National Institute of Technology Kurukshetra, Kurukshetra - 136119

Master Of Computer Application

MCA - 130

(2024-27)

Data Structures Lab

Lab coordinator: Prof. Sandeep Sood

Submitted by:	**Submitted to:**
Name: Jayesh Solanki	Ms. Monika Mam
Semester: 2st	**Date**: 29-04-2025
Section: G-4	
Roll no: 524110055	

DECLARATION

I, **Jayesh Solanki**, hereby declare that the **Data Structures Lab Manual** submitted by me is my own original and independently completed work.

While I have referred to online resources and sample code snippets to enhance my understanding of the concepts, the structure, implementation, and content of the manual are based entirely on my own logic and effort.

This manual is distinct from others because:

1. I have included **screenshots of program outputs**, organized in a folder named after me, as proof of successful execution on my own system.
2. Each program is **modular, well-structured**, and developed using my personal understanding of the topics.
3. **No part** of this manual has been copied from any student or external source.
4. The manual is systematically organized with a **comprehensive table of contents** for easy navigation.

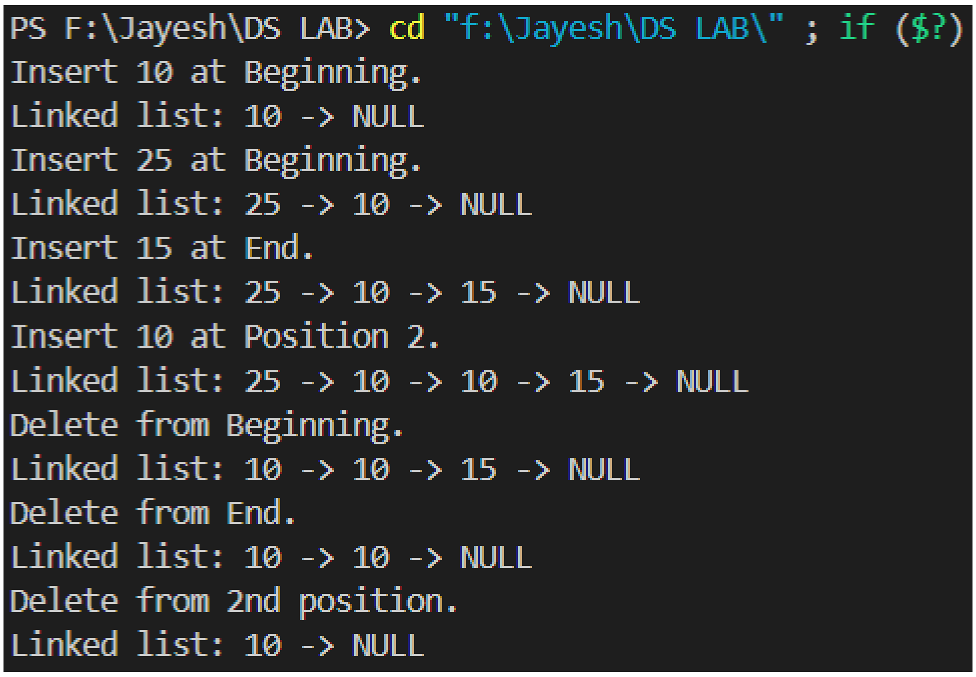
I fully understand and uphold the principles of academic integrity and take complete responsibility for the work I have submitted.

1. **Write a program to perform insert, delete and traverse operations on the singly linked list in the beginning, end and on any specific location.**

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Node* head = NULL;

void insertAtBeginning(int newData) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = newData;
   newNode->next = head;
   head = newNode;
}

void insertAtEnd(int newData) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = newData;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
      return;
   }
   struct Node* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}

void insertAtPosition(int newData, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = newData;
   if (position == 1) {
      newNode->next = head;
      head = newNode;
      return;
   }
   struct Node* temp = head;
   for (int i = 1; i < position - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position out of range.\n");
```

```
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
}

void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }
    struct Node* temp = head;
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}

void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    if (position == 1) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range.\n");
        return;
    }
    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
}
void traverse() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    printf("Insert 10 at Beginning.\n");
    insertAtBeginning(10);
    traverse();
    printf("Insert 25 at Beginning.\n");
    insertAtBeginning(25);
    traverse();
    printf("Insert 15 at End.\n");
    insertAtEnd(15);
    traverse();
    printf("Insert 10 at Position 2.\n");
    insertAtPosition(10, 2);
    traverse();
    printf("Delete from Beginning.\n");
    deleteAtBeginning();
    traverse();
    printf("Delete from End.\n");
    deleteAtEnd();
    traverse();
    printf("Delete from 2nd position.\n");
    deleteAtPosition(2);
    traverse();
    return 0;
}
```

Output:



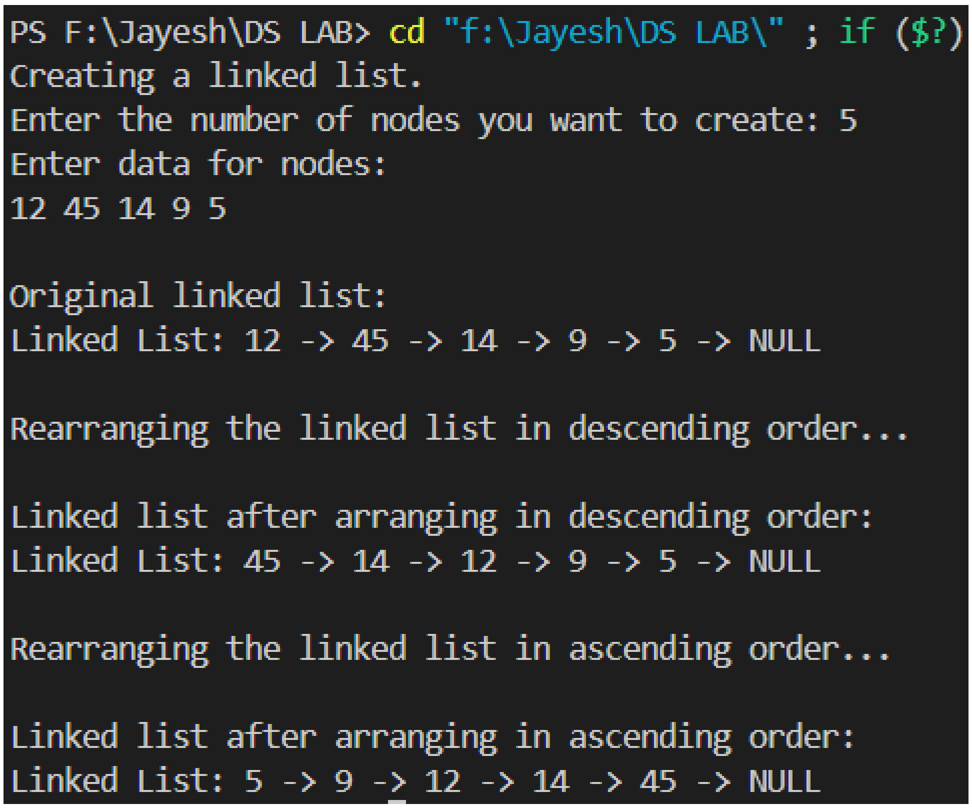
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Insert 10 at Beginning.
Linked list: 10 -> NULL
Insert 25 at Beginning.
Linked list: 25 -> 10 -> NULL
Insert 15 at End.
Linked list: 25 -> 10 -> 15 -> NULL
Insert 10 at Position 2.
Linked list: 25 -> 10 -> 10 -> 15 -> NULL
Delete from Beginning.
Linked list: 10 -> 10 -> 15 -> NULL
Delete from End.
Linked list: 10 -> 10 -> NULL
Delete from 2nd position.
Linked list: 10 -> NULL
```

2. Write a program to rearrange the elements of a singly linked list in ascending or descending order.

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

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

struct Node* head = NULL;
struct Node* temp = NULL;

void createLinkedList() {
    int n;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("Number of nodes must be positive.\n");
        return;
    }

    printf("Enter data for nodes:\n");
    for (int i = 0; i < n; ++i) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        scanf("%d", &newNode->data);
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void rearrange(int ascending) {
    printf("\nRearranging the linked list in %s order...\n", ascending ? "ascending" :
"descending");
```

```
    if (head == NULL || head->next == NULL)
        return;

    struct Node* current = head;

    while (current != NULL) {
        struct Node* index = current->next;
        while (index != NULL) {
            if (ascending ? (current->data > index->data) : (current->data < index->data)) {
                int tempData = current->data;
                current->data = index->data;
                index->data = tempData;
            }
            index = index->next;
        }
        current = current->next;
    }
}

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

int main() {
    printf("Creating a linked list.\n");
    createLinkedList();
    printf("\nOriginal linked list:\n");
    display();
    rearrange(0);
    printf("\nLinked list after arranging in descending order:\n");
    display();
    rearrange(1);
    printf("\nLinked list after arranging in ascending order:\n");
    display();

    return 0;
}
```

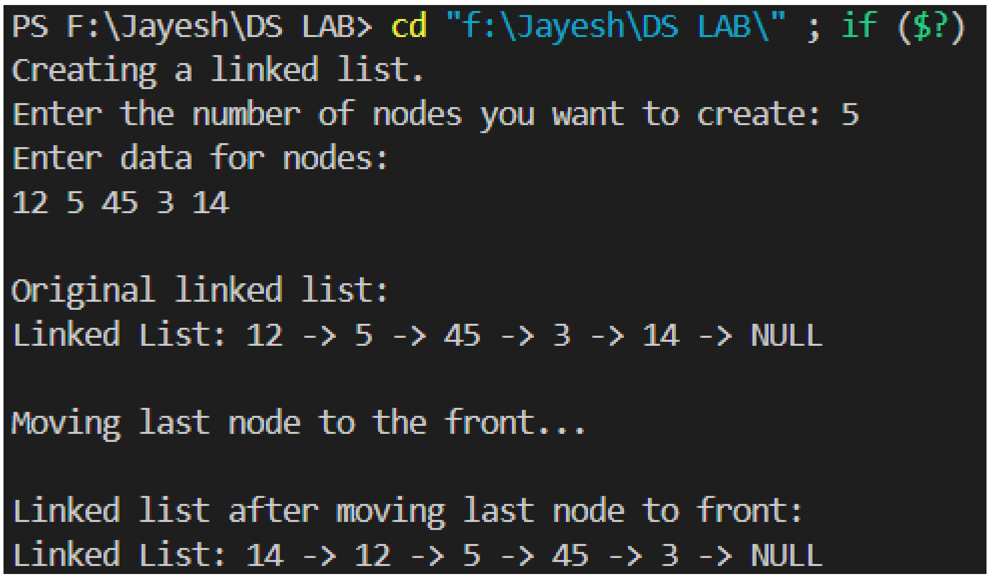
Output:


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Creating a linked list.
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 45 14 9 5

Original linked list:
Linked List: 12 -> 45 -> 14 -> 9 -> 5 -> NULL

Rearranging the linked list in descending order...

Linked list after arranging in descending order:
Linked List: 45 -> 14 -> 12 -> 9 -> 5 -> NULL

Rearranging the linked list in ascending order...

Linked list after arranging in ascending order:
Linked List: 5 -> 9 -> 12 -> 14 -> 45 -> NULL
```

3. Write a program to move the last node to the front of singly linked list.

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

struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;
struct Node* temp = NULL;

void createLinkedList() {
    int n;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Number of nodes must be positive.\n");
        return;
    }
    printf("Enter data for nodes:\n");
    for (int i = 0; i < n; ++i) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        scanf("%d", &newNode->data);
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void moveLastNodeToFront() {
    printf("\nMoving last node to the front...\n");
    if (head == NULL || head->next == NULL)
        return;
    struct Node* prev = NULL;
    struct Node* current = head;
    while (current->next != NULL) {
        prev = current;
        current = current->next;
    }
    prev->next = NULL;
    current->next = head;
    head = current;
}

