



NAME : KRIPANSH
SRIVASTAVA

BATCH NO : 46

SAP ID: 500122597

ENROL.NO: R21422543

CREATION OF ARRAY

1) STATIC

```
4  #include <stdio.h>
5
6  int main() {
7      // Declare a static array Let the size of array be 5
8      int arr[5] = {1, 2, 3, 4, 5};
9
10     // Print the elements
11     for(int i = 0; i < 5; i++) {
12         printf("%d ", arr[i]);
13     }
14     fflush(stdout);/// to clear the buffer
15
16     return 0;
17 }
```

FLUSH:- This function is used to clear output buffer.

OUTPUT :-

```
[Running] cd "c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA assignment\" && gcc DSACodes.c -o DSACodes &&
"c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA assignment\"DSACodes
1 2 3 4 5
[Done] exited with code=0 in 0.489 seconds
```

- With Pointer

```

4  #include <stdio.h>
5
6  int main() {
7      // Declare a static array of size 5
8      int arr[5] = {1, 2, 3, 4, 5};
9
10     // Declare a pointer and point it to the start of the array
11     int *ptr = arr;
12
13     // Print the elements
14     for(int i = 0; i < 5; i++) {
15         printf("%d ", *(ptr + i));
16     }
17     printf("\n"); // Add a newline character to flush the buffer
18
19     return 0;
20 }

```

• OUTPUT

```

[Running] cd "c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA assignment\" && gcc DSACodes.c -o DSACodes &&
"c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA assignment\DSACodes
1 2 3 4 5

```

• With Function and pointer

```

4  #include <stdio.h>
5
6  // Function to print elements of an array
7  void printArray(int *arr, int size) {
8      for(int i = 0; i < size; i++) {
9          printf("%d ", *(arr + i));
10     }
11     printf("\n"); // Add a newline character to flush the buffer
12 }
13
14 int main() {
15     // Declare a static array of size 5
16     int arr[5] = {1, 2, 3, 4, 5};
17
18     // Call the function to print the array
19     printArray(arr, 5);
20
21     return 0;
22 }

```

- OUTPUT

```
PROBLEMS    OUTPUT    DEBUG CONSOLE
[Running] cd "c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA"
1 2 3 4 5
```

- With Function

```
DSAcodes.c > main()
1  #include <stdio.h>
2
3  // Function to print elements of an array
4  void printArray(int arr[], int size) {
5      for(int i = 0; i < size; i++) {
6          printf("%d ", arr[i]);
7      }
8      printf("\n"); // Add a newline character to flush the buffer
9  }
10
11 int main() {
12     // Declare a static array of size 5
13     int arr[5] = {1, 2, 3, 4, 5};
14
15     // Call the function to print the array
16     printArray(arr, 5);
17
18     return 0;
19 }
```

- Output

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SERIAL MONITOR
[Running] cd "c:\Users\Kripansh Srivastava\OneDrive - UPES\Desktop\DSA"
1 2 3 4 5
```

• Dynamic

```
C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array
22         for(int i = 0; i < size; ++i) {
23             array[i] = i + 1;
24         }
25
26         // Print the array elements
27         printf("The elements of the array are: ");
28         for(int i = 0; i < size; ++i) {
29             printf("%d, ", array[i]);
30         }
31     }
32
33     return 0;
34 }
```

• Output

```
Enter size of the array: 12
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
...Program finished with exit code 0
Press ENTER to exit console.
```

- With Pointer

```
C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array
22         int *ptr = array;
23         for(int i = 0; i < size; ++i, ++ptr) {
24             *ptr = i + 1;
25         }
26
27         // Print the array elements
28         printf("The elements of the array are: ");
29         ptr = array;
30         for(int i = 0; i < size; ++i, ++ptr) {
31             printf("%d, ", *ptr);
32         }
33     }
34
35     return 0;
36 }
```

- OUTPUT

```
Enter size of the array: 8
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8,

...Program finished with exit code 0
Press ENTER to exit console.
```

- With Function

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fillAndPrintArray(int* array, int size) {
5      // Fill the array
6      for(int i = 0; i < size; ++i) {
7          array[i] = i + 1;
8      }
9
10     // Print the array elements
11     printf("The elements of the array are: ");
12     for(int i = 0; i < size; ++i) {
13         printf("%d, ", array[i]);
14     }
15 }
16
17 int main() {
18     int *array;
19     int size;
20
21     printf("Enter size of the array: ");
22     scanf("%d", &size);
23
24     // Allocate memory for the array
25     array = (int*) malloc(size * sizeof(int));
26
27     if(array == NULL) {
28         printf("Memory not allocated.\n");
29         exit(0);
30     }
31     else {
32         printf("Memory successfully allocated using malloc.\n");
33
34         fillAndPrintArray(array, size);
35     }
36
37     return 0;
38 }
```

- Output

```
Enter size of the array: 7
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7,
```

- With Pointer and Function

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void fillAndPrintArray(int* array, int size) {
5      // Get elements of the array
6      int *ptr = array;
7      for(int i = 0; i < size; ++i, ++ptr) {
8          *ptr = i + 1;
9      }
10
11     // Print the array elements
12     printf("The elements of the array are: ");
13     ptr = array;
14     for(int i = 0; i < size; ++i, ++ptr) {
15         printf("%d, ", *ptr);
16     }
17 }
18
19 int main() {
20     int *array;
21     int size;
22
23     printf("Enter size of the array: ");
24     scanf("%d", &size);
25
26     // Allocate memory for the array
27     array = (int*) malloc(size * sizeof(int));
28
29     if(array == NULL) {
30         printf("Memory not allocated.\n");
31         exit(0);
32     }
33     else {
34         printf("Memory successfully allocated using malloc.\n");
35
36         fillAndPrintArray(array, size);
37     }
38
39     return 0;
40 }
```

- Output


```
Enter size of the array: 8
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8,
```

TRAVERSING OF AN ARRAY

- WITHOUT POINTER AND FUNCTION

```
C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array from the user
22         printf("Enter elements of the array: ");
23         for(int i = 0; i < size; ++i) {
24             scanf("%d", &array[i]);
25         }
26
27         // Print the array elements
28         printf("The elements of the array are: ");
29         for(int i = 0; i < size; ++i) {
30             printf("%d, ", array[i]);
31         }
32     }
33
34     return 0;
35 }
```

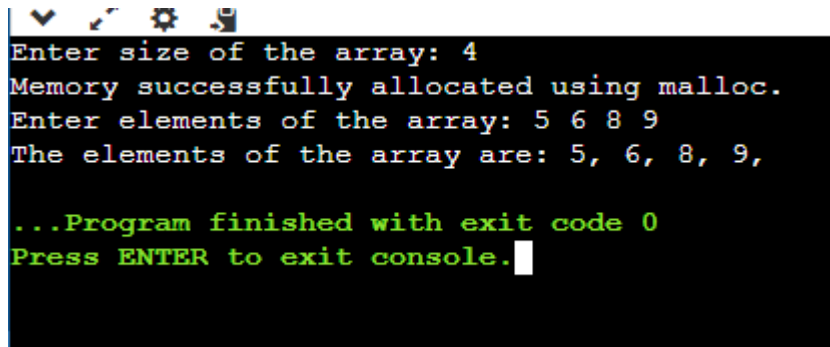
- Output

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter elements of the array: 15 4 4 5
The elements of the array are: 15, 4, 4, 5,
```

- With pointer

```
C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array from the user
22         printf("Enter elements of the array: ");
23         int *ptr = array;
24         for(int i = 0; i < size; ++i, ++ptr) {
25             scanf("%d", ptr);
26         }
27
28         // Print the array elements
29         printf("The elements of the array are: ");
30         ptr = array;
31         for(int i = 0; i < size; ++i, ++ptr) {
32             printf("%d, ", *ptr);
33         }
34     }
35
36     return 0;
37 }
```

- Output

A terminal window with a black background and white and green text. The text shows the execution of a program that asks for array size, allocates memory, takes input, and prints the array elements.

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter elements of the array: 5 6 8 9
The elements of the array are: 5, 6, 8, 9,

...Program finished with exit code 0
Press ENTER to exit console.
```

- With Function

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array from the user
22         printf("Enter elements of the array: ");
23         int *ptr = array;
24         for(int i = 0; i < size; ++i, ++ptr) {
25             scanf("%d", ptr);
26         }
27
28         // Print the array elements
29         printf("The elements of the array are: ");
30         ptr = array;
31         for(int i = 0; i < size; ++i, ++ptr) {
32             printf("%d, ", *ptr);
33         }
34     }
35
36     return 0;
37 }

```

- Output

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter elements of the array: 8 7 5 6
The elements of the array are: 8, 7, 5, 6,

...Program finished with exit code 0
Press ENTER to exit console.
```

- With Pointers

```
C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getAndPrintArray(int* array, int size) {
5      // Get elements of the array from the user
6      printf("Enter elements of the array: ");
7      int *ptr = array;
8      for(int i = 0; i < size; ++i, ++ptr) {
9          scanf("%d", ptr);
10     }
11
12     // Print the array elements
13     printf("The elements of the array are: ");
14     ptr = array;
15     for(int i = 0; i < size; ++i, ++ptr) {
16         printf("%d, ", *ptr);
17     }
18 }
19
20 int main() {
21     int *array;
22     int size;
23
24     printf("Enter size of the array: ");
25     scanf("%d", &size);
26
27     // Allocate memory for the array
28     array = (int*) malloc(size * sizeof(int));
29
30     if(array == NULL) {
31         printf("Memory not allocated.\n");
32         exit(0);
33     }
34     else {
35         printf("Memory successfully allocated using malloc.\n");
36
37         getAndPrintArray(array, size);
38     }
39
40     return 0;
41 }
```

And Functions

- Output

```
Enter size of the array: 7
Memory successfully allocated using malloc.
Enter elements of the array: 4 5 6 1 2 3 7
The elements of the array are: 4, 5, 6, 1, 2, 3, 7,

