
SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

*(Deemed to be* **University** *u/s 3 of UGC Act, 1956)*

DELHI NCR CAMPUS, GHAZIABAD, (U.P.)

# S R M INSTITUTE OF SCIENCE & TECHNOLOGY

**(FACULTY OF SCIENCE & HUMANITIES)**

DEPARTMENT OF COMPUTER APPLICATIONS

# PRACTICAL FILE

**Fundamentals Of Data Structures and Algorithms**

**[UDS23202J]**

**BCA 1ST YEAR, 2nd SEMESTER**

**Session: 2023-24**

**SUBMITTED TO :**

**Vishal Ravi Verma**
**Assistant Professor**
**Department of Computer Applications**

**SUBMITTED BY :**

**RA2331242030018**
**Ankit Kumar**
**Department of Computer Applications**

# S R M INSTITUTE OF SCIENCE & TECHNOLOGY

DELHI NCR CAMPUS, MODINAGAR

**(FACULTY OF SCIENCE & HUMANITIES)**

DEPARTMENT OF COMPUTER APPLICATIONS

**Register No. :** **RA2331242030018**

## BONAFIDE CERTIFICATE

Certified to be the Bonafide record of the work done by **Ankit Kumar** of BCA **1st Year/ 2nd** Semester for the Award of **Bachelor's Degree** in the **DEPARTMENT OF COMPUTER APPLICATIONS** in **Fundamentals of Data Structures and Algorithms[ UDS23202J ]** laboratory during the Academic Year **2023-24.**

**SUBJECT IN – CHARGE** **HEAD – COMPUTER APPLICATIONS**

**Submitted for the University Examination held on _____.**

**INTERNAL EXAMINER - 1** **INTERNAL EXAMINER - 2**

# TABLE OF CONTENTS

**Experiment 1: Write a program in C to traverse in array**

## Aim:

Using C programming write a program to traverse in array

**Algorithm/ Procedure:**

Here is the the steps to be followed sequentially to traverse in array :

1. Start a loop from **0** to **N-1**, where **N** is the size of array.

```
for(i = 0; i < N; i++)
```

2. Access every element of array with help of

```
arr[index]
```

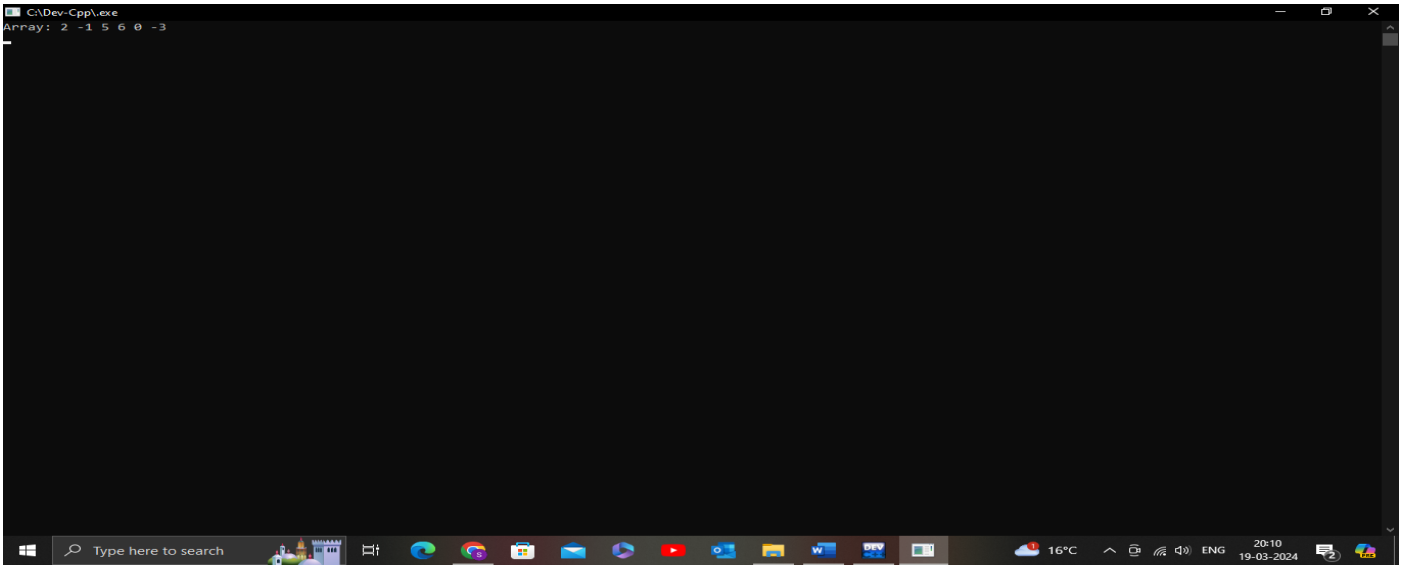
3. Print the elements.

```
printf("%d ", arr[i])
```

**Program:**

```c
#include <stdio.h>

// Function to traverse and print the array
void printArray(int* arr, int n)
{
    int i;

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

// Driver program
int main()
{
    int arr[] = { 2, -1, 5, 6, 0, -3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, n);

    return 0;
}
```

## Input/ Output




**Result:**

Thus the C program to traverse in array is successful.

**Experiment 2: Write a Program in C for deletion operation in array .**

**Aim : using c programming write a program to perform to deleting operation**

**Procedure:**

**Step 1:** Input the size of the array arr[] using num, and then declare the pos variable to define the position, and i represent the counter value.

