted arrays into a single sorted array

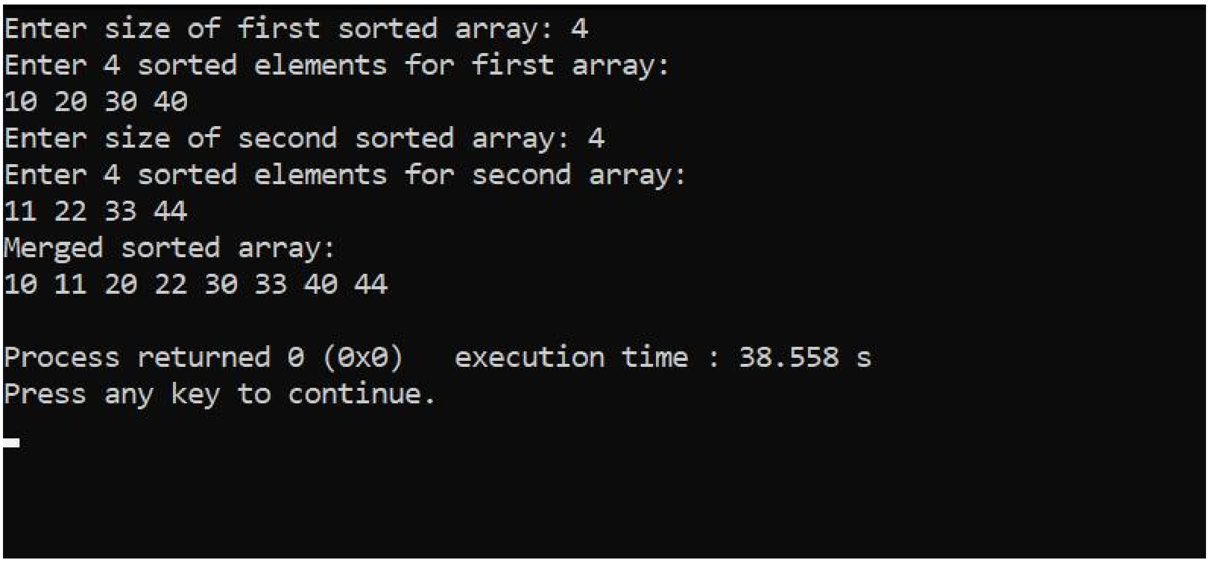
## Solution:

## Corresponding code:

```
#include <stdio.h>
void merge(int arr1[], int size1, int arr2[], int size2, int result[]) {
    int i = 0, j = 0, k = 0;
    while (i < size1 && j < size2) {
        if (arr1[i] < arr2[j]) {
            result[k++] = arr1[i++];
        } else {
            result[k++] = arr2[j++];
        }
    }
    while (i < size1) {
        result[k++] = arr1[i++];
    }
    while (j < size2) {
        result[k++] = arr2[j++];
    }
}

int main() {
    int arr1[100], arr2[100], result[200];
    int size1, size2, i;

    printf("Enter size of first sorted array: ");
```

```
        scanf("%d", &size1);
        printf("Enter %d sorted elements for first array:\n", size1);
        for (i = 0; i < size1; i++) {
                scanf("%d", &arr1[i]);
        }
        printf("Enter size of second sorted array: ");
        scanf("%d", &size2);
        printf("Enter %d sorted elements for second array:\n", size2);
        for (i = 0; i < size2; i++) {
                scanf("%d", &arr2[i]);
        }
      merge(arr1, size1, arr2, size2, result);
        printf("Merged sorted array:\n");
        for (i = 0; i < size1 + size2; i++) {
                printf("%d ", result[i]);
        }
        printf("\n");

        return 0;
}
```

**Output:**



```
Enter size of first sorted array: 4
Enter 4 sorted elements for first array:
10 20 30 40
Enter size of second sorted array: 4
Enter 4 sorted elements for second array:
11 22 33 44
Merged sorted array:
10 11 20 22 30 33 40 44

Process returned 0 (0x0)   execution time : 38.558 s
Press any key to continue.
```

**Experiment No: 8**

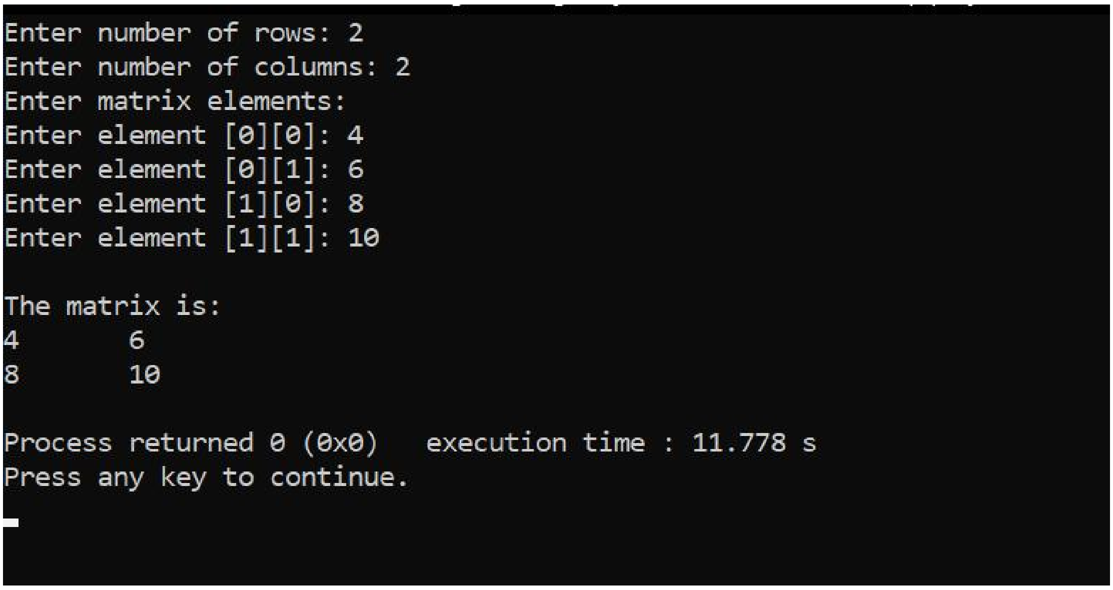
**Experiment name:** Write a C program that read and display a 2D array.

## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {
        int i,j,rows,cols;
        printf("Enter number of rows: ");
        scanf("%d", &rows);
        printf("Enter number of columns: ");
        scanf("%d", &cols);
        int mat[rows][cols];
        printf("Enter matrix elements:\n");
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        printf("Enter element [%d][%d]: ", i, j);
                        scanf("%d", &mat[i][j]);
                }
        }
        printf("\nThe matrix is:\n");
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        printf("%d\t", mat[i][j]);
                }
                printf("\n");
        }
        return 0;
}
```

## Output:



```
Enter number of rows: 2
Enter number of columns: 2
Enter matrix elements:
Enter element [0][0]: 4
Enter element [0][1]: 6
Enter element [1][0]: 8
Enter element [1][1]: 10

The matrix is:
4       6
8       10

Process returned 0 (0x0)   execution time : 11.778 s
Press any key to continue.
```

**Experiment No: 9**

**Experiment name:** Write a C program to read two matrices from the user and compute their sum

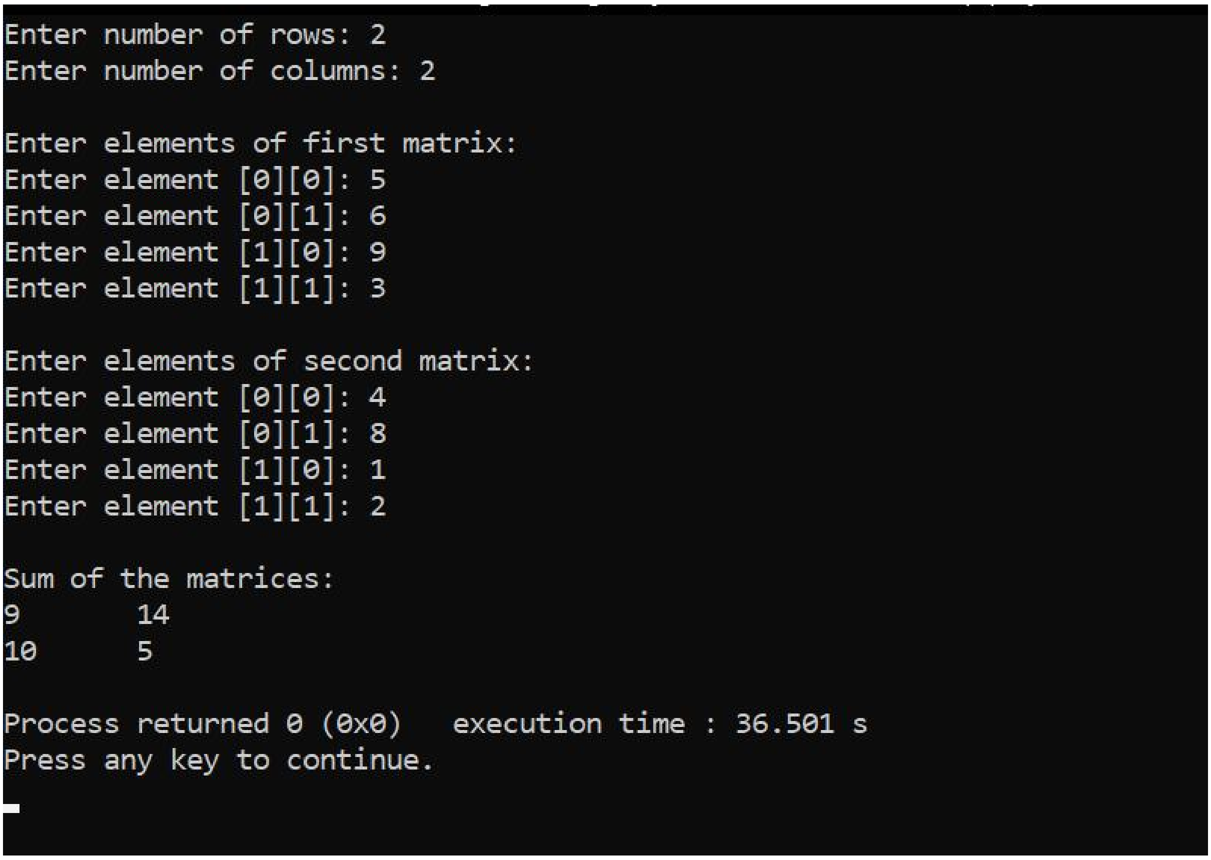
---

## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {
        int i,j,rows,cols;
        printf("Enter number of rows: ");
        scanf("%d", &rows);
        printf("Enter number of columns: ");
        scanf("%d", &cols);

        int mat1[rows][cols], mat2[rows][cols], sum[rows][cols];
        printf("\nEnter elements of first matrix:\n");
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        printf("Enter element [%d][%d]: ", i, j);
                        scanf("%d", &mat1[i][j]);
                }
        }
        printf("\nEnter elements of second matrix:\n");
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        printf("Enter element [%d][%d]: ", i, j);

                        scanf("%d", &mat2[i][j]);
                }
        }
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        sum[i][j] = mat1[i][j] + mat2[i][j];
                }
        }
        printf("\nSum of the matrices:\n");
        for(i = 0; i < rows; i++) {
                for(j = 0; j < cols; j++) {
                        printf("%d\t", sum[i][j]);
                }
                printf("\n");
        }
        return 0;
}
```

## Output:



```
Enter number of rows: 2
Enter number of columns: 2

Enter elements of first matrix:
Enter element [0][0]: 5
Enter element [0][1]: 6
Enter element [1][0]: 9
Enter element [1][1]: 3

Enter elements of second matrix:
Enter element [0][0]: 4
Enter element [0][1]: 8
Enter element [1][0]: 1
Enter element [1][1]: 2

Sum of the matrices:
9       14
10      5

Process returned 0 (0x0)   execution time : 36.501 s
Press any key to continue.
```

**Experiment No: 10**

**Experiment name:** Write a C program to read two matrices from the user and compute their product.

---

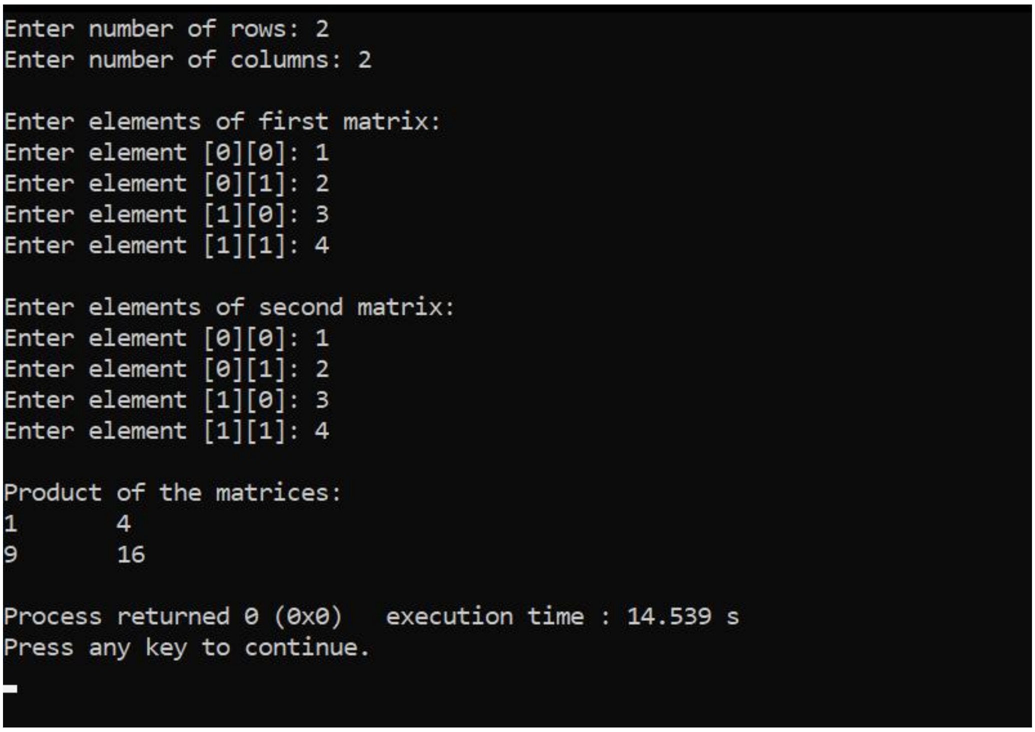
## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {
	int i,j,rows,cols;
	printf("Enter number of rows: ");
	scanf("%d", &rows);
	printf("Enter number of columns: ");
	scanf("%d", &cols);

	int mat1[rows][cols], mat2[rows][cols], product[rows][cols];
	printf("\nEnter elements of first matrix:\n");
	for(i = 0; i < rows; i++) {
		for(j = 0; j < cols; j++) {
```

```
                printf("Enter element [%d][%d]: ", i, j);
                scanf("%d", &mat1[i][j]);
            }
        }
        printf("\nEnter elements of second matrix:\n");
        for(i = 0; i < rows; i++) {
            for(j = 0; j < cols; j++) {
                printf("Enter element [%d][%d]: ", i, j);
                scanf("%d", &mat2[i][j]);
            }
        }
        for(i = 0; i < rows; i++) {
            for(j = 0; j < cols; j++) {
                product[i][j] = mat1[i][j] * mat2[i][j];
            }
        }
        printf("\nProduct of the matrices:\n");
        for(i = 0; i < rows; i++) {
            for(j = 0; j < cols; j++) {
                printf("%d\t", product[i][j]);
            }
            printf("\n");
        }

        return 0;
}
```

**Output:**



```
Enter number of rows: 2
Enter number of columns: 2

Enter elements of first matrix:
Enter element [0][0]: 1
Enter element [0][1]: 2
Enter element [1][0]: 3
Enter element [1][1]: 4

Enter elements of second matrix:
Enter element [0][0]: 1
Enter element [0][1]: 2
Enter element [1][0]: 3
Enter element [1][1]: 4