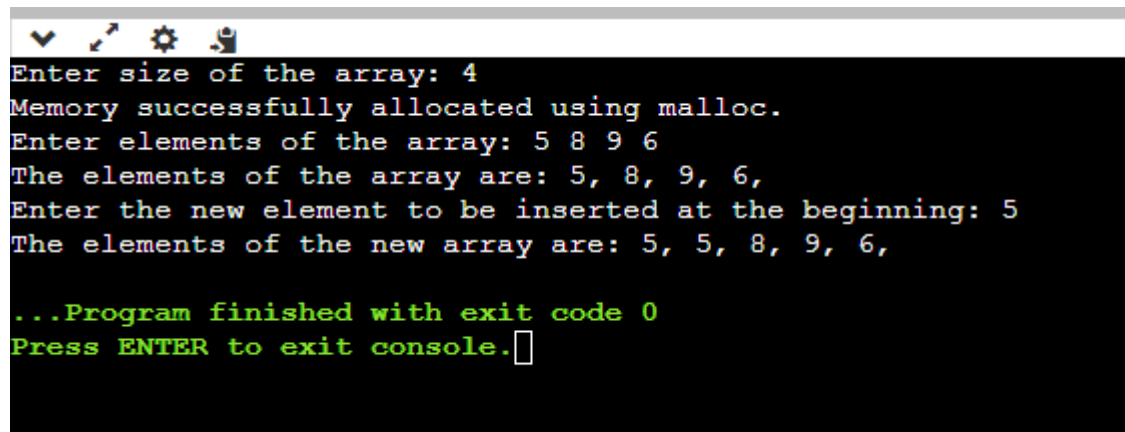
...Program finished with exit code 0
Press ENTER to exit console.
```

Insertion

1. In the Beginning

```
C dsa.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *array;
6      int size;
7
8      printf("Enter size of the array: ");
9      scanf("%d", &size);
10
11     // Allocate memory for the array
12     array = (int*) malloc(size * sizeof(int));
13
14     if(array == NULL) {
15         printf("Memory not allocated.\n");
16         exit(0);
17     }
18     else {
19         printf("Memory successfully allocated using malloc.\n");
20
21         // Get elements of the array from the user
22         printf("Enter elements of the array: ");
23         for(int i = 0; i < size; ++i) {
24             scanf("%d", &array[i]);
25         }
26
27         // Print the array elements
28         printf("The elements of the array are: ");
29         for(int i = 0; i < size; ++i) {
30             printf("%d, ", array[i]);
31         }
32
33         // Insert an element at the beginning of the array
34         int *newArray = (int*) malloc((size + 1) * sizeof(int));
35         if(newArray == NULL) {
36             printf("Memory not allocated for the new array.\n");
37             exit(0);
38         }
39
40         printf("\nEnter the new element to be inserted at the beginning: ");
41         int newElement;
42         scanf("%d", &newElement);
43
44         newArray[0] = newElement;
45         for(int i = 0; i < size; ++i) {
46             newArray[i + 1] = array[i];
47         }
48
49         free(array);
50         array = newArray;
51         size++;
52
53         // Print the new array elements
54         printf("The elements of the new array are: ");
55         for(int i = 0; i < size; ++i) {
56             printf("%d, ", array[i]);
57         }
58     }
59
60     return 0;
```

- Output

A screenshot of a terminal window with a dark background and light-colored text. The window has a title bar with standard icons (minimize, maximize, close, and a gear icon). The text inside the terminal shows the execution of a program that dynamically allocates memory and inserts an element at the beginning of an array. The output is as follows:

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter elements of the array: 5 8 9 6
The elements of the array are: 5, 8, 9, 6,
Enter the new element to be inserted at the beginning: 5
The elements of the new array are: 5, 5, 8, 9, 6,

...Program finished with exit code 0
Press ENTER to exit console.
```

- With Pointer


```

int main() {
    else {
        printf("Memory successfully allocated using malloc.\n");

        // Get elements of the array from the user
        for(int i = 0; i < size; ++i) {
            printf("Enter element %d: ", i);
            scanf("%d", &array[i]);
            // Flush the input buffer
            while (getchar() != '\n');
        }

        // Print the array elements
        printf("The elements of the array are: ");
        for(int i = 0; i < size; ++i) {
            printf("%d", array[i]);
            if(i < size - 1) {
                printf(", ");
            }
        }
        printf("\n");

        // Insert an element at the beginning of the array
        int *newArray = (int*) malloc((size + 1) * sizeof(int));
        if(newArray == NULL) {
            printf("Memory not allocated for the new array.\n");
            exit(0);
        }

        printf("\nEnter the new element to be inserted at the beginning: ");
        int newElement;
        scanf("%d", &newElement);
        // Flush the input buffer
        while (getchar() != '\n');

        newArray[0] = newElement;
        for(int i = 0; i < size; ++i) {
            newArray[i + 1] = array[i];
        }

        free(array);
        array = newArray;
        size++;

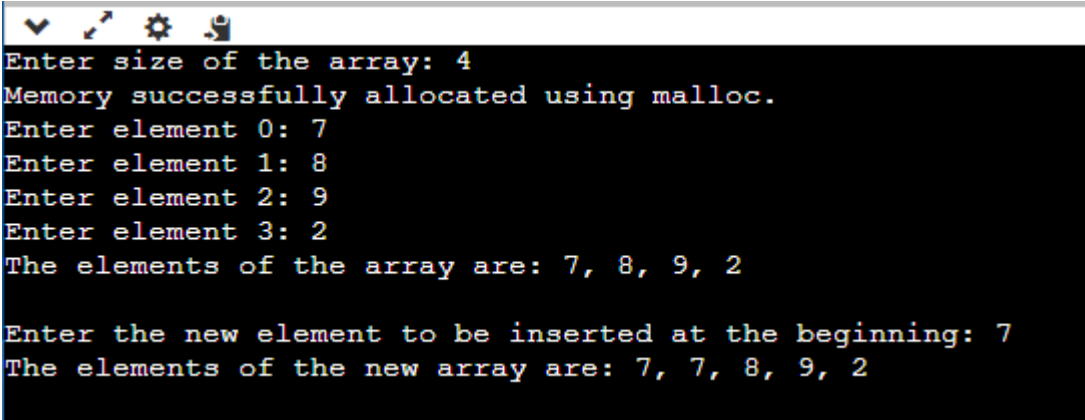
        // Print the new array elements
        printf("The elements of the new array are: ");
        for(int i = 0; i < size; ++i) {
            printf("%d", array[i]);
            if(i < size - 1) {
                printf(", ");
            }
        }
        printf("\n");
    }

    // Free the allocated memory
    free(array);

    return 0;
}

```

- OUTPUT

A terminal window with a black background and white text. The window has a title bar with standard Linux window controls (minimize, maximize, close) and a gear icon. The text inside the terminal shows a sequence of commands and outputs for an array program. The user enters the size of the array as 4, and the program confirms memory allocation. The user then enters four elements: 7, 8, 9, and 2. The program displays these elements. Next, the user enters a new element 7 to be inserted at the beginning, and the program displays the updated array: 7, 7, 8, 9, 2.

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 0: 7
Enter element 1: 8
Enter element 2: 9
Enter element 3: 2
The elements of the array are: 7, 8, 9, 2

Enter the new element to be inserted at the beginning: 7
The elements of the new array are: 7, 7, 8, 9, 2
```

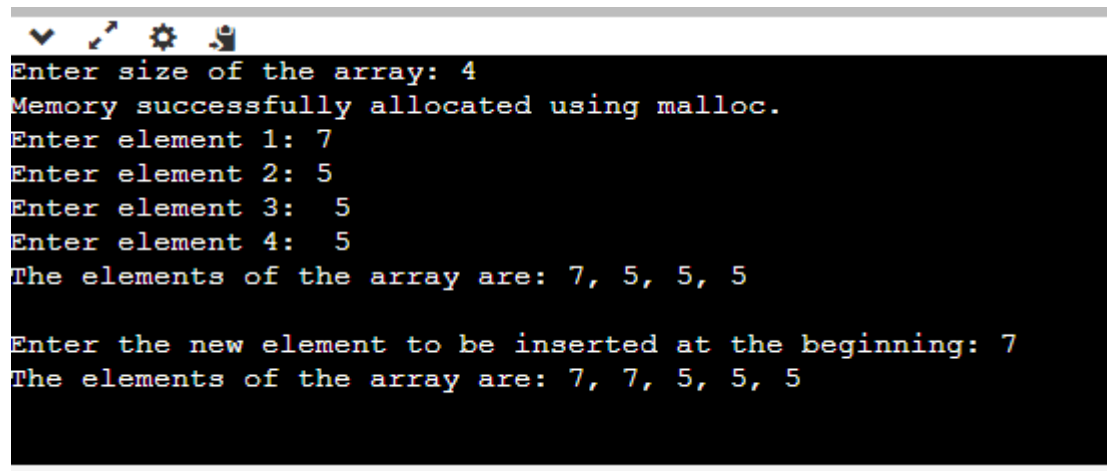
- With Function

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int array[], int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", &array[i]);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void insertAtBeginning(int array[], int size, int newElement) {
14     for(int i = size; i > 0; --i) {
15         array[i] = array[i - 1];
16     }
17     array[0] = newElement;
18 }
19
20 void printArray(int array[], int size) {
21     printf("The elements of the array are: ");
22     for(int i = 0; i < size; ++i) {
23         printf("%d", array[i]);
24         if(i < size - 1) {
25             printf(", ");
26         }
27     }
28     printf("\n");
29 }
30
31 int main() {
32     int size;
33
34     printf("Enter size of the array: ");
35     scanf("%d", &size);
36     // Flush the input buffer
37     while (getchar() != '\n');
38
39     // Allocate memory for the array
40     int *array = (int*) malloc((size + 1) * sizeof(int));
41
42     if(array == NULL) {
43         printf("Memory not allocated.\n");
44         exit(0);
45     }
46     else {
47         printf("Memory successfully allocated using malloc.\n");
48
49         // Get elements of the array from the user
50         getArray(array, size);
51
52         // Print the array elements
53         printArray(array, size);
54
55         // Insert an element at the beginning of the array
56         printf("\nEnter the new element to be inserted at the beginning: ");
57         int newElement;
58         scanf("%d", &newElement);
59         // Flush the input buffer
60         while (getchar() != '\n');
61
62         insertAtBeginning(array, size, newElement);
63         size++;
64
65         // Print the new array elements
66         printArray(array, size);
67     }
68
69     // Free the allocated memory
70     free(array);
71
72     return 0;
73 }

```

- Output

A terminal window with a dark background and light-colored text. The text shows the process of creating an array of size 4, allocating memory, entering elements 7, 5, 5, 5, and then inserting a new element 7 at the beginning, resulting in an array of 5 elements: 7, 7, 5, 5, 5.