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

int main() {
    printf("Creating a linked list.\n");
    createLinkedList();
    printf("\nOriginal linked list:\n");
    display();
    moveLastNodeToFront();
    printf("\nLinked list after moving last node to front:\n");
    display();
    return 0;
}
```

Output:



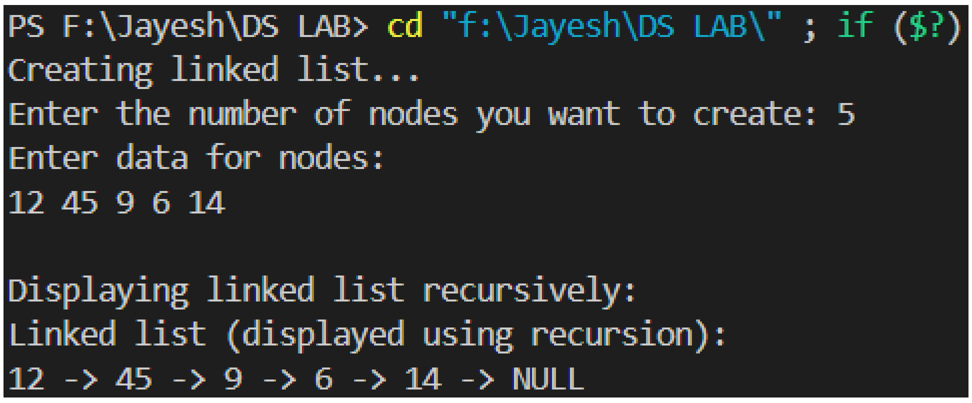
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Creating a linked list.
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 5 45 3 14

Original linked list:
Linked List: 12 -> 5 -> 45 -> 3 -> 14 -> NULL

Moving last node to the front...

Linked list after moving last node to front:
Linked List: 14 -> 12 -> 5 -> 45 -> 3 -> NULL
```

4. Write a program to print the elements of singly link list using recursion.

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

struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;
struct Node* temp = NULL;

void createLinkedList() {
    int n;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Number of nodes must be positive.\n");
        return;
    }
    printf("Enter data for nodes:\n");
    for (int i = 0; i < n; ++i) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        scanf("%d", &newNode->data);
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void displayRecursively(struct Node* current) {
    if (current == NULL) {
        printf("NULL\n");
        return;
    }
    printf("%d -> ", current->data);
    displayRecursively(current->next);
}
```

```
void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("Linked list (displayed using recursion):\n");
    displayRecursively(head);
}

int main() {
    printf("Creating linked list...\n");
    createLinkedList();
    printf("\nDisplaying linked list recursively:\n");
    display();
    return 0;
}
```

Output:

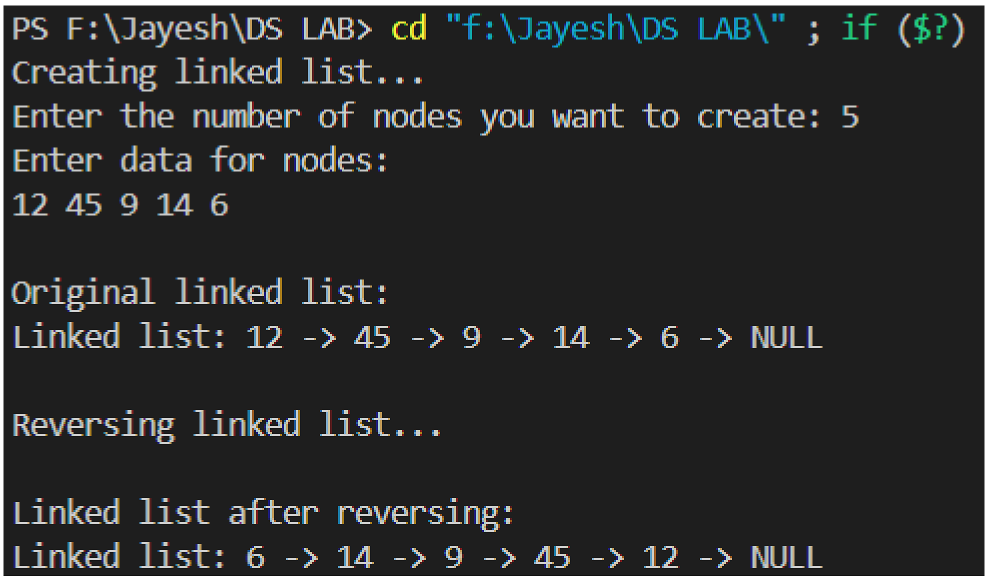


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Creating linked list...
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 45 9 6 14

Displaying linked list recursively:
Linked list (displayed using recursion):
12 -> 45 -> 9 -> 6 -> 14 -> NULL
```

5. Write a program to reverse link list using the iteration technique.

```c
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;
struct Node* temp = NULL;

void createLinkedList() {
    int n;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Number of nodes must be positive.\n");
        return;
    }
    printf("Enter data for nodes:\n");
    for (int i = 0; i < n; ++i) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        scanf("%d", &newNode->data);
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
}

void reverseLinkedList() {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}

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

int main() {
    printf("Creating linked list...\n");
    createLinkedList();
    printf("\nOriginal linked list:\n");
    display();
    printf("\nReversing linked list...\n");
    reverseLinkedList();
    printf("\nLinked list after reversing:\n");
    display();
    return 0;
}
```

Output:



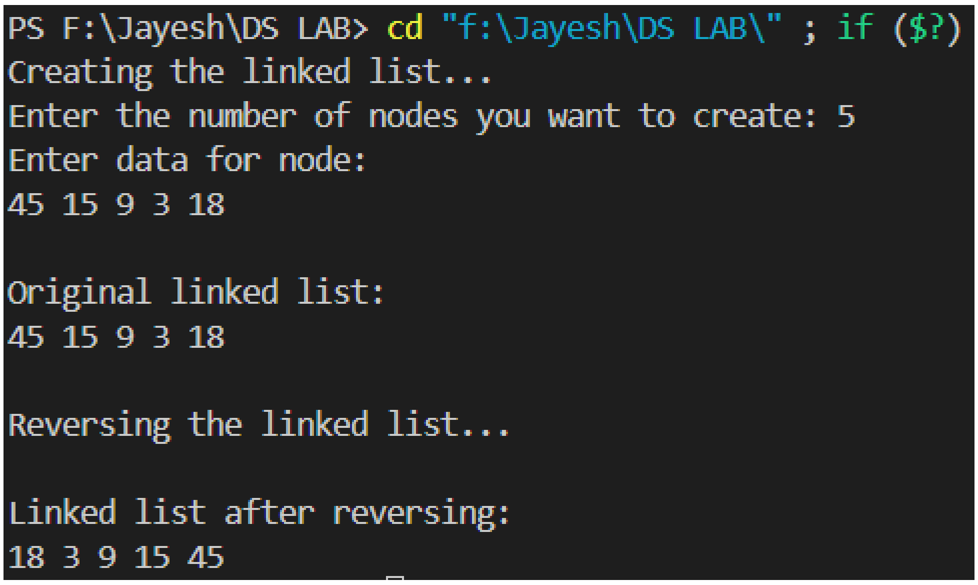
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Creating linked list...
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 45 9 14 6

Original linked list:
Linked list: 12 -> 45 -> 9 -> 14 -> 6 -> NULL

Reversing linked list...

Linked list after reversing:
Linked list: 6 -> 14 -> 9 -> 45 -> 12 -> NULL
```

6. Write a program to reverse the singly link list using recursion.

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

struct node {
   int data;
   struct node *next;
};
struct node *head = NULL, *temp = NULL;

void linkedlist() {
   int n;
   printf("Enter the number of nodes you want to create: ");
   scanf("%d", &n);
   printf("Enter data for node:\n");
   for (int i = 0; i < n; ++i) {
      struct node *newnode = (struct node*)malloc(sizeof(struct node));
      scanf("%d", &newnode->data);
      newnode->next = NULL;
      if (head == NULL) {
         head = newnode;
         temp = newnode;
      } else {
         temp->next = newnode;
         temp = newnode;
      }
   }
}

struct node* reverseRecursively(struct node *tempr) {
   if (tempr == NULL || tempr->next == NULL) {
      return tempr;
   }
   struct node *rest = reverseRecursively(tempr->next);
   tempr->next->next = tempr;
   tempr->next = NULL;
   return rest;
}

void display(struct node *tempr) {
   while (tempr != NULL) {
      printf("%d ", tempr->data);
      tempr = tempr->next;
   }
   printf("\n");
}
```

```
int main() {
    printf("Creating the linked list...\n");
    linkedlist();
    printf("\nOriginal linked list:\n");
    display(head);

    printf("\nReversing the linked list...\n");
    head = reverseRecursively(head);
    printf("\nLinked list after reversing:\n");
    display(head);

    return 0;
}
```

Output:

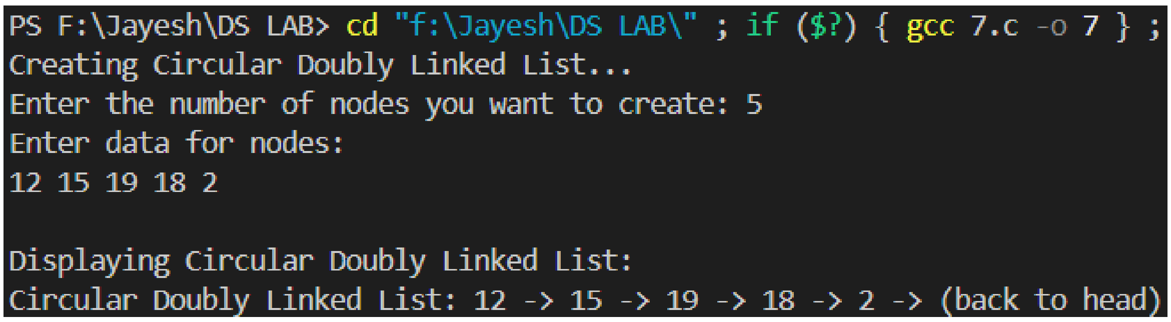


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Creating the linked list...
Enter the number of nodes you want to create: 5
Enter data for node:
45 15 9 3 18

Original linked list:
45 15 9 3 18

Reversing the linked list...

Linked list after reversing:
18 3 9 15 45
```

7. Write a program to implement a circular linked list.

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

struct node {
    int data;
    struct node *prev;
    struct node *next;
};
struct node *head = NULL, *temp = NULL;

void createCircularLinkedList() {
    int n;
    printf("Enter the number of nodes you want to create: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Number of nodes must be positive.\n");
        return;
    }
    printf("Enter data for nodes:\n");
    for (int i = 0; i < n; i++) {
        struct node *newnode = (struct node*)malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Memory allocation failed.\n");
            return;
        }
        scanf("%d", &newnode->data);
        newnode->next = NULL;
        newnode->prev = NULL;
        if (head == NULL) {
            head = newnode;
            temp = newnode;
        } else {
            newnode->prev = temp;
            temp->next = newnode;
            temp = newnode;
        }
    }
    temp->next = head;
    head->prev = temp;
}

void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct node *tempr = head;
    printf("Circular Doubly Linked List: ");
    do {
        printf("%d -> ", tempr->data);
        tempr = tempr->next;
    } while (tempr != head);
    printf("(back to head)\n");
}

int main() {
    printf("Creating Circular Doubly Linked List...\n");
    createCircularLinkedList();
    printf("\nDisplaying Circular Doubly Linked List:\n");
    display();
    return 0;
}
```

Output:

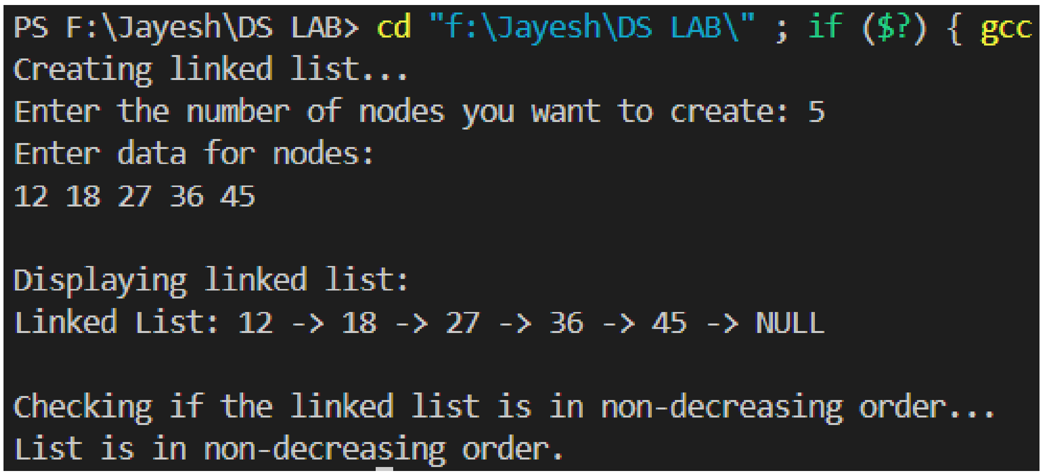


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 7.c -o 7 } ;
Creating Circular Doubly Linked List...
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 15 19 18 2