Product of the matrices:
1       4
9       16

Process returned 0 (0x0)   execution time : 14.539 s
Press any key to continue.
```

# Structure

**Experiment No: 1**

**Experiment name:** Create a structure called "Student" with members name, age, and total marks. Write a C program to input data for two students, display their information, and find the average of total marks.

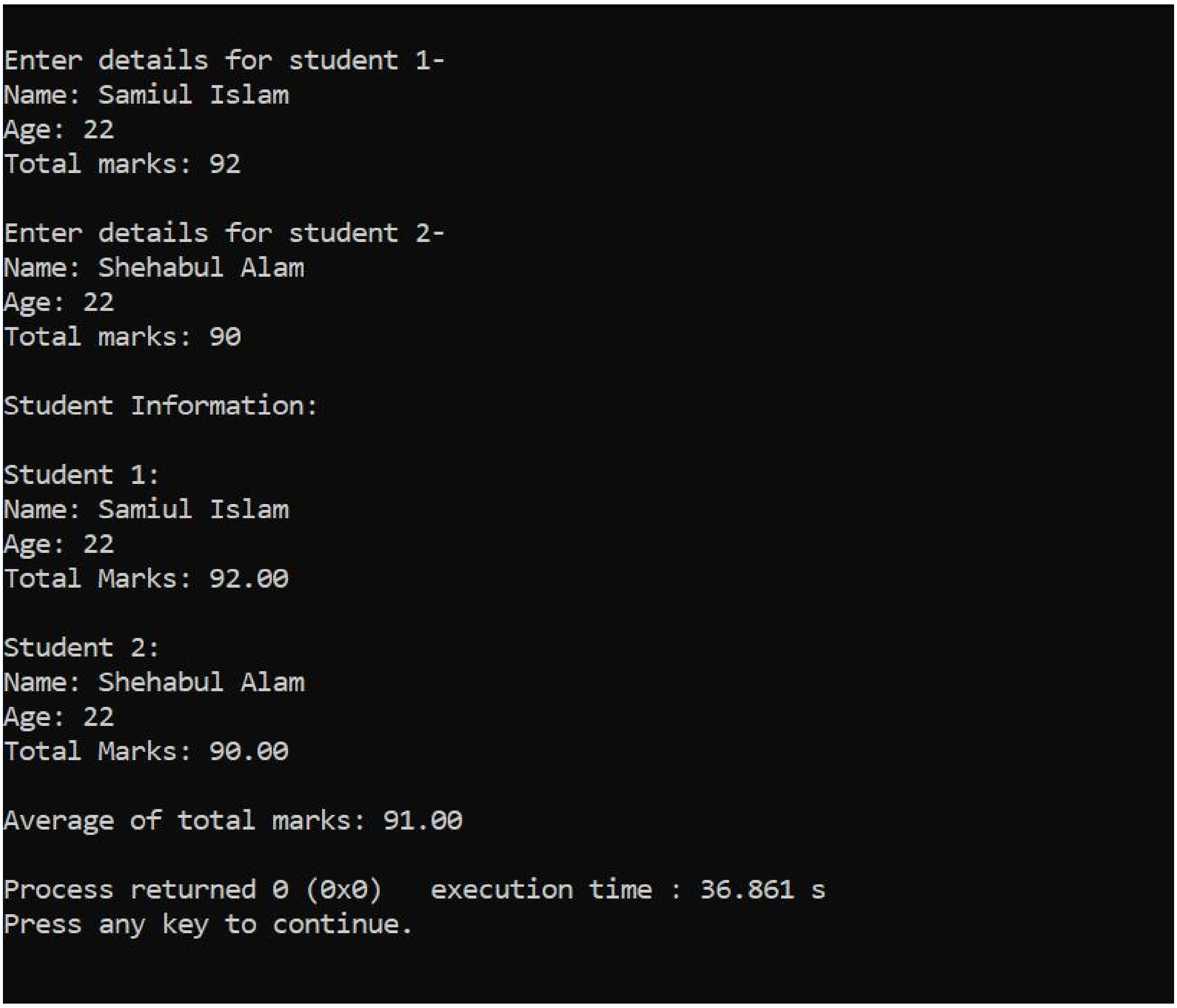
---

## Solution:

## Corresponding code:

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

struct Student {
        char name[50];
        int age;
        float total_marks;
};

int main() {
        struct Student students[2];
        float average = 0;
        for(int i = 0; i < 2; i++) {
                printf("\nEnter details for student %d-\n", i+1);

                printf("Name: ");
                fgets(students[i].name, 50, stdin);
                students[i].name[strcspn(students[i].name, "\n")] = '\0';

                printf("Age: ");
                scanf("%d", &students[i].age);

                printf("Total marks: ");
                scanf("%f", &students[i].total_marks);

                getchar();
        }

        printf("\nStudent Information:\n");
        for(int i = 0; i < 2; i++) {
                printf("\nStudent %d:\n", i+1);
                printf("Name: %s\n", students[i].name);
                printf("Age: %d\n", students[i].age);
                printf("Total Marks: %.2f\n", students[i].total_marks);
                average += students[i].total_marks;
        }

        average /= 2;
        printf("\nAverage of total marks: %.2f\n", average);
        return 0;
}
```

## Output:



```
Enter details for student 1-
Name: Samiul Islam
Age: 22
Total marks: 92

Enter details for student 2-
Name: Shehabul Alam
Age: 22
Total marks: 90

Student Information:

Student 1:
Name: Samiul Islam
Age: 22
Total Marks: 92.00

Student 2:
Name: Shehabul Alam
Age: 22
Total Marks: 90.00

Average of total marks: 91.00

Process returned 0 (0x0)   execution time : 36.861 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Create a structure named Book to store book details like title, author, and price. Write a C program to input details for three books, find the most expensive and the lowest priced books, and display their information.

---

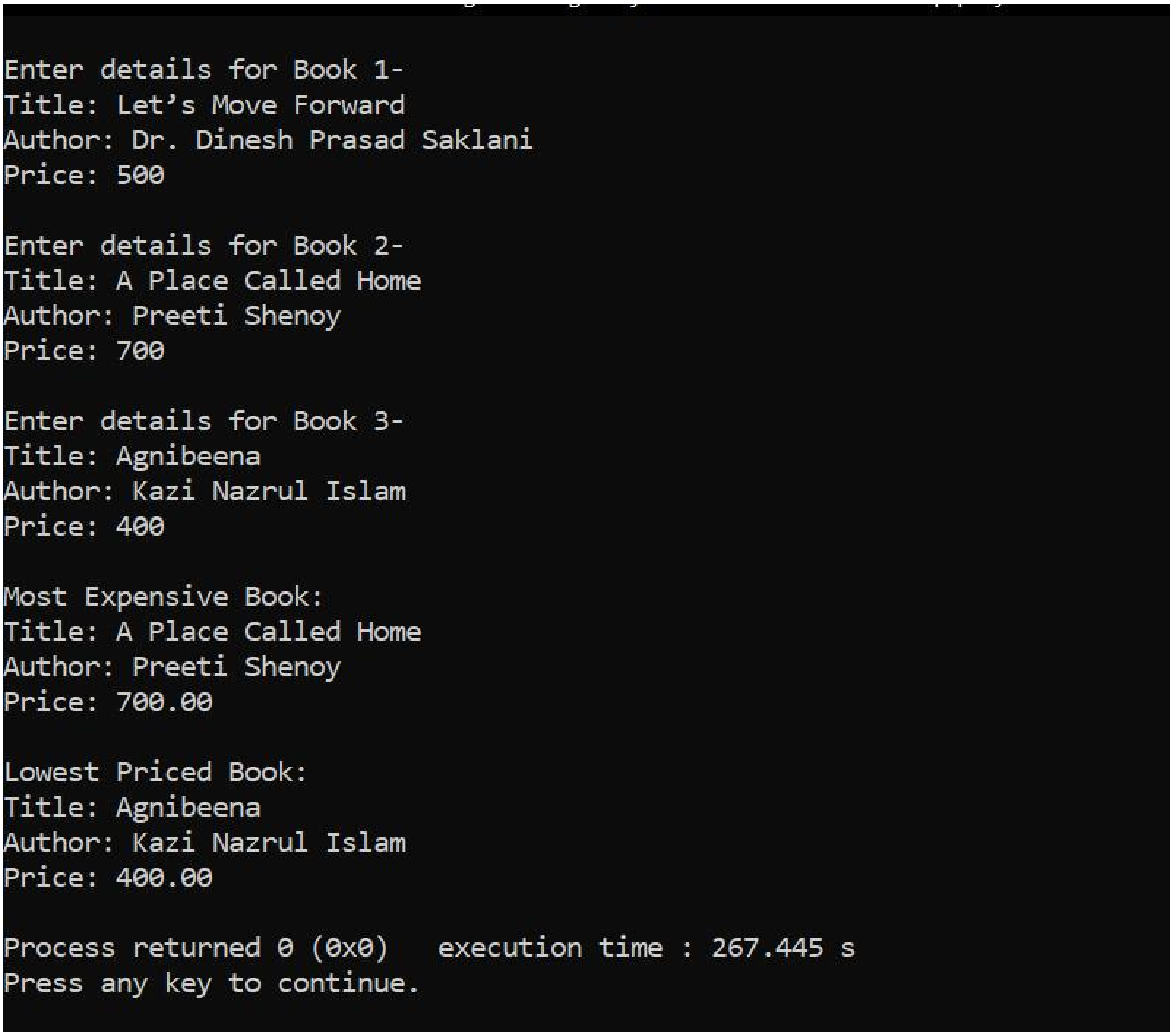
**Solution:**

**Corresponding code:**

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

struct Book {

        char title[100];
        char author[50];
        float price;
};

int main() {
        struct Book books[3];
        int max_index = 0, min_index = 0;
        for(int i = 0; i < 3; i++) {
                printf("\nEnter details for Book %d:\n", i+1);

                printf("Title: ");
                fgets(books[i].title, 100, stdin);
                books[i].title[strcspn(books[i].title, "\n")] = '\0';

                printf("Author: ");
                fgets(books[i].author, 50, stdin);
                books[i].author[strcspn(books[i].author, "\n")] = '\0';

                printf("Price: ");
                scanf("%f", &books[i].price);
                getchar();
        }

        for(int i = 1; i < 3; i++) {
                if(books[i].price > books[max_index].price) {
                        max_index = i;
                }
                if(books[i].price < books[min_index].price) {
                        min_index = i;
                }
        }
```

```
    printf("\nMost Expensive Book:\n");
    printf("Title: %s\n", books[max_index].title);
    printf("Author: %s\n", books[max_index].author);
    printf("Price: %.2f\n", books[max_index].price);

    printf("\nLowest Priced Book:\n");
    printf("Title: %s\n", books[min_index].title);
    printf("Author: %s\n", books[min_index].author);
    printf("Price: %.2f\n", books[min_index].price);

    return 0;
```

**Output:**



```
Enter details for Book 1-
Title: Let’s Move Forward
Author: Dr. Dinesh Prasad Saklani
Price: 500

Enter details for Book 2-
Title: A Place Called Home
Author: Preeti Shenoy
Price: 700

Enter details for Book 3-
Title: Agnibeena
Author: Kazi Nazrul Islam
Price: 400

Most Expensive Book:
Title: A Place Called Home
Author: Preeti Shenoy
Price: 700.00

Lowest Priced Book:
Title: Agnibeena
Author: Kazi Nazrul Islam
Price: 400.00

Process returned 0 (0x0)   execution time : 267.445 s
Press any key to continue.
```

**Experiment No: 3**

**Experiment name:** Write a C program that read some students name, roll and mark, and then sort and display in ascending order using structures.

---

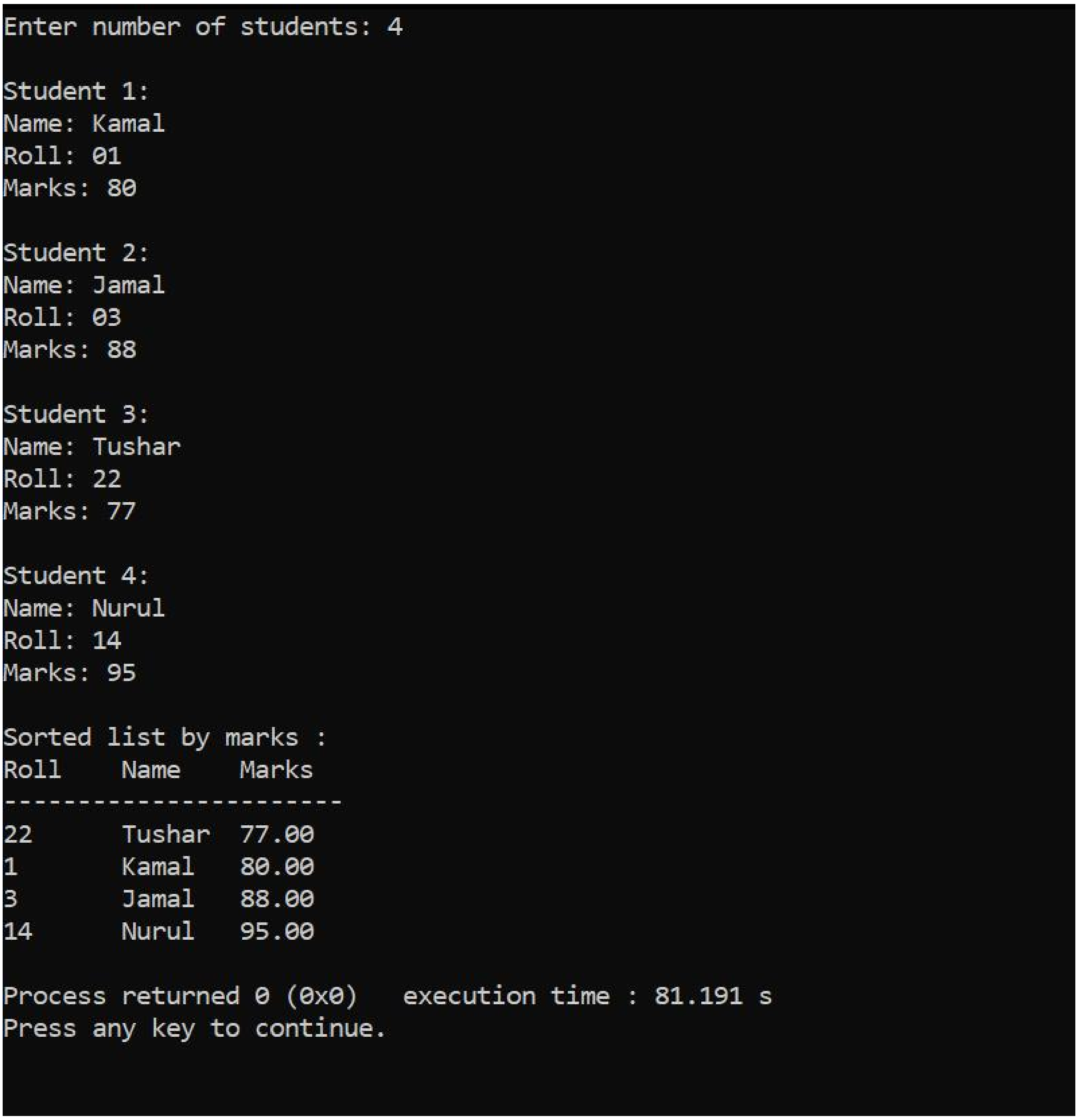
## Solution:

## Corresponding code:

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

struct Student {
        char name[50];
        int roll;
        float marks;
};

int main() {
        struct Student s[100], temp;
        int n, i, j;

        printf("Enter number of students: ");
        scanf("%d", &n);

        for(i = 0; i < n; i++) {
                printf("\nStudent %d:\n", i+1);
                printf("Name: ");
                scanf("%s", s[i].name);
                printf("Roll: ");
                scanf("%d", &s[i].roll);
                printf("Marks: ");
                scanf("%f", &s[i].marks);
        }

        for(i = 0; i < n; i++) {
                for(j = i+1; j < n; j++) {

                        if(s[i].marks > s[j].marks) {
                                temp = s[i];
                                s[i] = s[j];
                                s[j] = temp;
                        }
                }
        }

        printf("\nSorted list by marks :\n");
```

```
    printf("Roll\tName\tMarks\n");
    printf("-----------------------\n");
    for(i = 0; i < n; i++) {
        printf("%d\t%s\t%.2f\n", s[i].roll, s[i].name, s[i].marks);
    }

    return 0;
}
```

## Output:



```
Enter number of students: 4

Student 1:
Name: Kamal
Roll: 01
Marks: 80

Student 2:
Name: Jamal
Roll: 03
Marks: 88

Student 3:
Name: Tushar
Roll: 22
Marks: 77

Student 4:
Name: Nurul
Roll: 14
Marks: 95

Sorted list by marks :
Roll    Name    Marks
-----------------------
22      Tushar  77.00
1       Kamal   80.00
3       Jamal   88.00
14      Nurul   95.00

Process returned 0 (0x0)   execution time : 81.191 s
Press any key to continue.
```

## Experiment No: 4

**Experiment name:** Create a structure named "Employee" to store employee details such as employee ID, name, and salary. Write a program to input data for three employees, find the highest salary employee, and display their information.

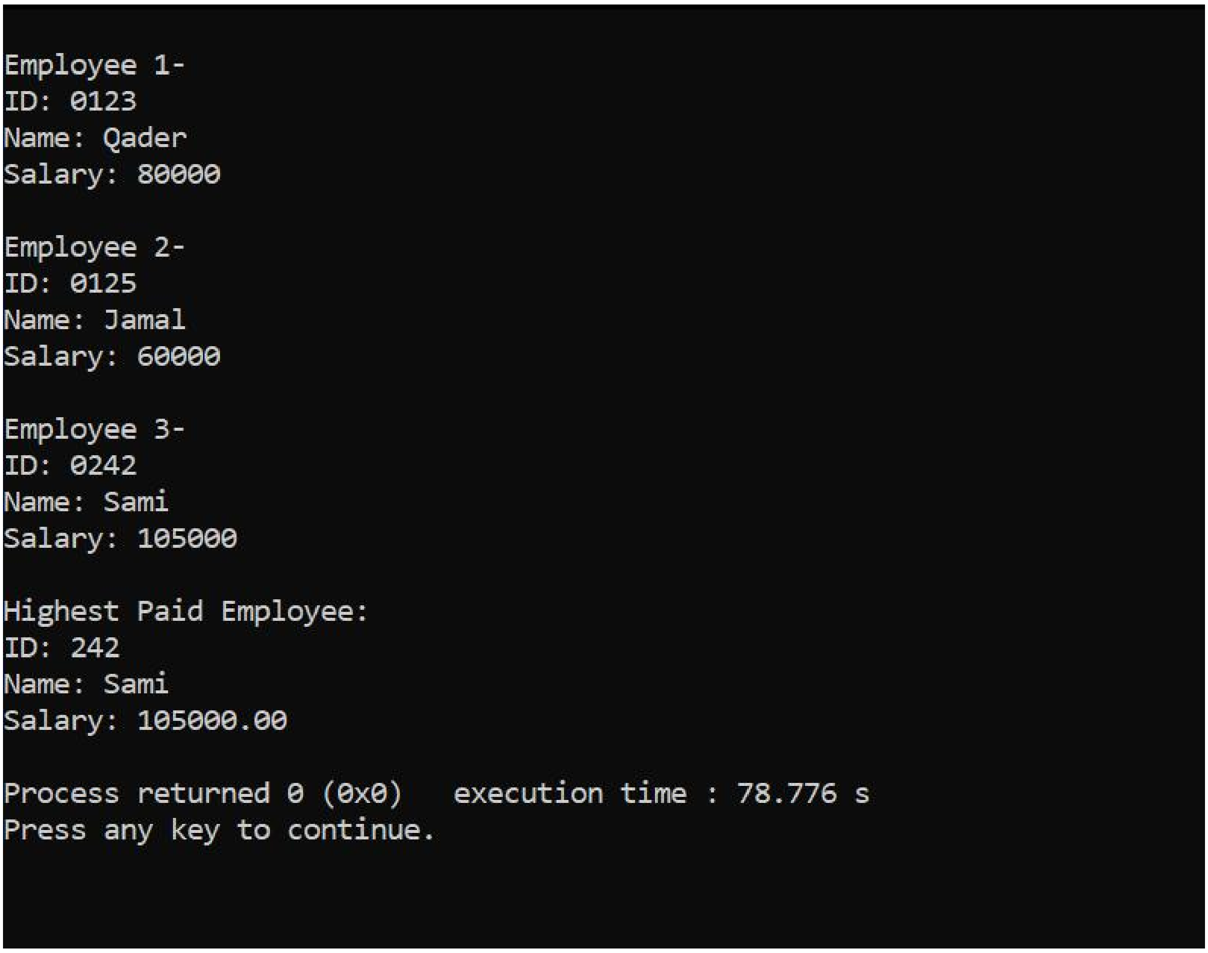
---

**Solution:**

**Corresponding code:**

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

struct emp {
        int id;
        char name[50];
        float sal;
};

int main() {
        struct emp e[3];
        int h = 0, i;
        char temp[50];

        for(i = 0; i < 3; i++) {
                printf("\nEmployee %d:\n", i+1);

                printf("ID: ");
                while(scanf("%d", &e[i].id) != 1) {
                        printf("Invalid ID. Enter numbers only: ");
                        while(getchar() != '\n');
                }

                printf("Name: ");
                while(getchar() != '\n');
                fgets(e[i].name, 50, stdin);
                e[i].name[strcspn(e[i].name, "\n")] = 0;

                printf("Salary: ");
                while(scanf("%f", &e[i].sal) != 1) {
                        printf("Invalid salary. Enter numbers only: ");
                        while(getchar() != '\n');
                }

                if(e[i].sal > e[h].sal) h = i;
        }
        printf("\nHighest Paid Employee:\n");
        printf("ID: %d\nName: %s\nSalary: %.2f\n", e[h].id, e[h].name, e[h].sal);

        return 0;
}
```