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 7
Enter element 2: 5
Enter element 3: 5
Enter element 4: 5
The elements of the array are: 7, 5, 5, 5

Enter the new element to be inserted at the beginning: 7
The elements of the array are: 7, 7, 5, 5, 5
```

- With Pointer And Function

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int *array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void insertAtBeginning(int *array, int size, int newElement) {
14     for(int i = size; i > 0; --i) {
15         *(array + i) = *(array + i - 1);
16     }
17     *array = newElement;
18 }
19
20 void printArray(int *array, int size) {
21     printf("The elements of the array are: ");
22     for(int i = 0; i < size; ++i) {
23         printf("%d", *(array + i));
24         if(i < size - 1) {
25             printf(", ");
26         }
27     }
28     printf("\n");
29 }
30
31 int main() {
32     int size;
33
34     printf("Enter size of the array: ");
35     scanf("%d", &size);
36     // Flush the input buffer
37     while (getchar() != '\n');
38
39     // Allocate memory for the array
40     int *array = (int*) malloc((size + 1) * sizeof(int));
41
42     if(array == NULL) {
43         printf("Memory not allocated.\n");
44         exit(0);
45     }
46     else {
47         printf("Memory successfully allocated using malloc.\n");
48
49         // Get elements of the array from the user
50         getArray(array, size);
51
52         // Print the array elements
53         printArray(array, size);
54
55         // Insert an element at the beginning of the array
56         printf("\nEnter the new element to be inserted at the beginning: ");
57         int newElement;
58         scanf("%d", &newElement);
59         // Flush the input buffer
60         while (getchar() != '\n');
61
62         insertAtBeginning(array, size, newElement);
63         size++;
64
65         // Print the new array elements
66         printArray(array, size);
67     }
68
69     // Free the allocated memory
70     free(array);
71
72     return 0;
73 }

```

- Output

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 6
Enter element 2: 63
Enter element 3: 36
Enter element 4: 69
Enter element 5: 96
The elements of the array are: 6, 63, 36, 69, 96

Enter the new element to be inserted at the beginning: 699
The elements of the array are: 699, 6, 63, 36, 69, 96
```

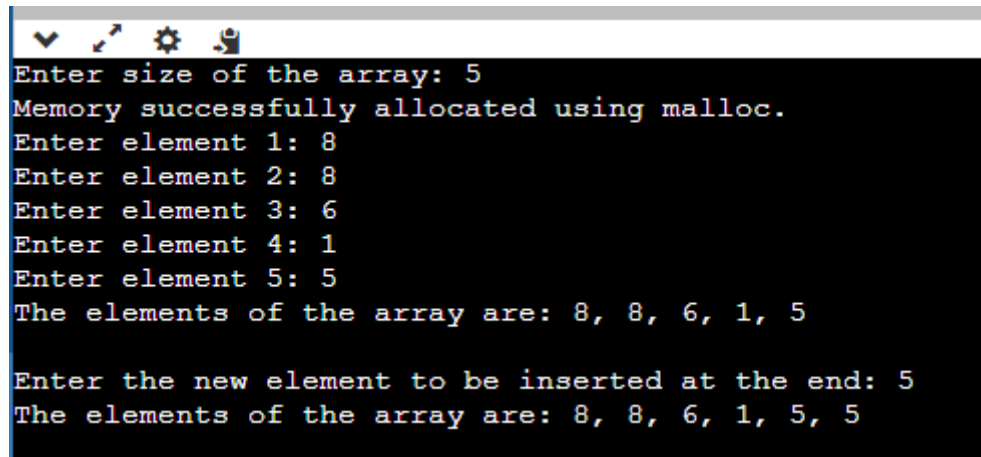
- IN the End

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc((size + 1) * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", &array[i]);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", array[i]);
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Insert an element at the end of the array
41         printf("\nEnter the new element to be inserted at the end: ");
42         int newElement;
43         scanf("%d", &newElement);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         array[size] = newElement;
48         size++;
49
50         // Print the new array elements
51         printf("The elements of the array are: ");
52         for(int i = 0; i < size; ++i) {
53             printf("%d", array[i]);
54             if(i < size - 1) {
55                 printf(", ");
56             }
57         }
58         printf("\n");
59     }
60
61     // Free the allocated memory
62     free(array);
63
64     return 0;
65 }

```

- Output

A terminal window with a dark background and light-colored text. The window has a title bar with standard icons (minimize, maximize, close, and a gear icon). The text inside the terminal shows a sequence of commands and their outputs for an array-based program.

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 8
Enter element 2: 8
Enter element 3: 6
Enter element 4: 1
Enter element 5: 5
The elements of the array are: 8, 8, 6, 1, 5

Enter the new element to be inserted at the end: 5
The elements of the array are: 8, 8, 6, 1, 5, 5
```

- With pointer


```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int *array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void insertAtBeginning(int *array, int size, int newElement) {
14     for(int i = size; i > 0; --i) {
15         *(array + i) = *(array + i - 1);
16     }
17     *array = newElement;
18 }
19
20 void printArray(int *array, int size) {
21     printf("The elements of the array are: ");
22     for(int i = 0; i < size; ++i) {
23         printf("%d", *(array + i));
24         if(i < size - 1) {
25             printf(", ");
26         }
27     }
28     printf("\n");
29 }
30
31 int main() {
32     int size;
33
34     printf("Enter size of the array: ");
35     scanf("%d", &size);
36     // Flush the input buffer
37     while (getchar() != '\n');
38
39     // Allocate memory for the array
40     int *array = (int*) malloc((size + 1) * sizeof(int));
41
42     if(array == NULL) {
43         printf("Memory not allocated.\n");
44         exit(0);
45     }
46     else {
47         printf("Memory successfully allocated using malloc.\n");
48
49         // Get elements of the array from the user
50         getArray(array, size);
51
52         // Print the array elements
53         printArray(array, size);
54
55         // Insert an element at the beginning of the array
56         printf("\nEnter the new element to be inserted at the beginning: ");
57         int newElement;
58         scanf("%d", &newElement);
59         // Flush the input buffer
60         while (getchar() != '\n');
61
62         insertAtBeginning(array, size, newElement);
63         size++;
64
65         // Print the new array elements
66         printArray(array, size);
67     }
68
69     // Free the allocated memory
70     free(array);
71
72     return 0;
73 }

```

- Output

```
Enter size of the array: 3
Memory successfully allocated using malloc.
Enter element 1: 5
Enter element 2: 69
Enter element 3: 96
The elements of the array are: 5, 69, 96

Enter the new element to be inserted at the end: 6
The elements of the array are: 5, 69, 96, 6
```

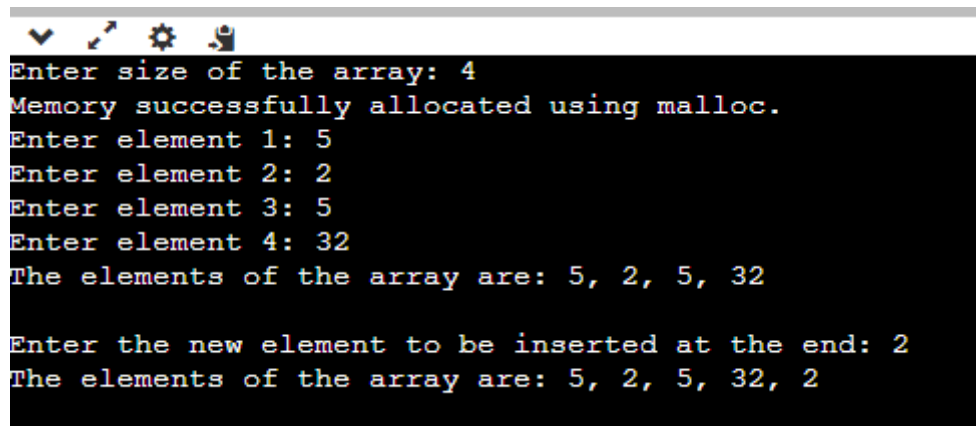
- With function

C dsac

C dsac > main()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int array[], int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", &array[i]);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void insertAtEnd(int array[], int size, int newElement) {
14     array[size] = newElement;
15 }
16
17 void printArray(int array[], int size) {
18     printf("The elements of the array are: ");
19     for(int i = 0; i < size; ++i) {
20         printf("%d", array[i]);
21         if(i < size - 1) {
22             printf(", ");
23         }
24     }
25     printf("\n");
26 }
27
28 int main() {
29     int size;
30
31     printf("Enter size of the array: ");
32     scanf("%d", &size);
33     // Flush the input buffer
34     while (getchar() != '\n');
35
36     // Allocate memory for the array
37     int *array = (int*) malloc((size + 1) * sizeof(int));
38
39     if(array == NULL) {
40         printf("Memory not allocated.\n");
41         exit(0);
42     }
43     else {
44         printf("Memory successfully allocated using malloc.\n");
45
46         // Get elements of the array from the user
47         getArray(array, size);
48
49         // Print the array elements
50         printArray(array, size);
51
52         // Insert an element at the end of the array
53         printf("\nEnter the new element to be inserted at the end: ");
54         int newElement;
55         scanf("%d", &newElement);
56         // Flush the input buffer
57         while (getchar() != '\n');
58
59         insertAtEnd(array, size, newElement);
60         size++;
61
62         // Print the new array elements
63         printArray(array, size);
64     }
65
66     // Free the allocated memory
67     free(array);
68
69     return 0;
70 }
```

- Output

A terminal window with a dark background and light-colored text. The text shows the process of creating an array, adding elements, and inserting a new element at the end. The window has a title bar with standard icons (checkmark, cursor, gear, and a person icon).