Displaying Circular Doubly Linked List:
Circular Doubly Linked List: 12 -> 15 -> 19 -> 18 -> 2 -> (back to head)
```

8. Write a program to check whether the given singly linked list is in non-decreasing order or not.

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;

void createList() {
   int n;
   struct node *temp = NULL;
   printf("Enter the number of nodes you want to create: ");
   scanf("%d", &n);
   if (n <= 0) {
      printf("Number of nodes must be positive.\n");
      return;
   }
   printf("Enter data for nodes:\n");
   for (int i = 0; i < n; i++) {
      struct node *newnode = (struct node*)malloc(sizeof(struct node));
      if (newnode == NULL) {
         printf("Memory allocation failed.\n");
         return;
      }
      scanf("%d", &newnode->data);
      newnode->next = NULL;
      if (head == NULL) {
         head = newnode;
         temp = newnode;
      } else {
         temp->next = newnode;
         temp = newnode;
      }
   }
}

void checkNonDecreasing() {
   if (head == NULL || head->next == NULL) {
      printf("List is non-decreasing.\n");
      return;
   }

   struct node *temp = head;
   while (temp->next != NULL) {
        if (temp->data > temp->next->data) {
            printf("List is not in non-decreasing order.\n");
            return;
        }
        temp = temp->next;
    }
    printf("List is in non-decreasing order.\n");
}

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

int main() {
    printf("Creating linked list...\n");
    createList();
    printf("\nDisplaying linked list:\n");
    displayList();
    printf("\nChecking if the linked list is in non-decreasing order...\n");
    checkNonDecreasing();
    return 0;
}
```

Output:



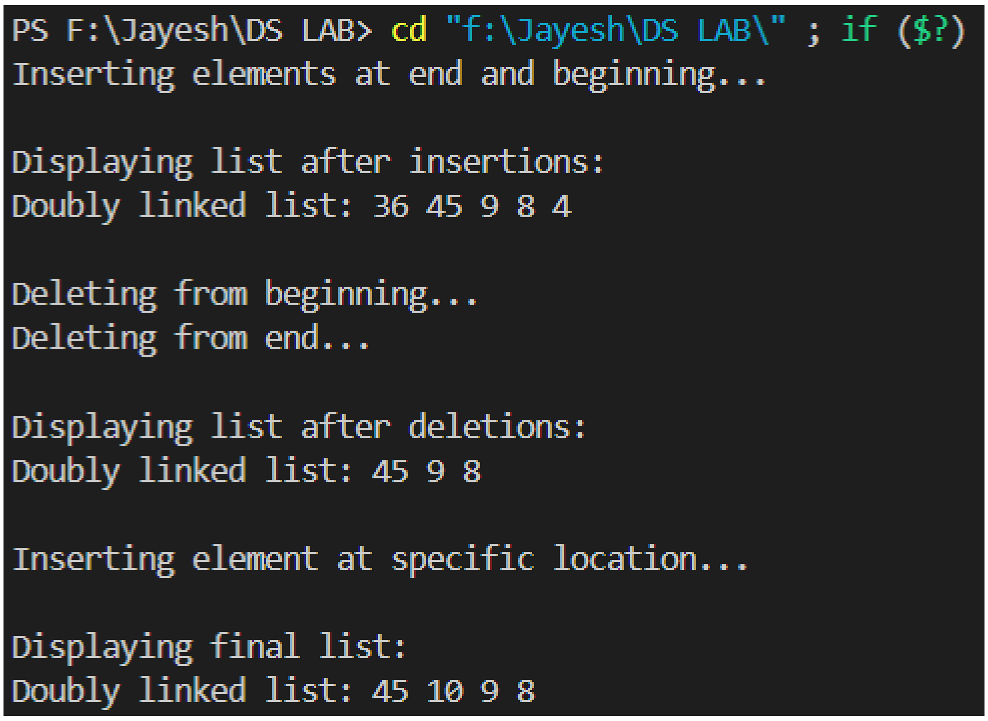
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Creating linked list...
Enter the number of nodes you want to create: 5
Enter data for nodes:
12 18 27 36 45

Displaying linked list:
Linked List: 12 -> 18 -> 27 -> 36 -> 45 -> NULL

Checking if the linked list is in non-decreasing order...
List is in non-decreasing order.
```

9. Write a program to perform insert, delete, and traverse operations on the doubly linked list in the beginning, end and on any specific location.

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

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

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

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

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

```
}

void insertAtLocation(struct Node** head, int data, int position) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        for (int i = 1; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Invalid position.\n");
            return;
        }
        newNode->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
    } else {
        struct Node* temp = *head;
        *head = (*head)->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
    }
}

void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty.\n");
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        if (temp->prev != NULL) {
            temp->prev->next = NULL;
        } else {
            *head = NULL;
        }
        free(temp);
    }
}

void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Doubly linked list: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    printf("Inserting elements at end and beginning...\n");
    insertAtEnd(&head, 9);
    insertAtBeginning(&head, 45);
    insertAtEnd(&head, 8);
    insertAtBeginning(&head, 36);
    insertAtEnd(&head, 4);
    printf("\nDisplaying list after insertions:\n");
    display(head);
    printf("\nDeleting from beginning...\n");
    deleteFromBeginning(&head);
    printf("Deleting from end...\n");
    deleteFromEnd(&head);
    printf("\nDisplaying list after deletions:\n");
    display(head);
    printf("\nInserting element at specific location...\n");
    insertAtLocation(&head, 10, 2);
    printf("\nDisplaying final list:\n");
    display(head);
    return 0;
}
```

Output:

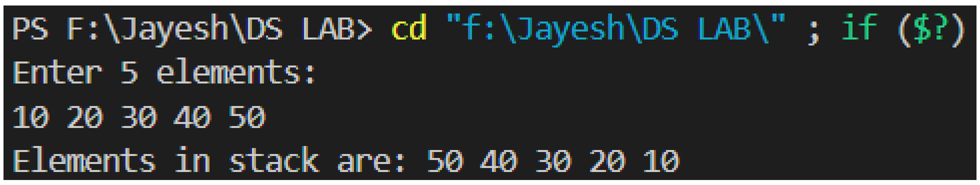


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Inserting elements at end and beginning...

Displaying list after insertions:
Doubly linked list: 36 45 9 8 4

Deleting from beginning...
Deleting from end...

Displaying list after deletions:
Doubly linked list: 45 9 8

Inserting element at specific location...

Displaying final list:
Doubly linked list: 45 10 9 8
```

10. Write a program to implement stack (push and pop operations) using array.

```
#include<stdio.h>

int Stack[10];
int top = -1;
void push(int data) {
    top++;
    Stack[top] = data;
}
void pop() {
    if (top == -1) {
        printf("Underflow\n");
    } else {
        top--;
    }
}

int main() {
    printf("Enter 5 elements:\n");
    int num;
    for (int i = 0; i < 5; i++) {
        scanf("%d", &num);
        push(num);
    }
    printf("Elements in stack are: ");
    while (top != -1) {
        printf("%d ", Stack[top]);
        pop();
    }
    printf("\n");
    return 0;
}
```

Output:

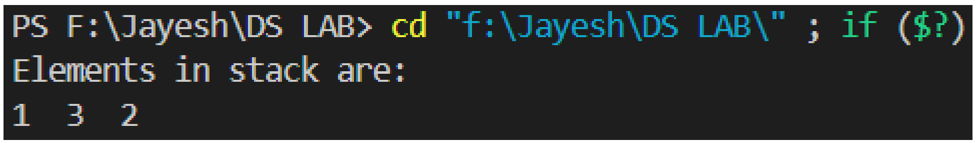


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Enter 5 elements:
10 20 30 40 50
Elements in stack are: 50 40 30 20 10
```

11. Write a program to implement stack using singly linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *top = NULL;
void push(int item) {
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newnode->data = item;
   newnode->next = top;
   top = newnode;
}

void pop() {
   struct node *temp = top;
   if (temp == NULL) {
      printf("Stack is empty\n");
      return;
   }
   printf("Elements in stack are:\n");
   while (temp != NULL) {
      printf("%d  ", temp->data);
      temp = temp->next;
   }
}

int main() {
   push(2);
   push(3);
   push(1);
   pop();
   return 0;
}
```

Output:



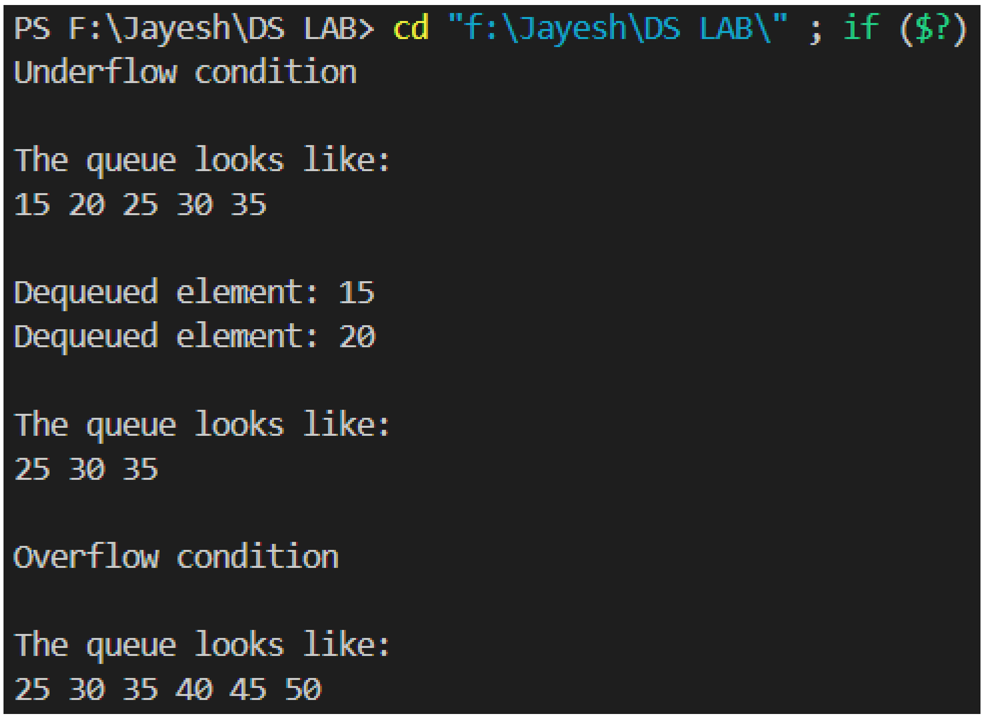
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Elements in stack are:
1  3  2
```

12. Write a program to implement a queue using a circular array.