**Output:**



```
Employee 1-
ID: 0123
Name: Qader
Salary: 80000

Employee 2-
ID: 0125
Name: Jamal
Salary: 60000

Employee 3-
ID: 0242
Name: Sami
Salary: 105000

Highest Paid Employee:
ID: 242
Name: Sami
Salary: 105000.00

Process returned 0 (0x0)   execution time : 78.776 s
Press any key to continue.
```

# Pointer

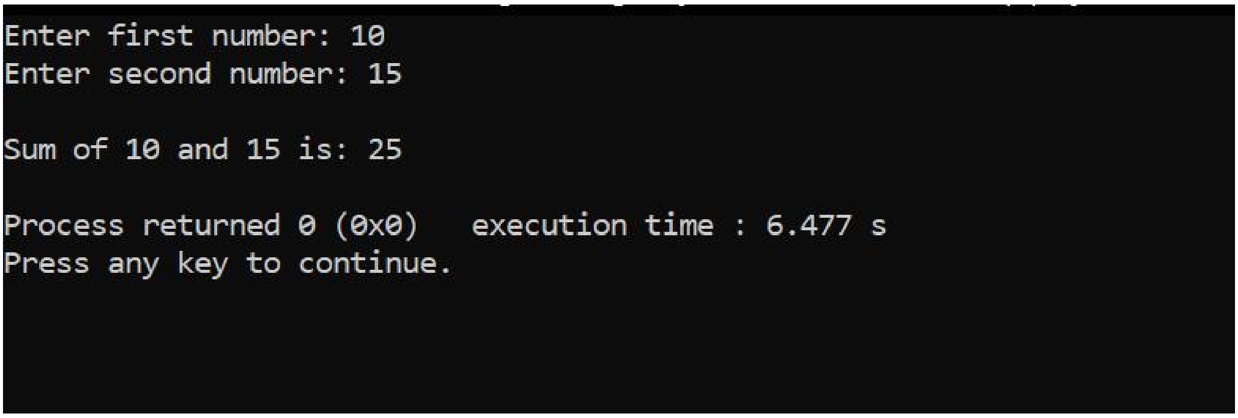
**Experiment No: 1**
**Experiment name:** Write a program in C to add two numbers using pointers.

## Solution:

## Corresponding code:

```
#include <stdio.h>
int main() {
        int n1,n2,sum;
        int *ptr1 = &n1, *ptr2 = &n2, *ptrSum = &sum;

        printf("Enter first number: ");
        scanf("%d", ptr1);

        printf("Enter second number: ");
        scanf("%d", ptr2);

        *ptrSum = *ptr1 + *ptr2;

        printf("\nSum of %d and %d is: %d\n", *ptr1, *ptr2, *ptrSum);

        return 0;
}
```

## Output:



```
Enter first number: 10
Enter second number: 15

Sum of 10 and 15 is: 25

Process returned 0 (0x0)   execution time : 6.477 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Write a program in C to add numbers using call by reference.

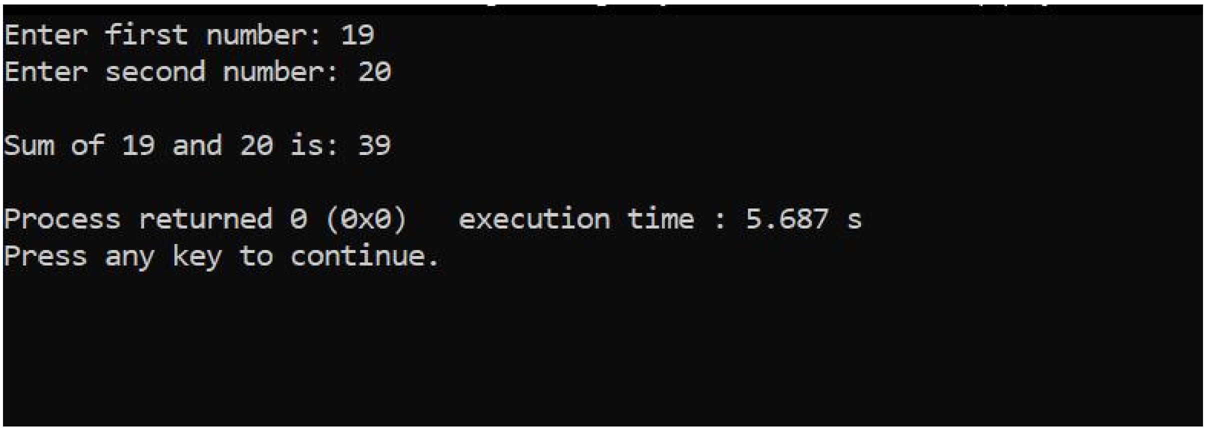
---

## Solution:

## Corresponding code:

```
#include <stdio.h>
void number(int *a, int *b, int *result) {
        *result = *a + *b;
}
int main() {
        int n1, n2, sum;
        printf("Enter first number: ");
        scanf("%d", &n1);
        printf("Enter second number: ");
        scanf("%d", &n2);

        number(&n1, &n2, &sum);

        printf("\nSum of %d and %d is: %d\n", n1, n2, sum);

        return 0;
}
```

## Output:



```
Enter first number: 19
Enter second number: 20

Sum of 19 and 20 is: 39

Process returned 0 (0x0)   execution time : 5.687 s
Press any key to continue.
```

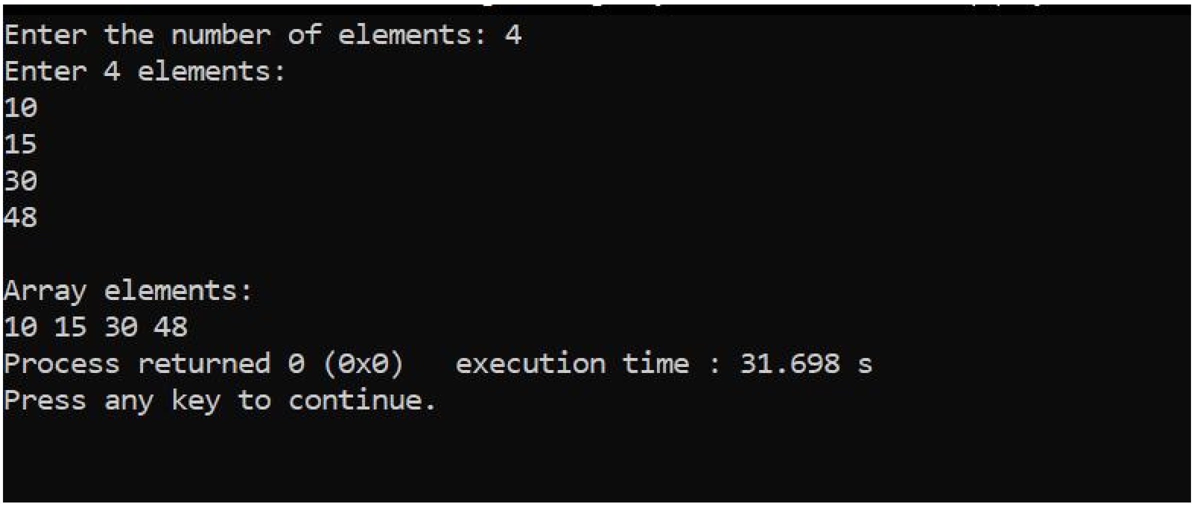
**Experiment No: 3**
**Experiment name:** Write a program in C to store n elements in an array and print the elements using a pointer.

---

**Solution:**
**Corresponding code:**

```
#include <stdio.h>
int main(){
        int n, i;

        printf("Enter the number of elements: ");
        scanf("%d", &n);

        int arr[n];
        int *ptr = arr;

        printf("Enter %d elements:\n", n);
        for(i = 0; i < n; i++) {
                scanf("%d", ptr + i);
        }

        printf("\nArray elements:\n");
        for(i = 0; i < n; i++) {
                printf("%d ", *(ptr + i));

        }
        return 0;
}
```

**Output:**



```
Enter the number of elements: 4
Enter 4 elements:
10
15
30
48

Array elements:
10 15 30 48
Process returned 0 (0x0)   execution time : 31.698 s
Press any key to continue.
```

**Experiment No: 4**
**Experiment name:** Write a program in C to find the largest element using Dynamic Memory Allocation.

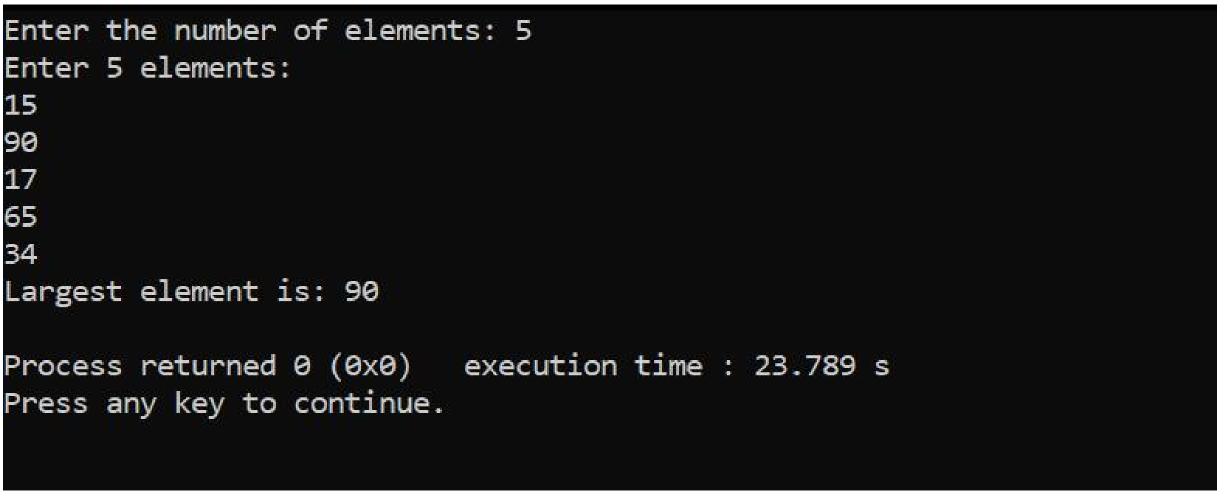
---

## Solution:

## Corresponding code:

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

int main(){
	int n,i;
	int *arr;
	int max;

	printf("Enter the number of elements: ");
	scanf("%d", &n);

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

	if(arr == NULL) {
		printf("Memory allocation failed!\n");
		return 1;
	}
	printf("Enter %d elements:\n", n);
	for(i = 0; i < n; i++) {
		scanf("%d", &arr[i]);
	}
	max = arr[0];
	for(i = 1; i < n; i++) {
		if(arr[i] > max) {
			max = arr[i];
		}
	}

	printf("Largest element is: %d\n", max);
	free(arr);

	return 0;
}
```

## Output:



```
Enter the number of elements: 5
Enter 5 elements:
15
90
17
65
34
Largest element is: 90

Process returned 0 (0x0)   execution time : 23.789 s
Press any key to continue.
```

**Experiment No: 5**

**Experiment name:** Write a program in C to calculate the length of a string using a pointer.

---

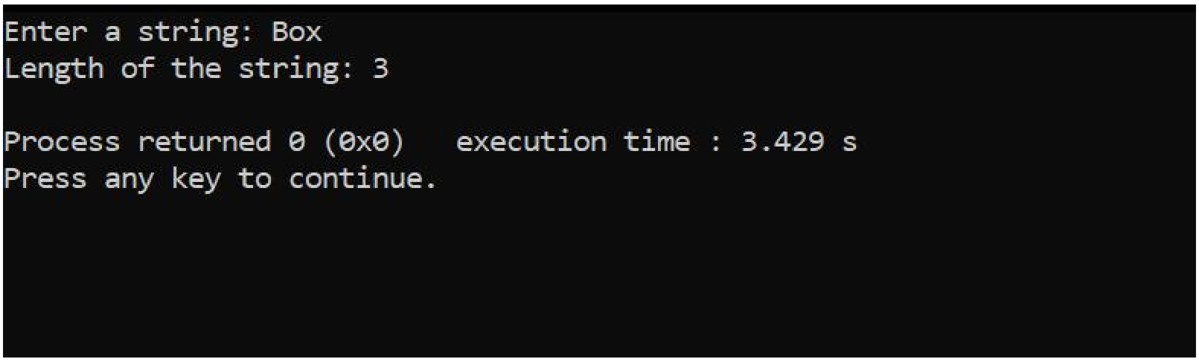
## Solution:

## Corresponding code:

```
#include <stdio.h>
int cl(char *str) {
        char *ptr = str;
        while(*ptr != '\0') {
                ptr++;
        }
        return ptr - str;
}

int main() {
        char input[100];

        printf("Enter a string: ");
        fgets(input, sizeof(input), stdin);

        int len = cl(input);
        if(len > 0 && input[len-1] == '\n') {
                input[len-1] = '\0';
        }

        len = cl(input);
```

```
    printf("Length of the string: %d\n", len);

    return 0;
}
```

**Output:**



```
Enter a string: Box
Length of the string: 3

Process returned 0 (0x0)   execution time : 3.429 s
Press any key to continue.
```

## DMA

**Experiment No: 1**

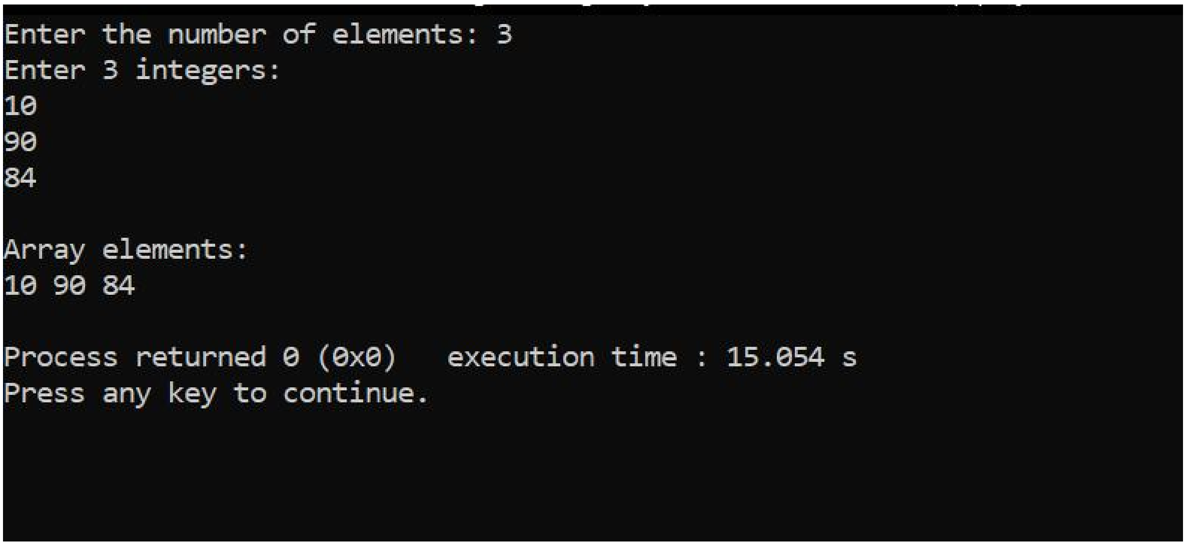
**Experiment name:** Write a C program to read and print integer array using malloc().