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 5
Enter element 2: 2
Enter element 3: 5
Enter element 4: 32
The elements of the array are: 5, 2, 5, 32

Enter the new element to be inserted at the end: 2
The elements of the array are: 5, 2, 5, 32, 2
```

- With function and Pointer

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int* array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void insertAtEnd(int* array, int size, int newElement) {
14     *(array + size) = newElement;
15 }
16
17 void printArray(int* array, int size) {
18     printf("The elements of the array are: ");
19     for(int i = 0; i < size; ++i) {
20         printf("%d", *(array + i));
21         if(i < size - 1) {
22             printf(", ");
23         }
24     }
25     printf("\n");
26 }
27
28 int main() {
29     int size;
30
31     printf("Enter size of the array: ");
32     scanf("%d", &size);
33     // Flush the input buffer
34     while (getchar() != '\n');
35
36     // Allocate memory for the array
37     int *array = (int*) malloc((size + 1) * sizeof(int));
38
39     if(array == NULL) {
40         printf("Memory not allocated.\n");
41         exit(0);
42     }
43     else {
44         printf("Memory successfully allocated using malloc.\n");
45
46         // Get elements of the array from the user
47         getArray(array, size);
48
49         // Print the array elements
50         printArray(array, size);
51
52         // Insert an element at the end of the array
53         printf("\nEnter the new element to be inserted at the end: ");
54         int newElement;
55         scanf("%d", &newElement);
56         // Flush the input buffer
57         while (getchar() != '\n');
58
59         insertAtEnd(array, size, newElement);
60         size++;
61
62         // Print the new array elements
63         printArray(array, size);
64     }
65
66     // Free the allocated memory
67     free(array);
68
69     return 0;
70 }

```

- Output

```
Enter size of the array: 2
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 2
The elements of the array are: 2, 2

Enter the new element to be inserted at the end: 1
The elements of the array are: 2, 2, 1
```

- At Particular Location

```

int main() {
    int size;

    printf("Enter size of the array: ");
    scanf("%d", &size);
    // Flush the input buffer
    while (getchar() != '\n');

    // Allocate memory for the array
    int *array = (int*) malloc((size + 1) * sizeof(int));

    if(array == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n");

        // Get elements of the array from the user
        for(int i = 0; i < size; ++i) {
            printf("Enter element %d: ", i + 1);
            scanf("%d", &array[i]);
            // Flush the input buffer
            while (getchar() != '\n');
        }

        // Print the array elements
        printf("The elements of the array are: ");
        for(int i = 0; i < size; ++i) {
            printf("%d", array[i]);
            if(i < size - 1) {
                printf(", ");
            }
        }
        printf("\n");

        // Insert an element at a particular location in the array
        printf("\nEnter the new element to be inserted: ");
        int newElement;
        scanf("%d", &newElement);
        // Flush the input buffer
        while (getchar() != '\n');

        printf("Enter the location at which the new element should be inserted (0 to %d): ", size);
        int location;
        scanf("%d", &location);
        // Flush the input buffer
        while (getchar() != '\n');

        // Shift the elements to the right of the insertion point
        for(int i = size; i > location; --i) {
            array[i] = array[i - 1];
        }

        // Insert the new element
        array[location] = newElement;
        size++;

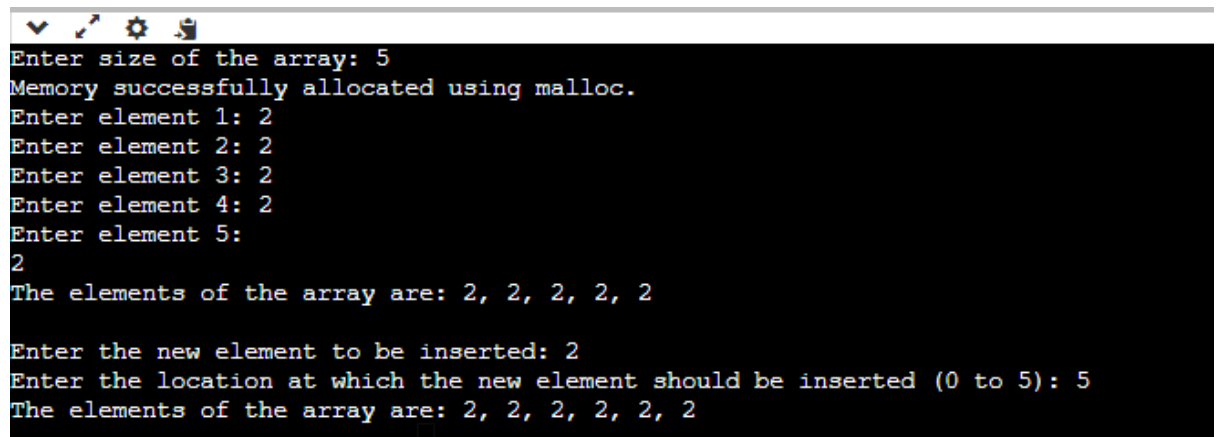
        // Print the new array elements
        printf("The elements of the array are: ");
        for(int i = 0; i < size; ++i) {
            printf("%d", array[i]);
            if(i < size - 1) {
                printf(", ");
            }
        }
        printf("\n");
    }

    // Free the allocated memory
    free(array);

    return 0;
}

```

- Output

A terminal window with a dark background and light-colored text. The window has a title bar with standard OS icons. The text inside shows a sequence of commands and outputs for an array program. The user enters '5' for the array size, and the program confirms memory allocation. Then, the user enters '2' five times for array elements. The program displays the array contents. Next, the user enters '2' for a new element and '5' for its insertion index. Finally, the program displays the updated array with six elements, all with the value 2.

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 2
Enter element 3: 2
Enter element 4: 2
Enter element 5:
2
The elements of the array are: 2, 2, 2, 2, 2

Enter the new element to be inserted: 2
Enter the location at which the new element should be inserted (0 to 5): 5
The elements of the array are: 2, 2, 2, 2, 2, 2
```

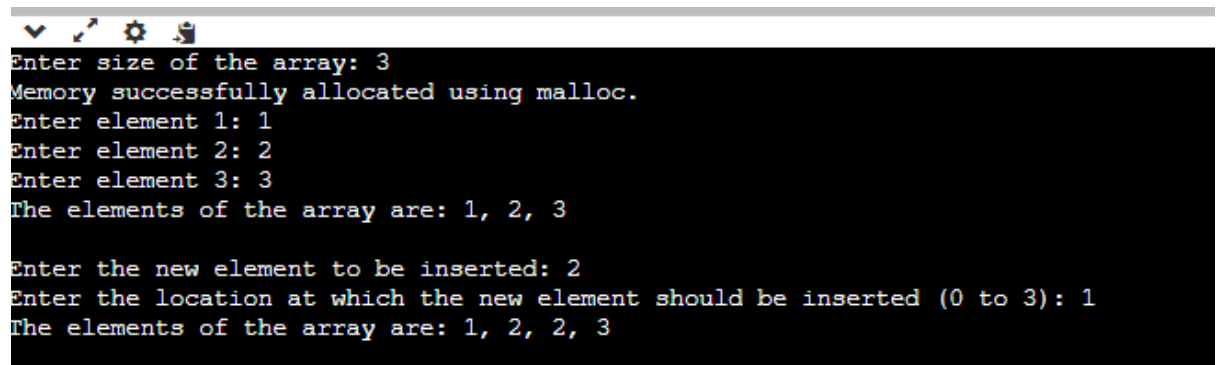
- With Pointer


```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc((size + 1) * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", array + i);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", *(array + i));
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Insert an element at a particular location in the array
41         printf("\nEnter the new element to be inserted: ");
42         int newElement;
43         scanf("%d", &newElement);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         printf("Enter the location at which the new element should be inserted (0 to %d): ", size);
48         int location;
49         scanf("%d", &location);
50         // Flush the input buffer
51         while (getchar() != '\n');
52
53         // Shift the elements to the right of the insertion point
54         for(int i = size; i > location; --i) {
55             *(array + i) = *(array + i - 1);
56         }
57
58         // Insert the new element
59         *(array + location) = newElement;
60         size++;
61
62         // Print the new array elements
63         printf("The elements of the array are: ");
64         for(int i = 0; i < size; ++i) {
65             printf("%d", *(array + i));
66             if(i < size - 1) {
67                 printf(", ");
68             }
69         }
70         printf("\n");
71     }
72
73     // Free the allocated memory
74     free(array);
75

```

- Output

A terminal window with a dark background and light-colored text. The window has a title bar with standard OS icons (back, forward, search, etc.). The text inside shows a sequence of commands and outputs for an array program. The user enters '3' for the array size, and the program confirms memory allocation. Then, the user enters '1', '2', and '3' for the first three elements. The program displays '1, 2, 3'. Next, the user enters '2' for a new element and '1' for its insertion index. The program then displays '1, 2, 2, 3'.

```
Enter size of the array: 3
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
The elements of the array are: 1, 2, 3

Enter the new element to be inserted: 2
Enter the location at which the new element should be inserted (0 to 3): 1
The elements of the array are: 1, 2, 2, 3
```

- With Function

```

void getArray(int array[], int size) {
}

void insertAtLocation(int array[], int size, int newElement, int location) {
    // Shift the elements to the right of the insertion point
    for(int i = size; i > location; --i) {
        array[i] = array[i - 1];
    }

    // Insert the new element
    array[location] = newElement;
}

void printArray(int array[], int size) {
    printf("The elements of the array are: ");
    for(int i = 0; i < size; ++i) {
        printf("%d", array[i]);
        if(i < size - 1) {
            printf(", ");
        }
    }
    printf("\n");
}

int main() {
    int size;

    printf("Enter size of the array: ");
    scanf("%d", &size);
    // Flush the input buffer
    while (getchar() != '\n');

    // Allocate memory for the array
    int *array = (int*) malloc((size + 1) * sizeof(int));

    if(array == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n");

        // Get elements of the array from the user
        getArray(array, size);

        // Print the array elements
        printArray(array, size);

        // Insert an element at a particular location in the array
        printf("\nEnter the new element to be inserted: ");
        int newElement;
        scanf("%d", &newElement);
        // Flush the input buffer
        while (getchar() != '\n');

        printf("Enter the location at which the new element should be inserted (0 to %d): ", size);
        int location;
        scanf("%d", &location);
        // Flush the input buffer
        while (getchar() != '\n');

        insertAtLocation(array, size, newElement, location);
        size++;

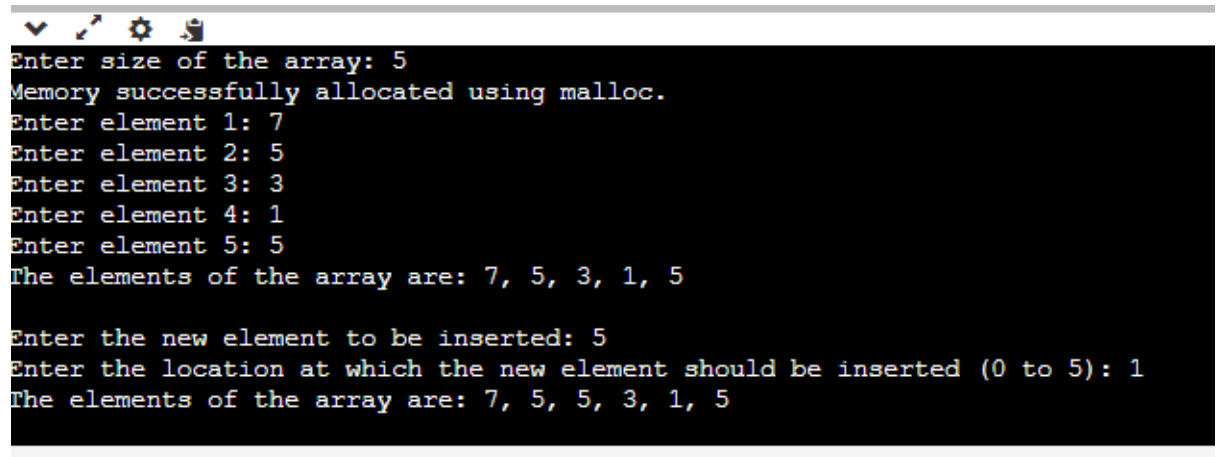
        // Print the new array elements
        printArray(array, size);
    }