**Step 2:** Use a loop to insert the elements in an array until (i < num) is satisfied.

**Step 3:** Now, input the position of the particular element that the user or programmer wants to delete from an array.

**Step 4:** Compare the position of an element (pos) from the total no. of elements (num+1). If the pos is greater than the num+1, the deletion of the element is not possible and jump to step 7.

**Step 5:** Else removes the particular element and shift the rest elements' position to the left side in an array.

**Step 6:** Display the resultant array after deletion or removal of the element from an array.

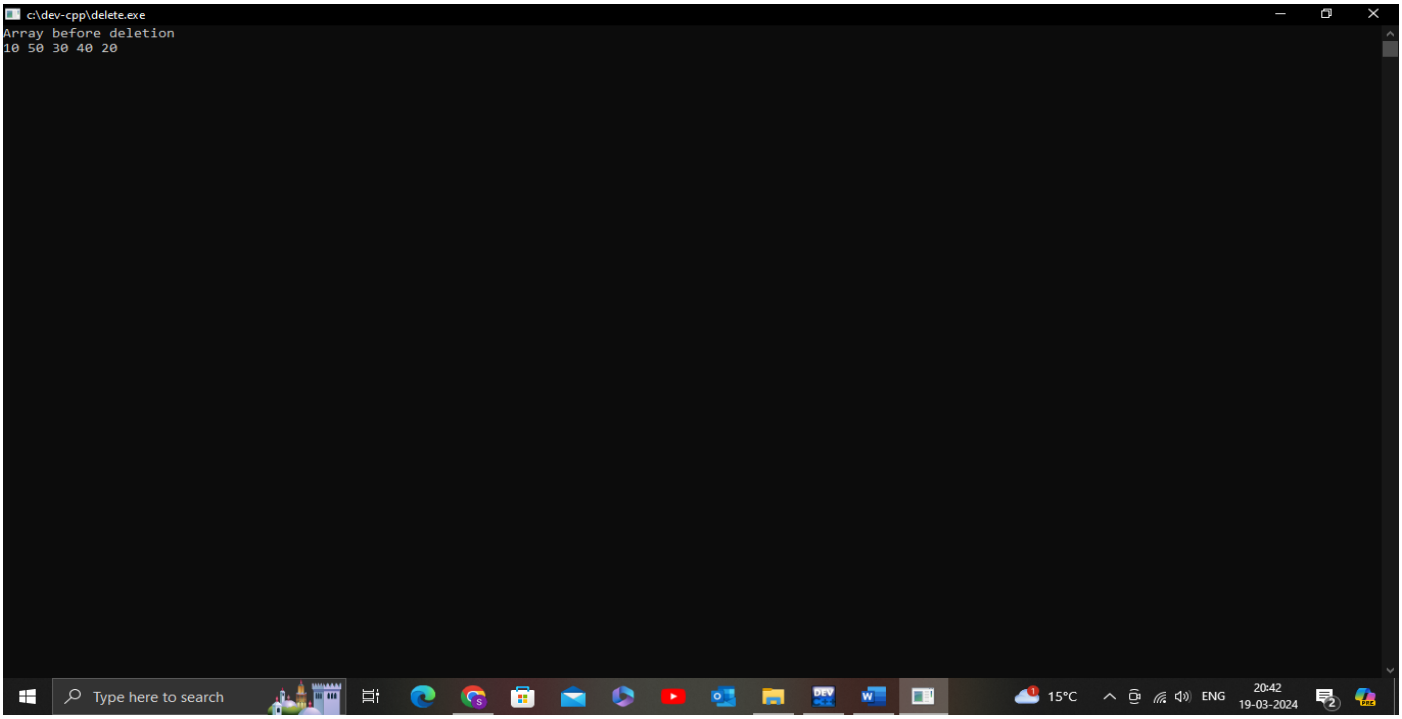
**Step 7:** Terminate or exit from the program.

**Program:**

```
// C program to implement delete operation in a
// unsorted array
#include <stdio.h>

// To search a key to be deleted
int findElement(int arr[], int n, int key);

// Function to delete an element
int deleteElement(int arr[], int n, int key)
{
        // Find position of element to be deleted
        int pos = findElement(arr, n, key);

        if (pos == -1) {
                printf("Element not found");
                return n;
        }

        // Deleting element
        int i;
        for (i = pos; i < n - 1; i++)
```

```
            arr[i] = arr[i + 1];

        return n - 1;
}

// Function to implement search operation
int findElement(int arr[], int n, int key)
{
        int i;
        for (i = 0; i < n; i++)
                if (arr[i] == key)
                        return i;

        return -1;
}

// Driver's code
int main()
{
        int i;
        int arr[] = { 10, 50, 30, 40, 20 };

        int n = sizeof(arr) / sizeof(arr[0]);
        int key = 30;

        printf("Array before deletion\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);

        // Function call
        n = deleteElement(arr, n, key);

        printf("\nArray after deletion\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);

        return 0;
}
```

**Input/Output**




**Result:** Thus the C program for deletion in array was executed and verified successfully.

**Experiment 3: Write a Program in C to update a number in array**

**Aim: Using C programming write a program to update a number in array**

**Procedure:**

1. **a :** Name of the array.
2. **size :** Size of the array (i.e., total number of elements in the array)
3. **i :** Loop counter or counter variable for the `for` loop.
4. **pos :** The position where you wish to update the element.
5. **x :** The updated element.