**Solution:**

**Corresponding code:**

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

int main(){
    int n, i;
    int *arr;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    arr = (int*)malloc(n * sizeof(int));
    if(arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    printf("Enter %d integers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("\nArray elements:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);

    return 0;
}
```

**Output:**



```
Enter the number of elements: 3
Enter 3 integers:
10
90
84

Array elements:
10 90 84

Process returned 0 (0x0)   execution time : 15.054 s
Press any key to continue.
```

**Experiment No: 2**

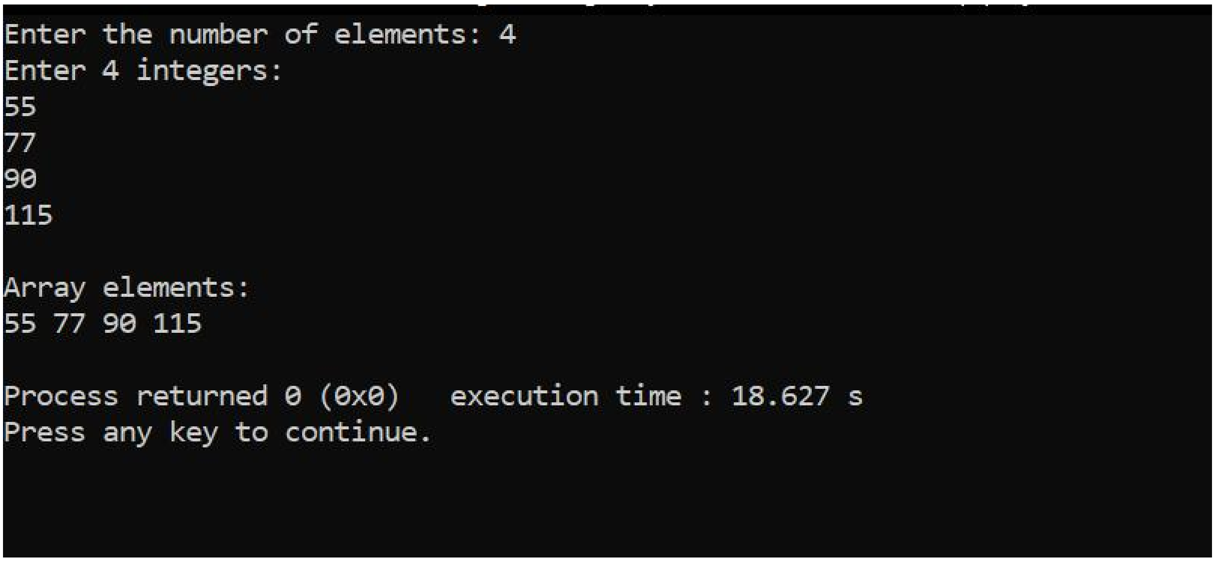
**Experiment name:** Write a C program to read and print integer array using calloc().

**Solution:**

**Corresponding code:**

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

int main() {
    int n, i;
    int *arr;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    arr = (int*)calloc(n, sizeof(int));
    if(arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    printf("Enter %d integers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", arr + i);
    }
    printf("\nArray elements:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", *(arr + i));
    }
    printf("\n");

    free(arr);

    return 0;
}
```

## **Output:**



```
Enter the number of elements: 4
Enter 4 integers:
55
77
90
115

Array elements:
55 77 90 115

Process returned 0 (0x0)   execution time : 18.627 s
Press any key to continue.
```

**Experiment No: 3**

**Experiment name:** Write a C program to calculate the sum of n numbers entered by the user using malloc() and free().

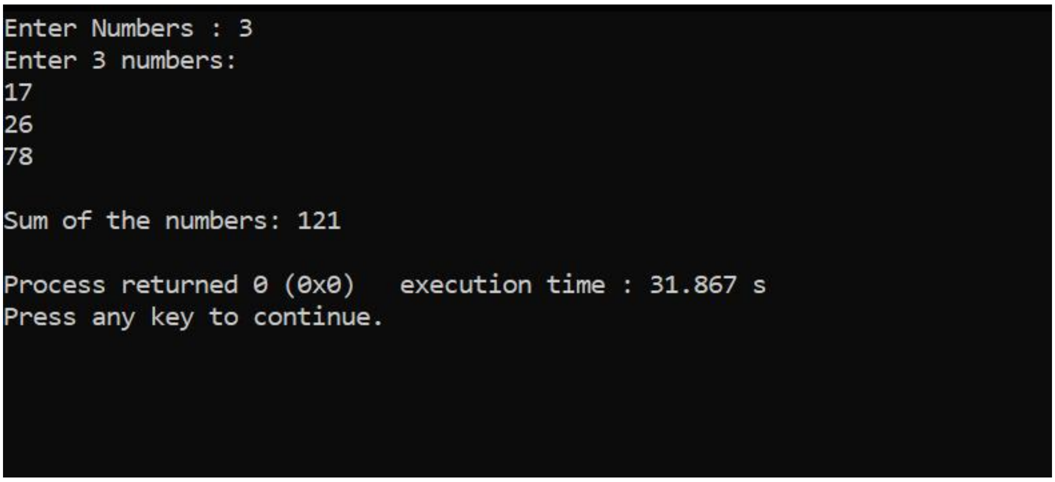
---

## Solution:

## Corresponding code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
        int n, i, sum = 0;
        int *numbers;

        printf("Enter Numbers : ");
        scanf("%d", &n);

        numbers = (int*)malloc(n * sizeof(int));
        if(numbers == NULL) {
                printf("Memory allocation failed!\n");
                return 1;
        }

        printf("Enter %d numbers:\n", n);
        for(i = 0; i < n; i++) {
                scanf("%d", numbers + i);
                sum += *(numbers + i);
        }
        printf("\nSum of the numbers: %d\n", sum);
        free(numbers);
        return 0;
}
```

## Output:



```
Enter Numbers : 3
Enter 3 numbers:
17
26
78

Sum of the numbers: 121

Process returned 0 (0x0)   execution time : 31.867 s
Press any key to continue.
```

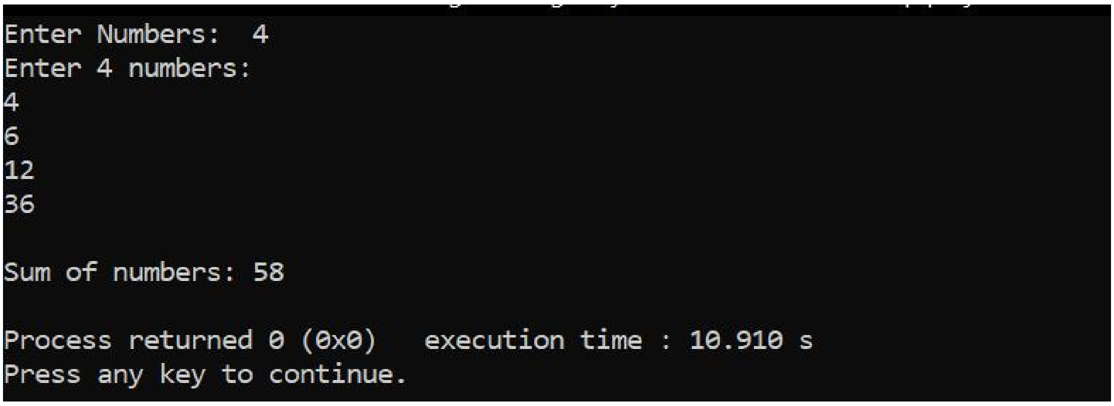
**Experiment No: 4**
**Experiment name:** Write a C program to calculate the sum of n numbers entered by the user using calloc() and free().

---

**Solution:**
**Corresponding code:**

```
#include <stdio.h>
#include <stdlib.h>
int main() {
        int n,i,sum = 0;
        int *numbers;

        printf("Enter Numbers:    ");
        scanf("%d", &n);
        numbers = (int*)calloc(n, sizeof(int));
        if(numbers == NULL) {
                printf("Memory allocation failed!\n");
                return 1;
        }

        printf("Enter %d numbers:\n", n);
        for(i = 0; i < n; i++) {

                scanf("%d", numbers + i);
                sum += numbers[i];
        }

        printf("\nSum: %d\n", sum);
        free(numbers);

        return 0;
}
```

**Output:**



```
Enter Numbers:  4
Enter 4 numbers:
4
6
12
36

Sum of numbers: 58

Process returned 0 (0x0)   execution time : 10.910 s
Press any key to continue.
```

**Experiment No: 5**

**Experiment name:** Write a C program that:

- dynamically allocates an array of 5 integers using malloc().
- fills the array with values from 1 to 5.
- uses realloc() to increase the size to 10 integers.
- assigns new values to the additional elements and prints the entire array.

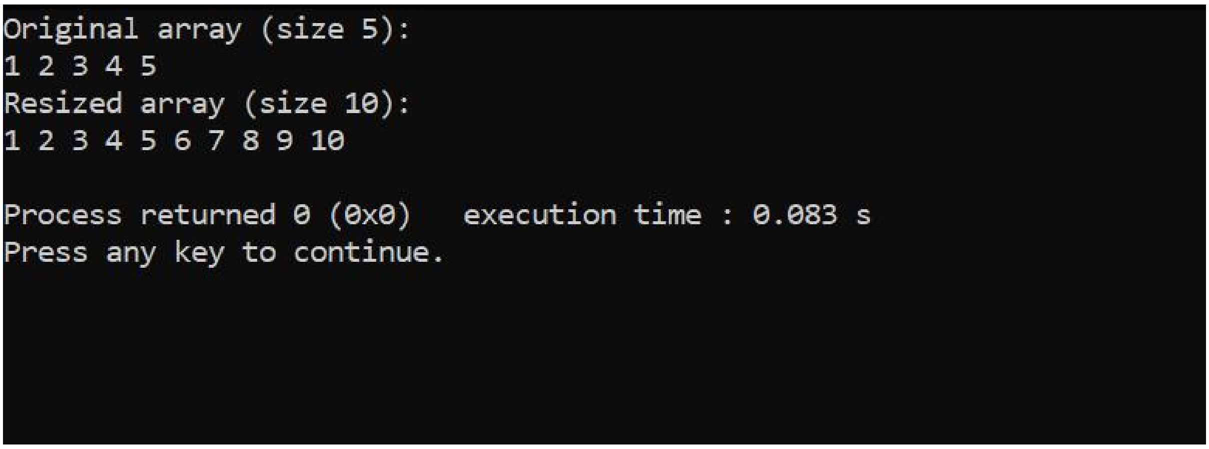
---

## Solution:

## Corresponding code:

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

int main() {
        int *arr;
        int i;
        arr = (int*)malloc(5 * sizeof(int));
        if(arr == NULL) {
                printf("Memory allocation failed!\n");
                return 1;
        }

        printf("Original array (size 5):\n");
        for(i = 0; i < 5; i++) {
                arr[i] = i + 1;
                printf("%d ", arr[i]);
        }
        printf("\n");

        int *new_arr = (int*)realloc(arr, 10 * sizeof(int));
        if(new_arr == NULL) {
                printf("Memory reallocation failed!\n");
                free(arr);
                return 1;
        }
        arr = new_arr;
        for(i = 5; i < 10; i++) {
                arr[i] = i + 1;
        }

        printf("Resized array (size 10):\n");
        for(i = 0; i < 10; i++) {
                printf("%d ", arr[i]);
        }
        printf("\n");
        free(arr);
        return 0;
}
```

**Output:**



```
Original array (size 5):
1 2 3 4 5
Resized array (size 10):
1 2 3 4 5 6 7 8 9 10

Process returned 0 (0x0)   execution time : 0.083 s
Press any key to continue.
```

## Stack

**Experiment No: 1**

**Experiment name:** Write a C program to implement a stack using an array. Include operations for push, pop, and displaying the stack contents.

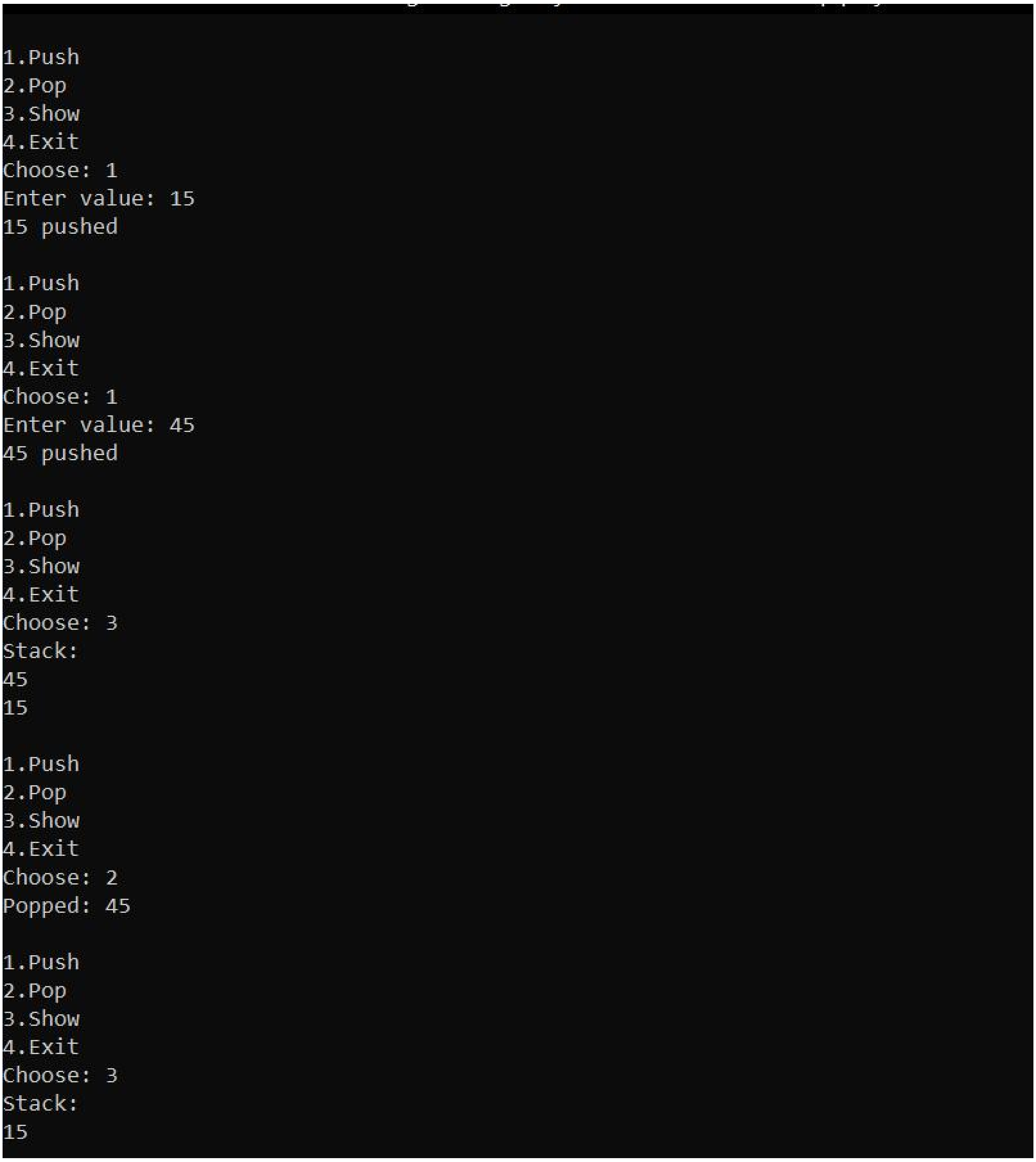
---

**Solution:**

**Corresponding code:**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

int stack[MAX];
int top = -1;

void push(int item) {
        if (top == MAX - 1) {
                printf("Stack full. Can't push %d\n", item);
        } else {
                stack[++top] = item;
                printf("%d pushed\n", item);
        }
}

int pop() {
        if (top == -1) {
                printf("Stack empty. Can't pop\n");
                return -1;
        }
        return stack[top--];
}

void show() {
        if (top == -1) {
                printf("Stack empty\n");
        } else {
                printf("Stack:\n");
                for (int i = top; i >= 0; i--) {
                        printf("%d\n", stack[i]);
                }
        }
}

int main() {
        int choice, x;
        while (1) {
                printf("\n1.Push\n2.Pop\n3.Show\n4.Exit\n");
                printf("Choose: ");
                scanf("%d", &choice);

                switch (choice) {
                        case 1:
                                printf("Enter value: ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                x = pop();
                                if (x != -1) printf("Popped: %d\n", x);
                                break;
                        case 3:
                                show();
                                break;
                        case 4:
                                exit(0);
                        default:
                                printf("Wrong choice\n");
                }
        }

        return 0;
}
```

**Output:**



```
1.Push
2.Pop
3.Show
4.Exit
Choose: 1
Enter value: 15
15 pushed

1.Push
2.Pop
3.Show
4.Exit
Choose: 1
Enter value: 45
45 pushed

1.Push
2.Pop
3.Show
4.Exit
Choose: 3
Stack:
45
15

1.Push
2.Pop
3.Show
4.Exit
Choose: 2
Popped: 45

1.Push
2.Pop
3.Show
4.Exit
Choose: 3
Stack:
15
```

**Experiment No: 2**

**Experiment name:** Write a C program to implement a stack using a linked list. Include functions for push, pop, and displaying the stack elements.

---

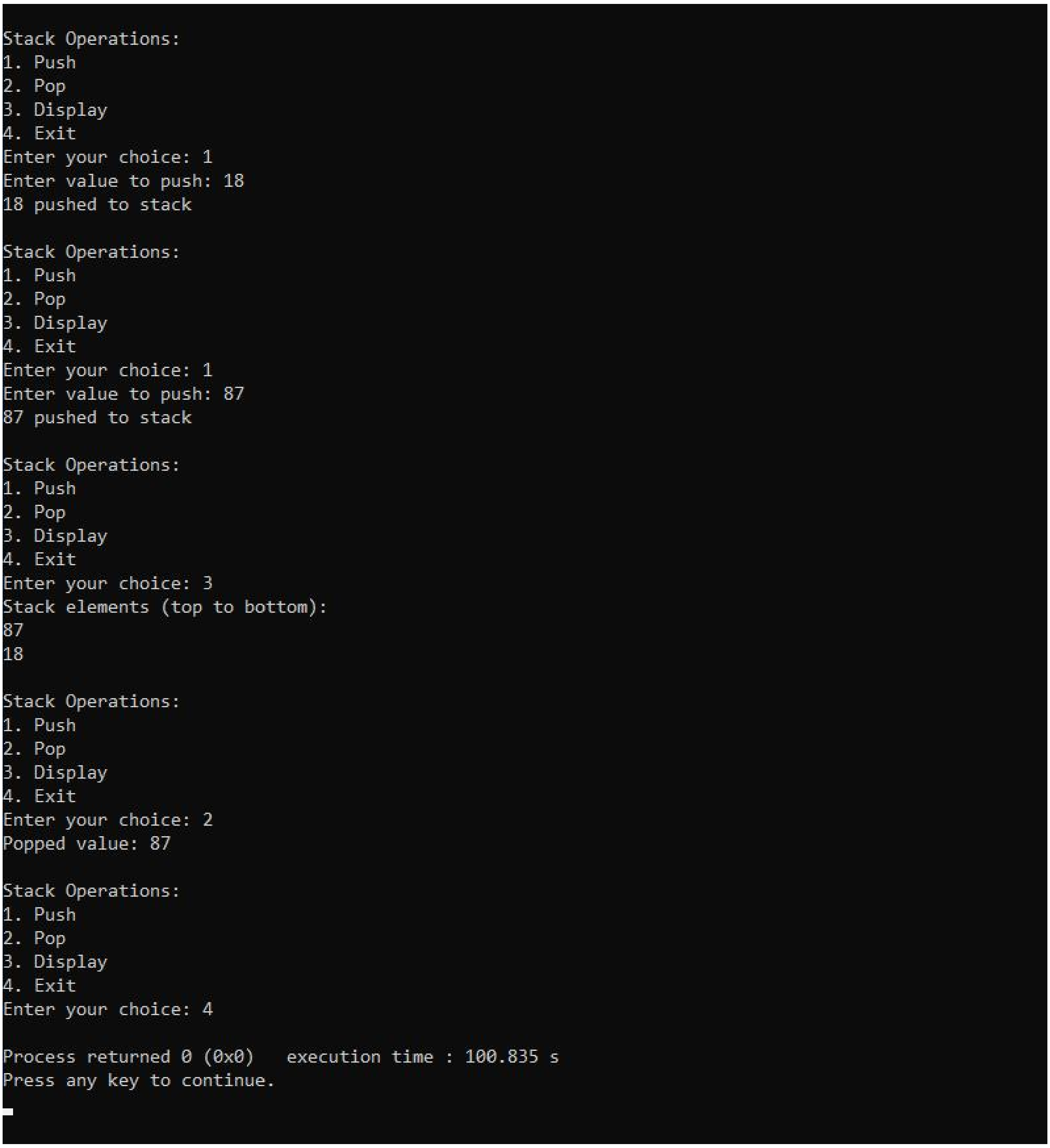
## Solution:

## Corresponding code:

```
  #include <stdio.h>
#include <stdlib.h>
struct Node {

        int data;
        struct Node* next;
};

struct Node* top = NULL;

void push(int value) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        printf("%d pushed to stack\n", value);
}

int pop() {
        if (top == NULL) {
                printf("Stack is empty\n");
                return -1;
        }
        struct Node* temp = top;
        int popped = temp->data;
        top = top->next;
        free(temp);
        return popped;
}

void display() {
        if (top == NULL) {
                printf("Stack is empty\n");
                return;
        }
        struct Node* current = top;
        printf("Stack elements (top to bottom):\n");
```

```
    while (current != NULL) {
        printf("%d\n", current->data);
        current = current->next;
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                value = pop();
                if (value != -1) {
                    printf("Popped value: %d\n", value);
                }
                break;
            case 3:
                display();
                break;

            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

## Output:



```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 18
18 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 87
87 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements (top to bottom):
87
18

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 87

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 100.835 s
Press any key to continue.
```

**Experiment No: 3**

**Experiment name:** Write a C function to implement the peek operation, which retrieves the top element of the stack without removing it.

---

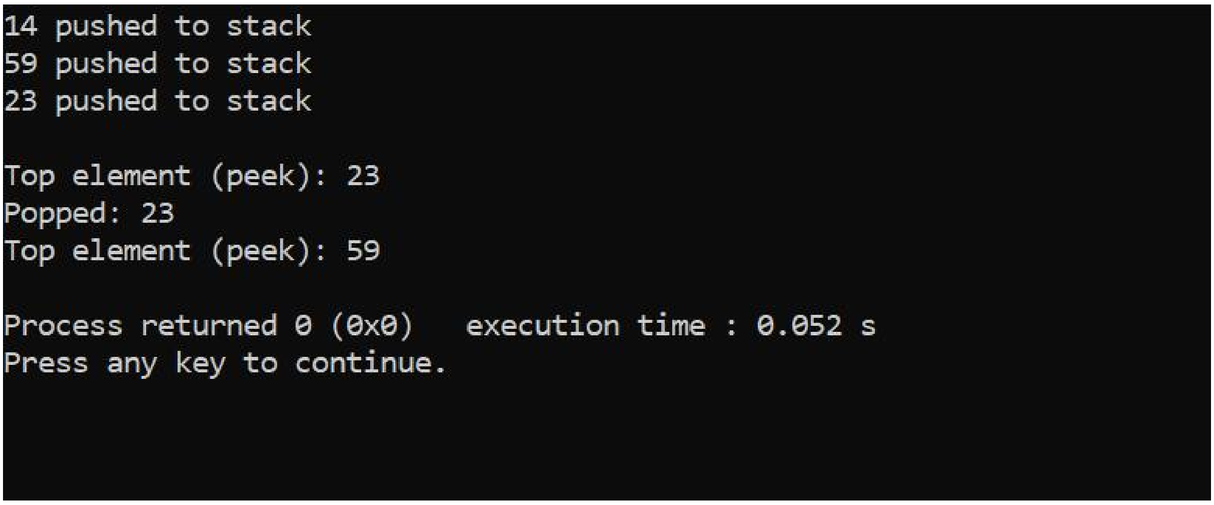
**Solution:**

**Corresponding code:**

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

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

struct Node* top = NULL;

void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("%d pushed to stack\n", value);
}

int pop() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return -1;
    }
    struct Node* temp = top;
    int popped = temp->data;
    top = top->next;
    free(temp);
    return popped;
}


int peek() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return -1;
    }
    return top->data;
}
```

```
void display() {
        if (top == NULL) {
                printf("Stack is empty\n");
                return;
        }
        struct Node* current = top;
        printf("Stack (top to bottom):\n");
        while (current != NULL) {
                printf("%d\n", current->data);
                current = current->next;
        }
}

int main() {
        push(14);
        push(59);
        push(23);

        printf("\nTop element (peek): %d\n", peek());
        printf("Popped: %d\n", pop());
        printf("Top element (peek): %d\n", peek());

        return 0;
}
```

**Output:**



```
14 pushed to stack
59 pushed to stack
23 pushed to stack

Top element (peek): 23
Popped: 23
Top element (peek): 59

Process returned 0 (0x0)   execution time : 0.052 s
Press any key to continue.
```

**Experiment No: 4**
**Experiment name:** Write a C program to convert an infix expression to a postfix expression using a stack.

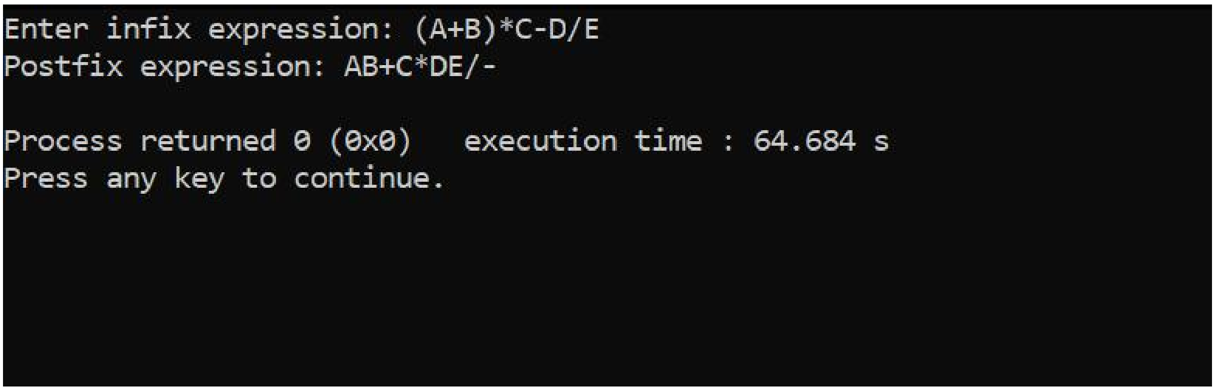
---

**Solution:**
**Corresponding code:**

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char item) {
	if (top >= MAX-1) {
		printf("Stack Overflow\n");
		return;
	}
	stack[++top] = item;
}


char pop() {
	if (top < 0) {
		printf("Stack Underflow\n");
		return '\0';
	}
	return stack[top--];
}

int precedence(char op) {
	switch(op) {
		case '+':
		case '-':
			return 1;
		case '*':
		case '/':
			return 2;
		case '^':
			return 3;
		default:
			return 0;
	}
```

```
}

void infixToPostfix(char* infix, char* postfix) {
        int i = 0, j = 0;
        char item, x;

        push('(');
        strcat(infix, ")");

        for (item = infix[i]; item != '\0'; item = infix[++i]) {
                if (item == '(') {

                        push(item);
                }
                else if (isalnum(item)) {
                        postfix[j++] = item;
                }
                else if (item == ')') {
                        while ((x = pop()) != '(') {
                                postfix[j++] = x;
                        }
                }
                else {
                        while (precedence(stack[top]) >= precedence(item)) {
                                postfix[j++] = pop();
                        }
                        push(item);
                }
        }
        postfix[j] = '\0';
}

int main() {
        char infix[MAX], postfix[MAX];
        printf("Enter infix expression: ");
        fgets(infix, MAX, stdin);
        infix[strcspn(infix, "\n")] = '\0';

        infixToPostfix(infix, postfix);
        printf("Postfix expression: %s\n", postfix);

        return 0;
}
```

## Output:



```
Enter infix expression: (A+B)*C-D/E
Postfix expression: AB+C*DE/-

Process returned 0 (0x0)   execution time : 64.684 s
Press any key to continue.
```

## Experiment No: 5

**Experiment name:** Write a C program to reverse a string using a stack. Push each character onto the stack and pop them to obtain the reversed string.

---

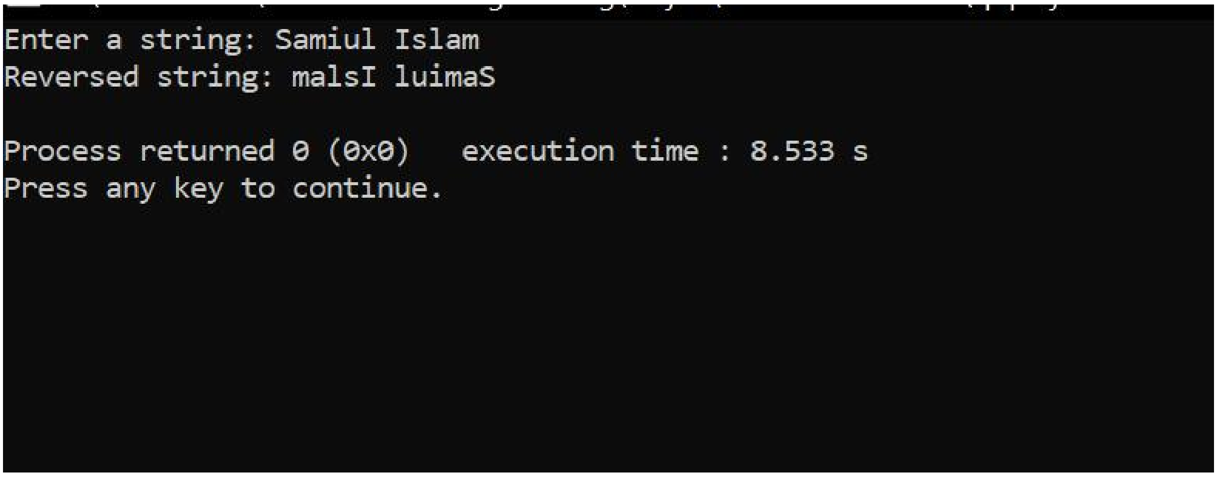
## Solution:

## Corresponding code:

```
#include <stdio.h>
#include <string.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
        if (top >= MAX-1) {
                printf("Stack Overflow\n");
                return;
        }
        stack[++top] = c;
}

char pop() {
        if (top < 0) {
                printf("Stack Underflow\n");
                return '\0';

        }
        return stack[top--];
}
```

```
void reverseString(char* str) {
        for (int i = 0; i < strlen(str); i++) {
                push(str[i]);
        }

        for (int i = 0; i < strlen(str); i++) {
                str[i] = pop();
        }
}

int main() {
        char str[MAX];

        printf("Enter a string: ");
        fgets(str, MAX, stdin);
        str[strcspn(str, "\n")] = '\0';

        reverseString(str);
        printf("Reversed string: %s\n", str);
        return 0;
}
```

**Output:**



```
Enter a string: Samiul Islam
Reversed string: malsI luimaS

Process returned 0 (0x0)   execution time : 8.533 s
Press any key to continue.
```

**Experiment No: 6**

**Experiment name:** Evaluate the following postfix expression using a stack in C: 10 5 + 60 6 / * 8 -.

---

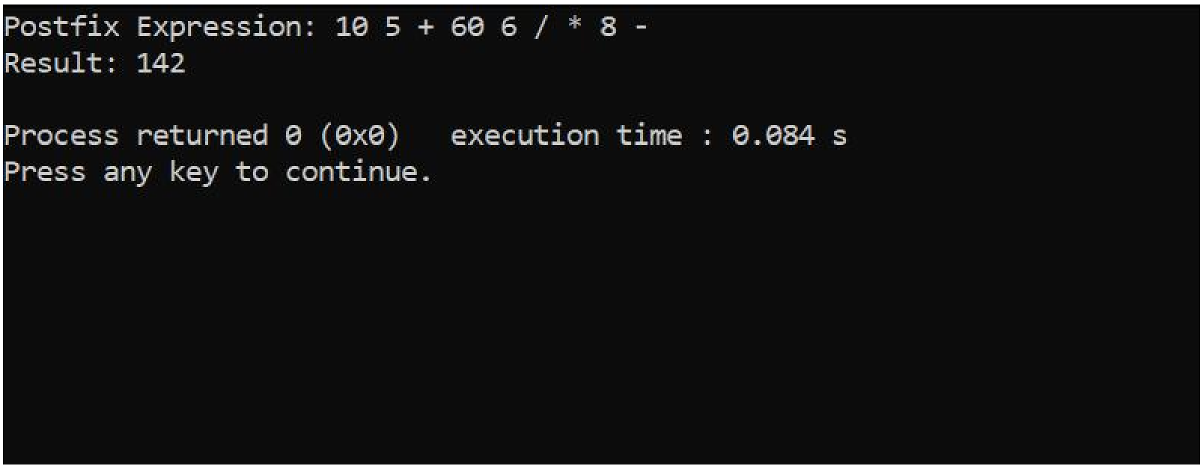
**Solution:**

**Corresponding code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 100

int stack[MAX];
int top = -1;

void push(int item) {
	if (top >= MAX-1) {
		printf("Stack Overflow\n");
		return;
	}
	stack[++top] = item;
}

int pop() {
	if (top < 0) {
		printf("Stack Underflow\n");
		exit(1);
	}
	return stack[top--];
}

int evaluatePostfix(char* exp) {
	int i, op1, op2, value;
	char ch;

	for (i = 0; exp[i] != '\0'; i++) {
		ch = exp[i];

		if (ch == ' ') continue;

		if (isdigit(ch)) {
			int num = 0;
			while (isdigit(exp[i])) {
				num = num * 10 + (exp[i] - '0');
				i++;
```