    // Free the allocated memory
    free(array);

    return 0;
}

```

- Output

A terminal window with a black background and white text. The window has a title bar with standard Linux window controls (minimize, maximize, close) and a gear icon for settings. The text inside the terminal shows a sequence of commands and their outputs for an array program. The user enters the size of the array (5), and the program confirms memory allocation using malloc. Then, the user enters five elements: 7, 5, 3, 1, and 5. The program displays these elements. Next, the user enters a new element (5) and a location (1) for insertion. The program then displays the updated array: 7, 5, 5, 3, 1, 5.

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 7
Enter element 2: 5
Enter element 3: 3
Enter element 4: 1
Enter element 5: 5
The elements of the array are: 7, 5, 3, 1, 5

Enter the new element to be inserted: 5
Enter the location at which the new element should be inserted (0 to 5): 1
The elements of the array are: 7, 5, 5, 3, 1, 5
```

- With Function and Pointer

```

C dsa.c > main()
4   void getArray(int *array, int size) {
10      }
11  }
12
13  void insertAtLocation(int *array, int size, int newElement, int location) {
14      // Shift the elements to the right of the insertion point
15      for(int i = size; i > location; --i) {
16          *(array + i) = *(array + i - 1);
17      }
18
19      // Insert the new element
20      *(array + location) = newElement;
21  }
22
23  void printArray(int *array, int size) {
24      printf("The elements of the array are: ");
25      for(int i = 0; i < size; ++i) {
26          printf("%d", *(array + i));
27          if(i < size - 1) {
28              printf(", ");
29          }
30      }
31      printf("\n");
32  }
33
34  int main() {
35      int size;
36
37      printf("Enter size of the array: ");
38      scanf("%d", &size);
39      // Flush the input buffer
40      while (getchar() != '\n');
41
42      // Allocate memory for the array
43      int *array = (int*) malloc((size + 1) * sizeof(int));
44
45      if(array == NULL) {
46          printf("Memory not allocated.\n");
47          exit(0);
48      }
49      else {
50          printf("Memory successfully allocated using malloc.\n");
51
52          // Get elements of the array from the user
53          getArray(array, size);
54
55          // Print the array elements
56          printArray(array, size);
57
58          // Insert an element at a particular location in the array
59          printf("\nEnter the new element to be inserted: ");
60          int newElement;
61          scanf("%d", &newElement);
62          // Flush the input buffer
63          while (getchar() != '\n');
64
65          printf("Enter the location at which the new element should be inserted (0 to %d): ", size);
66          int location;
67          scanf("%d", &location);
68          // Flush the input buffer
69          while (getchar() != '\n');
70
71          insertAtLocation(array, size, newElement, location);
72          size++;
73
74          // Print the new array elements
75          printArray(array, size);
76      }
77
78      // Free the allocated memory
79      free(array);
80
81      return 0;
82  }

```

- Output

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
Enter element 4: 4
The elements of the array are: 1, 2, 3, 4

Enter the new element to be inserted:
1
Enter the location at which the new element should be inserted (0 to 4): 0
The elements of the array are: 1, 1, 2, 3, 4
```

Deletion Of an Element

```

C dsa.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", &array[i]);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", array[i]);
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Delete an element at a particular location in the array
41         printf("\nEnter the location of the element to be deleted (0 to %d): ", size - 1);
42         int location;
43         scanf("%d", &location);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         // Shift the elements to the left of the deletion point
48         for(int i = location; i < size - 1; ++i) {
49             array[i] = array[i + 1];
50         }
51         size--;
52
53         // Print the new array elements
54         printf("The elements of the array are: ");
55         for(int i = 0; i < size; ++i) {
56             printf("%d", array[i]);
57             if(i < size - 1) {
58                 printf(", ");
59             }
60         }
61         printf("\n");
62     }
63
64     // Free the allocated memory
65     free(array);
66
67     return 0;
68 }
69

```

• OUTPUT

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 2
Enter element 3: 4
Enter element 4:
55
The elements of the array are: 1, 2, 4, 55

Enter the location of the element to be deleted (0 to 3): 2
The elements of the array are: 1, 2, 55
```

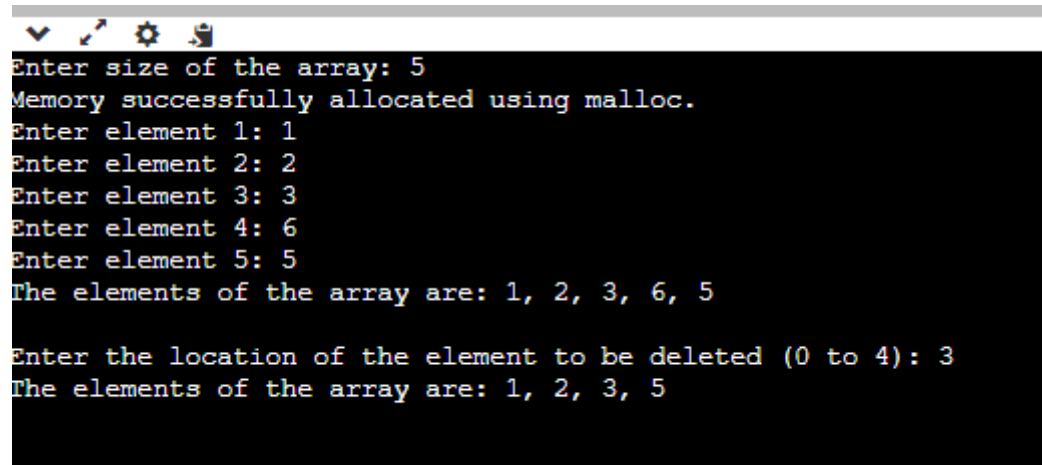
- WITH POINTER


```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", array + i);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", *(array + i));
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Delete an element at a particular location in the array
41         printf("\nEnter the location of the element to be deleted (0 to %d): ", size - 1);
42         int location;
43         scanf("%d", &location);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         // Shift the elements to the left of the deletion point
48         for(int i = location; i < size - 1; ++i) {
49             *(array + i) = *(array + i + 1);
50         }
51         size--;
52
53         // Print the new array elements
54         printf("The elements of the array are: ");
55         for(int i = 0; i < size; ++i) {
56             printf("%d", *(array + i));
57             if(i < size - 1) {
58                 printf(", ");
59             }
60         }
61         printf("\n");
62     }
63
64     // Free the allocated memory
65     free(array);
66
67     return 0;
68 }

```

- OUTPUT



```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
Enter element 4: 6
Enter element 5: 5
The elements of the array are: 1, 2, 3, 6, 5

Enter the location of the element to be deleted (0 to 4): 3
The elements of the array are: 1, 2, 3, 5
```

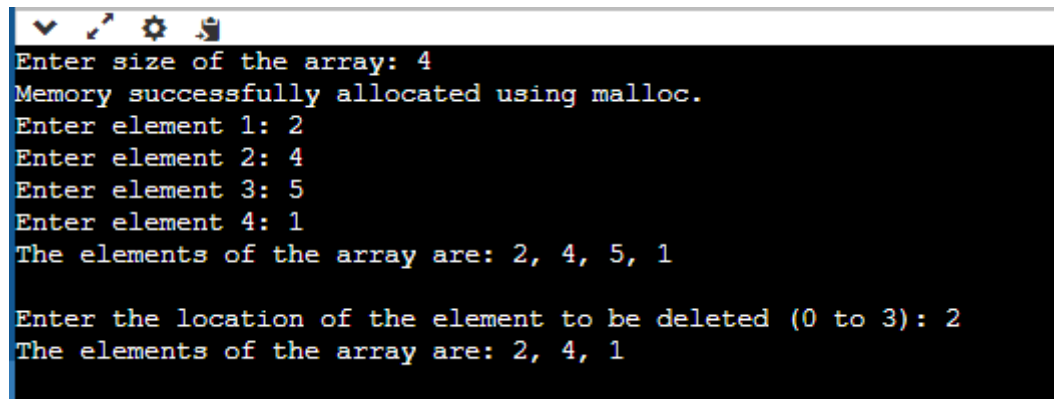
- WITH FUNCTION

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int array[], int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", &array[i]);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void deleteAtLocation(int array[], int *size, int location) {
14     // Shift the elements to the left of the deletion point
15     for(int i = location; i < *size - 1; ++i) {
16         array[i] = array[i + 1];
17     }
18     (*size)--;
19 }
20
21 void printArray(int array[], int size) {
22     printf("The elements of the array are: ");
23     for(int i = 0; i < size; ++i) {
24         printf("%d", array[i]);
25         if(i < size - 1) {
26             printf(", ");
27         }
28     }
29     printf("\n");
30 }
31
32 int main() {
33     int size;
34
35     printf("Enter size of the array: ");
36     scanf("%d", &size);
37     // Flush the input buffer
38     while (getchar() != '\n');
39
40     // Allocate memory for the array
41     int *array = (int*) malloc(size * sizeof(int));
42
43     if(array == NULL) {
44         printf("Memory not allocated.\n");
45         exit(0);
46     }
47     else {
48         printf("Memory successfully allocated using malloc.\n");
49
50         // Get elements of the array from the user
51         getArray(array, size);
52
53         // Print the array elements
54         printArray(array, size);
55
56         // Delete an element at a particular location in the array
57         printf("\nEnter the location of the element to be deleted (0 to %d): ", size - 1);
58         int location;
59         scanf("%d", &location);
60         // Flush the input buffer
61         while (getchar() != '\n');
62
63         deleteAtLocation(array, &size, location);
64
65         // Print the new array elements
66         printArray(array, size);
67     }
68
69     // Free the allocated memory
70     free(array);
71
72     return 0;
73 }

```

- OUTPUT

A terminal window with a black background and white text. The window has a title bar with standard Linux window controls (minimize, maximize, close) and a gear icon. The text inside the terminal shows the execution of a program that manages an array. It starts by asking for the array size (4), then prompts for four elements (2, 4, 5, 1). It then displays the array contents. Next, it asks for the index of an element to delete (2), and finally displays the updated array contents (2, 4, 1).

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 4
Enter element 3: 5
Enter element 4: 1
The elements of the array are: 2, 4, 5, 1

Enter the location of the element to be deleted (0 to 3): 2
The elements of the array are: 2, 4, 1
```