**Program:**

```
#include<stdio.h>
int main()
{
  int i,t,a[10],n,m,s,j=0,b[10];
  printf("\nEnter the Limit:");
  scanf("%d",&n);
  printf("\nEnter the Values:");
  for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  printf("\nGiven values are:");
  for(i=0;i<n;i++)
  {
    printf("a[%d]=%d",i,a[i]);
  }
  printf("\nEnter the position to be update:");
  scanf("%d",&t);
  printf("\nEnter the value to be update:");
  scanf("%d",&s);
  for(i=0;i<n;i++)
  {
    if(i==t)
    {
```

```
        a[i]=s;
      }
    }
    printf("\nUpdated value is:");
    for(i=0;i<n;i++)
    {
      printf("\na[%d]=%d",i,a[i]);
    }
    return 0;
}
```

**Output:**

```
Enter the Limit:5

Enter the Values:1
1
2
3

4

Given values are:a[0]=1a[1]=1a[2]=2a[3]=3a[4]=4
Enter the position to be update:3

Enter the value to be update:4

Updated value is:
a[0]=1
a[1]=1
a[2]=2
a[3]=4
a[4]=4
```

**Result: The program to update array was executed and verified successfully**

**Experiment 4: Write a program in C to implement the structure**

**Aim: Using C programming write a program in c to implement the structure**

**Procedure:**

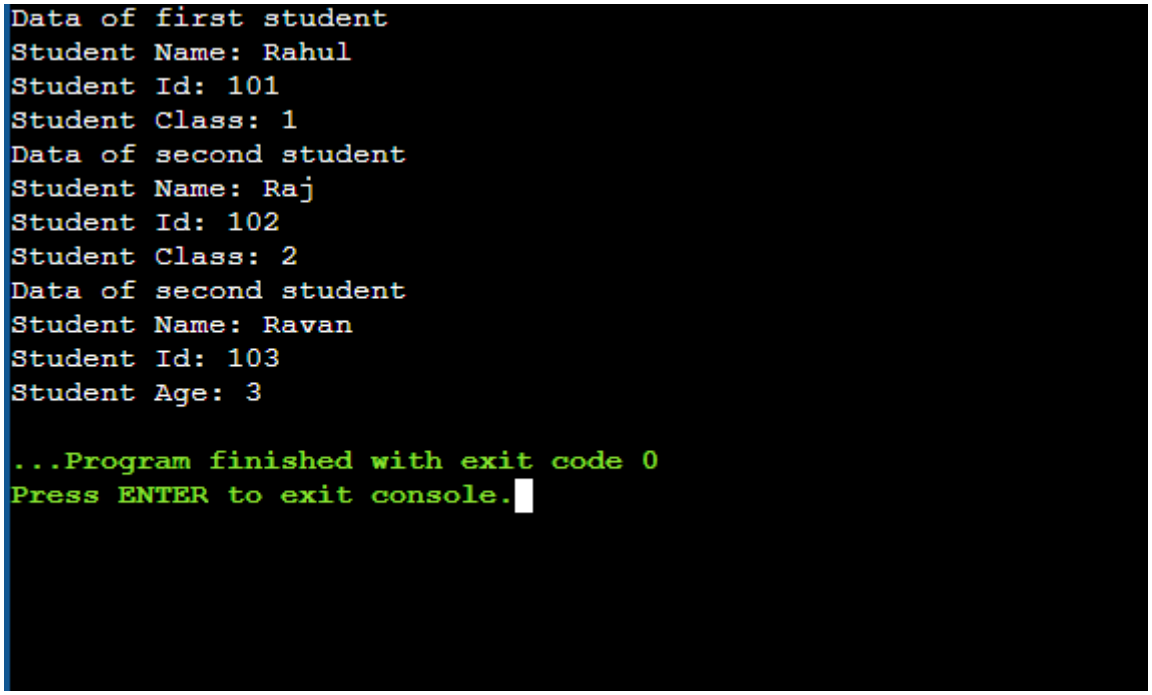
```
struct name_of_the_structure

{    data_type member1;

     data_type member2;

     ...

     data_type memberN;

};
```

**Program:**

```
#include <stdio.h>
#include<string.h>
//Declaring structure
struct Student
{
   char name[50];
   int id;
   int class;
};

int main()
{
    //declaring structure variable
    struct Student s;

    printf("Data of first student\n" );
```

```
//Initializing the member variables individually
    strcpy(s.name, "Rahul");
    s.id = 101;
    s.class = 1;
//Accessing the member variables
    printf("Student Name: %s", s.name);
    printf("\nStudent Id: %d", s.id);
    printf("\nStudent Class: %d", s.class);

    printf("\nData of second student\n" );
    strcpy(s.name, "Raj");
    s.id = 102;
    s.class = 2;
    printf("Student Name: %s", s.name);
    printf("\nStudent Id: %d", s.id);
    printf("\nStudent Class: %d", s.class);

    printf("\nData of second student\n" );
    strcpy(s.name, "Ravan");
    s.id = 103;
    s.class = 3;
    printf("Student Name: %s", s.name);
    printf("\nStudent Id: %d", s.id);
    printf("\nStudent Age: %d", s.class);


    return 0;
}
```

**Output:**




**Result :** Thus the C program for implement structure in c was executed and verified successfully.

**Experiment 5: Write a program in C to traverse a linked list**

**Aim: Using a C programming write a program in C to traverse a linked list**

**Procedure:**

**1.**Create a temporary variable for traversing. Assign reference of *head* node to it,

say temp = head.