```
            }
            i--;
            push(num);
        }
        else {
            op2 = pop();
            op1 = pop();
            switch(ch) {
                case '+': push(op1 + op2); break;
                case '-': push(op1 - op2); break;
                case '*': push(op1 * op2); break;
                case '/': push(op1 / op2); break;
                default:
                    printf("Invalid operator: %c\n", ch);
                    exit(1);
            }
        }
    }
    return pop();
}

int main() {
    char exp[] = "10 5 + 60 6 / * 8 -";
    printf("Postfix Expression: %s\n", exp);

    int result = evaluatePostfix(exp);
    printf("Result: %d\n", result);


    return 0;
}
```

**<u>Output:</u>**



```
Postfix Expression: 10 5 + 60 6 / * 8 -
Result: 142

Process returned 0 (0x0)   execution time : 0.084 s
Press any key to continue.
```

# Queue

**Experiment No: 1**

**Experiment name:** Write a C program to implement a queue using an array. Include operations for enqueue, dequeue, and displaying the queue contents.

---

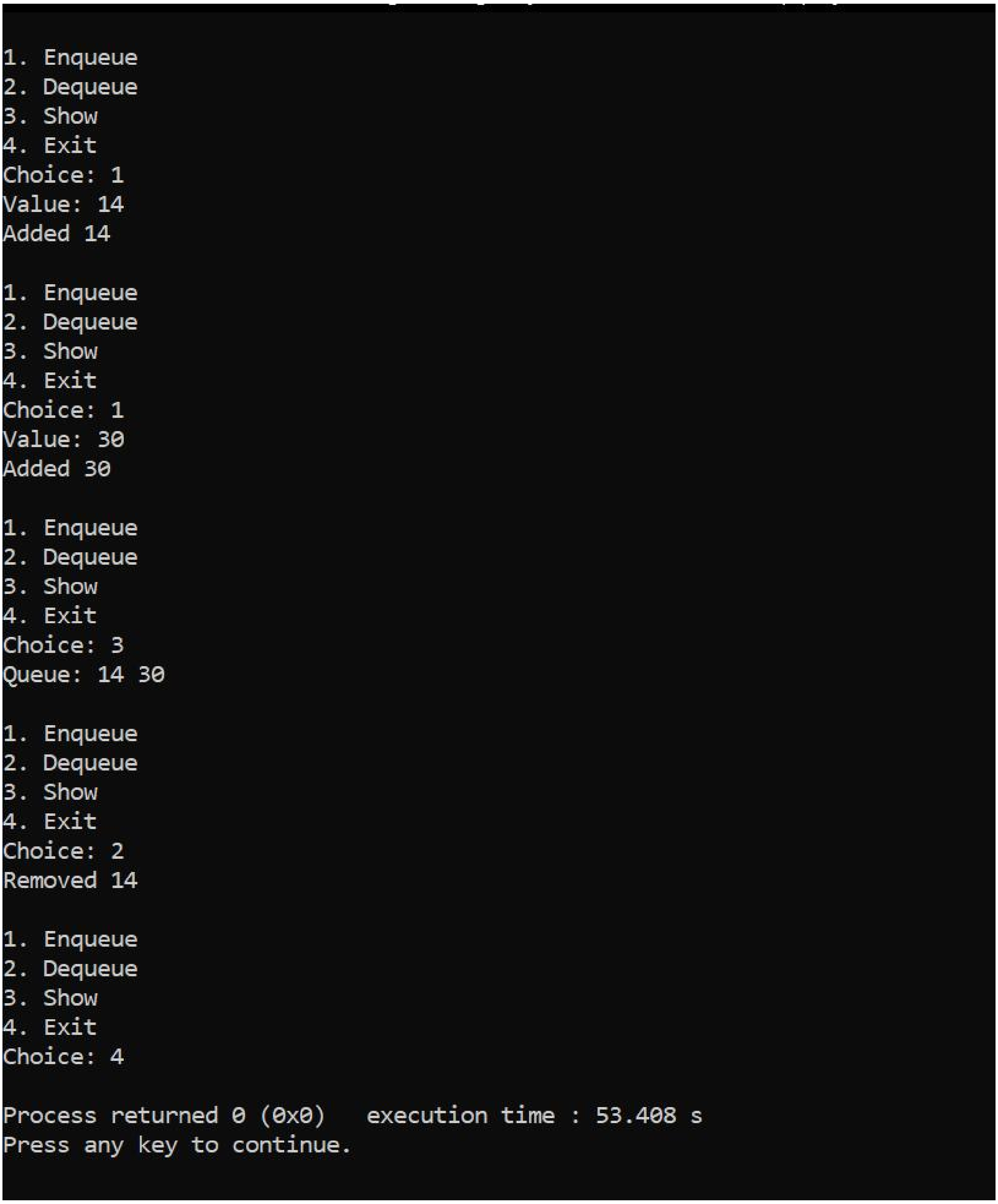
## Solution:

## Corresponding code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5

typedef struct {
        int items[MAX];
        int front;
        int rear;
} Queue;

void init(Queue *q) {
        q->front = -1;
        q->rear = -1;
}

bool full(Queue *q) {
        return (q->rear == MAX - 1 && q->front == 0) || (q->rear == q->front - 1);
}

bool empty(Queue *q) {
        return q->front == -1;
}

void enq(Queue *q, int val) {
        if (full(q)) {
                printf("Queue full\n");
                return;
        }

        if (empty(q)) {
                q->front = 0;
                q->rear = 0;
        } else if (q->rear == MAX - 1) {
```

```
        q->rear = 0;
    } else {
        q->rear++;
    }

    q->items[q->rear] = val;
    printf("Added %d\n", val);
}

int deq(Queue *q) {
    if (empty(q)) {
        printf("Queue empty\n");

        return -1;
    }

    int val = q->items[q->front];

    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else if (q->front == MAX - 1) {
        q->front = 0;
    } else {
        q->front++;
    }

    printf("Removed %d\n", val);
    return val;
}

void show(Queue *q) {
    if (empty(q)) {
        printf("Queue empty\n");
        return;
    }

    printf("Queue: ");

    if (q->rear >= q->front) {
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
    } else {
        for (int i = q->front; i < MAX; i++) {
            printf("%d ", q->items[i]);
        }
        for (int i = 0; i <= q->rear; i++) {

            printf("%d ", q->items[i]);
        }
    }
    printf("\n");
}

int main() {
    Queue q;
    init(&q);

    int ch, val;

    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Show\n4. Exit\nChoice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Value: ");
                scanf("%d", &val);
                enq(&q, val);
                break;
            case 2:
                deq(&q);
                break;
            case 3:
                show(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid\n");
        }
    }
    return 0;}
```

**Output:**



```
1. Enqueue
2. Dequeue
3. Show
4. Exit
Choice: 1
Value: 14
Added 14

1. Enqueue
2. Dequeue
3. Show
4. Exit
Choice: 1
Value: 30
Added 30

1. Enqueue
2. Dequeue
3. Show
4. Exit
Choice: 3
Queue: 14 30

1. Enqueue
2. Dequeue
3. Show
4. Exit
Choice: 2
Removed 14

1. Enqueue
2. Dequeue
3. Show
4. Exit
Choice: 4

Process returned 0 (0x0)   execution time : 53.408 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Write a C program to implement a queue using a linked list. Include functions for enqueue, dequeue, and displaying the queue elements.

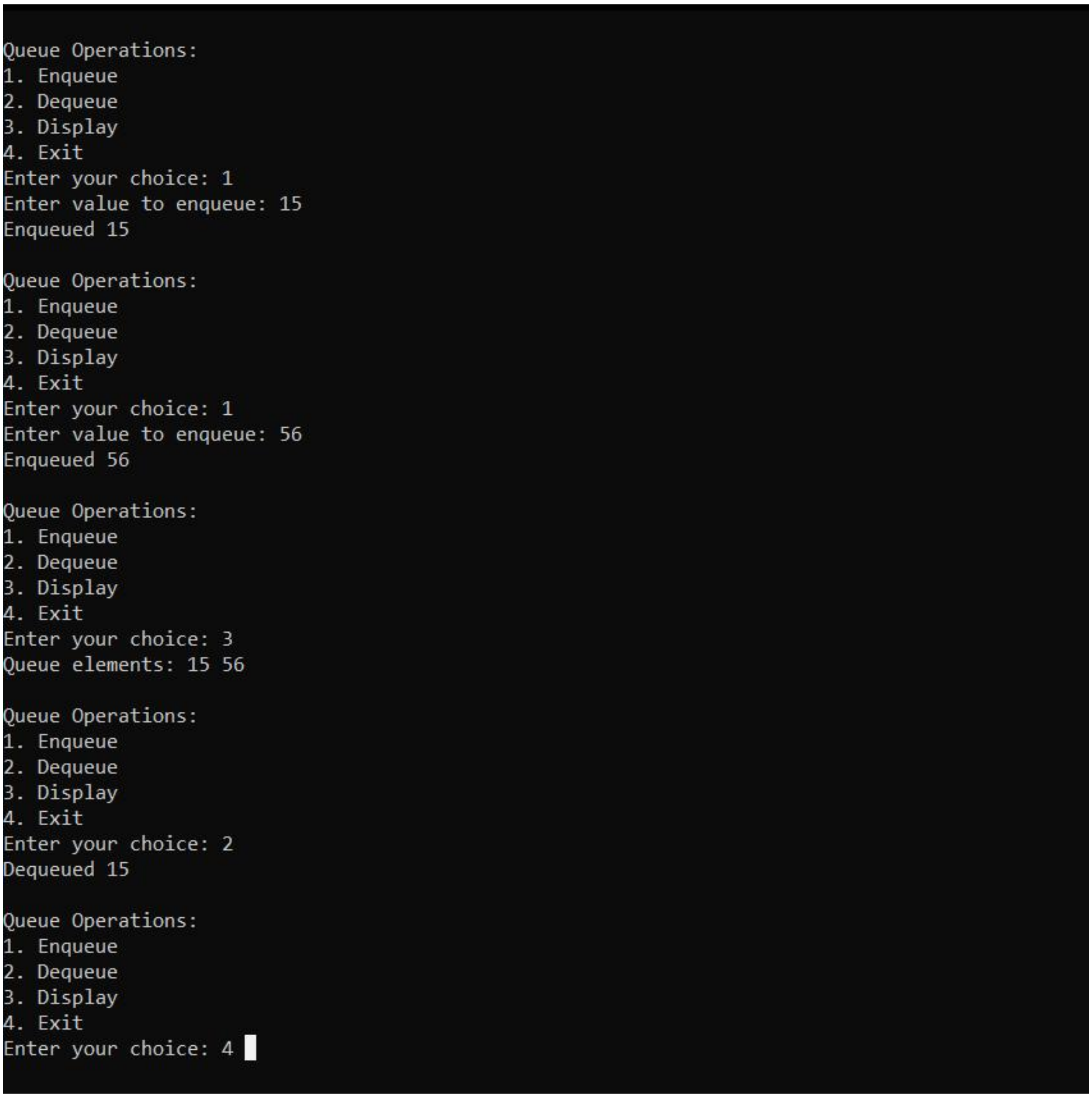
---

**Solution:**

**Corresponding code:**

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
        int data;
        struct Node* next;
} Node;
typedef struct {
        Node* front;
        Node* rear;
} Queue;
void initQueue(Queue* q) {
        q->front = q->rear = NULL;
}


int isEmpty(Queue* q) {
        return q->front == NULL;
}
void enqueue(Queue* q, int value) {
        Node* newNode = (Node*)malloc(sizeof(Node));
        if (!newNode) {
                printf("Memory allocation failed\n");
                return;
        }
        newNode->data = value;
        newNode->next = NULL;

        if (isEmpty(q)) {
                q->front = q->rear = newNode;
        } else {
                q->rear->next = newNode;
                q->rear = newNode;
        }
        printf("Enqueued %d\n", value);
}
int dequeue(Queue* q) {
        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return -1;
        }
        Node* temp = q->front;
        int value = temp->data;
        q->front = q->front->next;

        if (q->front == NULL) {
                q->rear = NULL;
        }

        free(temp);
        printf("Dequeued %d\n", value);

        return value;
}
void display(Queue* q) {
        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return;
        }
        printf("Queue elements: ");
        Node* current = q->front;
        while (current != NULL) {
                printf("%d ", current->data);
                current = current->next;
        }
        printf("\n");
}
int main(){
        Queue q;
        initQueue(&q);
        int choice, value;

        while (1) {
                printf("\nQueue Operations:\n");
                printf("1. Enqueue\n");
                printf("2. Dequeue\n");
                printf("3. Display\n");
                printf("4. Exit\n");
                printf("Enter your choice: ");
                scanf("%d", &choice);

                switch (choice) {
                    case 1:
                        printf("Enter value to enqueue: ");
                        scanf("%d", &value);
                        enqueue(&q, value);
                        break;

                    case 2:
                        dequeue(&q);
                        break;
                    case 3:
                        display(&q);
                        break;
                    case 4:
                        exit(0);
                    default:
                        printf("Invalid choice\n");
                }
        }
        return 0;
}
```

**Output:**



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 15
Enqueued 15

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued 56

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 15 56

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 15

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
```

**Experiment No: 3**
**Experiment name:** Write a C function to implement the peek operation, which retrieves the front element of the queue without removing it.

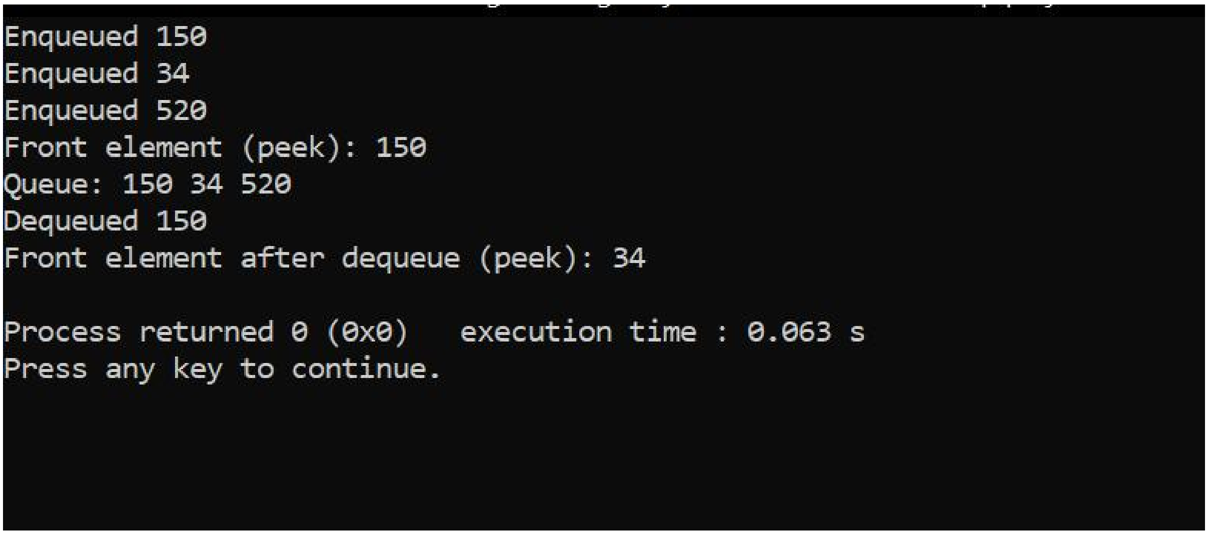
---

**Solution:**
**Corresponding code:**

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

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

typedef struct {
    Node* front;
    Node* rear;
} Queue;

void initQueue(Queue* q) {
    q->front = q->rear = NULL;
}

int isEmpty(Queue* q) {
    return q->front == NULL;
}

void enqueue(Queue* q, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (isEmpty(q)) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
    printf("Enqueued %d\n", value);
}
```

```
int dequeue(Queue* q) {
        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return -1;
        }
        Node* temp = q->front;
        int value = temp->data;
        q->front = q->front->next;
        if (q->front == NULL) {
                q->rear = NULL;
        }
        free(temp);
        printf("Dequeued %d\n", value);
        return value;
}

int peek(Queue* q) {
        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return -1;
        }
        return q->front->data;
}

void display(Queue* q) {

        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return;
        }
        printf("Queue: ");
        Node* current = q->front;
        while (current != NULL) {
                printf("%d ", current->data);
                current = current->next;
        }
        printf("\n");
}

int main() {
        Queue q;
        initQueue(&q);

        enqueue(&q, 150);
        enqueue(&q, 34);
        enqueue(&q, 520);
        printf("Front element (peek): %d\n", peek(&q));
        display(&q);

        dequeue(&q);
        printf("Front element after dequeue (peek): %d\n", peek(&q));

        return 0;
}
```

**Output:**



```
Enqueued 150
Enqueued 34
Enqueued 520
Front element (peek): 150
Queue: 150 34 520
Dequeued 150
Front element after dequeue (peek): 34

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

## Experiment No: 4

**Experiment name:** Write a C program to create a menu-driven queue system that performs the following operations:

- Add items
- Delete items
- Show the number of items
- Show the minimum and maximum items
- Find an item
- Print all items
- Exit

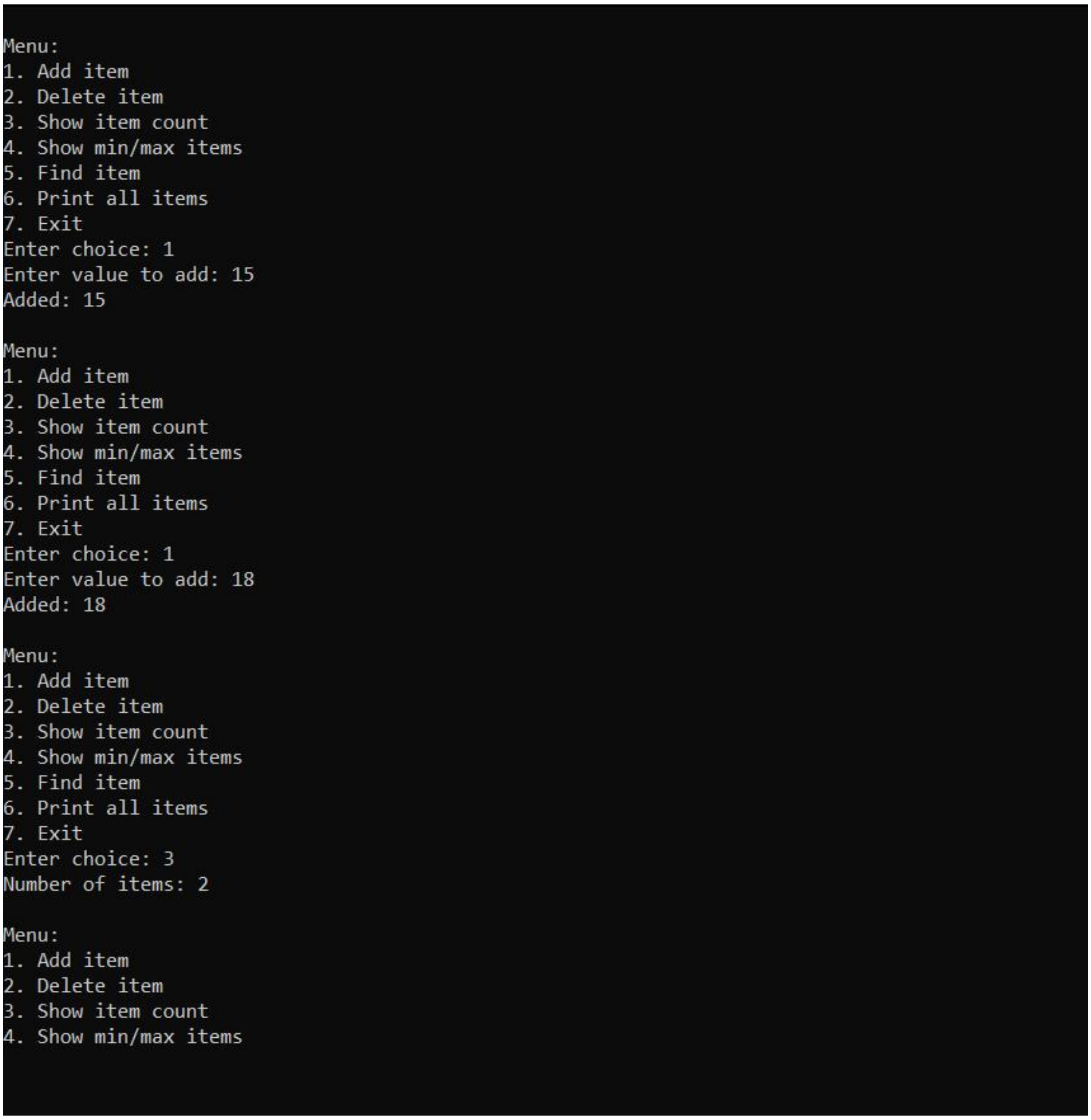
**Solution:**

**Corresponding code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define MAX_SIZE 100

typedef struct {
        int items[MAX_SIZE];
        int front;
        int rear;
        int count;
} Queue;


void initQueue(Queue *q) {
        q->front = 0;
        q->rear = -1;
        q->count = 0;
}

int isFull(Queue *q) {
        return q->count == MAX_SIZE;
}

int isEmpty(Queue *q) {
        return q->count == 0;
}

void enqueue(Queue *q, int value) {
        if (isFull(q)) {
                printf("Queue is full\n");
                return;
        }
        q->rear = (q->rear + 1) % MAX_SIZE;
        q->items[q->rear] = value;
        q->count++;
        printf("Added: %d\n", value);
}

int dequeue(Queue *q) {
        if (isEmpty(q)) {
                printf("Queue is empty\n");
                return -1;
        }
        int value = q->items[q->front];
```

```
    q->front = (q->front + 1) % MAX_SIZE;
    q->count--;
    printf("Removed: %d\n", value);

    return value;
}

int itemCount(Queue *q) {
    return q->count;
}

void findMinMax(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    int min = INT_MAX;
    int max = INT_MIN;
    int i = q->front;
    int cnt = 0;

    while (cnt < q->count) {
        if (q->items[i] < min) min = q->items[i];
        if (q->items[i] > max) max = q->items[i];
        i = (i + 1) % MAX_SIZE;
        cnt++;
    }
    printf("Minimum: %d, Maximum: %d\n", min, max);
}

int findItem(Queue *q, int value) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return 0;
    }
    int i = q->front;
    int cnt = 0;
    int found = 0;


    while (cnt < q->count) {
        if (q->items[i] == value) {
            found++;
        }
        i = (i + 1) % MAX_SIZE;
        cnt++;
    }
    if (found) {
        printf("Item %d found %d times\n", value, found);
    } else {
        printf("Item %d not found\n", value);
    }
    return found;
}

void printQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue items: ");
    int i = q->front;
    int cnt = 0;

    while (cnt < q->count) {
        printf("%d ", q->items[i]);
        i = (i + 1) % MAX_SIZE;
        cnt++;
    }
    printf("\n");
}

int main() {
    Queue q;
    initQueue(&q);

    int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add item\n");
        printf("2. Delete item\n");
        printf("3. Show item count\n");
        printf("4. Show min/max items\n");
        printf("5. Find item\n");
        printf("6. Print all items\n");
        printf("7. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to add: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                printf("Number of items: %d\n", itemCount(&q));
                break;
            case 4:
                findMinMax(&q);
                break;
            case 5:
                printf("Enter value to find: ");
                scanf("%d", &value);
                findItem(&q, value);
                break;
            case 6:

                printQueue(&q);
                break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

## Output:



```
Menu:
1. Add item
2. Delete item
3. Show item count
4. Show min/max items
5. Find item
6. Print all items
7. Exit
Enter choice: 1
Enter value to add: 15
Added: 15

Menu:
1. Add item
2. Delete item
3. Show item count
4. Show min/max items
5. Find item
6. Print all items
7. Exit
Enter choice: 1
Enter value to add: 18
Added: 18

Menu:
1. Add item
2. Delete item
3. Show item count
4. Show min/max items
5. Find item
6. Print all items
7. Exit
Enter choice: 3
Number of items: 2

Menu:
1. Add item
2. Delete item
3. Show item count
4. Show min/max items
```

# Linked List

**Experiment No: 1**

**Experiment name:** Write a C program to implement a singly linked list with operations for inserting a node at the beginning, end, and a specific position.

---

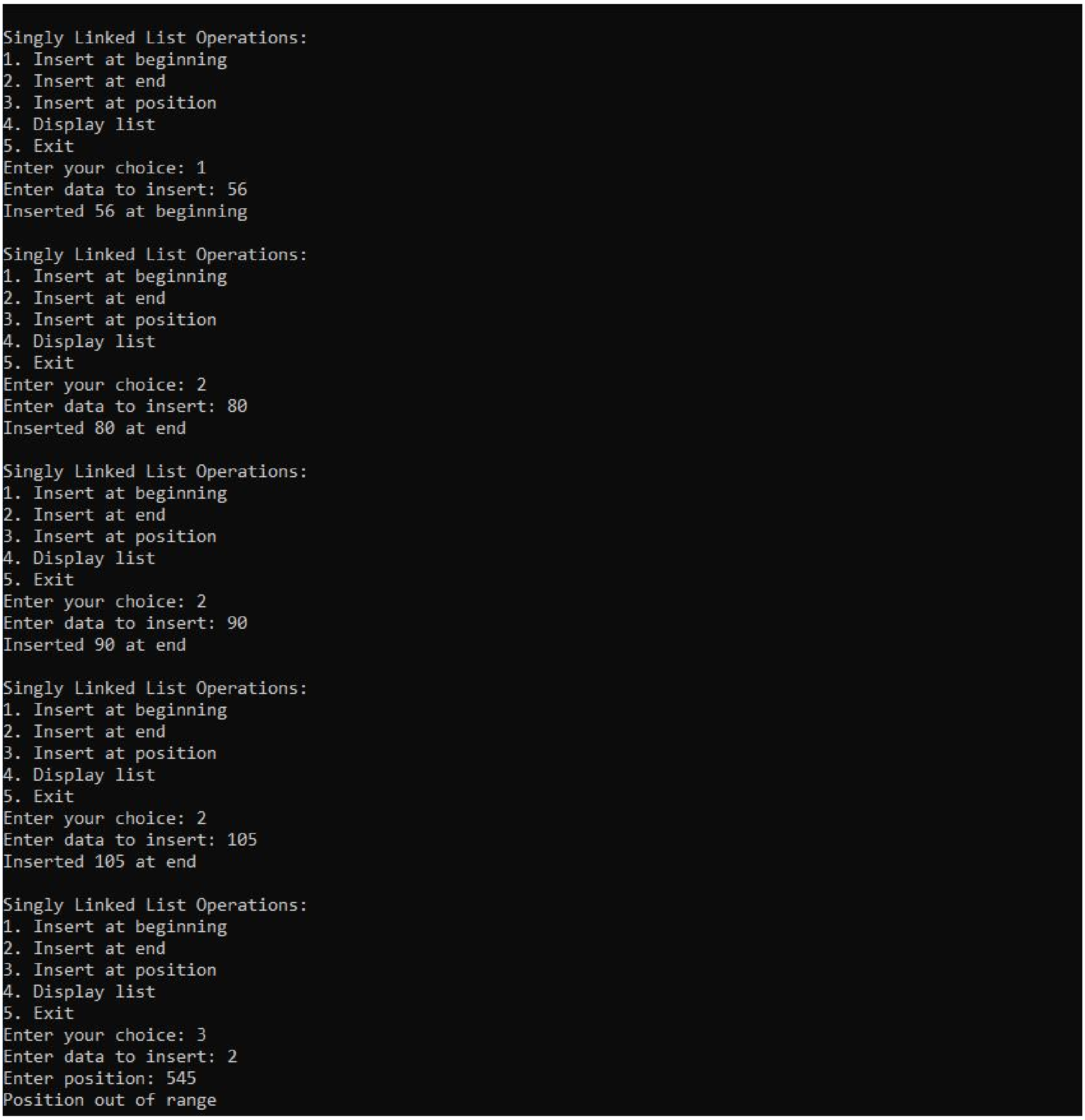
## Solution:

## Corresponding code:

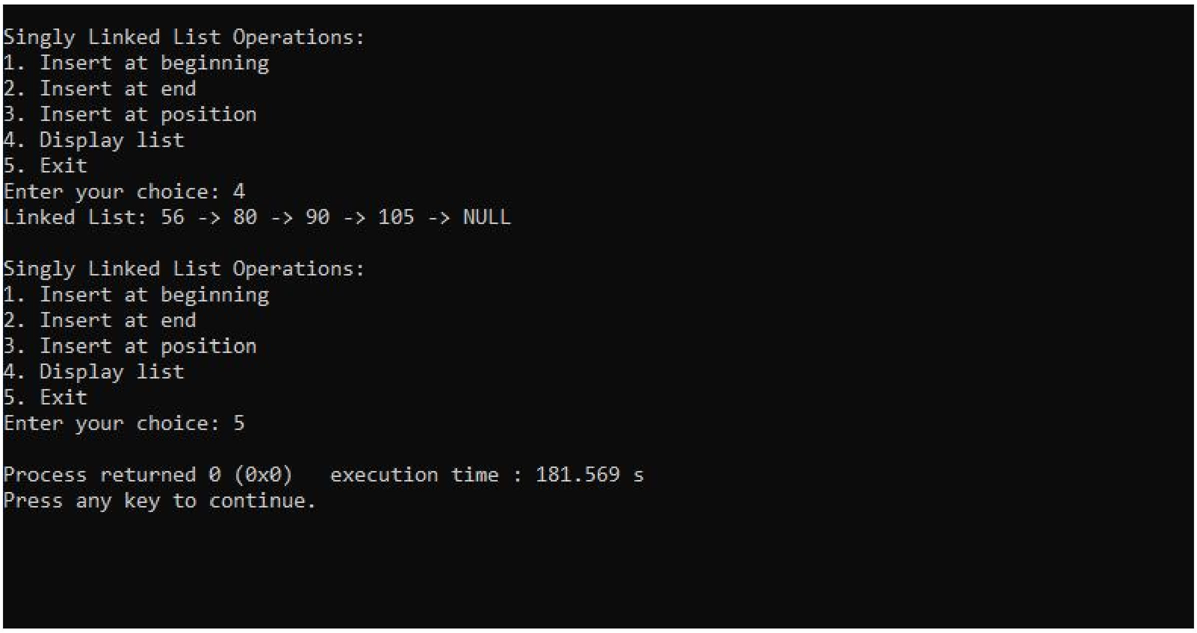
```
#include <stdio.h>
#include <stdlib.h>

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

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(Node** head, int data) {
    Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("Inserted %d at beginning\n", data);
}


void insertAtEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Inserted %d at end\n", data);
}

void insertAtPosition(Node** head, int data, int pos) {
    if (pos < 1) {
        printf("Invalid position\n");
        return;
    }

    if (pos == 1) {
        insertAtBeginning(head, data);
        return;
    }

    Node* newNode = createNode(data);
    Node* temp = *head;

    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {

        printf("Position out of range\n");
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", data, pos);
}

void displayList(Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Linked List: ");
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
```

```
void freeList(Node* head) {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {

    Node* head = NULL;
    int choice, data, pos;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at position\n");
        printf("4. Display list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position: ");
                scanf("%d", &pos);
                insertAtPosition(&head, data, pos);
                break;
            case 4:
                displayList(head);

                break;
            case 5:
                freeList(head);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

**Output:**



```
Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 1
Enter data to insert: 56
Inserted 56 at beginning

Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 2
Enter data to insert: 80
Inserted 80 at end

Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 2
Enter data to insert: 90
Inserted 90 at end

Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 2
Enter data to insert: 105
Inserted 105 at end

Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 3
Enter data to insert: 2
Enter position: 545
Position out of range
```



```
Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 4
Linked List: 56 -> 80 -> 90 -> 105 -> NULL

Singly Linked List Operations:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 181.569 s
Press any key to continue.
```

**Experiment No: 2**

**Experiment name:** Write a C function to insert a node at the beginning of a singly linked list.

---

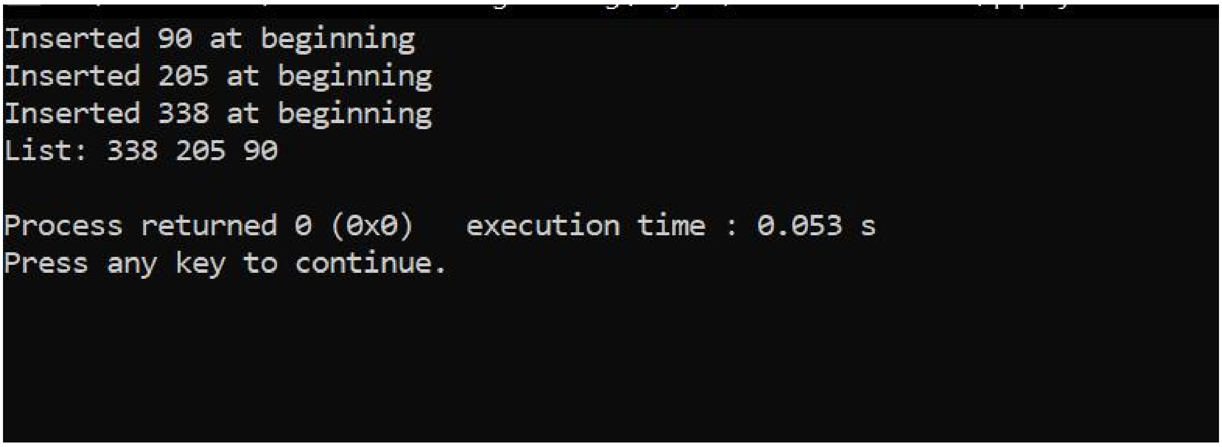
**Solution:**

**Corresponding code:**

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

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

void link_list(Node** head, int data) {
        Node* newNode = (Node*)malloc(sizeof(Node));
        if (newNode == NULL) {
                printf("Memory allocation failed\n");
                return;
        }
        newNode->data = data;
        newNode->next = *head;
        *head = newNode;
        printf("Inserted %d at beginning\n", data);
}
```

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

        printf("\n");
}

int main() {
        Node* head = NULL;

        link_list(&head, 90);
        link_list(&head, 205);
        link_list(&head, 338);

        printList(head);

        return 0;
}
```

**Output:**



```
Inserted 90 at beginning
Inserted 205 at beginning
Inserted 338 at beginning
List: 338 205 90

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

## Experiment No: 3

**Experiment name:** Write a C function to insert a node at the end of a singly linked list.

---

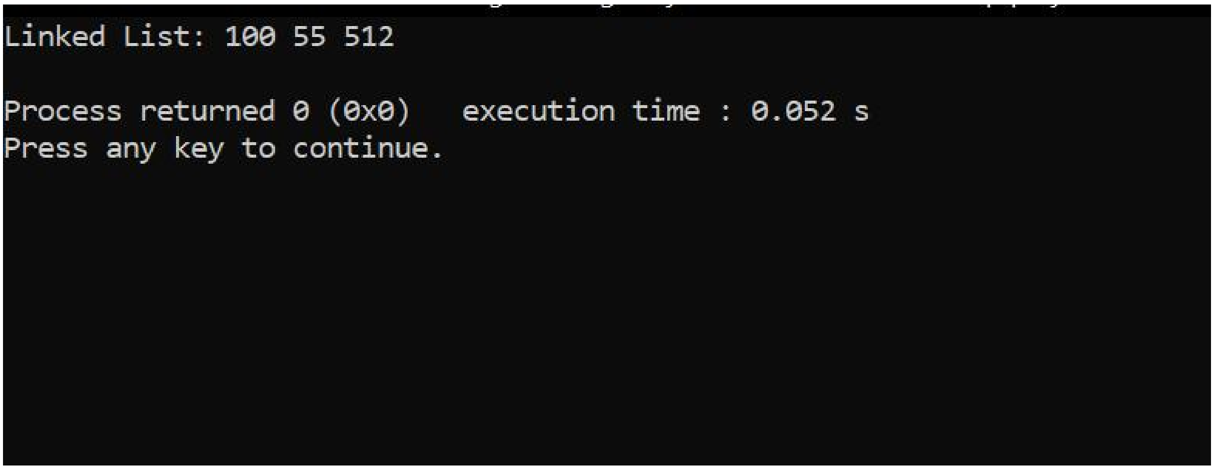
## Solution:

## Corresponding code:

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

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

void link_list(Node** head, int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;
	newNode->next = NULL;

	if (*head == NULL) {
		*head = newNode;
	} else {
		Node* temp = *head;
		while (temp->next != NULL) {
			temp = temp->next;
		}
		temp->next = newNode;
	}
}

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

}

int main() {
	Node* head = NULL;

	link_list(&head, 100);
```

```
        link_list(&head, 55);
        link_list(&head, 512);

        printf("Linked List: ");
        printList(head);

        return 0;
}
```

**Output:**



```
Linked List: 100 55 512

Process returned 0 (0x0)   execution time : 0.052 s
Press any key to continue.
```

**Experiment No: 4**
**Experiment name:** Write a C function to insert a new node after a specified node in a linked list.