- WITH POINTER AND FUNCTION

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int *array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void deleteAtLocation(int *array, int *size, int location) {
14     // Shift the elements to the left of the deletion point
15     for(int i = location; i < *size - 1; ++i) {
16         *(array + i) = *(array + i + 1);
17     }
18     (*size)--;
19 }
20
21 void printArray(int *array, int size) {
22     printf("The elements of the array are: ");
23     for(int i = 0; i < size; ++i) {
24         printf("%d", *(array + i));
25         if(i < size - 1) {
26             printf(", ");
27         }
28     }
29     printf("\n");
30 }
31
32 int main() {
33     int size;
34
35     printf("Enter size of the array: ");
36     scanf("%d", &size);
37     // Flush the input buffer
38     while (getchar() != '\n');
39
40     // Allocate memory for the array
41     int *array = (int*) malloc(size * sizeof(int));
42
43     if(array == NULL) {
44         printf("Memory not allocated.\n");
45         exit(0);
46     }
47     else {
48         printf("Memory successfully allocated using malloc.\n");
49
50         // Get elements of the array from the user
51         getArray(array, size);
52
53         // Print the array elements
54         printArray(array, size);
55
56         // Delete an element at a particular location in the array
57         printf("\nEnter the location of the element to be deleted (0 to %d): ", size - 1);
58         int location;
59         scanf("%d", &location);
60         // Flush the input buffer
61         while (getchar() != '\n');
62
63         deleteAtLocation(array, &size, location);
64
65         // Print the new array elements
66         printArray(array, size);
67     }
68
69     // Free the allocated memory
70     free(array);
71
72     return 0;
73 }

```

- OUTPUT

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 1
Enter element 3: 1
Enter element 4: 4
The elements of the array are: 2, 1, 1, 4

Enter the location of the element to be deleted (0 to 3): 2
The elements of the array are: 2, 1, 4
```

- Searching in an array

```

C dsac
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", &array[i]);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", array[i]);
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Search for an element in the array
41         printf("\nEnter the element to be searched: ");
42         int element;
43         scanf("%d", &element);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         int found = 0;
48         for(int i = 0; i < size; ++i) {
49             if(array[i] == element) {
50                 printf("Element %d found at location %d\n", element, i);
51                 found = 1;
52                 break;
53             }
54         }
55
56         if(!found) {
57             printf("Element %d not found in the array\n", element);
58         }
59     }
60
61     // Free the allocated memory
62     free(array);
63
64     return 0;
65 }

```

- OUTPUT

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 5
Enter element 2: 9
Enter element 3: 0
Enter element 4: 6
The elements of the array are: 5, 9, 0, 6

Enter the element to be searched: 9
Element 9 found at location 1
```

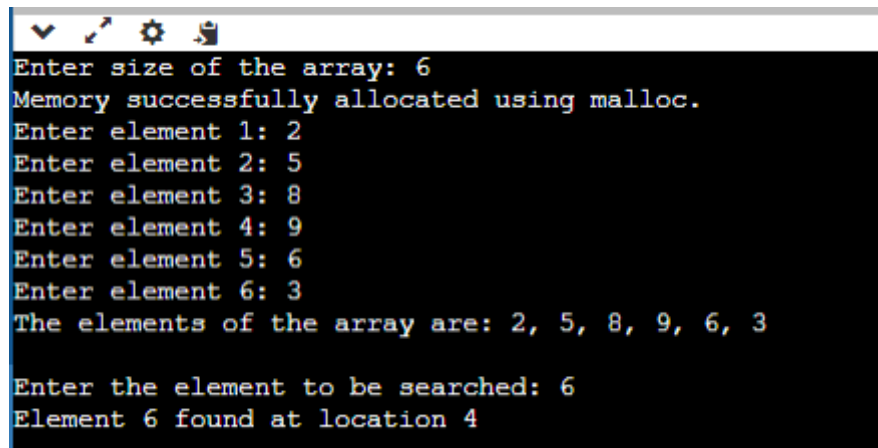
- WITH POINTERS


```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", array + i);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", *(array + i));
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Search for an element in the array
41         printf("\nEnter the element to be searched: ");
42         int element;
43         scanf("%d", &element);
44         // Flush the input buffer
45         while (getchar() != '\n');
46
47         int found = 0;
48         for(int i = 0; i < size; ++i) {
49             if(*(array + i) == element) {
50                 printf("Element %d found at location %d\n", element, i);
51                 found = 1;
52                 break;
53             }
54         }
55
56         if(!found) {
57             printf("Element %d not found in the array\n", element);
58         }
59     }
60
61     // Free the allocated memory
62     free(array);
63
64     return 0;
65 }

```

- OUTPUT

A terminal window with a dark background and light-colored text. The window has a title bar with standard icons (minimize, maximize, close, and a gear icon). The text inside the terminal shows the execution of a program that prompts for array size, allocates memory, takes six elements as input, displays them, and then searches for the value 6, finding it at index 4.

```
Enter size of the array: 6
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 5
Enter element 3: 8
Enter element 4: 9
Enter element 5: 6
Enter element 6: 3
The elements of the array are: 2, 5, 8, 9, 6, 3

Enter the element to be searched: 6
Element 6 found at location 4
```

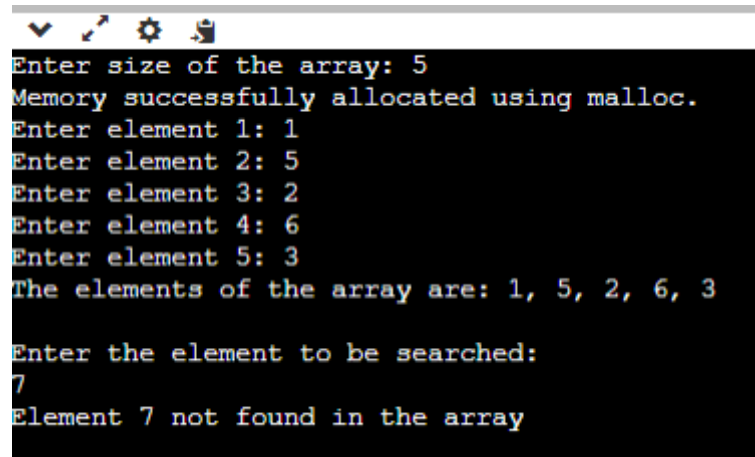
- WITH FUNCTION

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int *array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void printArray(int *array, int size) {
14     printf("The elements of the array are: ");
15     for(int i = 0; i < size; ++i) {
16         printf("%d", *(array + i));
17         if(i < size - 1) {
18             printf(", ");
19         }
20     }
21     printf("\n");
22 }
23
24 int searchElement(int *array, int size, int element) {
25     for(int i = 0; i < size; ++i) {
26         if(*(array + i) == element) {
27             return i;
28         }
29     }
30     return -1;
31 }
32
33 int main() {
34     int size;
35
36     printf("Enter size of the array: ");
37     scanf("%d", &size);
38     // Flush the input buffer
39     while (getchar() != '\n');
40
41     // Allocate memory for the array
42     int *array = (int*) malloc(size * sizeof(int));
43
44     if(array == NULL) {
45         printf("Memory not allocated.\n");
46         exit(0);
47     }
48     else {
49         printf("Memory successfully allocated using malloc.\n");
50
51         // Get elements of the array from the user
52         getArray(array, size);
53
54         // Print the array elements
55         printArray(array, size);
56
57         // Search for an element in the array
58         printf("\nEnter the element to be searched: ");
59         int element;
60         scanf("%d", &element);
61         // Flush the input buffer
62         while (getchar() != '\n');
63
64         int location = searchElement(array, size, element);
65
66         if(location != -1) {
67             printf("Element %d found at location %d\n", element, location);
68         }
69         else {
70             printf("Element %d not found in the array\n", element);
71         }
72     }
73
74     // Free the allocated memory
75     free(array);

```

- OUTPUT

A terminal window with a black background and white text. The window has a title bar with standard Linux window controls (minimize, maximize, close) and a gear icon. The text inside the terminal shows a sequence of commands and their outputs for an array program. The user enters the size of the array as 5, and the program confirms memory allocation. Then, the user enters five elements: 1, 5, 2, 6, and 3. The program displays these elements. Finally, the user enters 7 as the element to be searched, and the program outputs that the element was not found.

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 5
Enter element 3: 2
Enter element 4: 6
Enter element 5: 3
The elements of the array are: 1, 5, 2, 6, 3

Enter the element to be searched:
7
Element 7 not found in the array
```

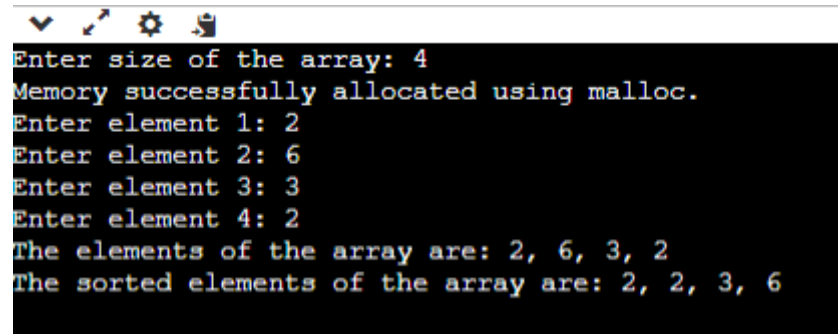
- Sort the element of an array

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", &array[i]);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", array[i]);
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Sort the array using Bubble Sort
41         for(int i = 0; i < size - 1; ++i) {
42             for(int j = 0; j < size - i - 1; ++j) {
43                 if(array[j] > array[j + 1]) {
44                     int temp = array[j];
45                     array[j] = array[j + 1];
46                     array[j + 1] = temp;
47                 }
48             }
49         }
50
51         // Print the sorted array
52         printf("The sorted elements of the array are: ");
53         for(int i = 0; i < size; ++i) {
54             printf("%d", array[i]);
55             if(i < size - 1) {
56                 printf(", ");
57             }
58         }
59         printf("\n");
60     }
61
62     // Free the allocated memory
63     free(array);
64
65     return 0;
66 }

```

•Output

A terminal window with a black background and white text. The text shows the process of entering array elements and the resulting sorted array. The window has a standard Linux terminal title bar with icons for window control and settings.