2.Repeat below step till temp != NULL.

3.temp->data contains the current node data. You can print it or can perform some calculation on it.

4.Once done, move to next node using temp = temp->next;.

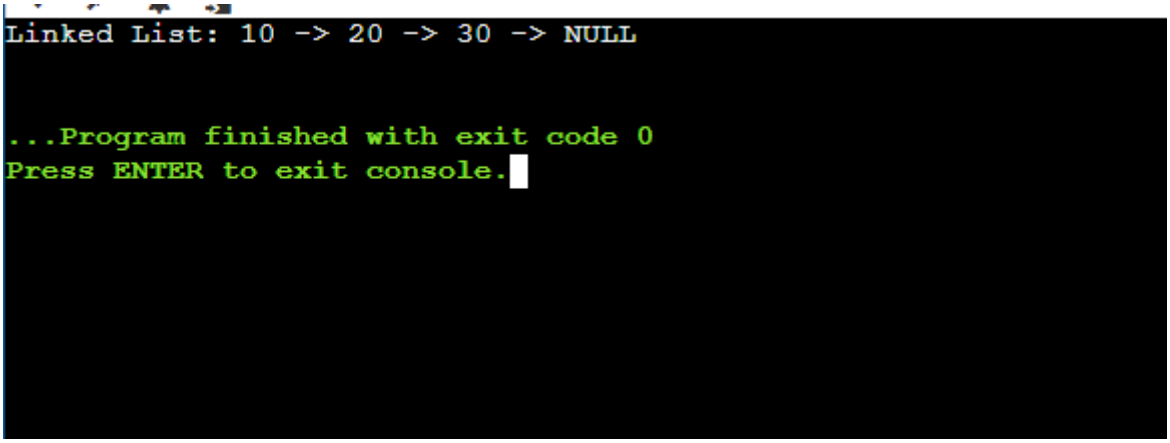
5.Go back to 2nd step.

**Program :**

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

// Define the Node structure
typedef struct Node {
   int data;
   struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory error\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
```

```
    return newNode;
}

// Function to traverse the linked list iteratively
void traverseList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);

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

    // Free the allocated memory (optional, for completeness)
    while (head) {
        Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

**Output:**




**Result:** Thus the C program for traverse a linked list was executed and verified successfully.

**Experiment 6 : Write a program in c in linked list to insert at specific location**

**Aim: Using C programming write a program in c to insert at specific location**

**Procedure:**

1. if the position is not 0 and the Head is null. Then, leave it.
2. If both Head and position is null, then. ...
3. If the position is 0 and the Head is not null. ...
4. If not, keep trying until you reach the Nth place or stop.

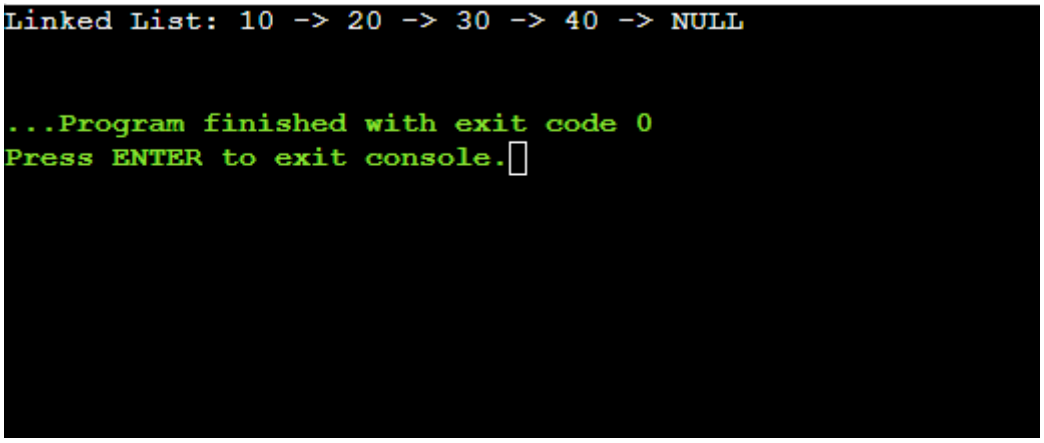
**Program:**

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

// Define the Node structure
typedef struct Node {
   int data;
   struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory error\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to insert a node after a specified location
void insertAfter(Node* prevNode, int key) {
   if (prevNode == NULL) {
      printf("The given previous node cannot be NULL.\n");
      return;
```

```
    }

    Node* newNode = createNode(key);
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

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

int main() {
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(40); // 10 -> 20 -> 40

    insertAfter(head->next, 30); // Insert 30 after 20
    // Resulting list: 10 -> 20 -> 30 -> 40