```
#include<stdio.h>
#define capacity 6

int queue[capacity];
int front = -1, rear = -1;

int checkFull() {
   if ((front == rear + 1) || (front == 0 && rear == capacity - 1))
      return 1;
   return 0;
}

int checkEmpty() {
   if (front == -1)
      return 1;
   return 0;
}

void enqueue(int value) {
   if (checkFull())
      printf("Overflow condition\n");
   else {
      if (front == -1)
         front = 0;
      rear = (rear + 1) % capacity;
      queue[rear] = value;
   }
}

int dequeue() {
   int variable;
   if (checkEmpty()) {
      printf("Underflow condition\n");
      return -1;
   } else {
      variable = queue[front];
      if (front == rear) {
         front = rear = -1;
      } else {
         front = (front + 1) % capacity;
      }
      printf("Dequeued element: %d\n", variable);
      return variable;
   }
}
```

```
void print() {
   int i;
   if (checkEmpty())
      printf("Nothing to display\n");
   else {
      printf("\nThe queue looks like: \n");
      for (i = front; i != rear; i = (i + 1) % capacity) {
         printf("%d ", queue[i]);
      }
      printf("%d \n\n", queue[i]);
   }
}

int main() {
   dequeue(); // Trying to dequeue initially

   enqueue(15);
   enqueue(20);
   enqueue(25);
   enqueue(30);
   enqueue(35);

   print();

   dequeue();
   dequeue();

   print();

   enqueue(40);
   enqueue(45);
   enqueue(50); // This enqueue will show Overflow (correct behavior)
   enqueue(55); // This enqueue will show Overflow (correct behavior)

   print();

   return 0;
}
```

Output:

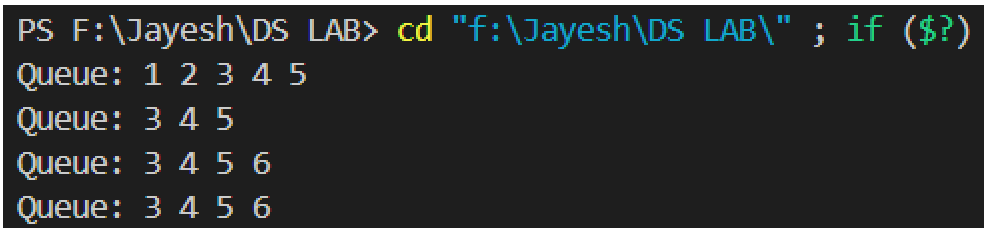


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Underflow condition

The queue looks like:
15 20 25 30 35

Dequeued element: 15
Dequeued element: 20

The queue looks like:
25 30 35

Overflow condition

The queue looks like:
25 30 35 40 45 50
```

13. Write a program to implement a queue using a circular linked list.

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

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

typedef struct {
    Node* rear;
} CircularQueue;

CircularQueue* createCircularQueue() {
    CircularQueue* cq = (CircularQueue*)malloc(sizeof(CircularQueue));
    cq->rear = NULL;
    return cq;
}

int is_empty(CircularQueue* cq) {
    return cq->rear == NULL;
}

void enqueue(CircularQueue* cq, int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = item;
    if (is_empty(cq)) {
        newNode->next = newNode;
        cq->rear = newNode;
    } else {
        newNode->next = cq->rear->next;
        cq->rear->next = newNode;
        cq->rear = newNode;
    }
}

int dequeue(CircularQueue* cq) {
    if (is_empty(cq)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    } else {
        int item;
        Node* temp = cq->rear->next;
        if (temp == cq->rear) {
            item = temp->data;
            free(temp);
            cq->rear = NULL;
        } else {
            item = temp->data;
            cq->rear->next = temp->next;
            free(temp);
        }
        return item;
    }
}

void display(CircularQueue* cq) {
    if (is_empty(cq)) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue: ");
        Node* temp = cq->rear->next;
        do {
            printf("%d ", temp->data);
            temp = temp->next;
        } while (temp != cq->rear->next);
        printf("\n");
    }
}

// New function to free all nodes before exiting
void freeQueue(CircularQueue* cq) {
    if (!is_empty(cq)) {
        Node* temp = cq->rear->next;
        Node* nextNode;
        do {
            nextNode = temp->next;
            free(temp);
            temp = nextNode;
        } while (temp != cq->rear->next);
    }
    free(cq);
}

int main() {
    CircularQueue* cq = createCircularQueue();

    enqueue(cq, 1);
    enqueue(cq, 2);
    enqueue(cq, 3);
    enqueue(cq, 4);
    enqueue(cq, 5);

    display(cq);
    dequeue(cq);
    dequeue(cq);

    display(cq);

    enqueue(cq, 6);

    display(cq);

    freeQueue(cq);  // Free the whole queue safely
    return 0;
}
```

Output:



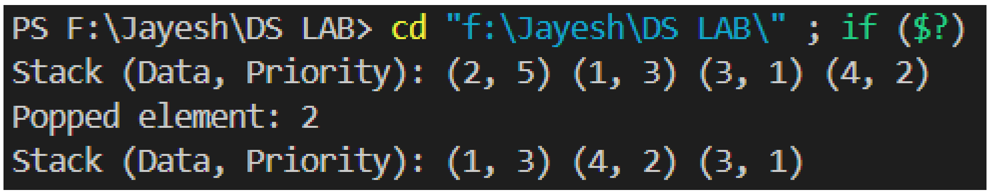
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Queue: 1 2 3 4 5
Queue: 3 4 5
Queue: 3 4 5 6
Queue: 3 4 5 6
```

14. Write a program to implement stack using priority queue.

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

typedef struct {
    int data;
    int priority;
} StackElement;

typedef struct {
    StackElement* elements;
    int capacity;
    int size;
} PriorityQueue;

PriorityQueue* createPriorityQueue(int capacity) {
    PriorityQueue* pq = (PriorityQueue*)malloc(sizeof(PriorityQueue));
    pq->elements = (StackElement*)malloc(capacity * sizeof(StackElement));
    pq->capacity = capacity;
    pq->size = 0;
    return pq;
}

void swap(StackElement* a, StackElement* b) {
    StackElement temp = *a;
    *a = *b;
    *b = temp;
}

void push(PriorityQueue* pq, int data, int priority) {
    if (pq->size == pq->capacity) {
        printf("Stack overflow. Cannot push.\n");
        return;
    }

    StackElement newElement = {data, priority};
    pq->elements[pq->size] = newElement;
    int current = pq->size;
    int parent = (current - 1) / 2;

    while (current > 0 && pq->elements[current].priority > pq->elements[parent].priority) {
        swap(&pq->elements[current], &pq->elements[parent]);
        current = parent;

        parent = (current - 1) / 2;
    }
    pq->size++;
}

int pop(PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Stack underflow. Cannot pop.\n");
        return -1;
    }
    int topData = pq->elements[0].data;
    pq->size--;
    pq->elements[0] = pq->elements[pq->size];

    int current = 0;
    while (1) {
        int leftChild = 2 * current + 1;
        int rightChild = 2 * current + 2;
        int largest = current;
        if (leftChild < pq->size && pq->elements[leftChild].priority > pq-
>elements[largest].priority) {
            largest = leftChild;
        }
        if (rightChild < pq->size && pq->elements[rightChild].priority > pq-
>elements[largest].priority) {
            largest = rightChild;
        }
        if (largest != current) {
            swap(&pq->elements[current], &pq->elements[largest]);
            current = largest;
        } else {
            break;
        }
    }

    return topData;
}

void display(PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack (Data, Priority): ");
    for (int i = 0; i < pq->size; i++) {
        printf("(%d, %d) ", pq->elements[i].data, pq->elements[i].priority);
    }
    printf("\n");
}
```

```
void freePriorityQueue(PriorityQueue* pq) {
    free(pq->elements);
    free(pq);
}

int main() {
    PriorityQueue* stack = createPriorityQueue(10);
    push(stack, 1, 3);
    push(stack, 2, 5);
    push(stack, 3, 1);
    push(stack, 4, 2);
    display(stack);
    int poppedElement = pop(stack);
    printf("Popped element: %d\n", poppedElement);
    display(stack);
    freePriorityQueue(stack);

    return 0;
}
```

Output:



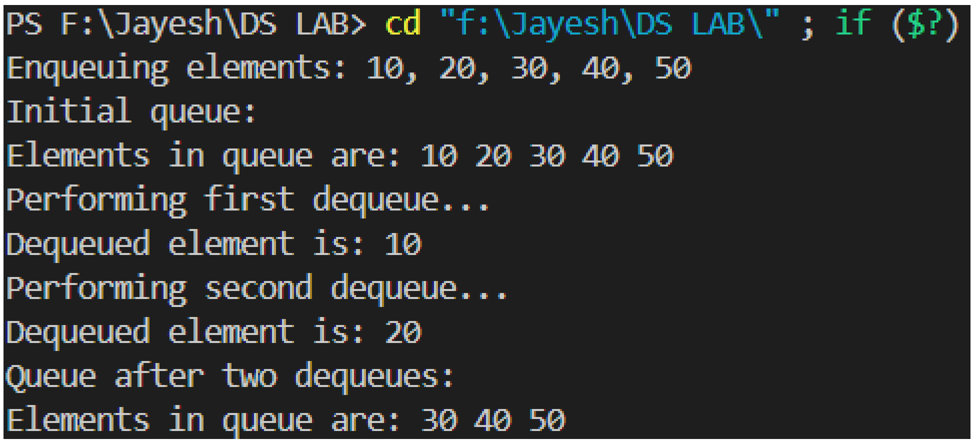
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Stack (Data, Priority): (2, 5) (1, 3) (3, 1) (4, 2)
Popped element: 2
Stack (Data, Priority): (1, 3) (4, 2) (3, 1)
```

15. Write a program to implement a queue using two stacks.

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

int Stack1[N];
int Stack2[N];
int top1 = -1;
int top2 = -1;
int count = 0;

void push1(int data) {
   Stack1[++top1] = data;
}

int pop1() {
   return Stack1[top1--];
}

int pop2() {
   return Stack2[top2--];
}

void push2(int data) {
   Stack2[++top2] = data;
}

void dequeue() {
   if (top1 == -1 && top2 == -1) {
      printf("Queue is empty\n");
      return;
   }
   for (int i = 0; i < count; i++) {
      push2(pop1());
   }
   printf("Dequeued element is: %d\n", pop2());
   count--;
   for (int i = 0; i < count; i++) {
      push1(pop2());
   }
}

void enqueue(int data) {
   push1(data);
   count++;
}
```

```
void display() {
    if (top1 == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Elements in queue are: ");
    for (int i = 0; i <= top1; i++) {
        printf("%d ", Stack1[i]);
    }
    printf("\n");
}

int main() {
    printf("Enqueuing elements: 10, 20, 30, 40, 50\n");
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);

    printf("Initial queue:\n");
    display();
    printf("Performing first dequeue...\n");
    dequeue();
    printf("Performing second dequeue...\n");
    dequeue();
    printf("Queue after two dequeues:\n");
    display();
    return 0;
}
```

Output:

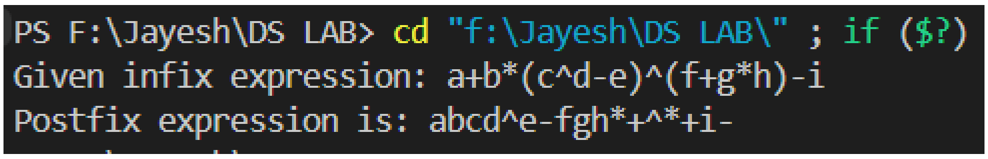


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Enqueuing elements: 10, 20, 30, 40, 50
Initial queue:
Elements in queue are: 10 20 30 40 50
Performing first dequeue...
Dequeued element is: 10
Performing second dequeue...
Dequeued element is: 20
Queue after two dequeues:
Elements in queue are: 30 40 50
```

16. Write a program to convert an infix expression to a postfix expression.

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

int prec(char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}

char associativity(char c) {
    if (c == '^')
        return 'R';
    return 'L';
}

void infixToPostfix(char s[]) {
    char result[1000];
    int resultIndex = 0;
    int len = strlen(s);
    char stack[1000];
    int stackIndex = -1;

    for (int i = 0; i < len; i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            result[resultIndex++] = c;
        } else if (c == '(') {
            stack[++stackIndex] = c;
        } else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--;
        } else {
            while (stackIndex >= 0 && (prec(s[i]) < prec(stack[stackIndex]) ||
                   (prec(s[i]) == prec(stack[stackIndex]) && associativity(s[i]) == 'L'))) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }

    while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
    }

    result[resultIndex] = '\0';
    printf("Postfix expression is: %s\n", result);
}

int main() {
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    printf("Given infix expression: %s\n", exp);
    infixToPostfix(exp);
    return 0;
}
```

Output:



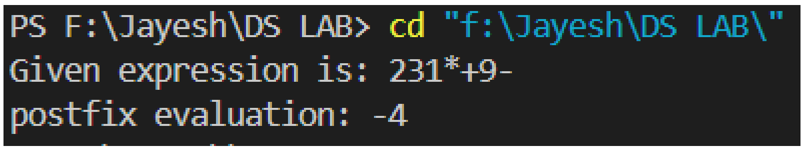
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Given infix expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression is: abcd^e-fgh*+^*+i-
```

17. Write a program to evaluate postfix expression.

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

struct Stack {
   int top;
   unsigned capacity;
   int* array;
};
struct Stack* createStack(unsigned capacity)
{
   struct Stack* stack
      = (struct Stack*)malloc(sizeof(struct Stack));

   if (!stack)
      return NULL;

   stack->top = -1;
   stack->capacity = capacity;
   stack->array
      = (int*)malloc(stack->capacity * sizeof(int));

   if (!stack->array)
      return NULL;

   return stack;
}

int isEmpty(struct Stack* stack)
{
   return stack->top == -1;
}

char peek(struct Stack* stack)
{
   return stack->array[stack->top];
}

char pop(struct Stack* stack)
{
   if (!isEmpty(stack))
      return stack->array[stack->top--];
   return '$';
}
```