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 2
Enter element 2: 6
Enter element 3: 3
Enter element 4: 2
The elements of the array are: 2, 6, 3, 2
The sorted elements of the array are: 2, 2, 3, 6
```

• With pointer

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int size;
6
7      printf("Enter size of the array: ");
8      scanf("%d", &size);
9      // Flush the input buffer
10     while (getchar() != '\n');
11
12     // Allocate memory for the array
13     int *array = (int*) malloc(size * sizeof(int));
14
15     if(array == NULL) {
16         printf("Memory not allocated.\n");
17         exit(0);
18     }
19     else {
20         printf("Memory successfully allocated using malloc.\n");
21
22         // Get elements of the array from the user
23         for(int i = 0; i < size; ++i) {
24             printf("Enter element %d: ", i + 1);
25             scanf("%d", array + i);
26             // Flush the input buffer
27             while (getchar() != '\n');
28         }
29
30         // Print the array elements
31         printf("The elements of the array are: ");
32         for(int i = 0; i < size; ++i) {
33             printf("%d", *(array + i));
34             if(i < size - 1) {
35                 printf(", ");
36             }
37         }
38         printf("\n");
39
40         // Sort the array using Bubble Sort
41         for(int i = 0; i < size - 1; ++i) {
42             for(int j = 0; j < size - i - 1; ++j) {
43                 if(*(array + j) > *(array + j + 1)) {
44                     int temp = *(array + j);
45                     *(array + j) = *(array + j + 1);
46                     *(array + j + 1) = temp;
47                 }
48             }
49         }
50
51         // Print the sorted array
52         printf("The sorted elements of the array are: ");
53         for(int i = 0; i < size; ++i) {
54             printf("%d", *(array + i));
55             if(i < size - 1) {
56                 printf(", ");
57             }
58         }
59         printf("\n");
60     }
61
62     // Free the allocated memory
63     free(array);
64
65     return 0;
66 }

```

- Output

```
Enter size of the array: 5
Memory successfully allocated using malloc.
Enter element 1: 1
Enter element 2: 4
Enter element 3: 2
Enter element 4: 6
Enter element 5: 3
The elements of the array are: 1, 4, 2, 6, 3
The sorted elements of the array are: 1, 2, 3, 4, 6
```

- With Function


```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int array[], int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", &array[i]);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void printArray(int array[], int size) {
14     for(int i = 0; i < size; ++i) {
15         printf("%d", array[i]);
16         if(i < size - 1) {
17             printf(", ");
18         }
19     }
20     printf("\n");
21 }
22
23 void bubbleSort(int array[], int size) {
24     for(int i = 0; i < size - 1; ++i) {
25         for(int j = 0; j < size - i - 1; ++j) {
26             if(array[j] > array[j + 1]) {
27                 int temp = array[j];
28                 array[j] = array[j + 1];
29                 array[j + 1] = temp;
30             }
31         }
32     }
33 }
34
35 int main() {
36     int size;
37
38     printf("Enter size of the array: ");
39     scanf("%d", &size);
40     // Flush the input buffer
41     while (getchar() != '\n');
42
43     // Allocate memory for the array
44     int *array = (int*) malloc(size * sizeof(int));
45
46     if(array == NULL) {
47         printf("Memory not allocated.\n");
48         exit(0);
49     }
50     else {
51         printf("Memory successfully allocated using malloc.\n");
52
53         // Get elements of the array from the user
54         getArray(array, size);
55
56         // Print the array elements
57         printf("The elements of the array are: ");
58         printArray(array, size);
59
60         // Sort the array using Bubble Sort
61         bubbleSort(array, size);
62
63         // Print the sorted array
64         printf("The sorted elements of the array are: ");
65         printArray(array, size);
66     }
67
68     // Free the allocated memory
69     free(array);
70
71     return 0;
72 }

```

- Output

```
Enter size of the array: 2
Memory successfully allocated using malloc.
Enter element 1: 4
Enter element 2: 1
The elements of the array are: 4, 1
The sorted elements of the array are: 1, 4
```

- With function and Pointer

```

C dsa.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void getArray(int *array, int size) {
5      for(int i = 0; i < size; ++i) {
6          printf("Enter element %d: ", i + 1);
7          scanf("%d", array + i);
8          // Flush the input buffer
9          while (getchar() != '\n');
10     }
11 }
12
13 void printArray(int *array, int size) {
14     for(int i = 0; i < size; ++i) {
15         printf("%d", *(array + i));
16         if(i < size - 1) {
17             printf(", ");
18         }
19     }
20     printf("\n");
21 }
22
23 void bubbleSort(int *array, int size) {
24     for(int i = 0; i < size - 1; ++i) {
25         for(int j = 0; j < size - i - 1; ++j) {
26             if(*(array + j) > *(array + j + 1)) {
27                 int temp = *(array + j);
28                 *(array + j) = *(array + j + 1);
29                 *(array + j + 1) = temp;
30             }
31         }
32     }
33 }
34
35 int main() {
36     int size;
37
38     printf("Enter size of the array: ");
39     scanf("%d", &size);
40     // Flush the input buffer
41     while (getchar() != '\n');
42
43     // Allocate memory for the array
44     int *array = (int*) malloc(size * sizeof(int));
45
46     if(array == NULL) {
47         printf("Memory not allocated.\n");
48         exit(0);
49     }
50     else {
51         printf("Memory successfully allocated using malloc.\n");
52
53         // Get elements of the array from the user
54         getArray(array, size);
55
56         // Print the array elements
57         printf("The elements of the array are: ");
58         printArray(array, size);
59
60         // Sort the array using Bubble Sort
61         bubbleSort(array, size);
62
63         // Print the sorted array
64         printf("The sorted elements of the array are: ");
65         printArray(array, size);
66     }
67
68     // Free the allocated memory
69     free(array);
70
71     return 0;
72 }

```

- Output

```
Enter size of the array: 4
Memory successfully allocated using malloc.
Enter element 1: 5
Enter element 2: 2
Enter element 3: 6
Enter element 4: 3
The elements of the array are: 5, 2, 6, 3
The sorted elements of the array are: 2, 3, 5, 6
```

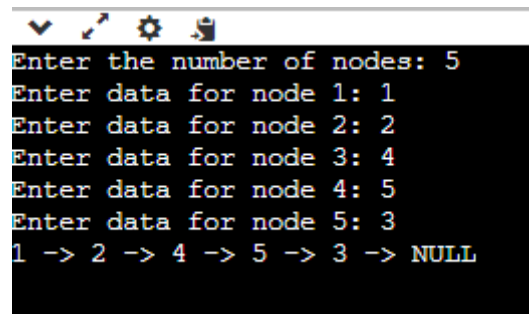
- Singular Linked List
 - Traversing

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Define the structure for a node
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Function to create a new node
11 Node* createNode(int data) {
12     Node* newNode = (Node*) malloc(sizeof(Node));
13     if(newNode == NULL) {
14         printf("Error! memory not allocated.");
15         exit(0);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // Function to add a node at the end of the list
23 void addNode(Node** head, int data) {
24     Node* newNode = createNode(data);
25     if(*head == NULL) {
26         *head = newNode;
27         return;
28     }
29     Node* temp = *head;
30     while(temp->next != NULL) {
31         temp = temp->next;
32     }
33     temp->next = newNode;
34 }
35
36 // Function to print the list
37 void printList(Node* head) {
38     Node* temp = head;
39     while(temp != NULL) {
40         printf("%d -> ", temp->data);
41         temp = temp->next;
42     }
43     printf("NULL\n");
44 }
45
46 int main() {
47     Node* head = NULL;
48
49     addNode(&head, 1);
50     addNode(&head, 2);
51     addNode(&head, 3);
52     addNode(&head, 4);
53     addNode(&head, 5);
54
55     printList(head);
56
57     return 0;
58 }

```

OUTPUT



A terminal window with a dark background and light blue text. The window has a title bar with standard icons (checkmark, cursor, gear, and a document). The text inside shows the process of creating a linked list with 5 nodes. The data for each node is entered sequentially, and the final output shows the sequence of nodes: 1 -> 2 -> 4 -> 5 -> 3 -> NULL.

```
Enter the number of nodes: 5
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 4
Enter data for node 4: 5
Enter data for node 5: 3
1 -> 2 -> 4 -> 5 -> 3 -> NULL
```

- Insertion

```

Node* createNode(int data) {
    printf("Error! memory not allocated.");
    exit(0);
}
newNode->data = data;
newNode->next = NULL;
return newNode;
}

// Function to add a node at the end of the list
void addNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if(*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while(temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to insert a node at a specific position
void insertNode(Node** head, int data, int position) {
    Node* newNode = createNode(data);
    if(position == 0) {
        newNode->next = *head;
        *head = newNode;
        return;
    }
    Node* temp = *head;
    for(int i = 0; i < position - 1; i++) {
        if(temp != NULL) {
            temp = temp->next;
        }
    }
    if(temp != NULL) {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

// Function to print the list
void printList(Node* head) {
    Node* temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    Node* head = NULL;
    int size, data, position;

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

    for(int i = 0; i < size; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        addNode(&head, data);
    }

    printf("Enter data and position for new node: ");
    scanf("%d %d", &data, &position);
    insertNode(&head, data, position);

    printList(head);

    return 0;
}

```

- Output

```
Enter the number of nodes: 4
Enter data for node 1: 2
Enter data for node 2: 1
Enter data for node 3: 4
Enter data for node 4: 5
Enter data and position for new node: 2

4
2 -> 1 -> 4 -> 5 -> 2 -> NULL
```

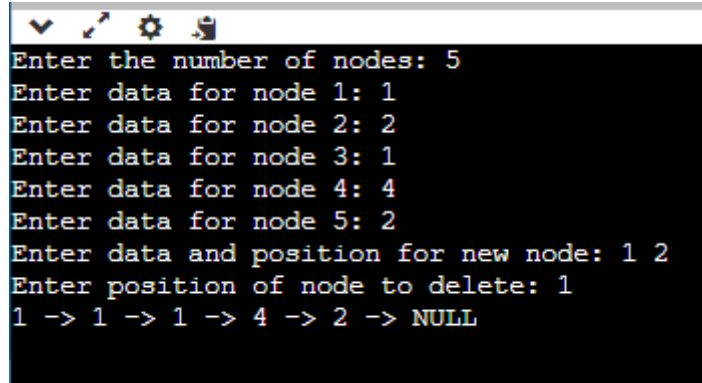
- Deletion