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

    // Free the allocated memory (optional, for completeness)
    while (head) {
        Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
```

}

**Output:**




**Result: Thus** the C program for inserting at specific location was executed and verified successfully

.

## Experiment 7: write a program in C to reverse a linked list

**Aim: Using a C programming write a program in c to reverse a linked list**

**Procedure :**

1. Given the head pointer of the linked list.
2. Reverse the order of the linked list by updating the pointer of the node.
3. Update the head of the list.
4. Print the linked list before and after reversing it.
5. Exit.

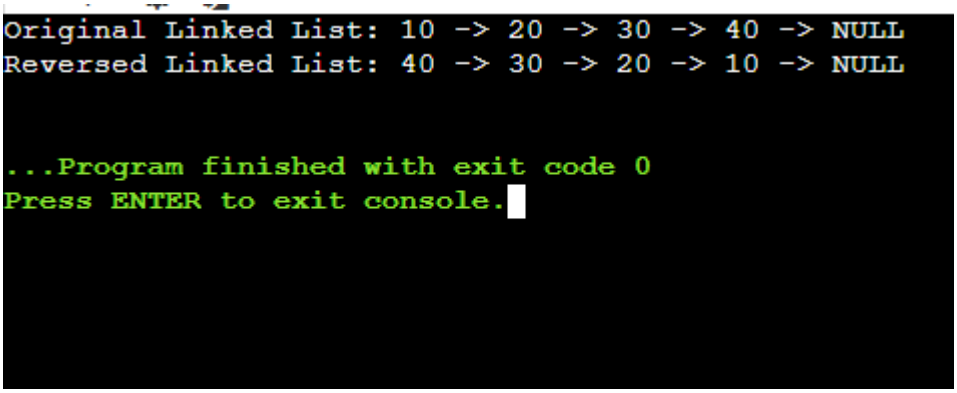
**Program:**

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

// Define the Node structure
typedef struct Node {
   int data;
   struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory error\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to reverse the linked list iteratively
void reverseListIterative(Node** head) {
   Node* prev = NULL;
   Node* current = *head;
   Node* next = NULL;
```

```
    while (current != NULL) {
        next = current->next;  // Store the next node
        current->next = prev;  // Reverse the current node's pointer
        prev = current;        // Move prev to this node for the next iteration
        current = next;        // Proceed to the next node
    }
    *head = prev;  // Reset the head pointer to the last node in the original list
}

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

int main() {
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);

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

    reverseListIterative(&head);

    printf("Reversed Linked List: ");
    displayList(head);

    // Free the allocated memory (optional, for completeness)
    while (head) {
        Node* temp = head;
```

```
        head = head->next;
        free(temp);
    }

    return 0;
}
```

**Output:**




**Result: Thus the program to reverse linked list was implemented and executed successfully**

## Experiment 8: Write a program in C for stack implementation using array

**Aim: Using C programming write a program in c for stack implementation using array**

**Procedure :**

1. Define Constants and Variables. First, we need to define the maximum size of the stack and declare an array to hold the stack elements. ...
2. Implement the push Function. The push() function adds an element to the top of the stack.
3. Implement the pop Function. ...
4. Implement the main Function.

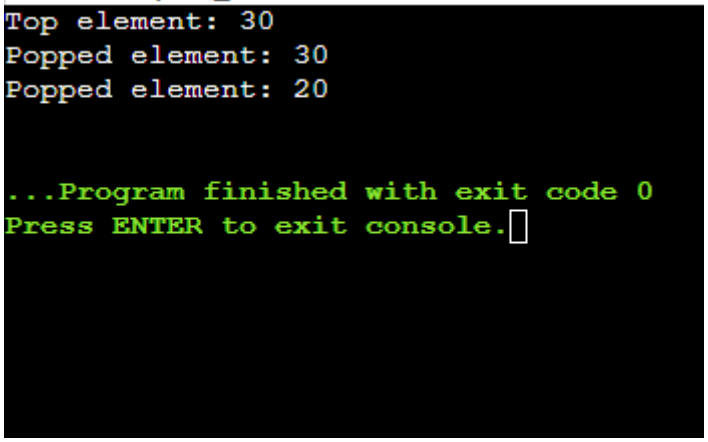
**Program:**

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

#define MAX_SIZE 100 // Define the maximum size of the stack

typedef struct {
   int arr[MAX_SIZE];
   int top;
} Stack;

// Initialize the stack
void initialize(Stack *s) {
   s->top = -1;
}

// Check if the stack is empty
int isEmpty(Stack *s) {
   return s->top == -1;
}

// Check if the stack is full
int isFull(Stack *s) {
   return s->top == MAX_SIZE - 1;
}

// Push an item onto the stack
```

```
void push(Stack *s, int item) {
    if (isFull(s)) {
        printf("Stack is full!\n");
        return;
    }
    s->arr[++(s->top)] = item;
}

// Pop an item from the stack
int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
        exit(1); // Exit with an error code
    }
    return s->arr[(s->top)--];
}

// Peek the top item without removing it
int peek(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
        exit(1); // Exit with an error code
    }
    return s->arr[s->top];
}

int main() {
    Stack s;
    initialize(&s);