```
void push(struct Stack* stack, char op)
{
   stack->array[++stack->top] = op;
}
int evaluatePostfix(char* exp)
{
   struct Stack* stack = createStack(strlen(exp));
   int i;
   if (!stack)
      return -1;
   for (i = 0; exp[i]; ++i) {
      if (isdigit(exp[i]))
         push(stack, exp[i] - '0');
      else {
         int val1 = pop(stack);
         int val2 = pop(stack);
         switch (exp[i]) {
         case '+':
            push(stack, val2 + val1);
            break;
         case '-':
            push(stack, val2 - val1);
            break;
         case '*':
            push(stack, val2 * val1);
            break;
         case '/':
            push(stack, val2 / val1);
            break;
         }
      }
   }
   return pop(stack);
}
int main(){
   char exp[] = "231*+9-";
   printf("Given expression is: 231*+9-\n");
   printf("postfix evaluation: %d\n", evaluatePostfix(exp));
   return 0;
}
```

Output:



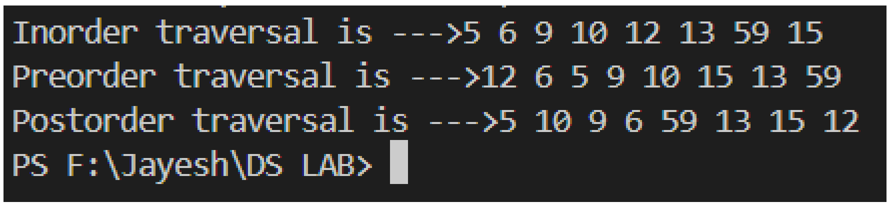
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Given expression is: 231*+9-
postfix evaluation: -4
```

18. Write a program to find out the preorder, inorder and postorder traversal of the tree.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *left,*right;
};
struct node* create(){
    struct node *newnode=(struct node*)malloc(sizeof(struct node));
    int x;
    printf("Enter data (-1  for no node):");
    scanf("%d",&x);
    if(x==-1) return 0;
    newnode->data=x;
    printf("Enter data for left child of %d\n",x);
    newnode->left=create();
    printf("Enter data for right child of %d\n",x);
    newnode->right=create();
    return newnode;
}
void inorder(struct node *root){
   if(root==0) return;
   inorder(root->left);
   printf("%d ",root->data);
   inorder(root->right);
}
void preorder(struct node *root){
   if(root==0) return;
   printf("%d ",root->data);
   preorder(root->left);
   preorder(root->right);
}
void postorder(struct node *root){
   if(root==0) return;
   postorder(root->left);
   postorder(root->right);
    printf("%d ",root->data);
}  int main(){
   struct node *root;
   root=0;
   root=create();
   printf("Inorder traversal is --->");
   inorder(root);
   printf("\n");
```

```
    printf("Preorder traversal is --->");
    preorder(root);
     printf("\n");
     printf("Postorder traversal is --->");
    postorder(root);
     printf("\n");
}
```

Output:



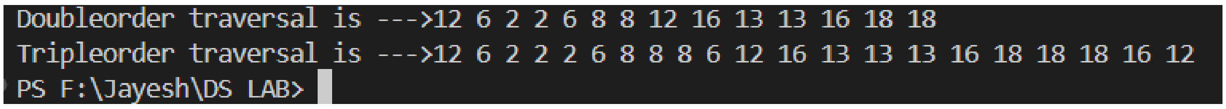
```
Inorder traversal is --->5 6 9 10 12 13 59 15
Preorder traversal is --->12 6 5 9 10 15 13 59
Postorder traversal is --->5 10 9 6 59 13 15 12
PS F:\Jayesh\DS LAB>
```

19. Write a program to perform double-order traversal and triple-order traversal on the tree.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *left,*right;
};
struct node* create(){
    struct node *newnode=(struct node*)malloc(sizeof(struct node));
    int x;
    printf("Enter data (-1  for no node):");
    scanf("%d",&x);
    if(x==-1) return 0;
    newnode->data=x;
    printf("Enter data for left child of %d\n",x);
    newnode->left=create();
    printf("Enter data for right child of %d\n",x);
    newnode->right=create();
    return newnode;
}

void doubleorder(struct node *root){
   if(root==0) return;
   printf("%d ",root->data);
   doubleorder(root->left);
   printf("%d ",root->data);
   doubleorder(root->right);
}

void tripleorder(struct node *root){
   if(root==0) return;
   printf("%d ",root->data);
   tripleorder(root->left);
   printf("%d ",root->data);
   tripleorder(root->right);
   printf("%d ",root->data);
}

int main(){
   struct node *root;
   root=0;
   root=create();
   printf("Doubleorder traversal is --->");
   doubleorder(root);
   printf("\n");
```

```
    printf("Tripleorder traversal is --->");
    tripleorder(root);
     printf("\n");
}
```

Output:

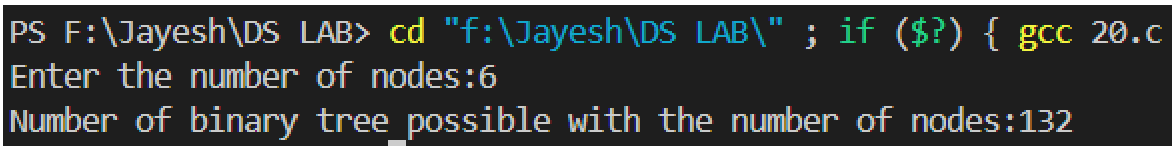


```
Doubleorder traversal is --->12 6 2 2 6 8 8 12 16 13 13 16 18 18
Tripleorder traversal is --->12 6 2 2 2 6 8 8 8 6 12 16 13 13 13 16 18 18 18 16 12
PS F:\Jayesh\DS LAB>
```

20. Write a program to find the number of binary trees possible with given number of nodes.

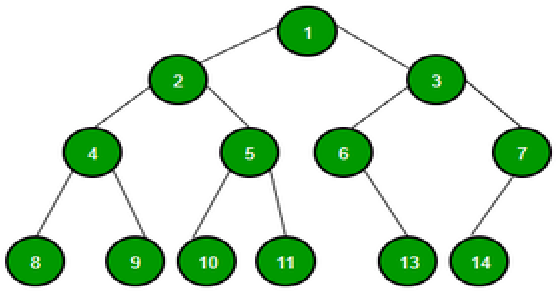
```
#include <stdio.h>
int factorial(int n){
    if(n==0 || n==1) return 1;
    else return(n*factorial(n-1));
}

int main()
{
    int n;
    printf("Enter the number of nodes:");
    scanf("%d",&n);
    int res=factorial(2 * n) / (factorial(n + 1) * factorial(n));
    printf("Number of binary tree possible with the number of nodes:%d \n",res);
    return 0;
}
```

Output:



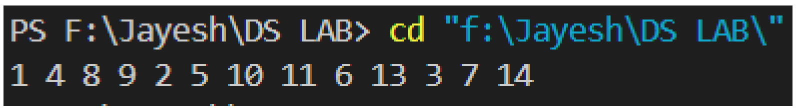
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 20.c
Enter the number of nodes:6
Number of binary tree possible with the number of nodes:132
```

21. Write a program to perform indirect recursion on the tree.



```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
struct node *createNode(int data)
{
    struct node *n = (struct node*)malloc(sizeof(struct node));
    n->data = data;
     n->right = NULL;
     n->left = NULL;
     return n;
}
// indirect recursion
void B(struct node *r);
void A(struct node *r)
{
    if(r)
    {
        printf("%d ",r->data);
        B(r->left);
        B(r->right);
     }
}
void B(struct node *r)
{
    if(r) {
        A(r->left);
        printf("%d ",r->data);
        A(r->right);
    }
}
int main()
{
struct node *r = createNode(1);
struct node *r1 = createNode(2);
struct node *r2 = createNode(3);
struct node *r3 = createNode(4);
struct node *r4 = createNode(5);
struct node *r5 = createNode(6);
struct node *r6 = createNode(7);
struct node *r7 = createNode(8);
struct node *r8 = createNode(9);
struct node *r9 = createNode(10);
struct node *r10 = createNode(11);
struct node *r11 = createNode(13);
struct node *r12 = createNode(14);
r->left = r1;
r->right = r2;
r1->left = r3;
r1->right = r4;
r2->left = r5;
r2->right = r6;
r3->left = r7;
r3->right = r8;
r4->left = r9;
r4->right = r10;
r5->right = r11;
r6->left = r12;
A(r);
return 0;
}
```

Output:

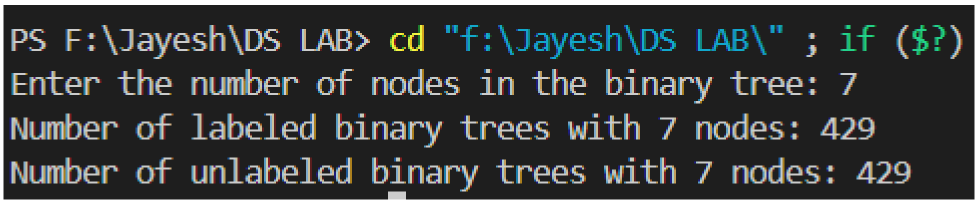


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
1 4 8 9 2 5 10 11 6 13 3 7 14
```

22. Write a program to find out possible labelled and unlabeled binary trees with the given number of nodes.

```
#include <stdio.h>
unsigned long long factorial(int n) {
   if (n == 0 || n == 1)
      return 1;
   else
      return n * factorial(n - 1);
}
unsigned long long labeled_binary_trees(int n) {
     return factorial(2 * n) / (factorial(n + 1) * factorial(n));
}
unsigned long long unlabeled_binary_trees(int n) {
     return labeled_binary_trees(n);
}
int main() {
   int num_nodes;
   printf("Enter the number of nodes in the binary tree: ");
   scanf("%d", &num_nodes);
   unsigned long long labeled_trees = labeled_binary_trees(num_nodes);
   printf("Number of labeled binary trees with %d nodes: %llu\n", num_nodes, labeled_trees);
      unsigned long long unlabeled_trees = unlabeled_binary_trees(num_nodes);
   printf("Number of unlabeled binary trees with %d nodes: %llu\n", num_nodes,
unlabeled_trees);
   return 0;
}
```

Output:



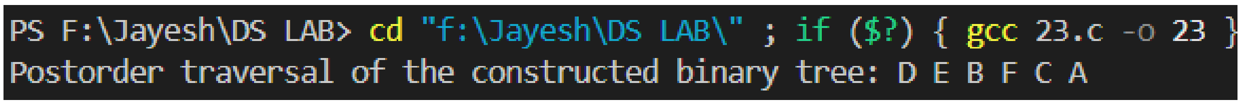
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Enter the number of nodes in the binary tree: 7
Number of labeled binary trees with 7 nodes: 429
Number of unlabeled binary trees with 7 nodes: 429
```

23. Write a program to construct the unique binary tree using inorder and preorder traversal and hence find postorder.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct TreeNode {
    char data;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* newNode(char data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int search(char arr[], int start, int end, char value) {
    int i;
    for (i = start; i <= end; i++) {
        if (arr[i] == value)
            return i;
    }
    return -1;
}
struct TreeNode* buildTree(char inorder[], char preorder[], int inStart, int inEnd) {
    static int preIndex = 0;
    if (inStart > inEnd)
        return NULL;
      struct TreeNode* node = newNode(preorder[preIndex++]);
    if (inStart == inEnd)
        return node;
    int inIndex = search(inorder, inStart, inEnd, node->data);
    node->left = buildTree(inorder, preorder, inStart, inIndex - 1);
    node->right = buildTree(inorder, preorder, inIndex + 1, inEnd);
    return node;
}
void printPostorder(struct TreeNode* node) {
    if (node == NULL)
        return;
    printPostorder(node->left);
    printPostorder(node->right);
    printf("%c ", node->data);
}
```

```
int main() {
    char inorder[] = {'D', 'B', 'E', 'A', 'F', 'C'};
    char preorder[] = {'A', 'B', 'D', 'E', 'C', 'F'};
    int len = strlen(inorder);
    struct TreeNode* root = buildTree(inorder, preorder, 0, len - 1);
    printf("Postorder traversal of the constructed binary tree: ");
    printPostorder(root);
    return 0;
}
```