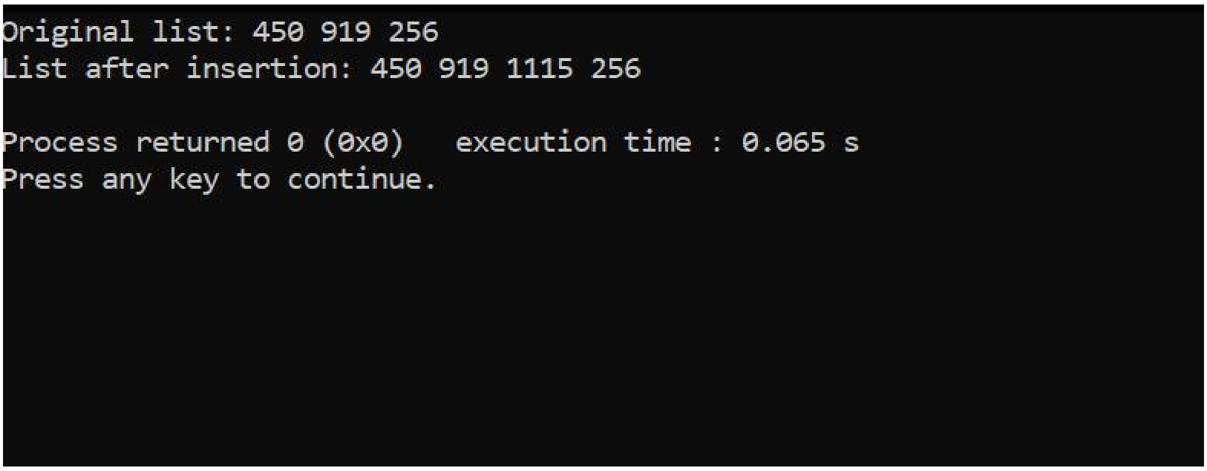
---

## Solution:
## Corresponding code:

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

typedef struct Node {

	int data;
	struct Node* next;
} Node;

void link_list(Node* prevNode, int newData) {
	if (prevNode == NULL) {
		printf("Previous node cannot be NULL\n");
		return;
	}

	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = newData;
	newNode->next = prevNode->next;
	prevNode->next = newNode;
}

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

int main() {
	Node* head = NULL;
	Node* second = NULL;
	Node* third = NULL;

	head = (Node*)malloc(sizeof(Node));
	second = (Node*)malloc(sizeof(Node));
	third = (Node*)malloc(sizeof(Node));
```

```
    head->data = 450;
    head->next = second;

    second->data = 919;
    second->next = third;

    third->data = 256;
    third->next = NULL;

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

    link_list(second, 1115);

    printf("List after insertion: ");
    printList(head);

    return 0;
}
```

**Output:**



```
Original list: 450 919 256
List after insertion: 450 919 1115 256

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

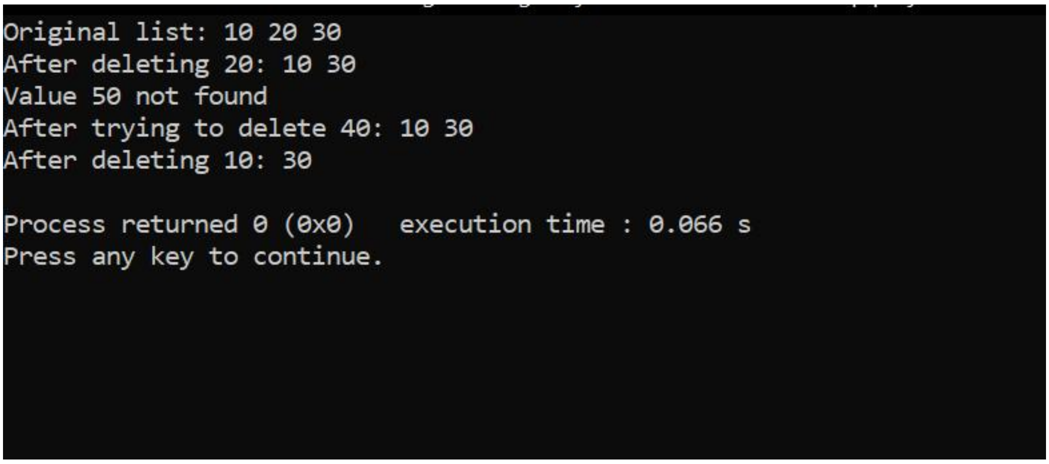
**Experiment No: 5**
**Experiment name:** Write a C function to delete a node with a specific value from a linked list.

---

**Solution:**
**Corresponding code:**

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

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

void link_list(Node** head, int key){
        Node* temp = *head;
        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) {
                printf("Value %d not found\n", key);
                return;
        }

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

}

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

void push(Node** head, int new_data) {
        Node* new_node = (Node*)malloc(sizeof(Node));
        new_node->data = new_data;
        new_node->next = *head;
        *head = new_node;
}

int main() {
        Node* head = NULL;

        push(&head, 30);
        push(&head, 20);
        push(&head, 10);

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

        link_list(&head, 20);
        printf("After deleting 20: ");
        printList(head);

        link_list(&head, 50);
        printf("After trying to delete 40: ");

        printList(head);

        link_list(&head, 10);
        printf("After deleting 10: ");
        printList(head);

        return 0;
}
```

**<u>Output:</u>**



```
Original list: 10 20 30
After deleting 20: 10 30
Value 50 not found
After trying to delete 40: 10 30
After deleting 10: 30

Process returned 0 (0x0)   execution time : 0.066 s
Press any key to continue.
```

**Experiment No: 6**

**Experiment name:** Write a C function to update the value of a specific node in a linked list.

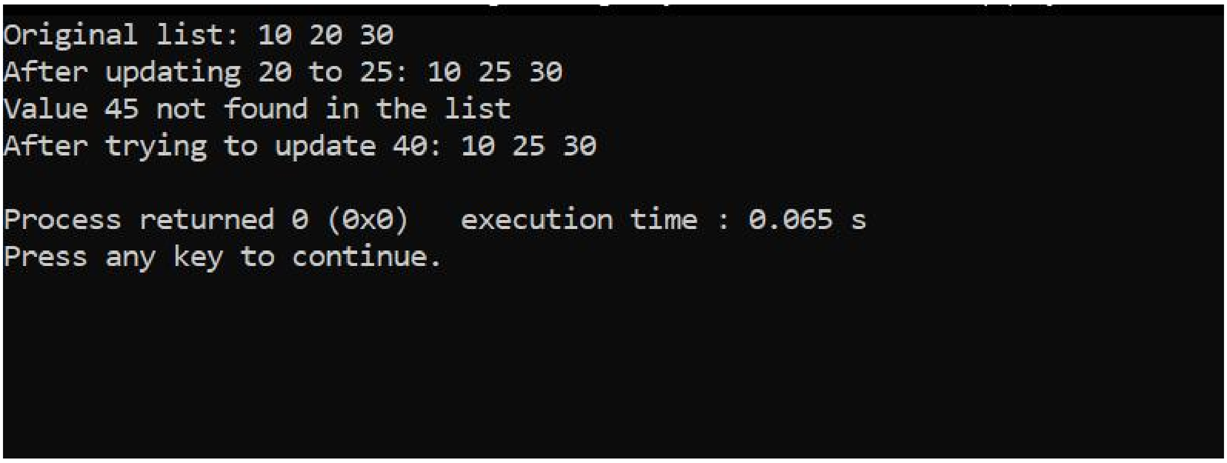
## Solution:

## Corresponding code:

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

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


void link_list(Node* head, int oldValue, int newValue) {
        Node* current = head;

        while (current != NULL) {
                if (current->data == oldValue) {
                        current->data = newValue;
                        return;
                }
                current = current->next;
        }

        printf("Value %d not found in the list\n", oldValue);
}

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

void push(Node** head, int new_data) {
        Node* new_node = (Node*)malloc(sizeof(Node));
        new_node->data = new_data;
        new_node->next = *head;
        *head = new_node;
}
```

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

        push(&head, 30);

        push(&head, 20);
        push(&head, 10);

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

        link_list(head, 20, 25);
        printf("After updating 20 to 25: ");
        printList(head);

        link_list(head, 45, 50);
        printf("After trying to update 40: ");
        printList(head);

        return 0;
}
```

**Output:**



```
Original list: 10 20 30
After updating 20 to 25: 10 25 30
Value 45 not found in the list
After trying to update 40: 10 25 30

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

**Experiment No: 7**

**Experiment name:** Write a C function to search for a given value in a linked list and return its position.

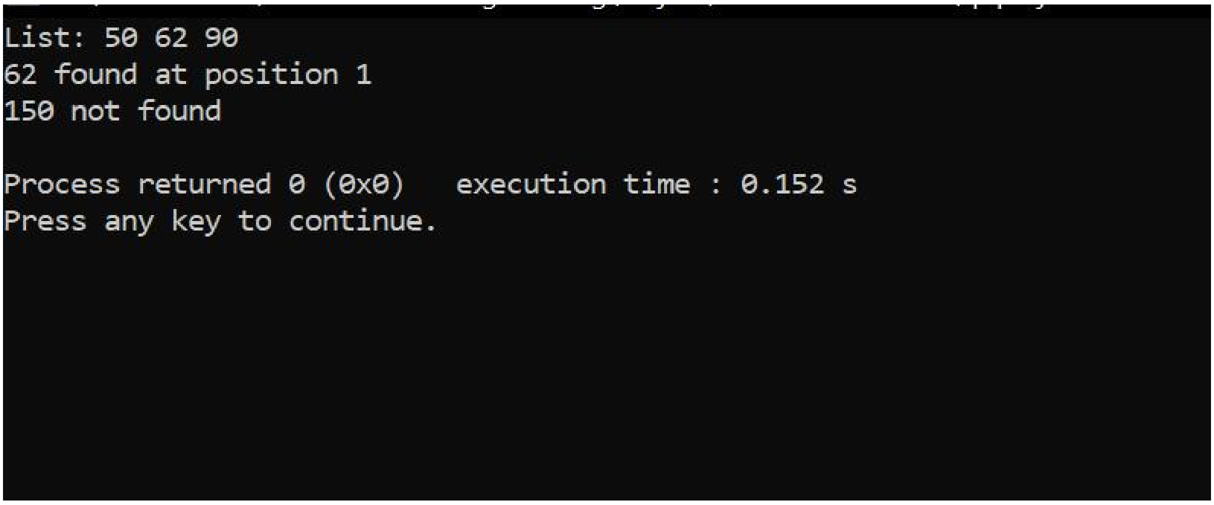
---

**Solution:**

**Corresponding code:**

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

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

int link_list(Node* head, int value) {
    int position = 0;
    Node* current = head;

    while (current != NULL) {
        if (current->data == value) {
            return position;
        }
        current = current->next;
        position++;
    }

    return -1;
}

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

    printf("\n");
}

void push(Node** head, int new_data) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

int main() {
    Node* head = NULL;

    push(&head, 90);
    push(&head, 62);
    push(&head, 50);

    printf("List: ");
    printList(head);

    int value = 62;
    int pos = link_list(head, value);
    if (pos != -1) {
        printf("%d found at position %d\n", value, pos);
    } else {
        printf("%d not found\n", value);
    }

    value = 150;
    pos = link_list(head, value);
    if (pos != -1) {
        printf("%d found at position %d\n", value, pos);
    } else {

        printf("%d not found\n", value);
    }

    return 0;
}
```

**Output:**



```
List: 50 62 90
62 found at position 1
150 not found

Process returned 0 (0x0)   execution time : 0.152 s
Press any key to continue.
```

**Experiment No: 8**
**Experiment name:** Write a C program to implement a doubly linked list with insertion and deletion operations.

---

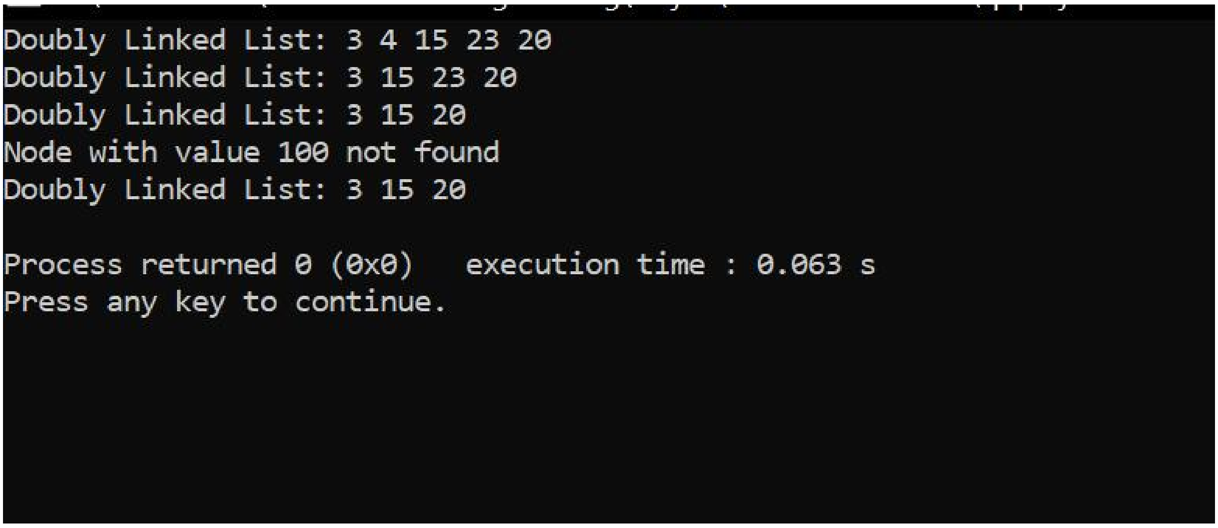
**Solution:**
**Corresponding code:**

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

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

Node* link_list(int data) {
        Node* newNode = (Node*)malloc(sizeof(Node));

        newNode->data = data;
        newNode->prev = NULL;
        newNode->next = NULL;
        return newNode;
}

void insertAtBeginning(Node** head, int data) {
        Node* newNode = link_list(data);
        if (*head == NULL) {
                *head = newNode;
                return;
        }
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
}

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

```
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void deleteNode(Node** head, int key) {
    if (*head == NULL) return;


    Node* temp = *head;

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

    if (temp == NULL) {
        printf("Node with value %d not found\n", key);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
}

void printList(Node* head) {
    printf("Doubly Linked List: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}


int main() {
    Node* head = NULL;

    insertAtEnd(&head, 15);
```

```
        insertAtBeginning(&head, 4);
        insertAtEnd(&head, 23);
        insertAtBeginning(&head, 3);
        insertAtEnd(&head, 20);

        printList(head);

        deleteNode(&head, 4);
        printList(head);

        deleteNode(&head, 23);
        printList(head);

        deleteNode(&head, 100);
        printList(head);

        return 0;
}
```

**Output:**



```
Doubly Linked List: 3 4 15 23 20
Doubly Linked List: 3 15 23 20
Doubly Linked List: 3 15 20
Node with value 100 not found
Doubly Linked List: 3 15 20

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

## Experiment No: 9

**Experiment name:** Draw the memory representation of the following linked list:
8 → 2 → 4 → 3 → 7
Insert 9 between 4 and 3, then delete 7, and update the diagram accordingly.

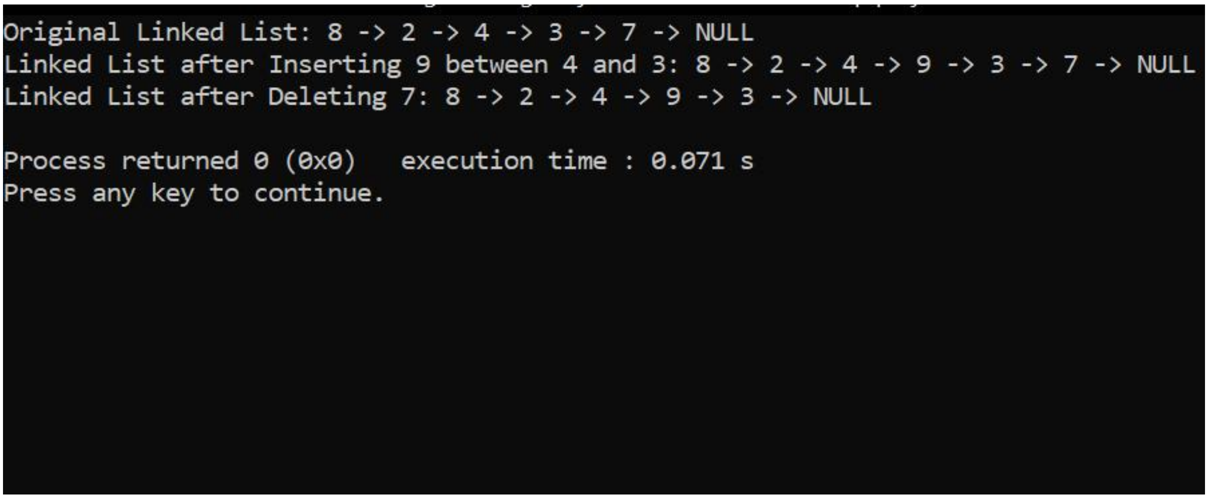
---

## Solution:

## Corresponding code:

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

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


void insertAfter(struct Node* prevNode, int newData) {
        if (prevNode == NULL) {
                printf("The previous node cannot be NULL.\n");
                return;
        }

        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = newData;
        newNode->next = prevNode->next;
        prevNode->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* node) {
        while (node != NULL) {
                printf("%d -> ", node->data);
                node = node->next;
        }
        printf("NULL\n");
}

int main() {

        struct Node* head = (struct Node*)malloc(sizeof(struct Node));
        head->data = 8;
        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 = 4;
        head->next->next->next = (struct Node*)malloc(sizeof(struct Node));
        head->next->next->next->data = 3;
        head->next->next->next->next = (struct Node*)malloc(sizeof(struct Node));
        head->next->next->next->next->data = 7;
        head->next->next->next->next->next = NULL;

        printf("Original Linked List: ");
        printList(head);


        struct Node* temp = head;
        while (temp != NULL && temp->data != 4) {
                temp = temp->next;
        }
        insertAfter(temp, 9);

        printf("Linked List after Inserting 9 between 4 and 3: ");
        printList(head);

        deleteNode(&head, 7);

        printf("Linked List after Deleting 7: ");
        printList(head);
        return 0;
}
```

## Output:



```
Original Linked List: 8 -> 2 -> 4 -> 3 -> 7 -> NULL
Linked List after Inserting 9 between 4 and 3: 8 -> 2 -> 4 -> 9 -> 3 -> 7 -> NULL
Linked List after Deleting 7: 8 -> 2 -> 4 -> 9 -> 3 -> NULL

Process returned 0 (0x0)   execution time : 0.071 s
Press any key to continue.
```