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

    printf("Top element: %d\n", peek(&s)); // Should print 30
```

```
    printf("Popped element: %d\n", pop(&s)); // Should print 30
    printf("Popped element: %d\n", pop(&s)); // Should print 20

    return 0;
}
```

**Output:**




**Result: Thus the program in C for stack implementation using array was implemented and executes successfully**

**Experiment 9: Write a program in C to stack implementation in linked list**

**Aim:**

Using C programming write a program in c to stack implementation in linked list

**Procedure:**

1. Create a new node.
2. Assign the value to the data field of the node.
3. Set the next field of the node to the current top.
4. Update the top to point to the new node.

**Program:**

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

// Define the Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Define the Stack structure
typedef struct {
    Node* top;
} Stack;

// Initialize the stack
void initialize(Stack* s) {
    s->top = NULL;
}

// Check if the stack is empty
int isEmpty(Stack* s) {
```

```
    return s->top == NULL;
}

// Push an item onto the stack
void push(Stack* s, int item) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Stack overflow!\n");
        exit(1); // Exit with an error code
    }
    newNode->data = item;
    newNode->next = s->top;
    s->top = newNode;
}

// Pop an item from the stack
int pop(Stack* s) {
    if (isEmpty(s)) {
        printf("Stack underflow!\n");
        exit(1); // Exit with an error code
    }
    Node* temp = s->top;
    int poppedData = temp->data;
    s->top = s->top->next;
    free(temp);
    return poppedData;
}

// Peek the top item without removing it
int peek(Stack* s) {
    if (isEmpty(s)) {
```

```
        printf("Stack is empty!\n");
        exit(1); // Exit with an error code
    }
    return s->top->data;
}

int main() {
    Stack s;
    initialize(&s);

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

    printf("Top element: %d\n", peek(&s)); // Should print 30

    printf("Popped element: %d\n", pop(&s)); // Should print 30
    printf("Popped element: %d\n", pop(&s)); // Should print 20

    return 0;
}
```

**Output:**

```
Top element: 30
Popped element: 30
Popped element: 20


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

**Result : Thus the program in c to stack implementation using linked list was implemented and successfully executed .**

**Experiment 10: Write a program in c for linear queue using array**

**Aim: Using C programming write a program in c for linear queue using array**
**Procedure:**

1. Use three functions for three operations like insert, delete and display.
2. Use switch statement to access these functions.
3. Exit.

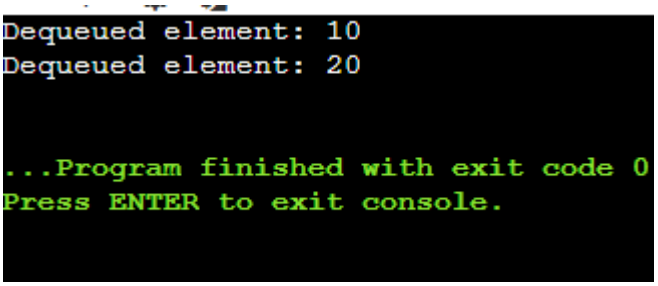
## Program:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_SIZE 100 // Define the maximum size of the queue


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


// Initialize the queue
void initialize(Queue *q) {
    q->front = -1;
    q->rear = -1;
}


// Check if the queue is empty
int isEmpty(Queue *q) {
```

```
    return q->front == -1;
}

// Check if the queue is full
int isFull(Queue *q) {
    return q->rear == MAX_SIZE - 1;
}

// Enqueue an item to the queue
void enqueue(Queue *q, int item) {
    if (isFull(q)) {
        printf("Queue is full!\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->arr[++(q->rear)] = item;
}

// Dequeue an item from the queue
int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        exit(1);
```

```
    }
    int dequeuedItem = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front++;
    }
    return dequeuedItem;
}

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

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);

    printf("Dequeued element: %d\n", dequeue(&q)); // Should print 10
    printf("Dequeued element: %d\n", dequeue(&q)); // Should print 20

    return 0;
}
```

**Output:**




**Result: Thus the program in c for linear queue in array was implemented and successfully executed.**

**Experiment 11: Write a program in c to for circular queue using array .**

**Aim**: Using c programming write a program in c for circular queue using array

**Procedure:**

1. Check if the queue is empty by checking if front is -1. ...
2. Set x to queue[front].
3. If front is equal to rear, set front and rear to -1.
4. Otherwise, increment front by 1 and if front is equal to n, set front to 0.
5. Return x.

**Program:**

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

#define MAX_SIZE 100  // Define the maximum size of the queue

typedef struct {
    int arr[MAX_SIZE];
    int front;
    int rear;
} CircularQueue;

// Initialize the circular queue
void initialize(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

// Check if the queue is empty
int isEmpty(CircularQueue *q) {
    return q->front == -1;
}

// Check if the queue is full
int isFull(CircularQueue *q) {
    return (q->rear + 1) % MAX_SIZE == q->front;
}
```

```
// Enqueue an item to the queue
void enqueue(CircularQueue *q, int item) {
    if (isFull(q)) {
        printf("Queue is full!\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
        q->rear = 0;
    } else {
        q->rear = (q->rear + 1) % MAX_SIZE;
    }
    q->arr[q->rear] = item;
}

// Dequeue an item from the queue
int dequeue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        exit(1);
    }
    int dequeuedItem = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX_SIZE;
    }
    return dequeuedItem;
}

int main() {
    CircularQueue q;
    initialize(&q);

    enqueue(&q, 10);
```

```
    enqueue(&q, 20);
    enqueue(&q, 30);

    printf("Dequeued element: %d\n", dequeue(&q));  // Should print 10
    printf("Dequeued element: %d\n", dequeue(&q));  // Should print 20

    enqueue(&q, 40);
    enqueue(&q, 50);
    enqueue(&q, 60);

    printf("Dequeued element: %d\n", dequeue(&q));  // Should print 30
    printf("Dequeued element: %d\n", dequeue(&q));  // Should print 40

    return 0;
}
```

**Output:**

```
Dequeued element: 10
Dequeued element: 20
Dequeued element: 30
Dequeued element: 40