Output:

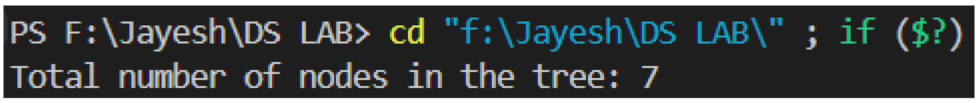


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 23.c -o 23 }
Postorder traversal of the constructed binary tree: D E B F C A
```

24. Write a recursive program to count the total number of nodes in the tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int countNodes(struct TreeNode* root) {
    if (root == NULL)
        return 0;
    else
        return 1 + countNodes(root->left) + countNodes(root->right);
}
int main() {
    struct TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    int totalNodes = countNodes(root);
    printf("Total number of nodes in the tree: %d\n", totalNodes);
    return 0;
}
```

Output:

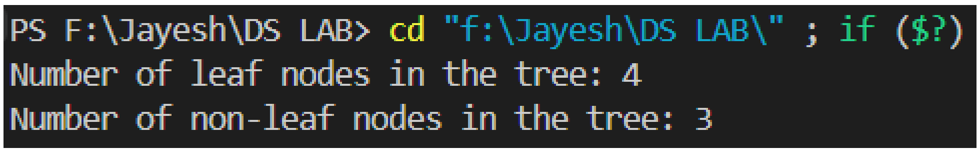


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Total number of nodes in the tree: 7
```

25. Write a recursive program to count the number of the leaf or non-leaf nodes of the tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int countLeafNodes(struct TreeNode* root) {
    if (root == NULL)
        return 0;
    else if (root->left == NULL && root->right == NULL)
        return 1; // Leaf node
    else
        return countLeafNodes(root->left) + countLeafNodes(root->right);
}
int countNonLeafNodes(struct TreeNode* root) {
    if (root == NULL || (root->left == NULL && root->right == NULL))
        return 0; // Empty node or leaf node
    else
        return 1 + countNonLeafNodes(root->left) + countNonLeafNodes(root->right);
}
int main() {
    struct TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    int leafNodes = countLeafNodes(root);
    printf("Number of leaf nodes in the tree: %d\n", leafNodes);
    int nonLeafNodes = countNonLeafNodes(root);
    printf("Number of non-leaf nodes in the tree: %d\n", nonLeafNodes);
    return 0;
}
```

Output:

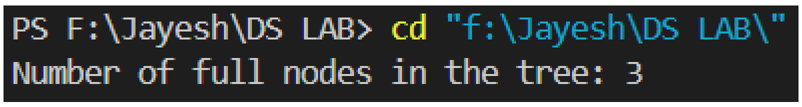


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Number of leaf nodes in the tree: 4
Number of non-leaf nodes in the tree: 3
```

26. Write a recursive program to count the number of full nodes of the tree (Full Nodes are nodes which has both left and right children as non-empty).

```c
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int countFullNodes(struct TreeNode* root) {
    if (root == NULL)
        return 0;
    else if (root->left != NULL && root->right != NULL)
        return 1 + countFullNodes(root->left) + countFullNodes(root->right);
    else
        return countFullNodes(root->left) + countFullNodes(root->right);
}

int main() {
     struct TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->left->left->left = newNode(8);
    root->left->right->right = newNode(9);
    int fullNodes = countFullNodes(root);
    printf("Number of full nodes in the tree: %d\n", fullNodes);
    return 0;
}
```

Output:

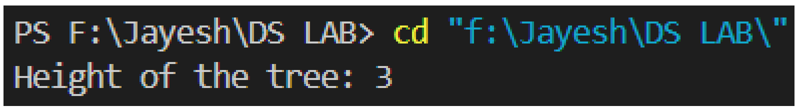


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Number of full nodes in the tree: 3
```

27. Write a recursive program to find the height of the tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int treeHeight(struct TreeNode* root) {
    if (root == NULL)
        return -1; // Height of an empty tree is -1
    else {
        int leftHeight = treeHeight(root->left);
        int rightHeight = treeHeight(root->right);
            return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);
    }
}
int main() {
    struct TreeNode* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->right->right = newNode(8);
    int height = treeHeight(root);
    printf("Height of the tree: %d\n", height);
    return 0;
}
```

Output:



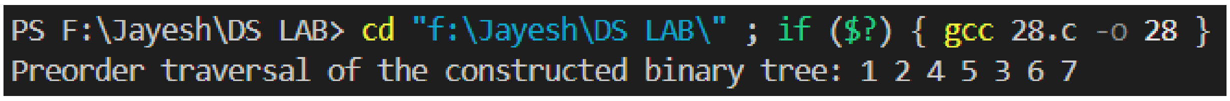
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Height of the tree: 3
```

28. Write a program to construct BST using inorder and postorder traversal and hence find preorder.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
int search(int inorder[], int start, int end, int value) {
    int i;
    for (i = start; i <= end; i++) {
        if (inorder[i] == value)
            return i;
    }
    return -1;
}
struct TreeNode* buildTree(int inorder[], int postorder[], int inStart, int inEnd, int* postIndex) {
    if (inStart > inEnd)
        return NULL;
    struct TreeNode* node = newNode(postorder[*postIndex]);
    (*postIndex)--;
    if (inStart == inEnd)
        return node;
    int inIndex = search(inorder, inStart, inEnd, node->data);
    node->right = buildTree(inorder, postorder, inIndex + 1, inEnd, postIndex);
    node->left = buildTree(inorder, postorder, inStart, inIndex - 1, postIndex);
    return node;
}
void printPreorder(struct TreeNode* node) {
    if (node == NULL)
        return;
    printf("%d ", node->data);
    printPreorder(node->left);
    printPreorder(node->right);
}
```

```
int main() {
    int inorder[] = {4, 2, 5, 1, 6, 3, 7};
    int postorder[] = {4, 5, 2, 6, 7, 3, 1};
    int len = sizeof(inorder) / sizeof(inorder[0]);
    int postIndex = len - 1;
    struct TreeNode* root = buildTree(inorder, postorder, 0, len - 1, &postIndex);
    printf("Preorder traversal of the constructed binary tree: ");
    printPreorder(root);
    return 0;
}
```

Output:

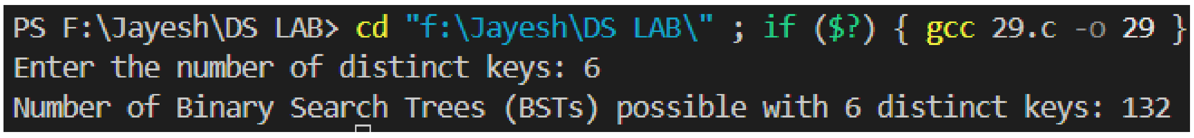


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 28.c -o 28 }
Preorder traversal of the constructed binary tree: 1 2 4 5 3 6 7
```

29. Write a program to find how many BSTs are possible with given distinct keys.

```
#include <stdio.h>
#include <stdlib.h>
int countBSTs(int n) {
    int* dp = (int*)malloc((n + 1) * sizeof(int));
    dp[0] = dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = 0;
        for (int j = 0; j < i; j++) {
            dp[i] += dp[j] * dp[i - j - 1];
        }
    }
    int result = dp[n];
    free(dp);
    return result;
}
int main() {
    int num_keys;
    printf("Enter the number of distinct keys: ");
    scanf("%d", &num_keys);
    int num_BSTs = countBSTs(num_keys);
    printf("Number of Binary Search Trees (BSTs) possible with %d distinct keys: %d\n",
num_keys, num_BSTs);
    return 0;
}
```

Output:



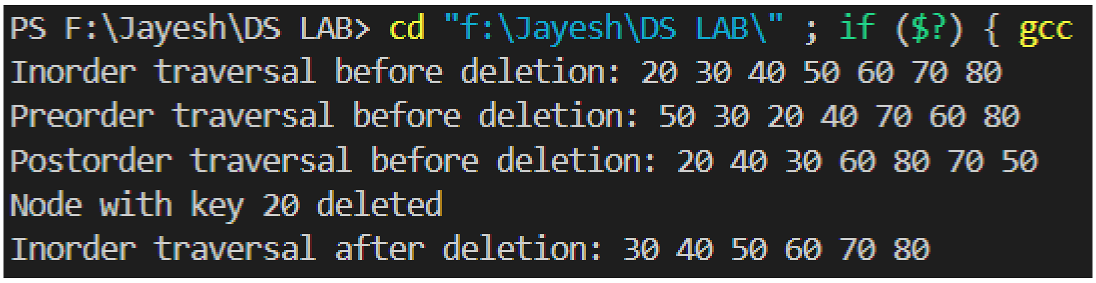
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 29.c -o 29 }
Enter the number of distinct keys: 6
Number of Binary Search Trees (BSTs) possible with 6 distinct keys: 132
```

30. Write a program to find out the postorder, preorder and inorder traversal on constructed BST and then perform delete operation on the tree and again perform inorder traversal.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL)
        return newNode(data);
    if (data < root->data)
        root->left = insertNode(root->left, data);
    else if (data > root->data)
        root->right = insertNode(root->right, data);
    return root;
}
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void preorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
```

```
}
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL)
        return root;

        if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // Node with only one child or no child
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        struct TreeNode* temp = minValueNode(root->right);
             root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
int main() {
    struct TreeNode* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);
    printf("Inorder traversal before deletion: ");
    inorderTraversal(root);
    printf("\n");
    printf("Preorder traversal before deletion: ");
    preorderTraversal(root);
    printf("\n");
    printf("Postorder traversal before deletion: ");
    postorderTraversal(root);
    printf("\n");
    root = deleteNode(root, 20);
    printf("Node with key 20 deleted\n");
    printf("Inorder traversal after deletion: ");
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

Output:

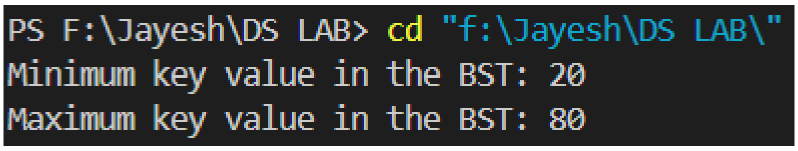


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Inorder traversal before deletion: 20 30 40 50 60 70 80
Preorder traversal before deletion: 50 30 20 40 70 60 80
Postorder traversal before deletion: 20 40 30 60 80 70 50
Node with key 20 deleted
Inorder traversal after deletion: 30 40 50 60 70 80
```

31. Write a program to find the minimum and maximum key values from BSTs.

```c
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
struct TreeNode* insertNode(struct TreeNode* root, int data) {
    if (root == NULL)
        return newNode(data);
    if (data < root->data)
        root->left = insertNode(root->left, data);
    else if (data > root->data)
        root->right = insertNode(root->right, data);
    return root;
}
int findMinValue(struct TreeNode* root) {
    if (root == NULL) {
        printf("BST is empty.\n");
        return -1;
    }
    while (root->left != NULL)
        root = root->left;
    return root->data;
}
int findMaxValue(struct TreeNode* root) {
    if (root == NULL) {
        printf("BST is empty.\n");
        return -1;
    }
    while (root->right != NULL)
        root = root->right;
    return root->data;
}
int main() {
    struct TreeNode* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);
    int minValue = findMinValue(root);
    printf("Minimum key value in the BST: %d\n", minValue);
    int maxValue = findMaxValue(root);
    printf("Maximum key value in the BST: %d\n", maxValue);
    return 0;
}
```

Output:



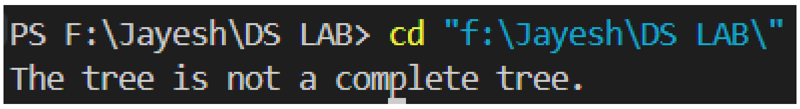
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Minimum key value in the BST: 20
Maximum key value in the BST: 80
```

32. Write a recursive program to check whether given tree is complete tree or not.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
   int data;
   struct TreeNode* left;
   struct TreeNode* right;
};
struct TreeNode* newNode(int data) {
   struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return node;
}
int height(struct TreeNode* root) {
   if (root == NULL)
      return 0;
   else {
      int leftHeight = height(root->left);
      int rightHeight = height(root->right);
      return (leftHeight > rightHeight) ? (leftHeight + 1) : (rightHeight + 1);
   }
}
int isCompleteTree(struct TreeNode* root, int index, int numNodes) {
   if (root == NULL)
      return 1;
   if (index >= numNodes)
      return 0;
   return (isCompleteTree(root->left, 2 * index + 1, numNodes) && isCompleteTree(root->right,
2 * index + 2, numNodes));
}
int main() {    struct TreeNode* root = newNode(1);
   root->left = newNode(2);
   root->right = newNode(3);
   root->left->left = newNode(4);
   root->left->right = newNode(5);
   root->right->left = newNode(6);
   int numNodes = height(root);
   int index = 0;
```

```
    if (isCompleteTree(root, index, numNodes))
        printf("The tree is a complete tree.\n");
    else
        printf("The tree is not a complete tree.\n");
    return 0;
}
```