```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Define the structure for a node
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Function to create a new node
11 Node* createNode(int data) {
12     Node* newNode = (Node*) malloc(sizeof(Node));
13     if(newNode == NULL) {
14         printf("Error! memory not allocated.");
15         exit(0);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // Function to add a node at the end of the list
23 void addNode(Node** head, int data) {
24     Node* newNode = createNode(data);
25     if(*head == NULL) {
26         *head = newNode;
27         return;
28     }
29     Node* temp = *head;
30     while(temp->next != NULL) {
31         temp = temp->next;
32     }
33     temp->next = newNode;
34 }
35
36 // Function to insert a node at a specific position
37 void insertNode(Node** head, int data, int position) {
38     Node* newNode = createNode(data);
39     if(position == 0) {
40         newNode->next = *head;
41         *head = newNode;
42         return;
43     }
44     Node* temp = *head;
45     for(int i = 0; i < position - 1; i++) {
46         if(temp != NULL) {
47             temp = temp->next;
48         }
49     }
50     if(temp != NULL) {
51         newNode->next = temp->next;
52         temp->next = newNode;
53     }
54 }
55
56 // Function to delete a node at a specific position
57 void deleteNode(Node** head, int position) {
58     if(*head == NULL) {
59         return;
60     }
61     Node* temp = *head;
62     if(position == 0) {
63         *head = temp->next;
64         free(temp);
65         return;
66     }
67     for(int i = 0; temp != NULL && i < position - 1; i++) {
68         temp = temp->next;
69     }
70     if(temp == NULL || temp->next == NULL) {
71         return;
72     }
73     Node* next = temp->next->next;
74     free(temp->next);
75     temp->next = next;

```

- Output

A screenshot of a terminal window with a black background and white text. The window has a standard OS title bar with a checkmark, a cursor, a gear icon, and a document icon. The text in the terminal shows the user entering 5 nodes, then data for each node (1, 2, 1, 4, 2). Then, they enter '1 2' for a new node and '1' for the position to delete. The final output shows the linked list: 1 -> 1 -> 1 -> 4 -> 2 -> NULL.

```
Enter the number of nodes: 5
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 1
Enter data for node 4: 4
Enter data for node 5: 2
Enter data and position for new node: 1 2
Enter position of node to delete: 1
1 -> 1 -> 1 -> 4 -> 2 -> NULL
```

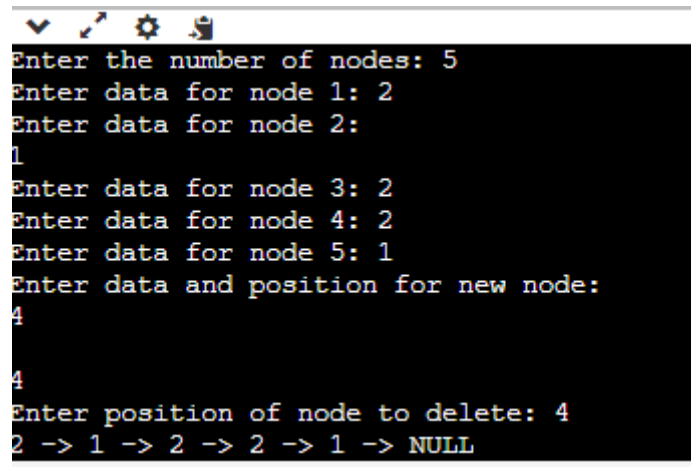
- Circular linked list
 - Traversing

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Define the structure for a node
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Function to create a new node
11 Node* createNode(int data) {
12     Node* newNode = (Node*) malloc(sizeof(Node));
13     if(newNode == NULL) {
14         printf("Error! memory not allocated.");
15         exit(0);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // Function to add a node at the end of the list
23 void addNode(Node** head, int data) {
24     Node* newNode = createNode(data);
25     if(*head == NULL) {
26         *head = newNode;
27         return;
28     }
29     Node* temp = *head;
30     while(temp->next != NULL) {
31         temp = temp->next;
32     }
33     temp->next = newNode;
34 }
35
36 // Function to insert a node at a specific position
37 void insertNode(Node** head, int data, int position) {
38     Node* newNode = createNode(data);
39     if(position == 0) {
40         newNode->next = *head;
41         *head = newNode;
42         return;
43     }
44     Node* temp = *head;
45     for(int i = 0; i < position - 1; i++) {
46         if(temp != NULL) {
47             temp = temp->next;
48         }
49     }
50     if(temp != NULL) {
51         newNode->next = temp->next;
52         temp->next = newNode;
53     }
54 }
55
56 // Function to delete a node at a specific position
57 void deleteNode(Node** head, int position) {
58     if(*head == NULL) {
59         return;
60     }
61     Node* temp = *head;
62     if(position == 0) {
63         *head = temp->next;
64         free(temp);
65         return;
66     }
67     for(int i = 0; temp != NULL && i < position - 1; i++) {
68         temp = temp->next;
69     }
70     if(temp == NULL || temp->next == NULL) {
71         return;
72     }
73     Node* next = temp->next->next;
74     free(temp->next);
75     temp->next = next;

```

- Output

A terminal window with a black background and white text. The text shows the execution of a program that creates a linked list, inserts a new node, and then prints the list. The output is as follows:

```
Enter the number of nodes: 5
Enter data for node 1: 2
Enter data for node 2:
1
Enter data for node 3: 2
Enter data for node 4: 2
Enter data for node 5: 1
Enter data and position for new node:
4
4
Enter position of node to delete: 4
2 -> 1 -> 2 -> 2 -> 1 -> NULL
```

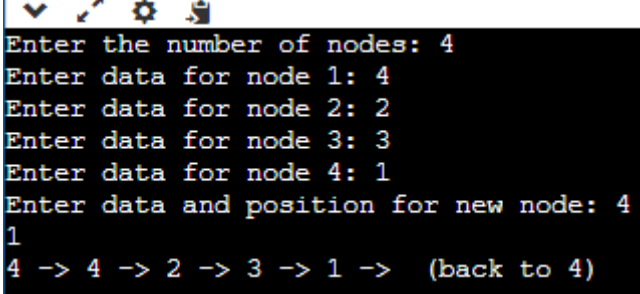
- INSERTION

```

C dsa.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Define the structure for a node
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Function to create a new node
11 Node* createNode(int data) {
12     Node* newNode = (Node*) malloc(sizeof(Node));
13     if(newNode == NULL) {
14         printf("Error! memory not allocated.");
15         exit(0);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // Function to add a node at the end of the list
23 void addNode(Node** head, int data) {
24     Node* newNode = createNode(data);
25     if(*head == NULL) {
26         *head = newNode;
27         newNode->next = newNode; // Point next of new node to itself
28     } else {
29         Node* temp = *head;
30         while(temp->next != *head) { // Traverse till the last node
31             temp = temp->next;
32         }
33         temp->next = newNode;
34         newNode->next = *head; // Point next of new node to head
35     }
36 }
37
38 // Function to insert a node at a specific position
39 void insertNode(Node** head, int data, int position) {
40     Node* newNode = createNode(data);
41     if(position == 0) {
42         if(*head == NULL) {
43             newNode->next = newNode; // Point next of new node to itself
44             *head = newNode;
45         } else {
46             Node* temp = *head;
47             while(temp->next != *head) { // Traverse till the last node
48                 temp = temp->next;
49             }
50             newNode->next = *head;
51             temp->next = newNode;
52             *head = newNode;
53         }
54         return;
55     }
56     Node* temp = *head;
57     for(int i = 0; i < position - 1; i++) {
58         if(temp != NULL) {
59             temp = temp->next;
60         }
61     }
62     if(temp != NULL) {
63         newNode->next = temp->next;
64         temp->next = newNode;
65     }
66 }
67
68 // Function to print the list
69 void printList(Node* head) {
70     if(head != NULL) {
71         Node* temp = head;
72         do {
73             printf("%d -> ", temp->data);
74             temp = temp->next;
75         } while(temp != head);

```

- Output

A terminal window with a black background and white text. At the top, there is a toolbar with four icons: a checkmark, a cursor, a gear, and a document. The text in the terminal shows the following sequence of inputs and outputs:

```
Enter the number of nodes: 4
Enter data for node 1: 4
Enter data for node 2: 2
Enter data for node 3: 3
Enter data for node 4: 1
Enter data and position for new node: 4
1
4 -> 4 -> 2 -> 3 -> 1 -> (back to 4)
```

- DELETION

```

void insertNode(Node** head, int data, int position) {
    temp->next = newNode;
}

// Function to delete a node at a specific position
void deleteNode(Node** head, int position) {
    if(*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Node* temp = *head;
    if(position == 0) {
        while(temp->next != *head) { // Traverse till the last node
            temp = temp->next;
        }
        if(*head == (*head)->next) { // If only one node
            free(*head);
            *head = NULL;
        } else {
            Node* temp2 = *head;
            temp->next = (*head)->next;
            *head = (*head)->next;
            free(temp2);
        }
        return;
    }
    for(int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if(temp != NULL) {
        Node* next = temp->next;
        temp->next = next->next;
        free(next);
    }
}

// Function to print the list
void printList(Node* head) {
    if(head != NULL) {
        Node* temp = head;
        do {
            printf("%d -> ", temp->data);
            temp = temp->next;
        } while(temp != head);
        printf(" (back to %d)\n", head->data);
    }
}

```

```

int main() {
    Node* head = NULL;
    int size, data, position;

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

    for(int i = 0; i < size; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        addNode(&head, data);
    }

    printf("Enter data and position for new node: ");
    scanf("%d %d", &data, &position);
    insertNode(&head, data, position);


    printf("Enter position of node to delete: ");
    scanf("%d", &position);
    deleteNode(&head, position);

    printList(head);

    return 0;
}

```

• Output



```
Enter the number of nodes: 4
Enter data for node 1: 1
Enter data for node 2:
12
Enter data for node 3:
1
Enter data for node 4:
1
Enter data and position for new node: 2
1
Enter position of node to delete: 2
1 -> 2 -> 1 -> 1 -> (back to 1)
```