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

**Result:** Thus the program in c for circular queue using array id implemented and successfully executed **.**

**Experiment 12: Write a program in c to implement a binary tree .**

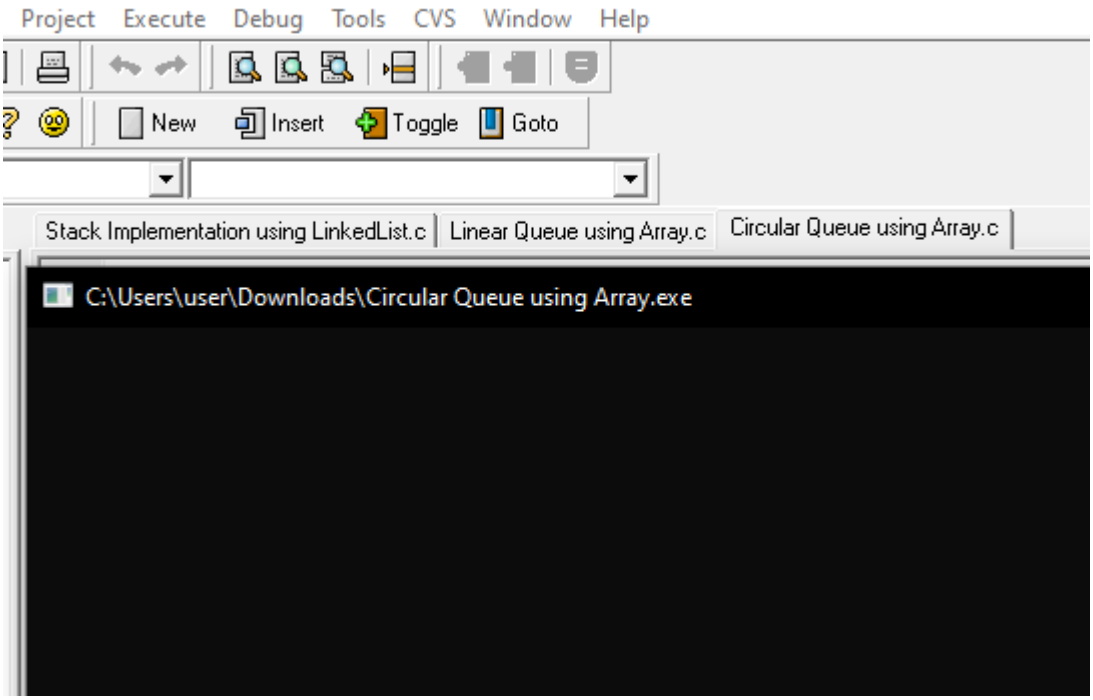
**Aim**: using C programming write a program in c to implement a binary tree .

**Procedure:**

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

struct node {
	int data;
	struct node* left;
	struct node* right;
};

// newNode() allocates a new node
// with the given data and NULL left
// and right pointers.
struct node* newNode(int data)
{
	// Allocate memory for new node
	struct node* node
		= (struct node*)malloc(sizeof(struct node));

	// Assign data to this node
	node->data = data;

	// Initialize left and
	// right children as NULL
	node->left = NULL;
	node->right = NULL;
	return (node);
}

int main()
{
	// Create root
	struct node* root = newNode(1);
```

```
        /* following is the tree after above statement
                1
        / \
        NULL NULL
        */
        root->left = newNode(2);
        root->right = newNode(3);

        /* 2 and 3 become left and right children of 1
                        1
                / \
                2 3
                / \ / \
        NULL NULL NULL NULL
        */
        root->left->left = newNode(4);

        /* 4 becomes left child of 2
                1
                / \
        2 3
        / \ / \
        4 NULL NULL NULL
        / \
        NULL NULL
        */
        getchar();
        return 0;
});
```

**Output:**




Result: Thus the program to implement bubble tree was successfully executed

**Experiment13. Write a program in c traverse a tree**

**Aim: Using C programming write a program in C to traverse a tree**

**Procedure/Algorithm:**

1. Traverse the left subtree by recursively calling the in-order function.
2. Return the root node value.
3. Traverse the right subtree by recursively calling the in-order function

**Program:**

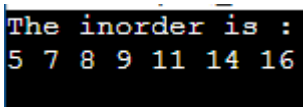
```
// Program for tree traversal inorder in Binary Tree

#include<stdio.h>

#include<stdlib.h>


// We are creating struct for the binary tree below

struct node

{

  int data;

  struct node *left, *right;

};


// newNode function for initialisation of the newly created node

struct node *newNode (int item)

{

  struct node *temporary = (struct node *) malloc (sizeof (struct node));

  temporary->data = item;

  temporary->left = temporary->right = NULL;

  return temporary;

}
```

```
// Here we print the inorder recursively
void inorder (struct node *root)
{
  if (root != NULL)
    {
      inorder (root->left);
      printf ("%d ", root->data);
      inorder (root->right);
    }
}

// Basic Program to insert new node at the correct position in BST
struct node *insert (struct node *node, int data)
{
  /* When there no node in the tree(subtree) then create
   and return new node using newNode function */
  if (node == NULL)
    return newNode (data);