Output:



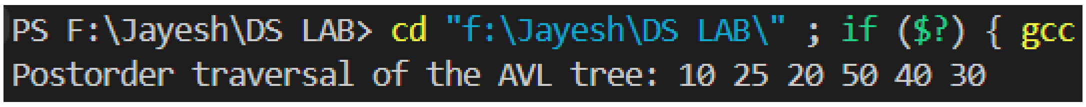
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
The tree is not a complete tree.
```

33. Write a program to construct an AVL tree and perform postorder traversal.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
    int height;
};
struct TreeNode* newNode(int data) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}
int height(struct TreeNode* node) {
    if (node == NULL)
        return 0;
    return node->height;
}
struct TreeNode* rightRotate(struct TreeNode* y) {
    struct TreeNode* x = y->left;
    struct TreeNode* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = (height(y->left) > height(y->right) ? height(y->left) : height(y->right)) + 1;
    x->height = (height(x->left) > height(x->right) ? height(x->left) : height(x->right)) + 1;
    return x;
}
struct TreeNode* leftRotate(struct TreeNode* x) {
    struct TreeNode* y = x->right;
    struct TreeNode* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = (height(x->left) > height(x->right) ? height(x->left) : height(x->right)) + 1;
    y->height = (height(y->left) > height(y->right) ? height(y->left) : height(y->right)) + 1;
    return y;
}
int getBalance(struct TreeNode* node) {
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}
struct TreeNode* insertNode(struct TreeNode* node, int data) {
    if (node == NULL)
```

```
        return newNode(data);
    if (data < node->data)
        node->left = insertNode(node->left, data);
    else if (data > node->data)
        node->right = insertNode(node->right, data);
    else
        return node;
    node->height = (height(node->left) > height(node->right) ? height(node->left) : height(node-
>right)) + 1;
    int balance = getBalance(node);
    if (balance > 1 && data < node->left->data)
        return rightRotate(node);
    if (balance < -1 && data > node->right->data)
        return leftRotate(node);
    if (balance > 1 && data > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && data < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
void postorderTraversal(struct TreeNode* node) {
    if (node != NULL) {
        postorderTraversal(node->left);
        postorderTraversal(node->right);
        printf("%d ", node->data);
    }
}
int main() {
    struct TreeNode* root = NULL;
    root = insertNode(root, 10);
    root = insertNode(root, 20);
    root = insertNode(root, 30);
    root = insertNode(root, 40);
    root = insertNode(root, 50);
    root = insertNode(root, 25);
    printf("Postorder traversal of the AVL tree: ");
    postorderTraversal(root);
    printf("\n");
    return 0;
}
```

Output:

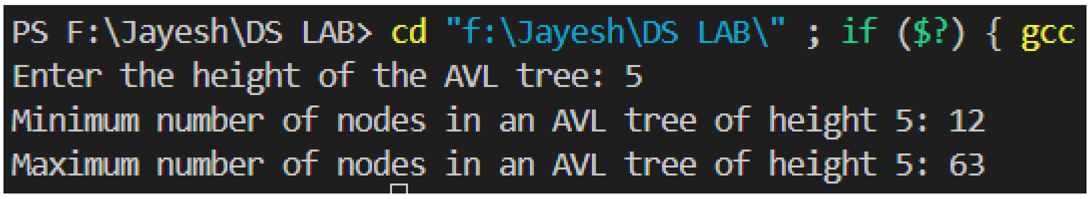


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Postorder traversal of the AVL tree: 10 25 20 50 40 30
```

34. Write a program to find the minimum and maximum nodes in an AVL tree of given height.

```c
#include <stdio.h>
#include <stdlib.h>
int fibonacci(int h) {
    if (h <= 1)
        return h;
    return fibonacci(h - 1) + fibonacci(h - 2);
}
int minNodes(int h) {
    return fibonacci(h + 2) - 1;
}
int maxNodes(int h) {
    return (1 << (h + 1)) - 1;
}
int main() {
    int height;
    printf("Enter the height of the AVL tree: ");
    scanf("%d", &height);
    int min = minNodes(height);
    int max = maxNodes(height);
    printf("Minimum number of nodes in an AVL tree of height %d: %d\n", height, min);
    printf("Maximum number of nodes in an AVL tree of height %d: %d\n", height, max);
    return 0;
}
```

Output:

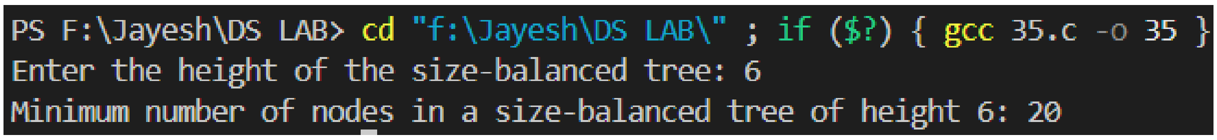


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Enter the height of the AVL tree: 5
Minimum number of nodes in an AVL tree of height 5: 12
Maximum number of nodes in an AVL tree of height 5: 63
```

35. Write a program to find the minimum number of nodes on a size-balanced tree.

```
#include <stdio.h>
#include <stdlib.h>
int minNodesSizeBalancedTree(int h) {
    if (h <= 0)
        return 0;
    if (h == 1)
        return 1;
    return minNodesSizeBalancedTree(h - 1) + minNodesSizeBalancedTree(h - 2) + 1;
}
int main() {
    int height;
    printf("Enter the height of the size-balanced tree: ");
    scanf("%d", &height);
    int minNodes = minNodesSizeBalancedTree(height);
    printf("Minimum number of nodes in a size-balanced tree of height %d: %d\n", height,
minNodes);
    return 0;
}
```

Output:



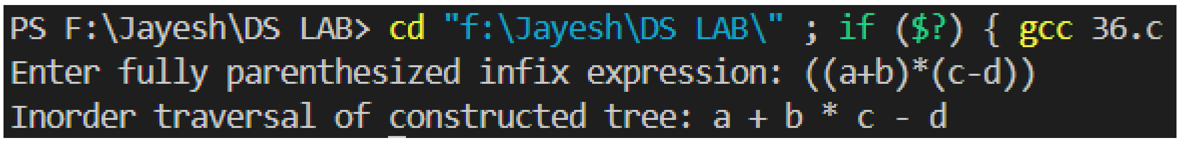
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 35.c -o 35 }
Enter the height of the size-balanced tree: 6
Minimum number of nodes in a size-balanced tree of height 6: 20
```

36. Write a program to implement a tree for a given infix expression.

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

struct Node {
    char data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(char data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

struct Node* constructTree(char* expr, int* index) {
    if (expr[*index] == '\0') return NULL;
    if (expr[*index] == '(') {
        (*index)++;
        struct Node* left = constructTree(expr, index);
        char op = expr[(*index)++];
        struct Node* right = constructTree(expr, index);
        (*index)++;
        struct Node* node = createNode(op);
        node->left = left;
        node->right = right;
        return node;
    } else if (isalnum(expr[*index])) {
        return createNode(expr[(*index)++]);
    }
    return NULL;
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%c ", root->data);
    inorder(root->right);
}
```

```
int main() {
    char expr[100];
    printf("Enter fully parenthesized infix expression: ");
    scanf("%s", expr);

    int index = 0;
    struct Node* root = constructTree(expr, &index);

    printf("Inorder traversal of constructed tree: ");
    inorder(root);

    return 0;
}
```

Output:

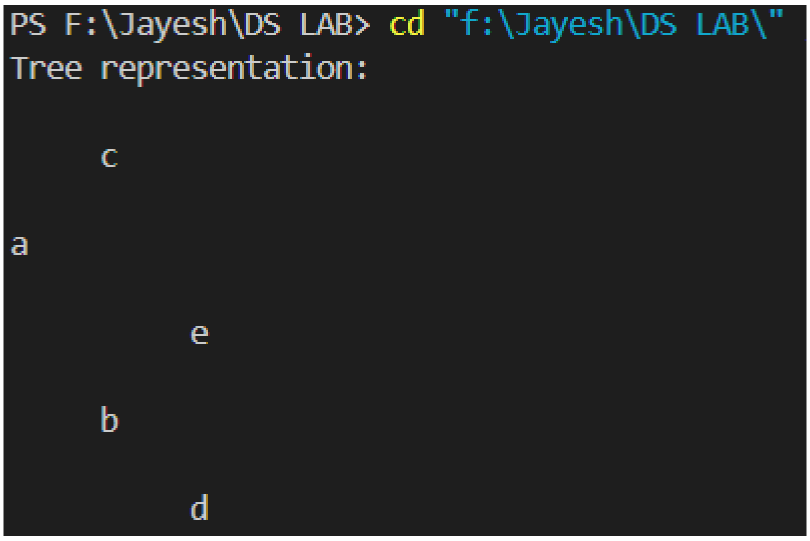


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 36.c
Enter fully parenthesized infix expression: ((a+b)*(c-d))
Inorder traversal of constructed tree: a + b * c - d
```

37. Write a program to draw a tree for a given nested tree representation expression.

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

struct Node {
    char data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(char data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

void drawTree(struct Node* root, int space) {
    if (root == NULL) return;

    space += 5;
    drawTree(root->right, space);

    printf("\n");
    for (int i = 5; i < space; i++) {
        printf(" ");
    }
    printf("%c\n", root->data);

    drawTree(root->left, space);
}

int main() {
    struct Node* root = newNode('a');
    root->left = newNode('b');
    root->right = newNode('c');
    root->left->left = newNode('d');
    root->left->right = newNode('e');

    printf("Tree representation:\n");
    drawTree(root, 0);
    return 0;
}
```

Output:

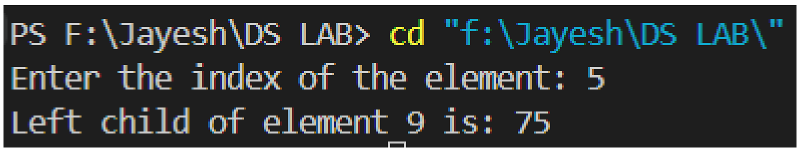


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Tree representation:

     c

a

          e

     b

          d
```

38. Write a program to find the left child of kth element from the given array representation tree.

```
#include<stdio.h>

int findLeftChild(int arr[], int k, int n) {
   int index = k - 1;
   int leftChild = 2 * index + 1;
   if (leftChild < n) return arr[leftChild];
   return -1;
}

int main() {
   int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
   int n = sizeof(arr) / sizeof(arr[0]);
   int k;
   printf("Enter the index of the element: ");
   scanf("%d", &k);
   int leftChild = findLeftChild(arr, k, n);
   if (leftChild == -1) {
      printf("No left child for the given element.\n");
   } else {
      printf("Left child of element %d is: %d\n", arr[k-1], leftChild);
   }
   return 0;
}
```

Output:



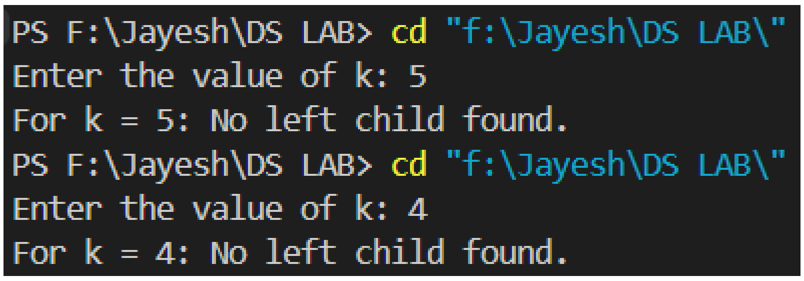
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Enter the index of the element: 5
Left child of element 9 is: 75
```

39. Write a program to find the left child of kth element from the given leftmost child right sibling representation tree.