  /* If not then we recur down the tree to find correct position for insertion */
  if (data < node->data)
    node->left = insert (node->left, data);
  else if (data > node->data)
    node->right = insert (node->right, data);

  return node;
```

}

**int main ()**

**Output:**




**Result: Thus the program in C for tree traversing is successfully executed**

**Experiment 14 : Write a program in c to implement bubble sort**

**Aim: Program in c to implement bubble sort**

**Procedure**

1. Starts from the first index: arr[0] and compares the first and second element: arr[0] and arr[1]
2. If arr[0] is greater than arr[1], they are swapped.
3. Similarly, if arr[1] is greater than arr[2], they are swapped.
4. The above process continues until the last element arr[n-1]

**Program:**

```
// C program for implementation of Bubble sort
#include <stdio.h>

// Swap function
void swap(int* arr, int i, int j)
{
	int temp = arr[i];
	arr[i] = arr[j];
	arr[j] = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
	int i, j;
	for (i = 0; i < n - 1; i++)

		// Last i elements are already
		// in place
		for (j = 0; j < n - i - 1; j++)
			if (arr[j] > arr[j + 1])
				swap(arr, j, j + 1);
}

// Function to print an array
void printArray(int arr[], int size)
```

```
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

// Driver code
int main()
{
        int arr[] = { 5, 1, 4, 2, 8 };
        int N = sizeof(arr) / sizeof(arr[0]);
        bubbleSort(arr, N);
        printf("Sorted array: ");
        printArray(arr, N);
        return 0;
}
```

**Output:**

```
Sorted array: 1 2 4 5 8


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

**Result: Thus the program in c for bubble sort is successfully executed .**

**Experiment 15 : Write a program in c to implement insertion sort**

**Aim: Using c programming to implement insertion sort**

**Procedure:**

1. // Function to perform insertion sort.
2. void insertionSort(int arr[], int n) {
3. int i, key, j;
4. // Loop from the second element of the array.
5. for (i = 1; i < n; i++) {
6. key = arr[i]; // The element to be inserted.
7. j = i - 1;
8. /* Move elements of arr[0..i-1], that are greater than key

**Program:**

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>

/* Function to sort an array
using insertion sort*/
void insertionSort(int arr[], int n)
{
        int i, key, j;
        for (i = 1; i < n; i++)
        {
                key = arr[i];
                j = i - 1;

                /* Move elements of arr[0..i-1],
                that are greater than key,
                to one position ahead of
                their current position */
                while (j >= 0 && arr[j] > key)
```

```
                {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
}

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

// Driver code
int main()
{
        int arr[] = {12, 11, 13, 5, 6};
        int n = sizeof(arr) / sizeof(arr[0]);

        insertionSort(arr, n);
        printArray(arr, n);

        return 0;
}
```

**Output**

```
5 6 11 12 13


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

**Result: Thus the program in c to implement insertion sort is successfully executed .**

**Experiment 16: Write a program in c to implement quick sort**

**Aim: Using C programming to impement quick sort .**

**Procedure:**

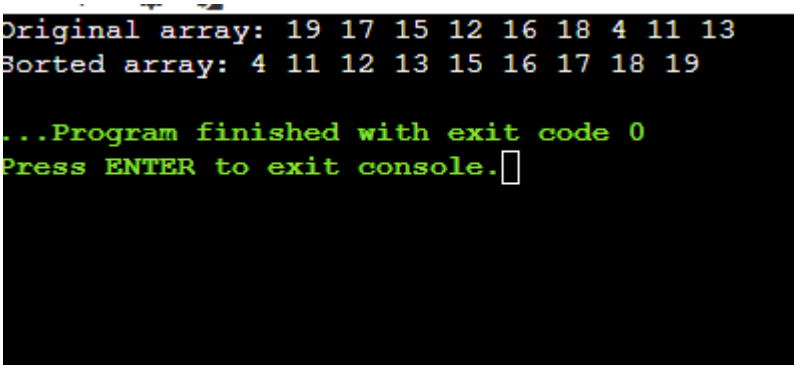
**Program:**

```
// C program to implement Quick Sort Algorithm
#include <stdio.h>

// Function to swap two elements
void swap(int* a, int* b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

// Partition function
int partition(int arr[], int low, int high)
{

	// initialize pivot to be the first element
	int pivot = arr[low];
	int i = low;
	int j = high;

	while (i < j) {

		// condition 1: find the first element greater than
		// the pivot (from starting)
		while (arr[i] <= pivot && i <= high - 1) {
			i++;
		}

		// condition 2: find the first element smaller than
		// the pivot (from last)
		while (arr[j] > pivot && j >= low + 1) {
```

```
                j--;
            }
            if (i < j) {
                swap(&arr[i], &arr[j]);
            }
        }
        swap(&arr[low], &arr[j]);
        return j;
}

// QuickSort function
void quickSort(int arr[], int low, int high)
{
        if (low < high) {

            // call Partition function to find Partition Index
            int partitionIndex = partition(arr, low, high);

            // Recursively call quickSort() for left and right
            // half based on partition Index
            quickSort(arr, low, partitionIndex - 1);
            quickSort(arr, partitionIndex + 1, high);
        }
}

// driver code
int main()
{
        int arr[] = { 19, 17, 15, 12, 16, 18, 4, 11, 13 };
        int n = sizeof(arr) / sizeof(arr[0]);

        // printing the original array
        printf("Original array: ");
        for (int i = 0; i < n; i++) {
            printf("%d ", arr[i]);
        }
```

```
    // calling quickSort() to sort the given array
    quickSort(arr, 0, n - 1);

    // printing the sorted array
    printf("\nSorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

**Output:**




**Result : Thus the program in c to implement quick sort is successfully executed .**