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
   int data;
   struct TreeNode* leftmost_child;
   struct TreeNode* right_sibling;
};
struct TreeNode* newNode(int data) {
   struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
   node->data = data;
   node->leftmost_child = NULL;
   node->right_sibling = NULL;
   return node;
}
struct TreeNode* findKthElement(struct TreeNode* root, int k) {
   if (root == NULL)
      return NULL;
   if (k == 0)
      return root->leftmost_child;
   struct TreeNode* sibling = root->leftmost_child;
   while (k > 0 && sibling != NULL) {
      sibling = sibling->right_sibling;
      k--;
   }
   return sibling;
}
void printLeftChild(struct TreeNode* leftChild) {
   if (leftChild != NULL)
      printf("Left child: %d\n", leftChild->data);
   else
      printf("No left child found.\n");
}
int main() {
   struct TreeNode* root = newNode(1);
   root->leftmost_child = newNode(2);
   root->leftmost_child->right_sibling = newNode(3);
   root->leftmost_child->right_sibling->right_sibling = newNode(4);
   root->leftmost_child->leftmost_child = newNode(5);
   root->leftmost_child->leftmost_child->right_sibling = newNode(6);
   root->leftmost_child->leftmost_child->right_sibling->right_sibling = newNode(7);
   int k;
   printf("Enter the value of k: ");
   scanf("%d", &k);
   struct TreeNode* kthElement = findKthElement(root, k);
```

```
    struct TreeNode* leftChild = NULL;
    if (kthElement != NULL)
        leftChild = kthElement->leftmost_child;
    printf("For k = %d: ", k);
    printLeftChild(leftChild);
    return 0;
}
```

Output:

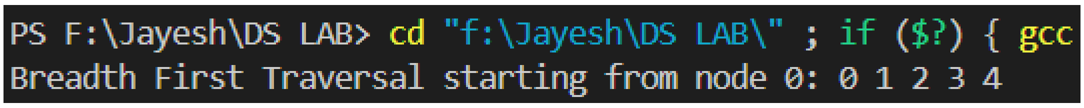


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Enter the value of k: 5
For k = 5: No left child found.
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Enter the value of k: 4
For k = 4: No left child found.
```

40. Write a program for breadth first traversal on graph.

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

#define MAX 10

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

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

int isFull(struct Queue* q) {
    return (q->rear == MAX - 1);
}

int isEmpty(struct Queue* q) {
    return (q->front == -1);
}

void enqueue(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full.\n");
        return;
    }
    if (q->front == -1) q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) q->front = q->rear = -1;
    return item;
}

void bfs(int graph[MAX][MAX], int start, int n) {
    struct Queue q;
    initQueue(&q);
    int visited[MAX] = {0};
    enqueue(&q, start);
    visited[start] = 1;

    while (!isEmpty(&q)) {
        int node = dequeue(&q);
        printf("%d ", node);

        for (int i = 0; i < n; i++) {
            if (graph[node][i] == 1 && !visited[i]) {
                enqueue(&q, i);
                visited[i] = 1;
            }
        }
    }
}

int main() {
    int graph[MAX][MAX] = {{0, 1, 1, 0, 0},
                           {1, 0, 1, 1, 0},
                           {1, 1, 0, 0, 0},
                           {0, 1, 0, 0, 1},
                           {0, 0, 0, 1, 0}};

    printf("Breadth First Traversal starting from node 0: ");
    bfs(graph, 0, 5);

    return 0;
}
```

Output:

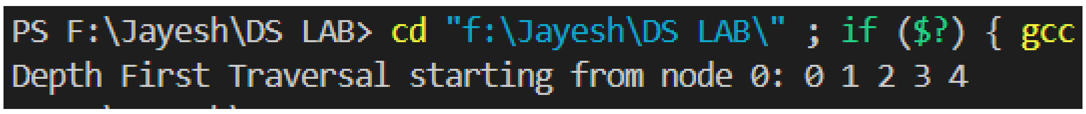


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Breadth First Traversal starting from node 0: 0 1 2 3 4
```

41. Write a program for depth first traversal on graph.

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

#define MAX 10

void dfs(int graph[MAX][MAX], int visited[MAX], int node, int n) {
    visited[node] = 1;
    printf("%d ", node);

    for (int i = 0; i < n; i++) {
        if (graph[node][i] == 1 && !visited[i]) {
            dfs(graph, visited, i, n);
        }
    }
}

int main() {
    int graph[MAX][MAX] = {{0, 1, 1, 0, 0},
                           {1, 0, 1, 1, 0},
                           {1, 1, 0, 0, 0},
                           {0, 1, 0, 0, 1},
                           {0, 0, 0, 1, 0}};

    int visited[MAX] = {0};
    printf("Depth First Traversal starting from node 0: ");
    dfs(graph, visited, 0, 5);

    return 0;
}
```

Output:



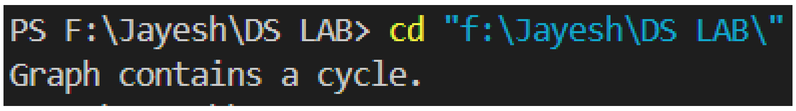
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc
Depth First Traversal starting from node 0: 0 1 2 3 4
```

42. Write a program to check whether there is a cycle in a given directed graph or not.

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

#define MAX 10

int graph[MAX][MAX];
int visited[MAX];
int recStack[MAX];

int isCycleUtil(int v, int n) {
   if (recStack[v]) return 1; // Node is in the recursion stack, cycle detected
   if (visited[v]) return 0;  // Node already visited, no cycle

   visited[v] = 1;
   recStack[v] = 1;

   for (int i = 0; i < n; i++) {
      if (graph[v][i] && isCycleUtil(i, n)) return 1;
   }

   recStack[v] = 0;
   return 0;
}

int isCycle(int n) {
   for (int i = 0; i < n; i++) {
      if (!visited[i] && isCycleUtil(i, n)) return 1;
   }
   return 0;
}

int main() {
   int n = 4;
   int edges[][2] = {{0, 1}, {1, 2}, {2, 3}, {3, 1}}; // Example graph with a cycle

   for (int i = 0; i < 4; i++) {
      graph[edges[i][0]][edges[i][1]] = 1;
   }
   for (int i = 0; i < n; i++) {
      visited[i] = 0;
      recStack[i] = 0;
   }
```

```
    if (isCycle(n)) {
        printf("Graph contains a cycle.\n");
    } else {
        printf("Graph does not contain a cycle.\n");
    }

    return 0;
}
```

Output:

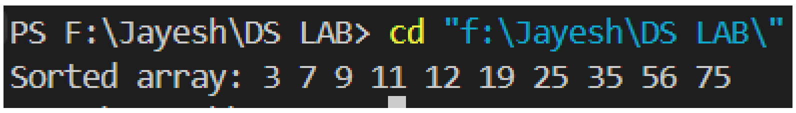


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Graph contains a cycle.
```

43. Write a program to sort given elements using insertion sort method.

```c
#include<stdio.h>
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

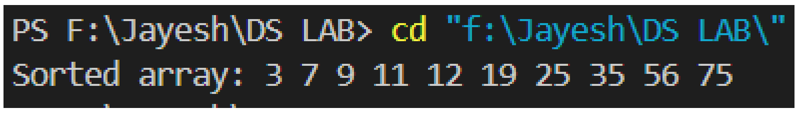


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

44. Write a program to sort given elements using bubble sort method.

```c
#include<stdio.h>
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

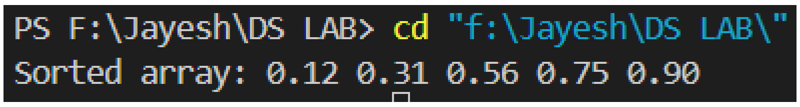


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

45. Write a program to sort given elements using bucket sort method.

```
#include<stdio.h>
#include<stdlib.h>
void bucketSort(float arr[], int n) {
    if (n <= 0) return;

    // Create buckets
    float bucket[n];
    for (int i = 0; i < n; i++) {
        bucket[i] = arr[i];
    }

    // Sort the buckets
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (bucket[i] > bucket[j]) {
                float temp = bucket[i];
                bucket[i] = bucket[j];
                bucket[j] = temp;
            }
        }
    }

    // Copy the sorted elements back to the original array
    for (int i = 0; i < n; i++) {
        arr[i] = bucket[i];
    }
}

int main() {
    float arr[] = {0.12, 0.56, 0.31, 0.75, 0.9};
    int n = sizeof(arr) / sizeof(arr[0]);

    bucketSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%.2f ", arr[i]);
    }
    return 0;
}
```

Output:

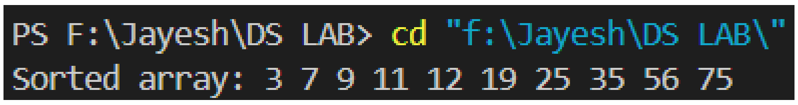


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 0.12 0.31 0.56 0.75 0.90
```

46. Write a program to sort given elements using merge sort method.

```
#include<stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int i = 0; i < n2; i++) {
        R[i] = arr[mid + 1 + i];
    }

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n - 1);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

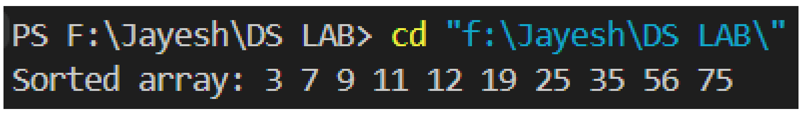


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

47. Write a program to sort given elements using quick sort method.

```c
#include <stdio.h>

void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pivot = arr[high];
      int i = low - 1;

      for (int j = low; j < high; j++) {
         if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }

      int temp = arr[i + 1];
      arr[i + 1] = arr[high];
      arr[high] = temp;

      int pi = i + 1;

      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

int main() {
   int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
   int n = sizeof(arr) / sizeof(arr[0]);

   quickSort(arr, 0, n - 1);

   printf("Sorted array: ");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   return 0;
}
```

Output:

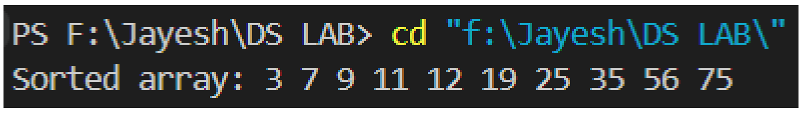


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

48. Write a program to sort given elements using heap sort method.

```
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest]) {
        largest = left; }
    if (right < n && arr[right] > arr[largest]) {
        largest = right; }
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}
int main() {
    int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arr) / sizeof(arr[0]);
    heapSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

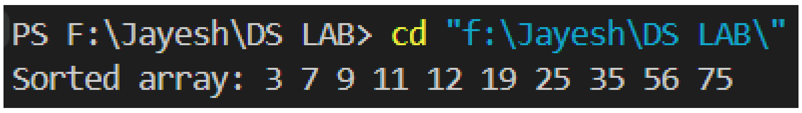


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

49. Write a program to sort given elements using insertion sort method.

```c
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

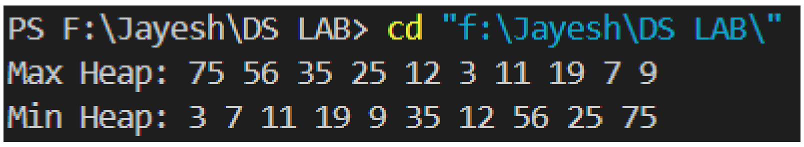


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Sorted array: 3 7 9 11 12 19 25 35 56 75
```

50. Write a program to construct min heap and max heap.

```
#include <stdio.h>

void heapifyMax(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        heapifyMax(arr, n, largest);
    }
}

void heapifyMin(int arr[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] < arr[smallest]) {
        smallest = left;
    }
    if (right < n && arr[right] < arr[smallest]) {
        smallest = right;
    }

    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;

        heapifyMin(arr, n, smallest);
    }
}

void buildMaxHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapifyMax(arr, n, i);
    }
}

void buildMinHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapifyMin(arr, n, i);
    }
}

int main() {
    int arrMax[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int arrMin[] = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
    int n = sizeof(arrMax) / sizeof(arrMax[0]);

    buildMaxHeap(arrMax, n);
    buildMinHeap(arrMin, n);

    printf("Max Heap: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arrMax[i]);
    }

    printf("\nMin Heap: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arrMin[i]);
    }

    return 0;
}
```

Output:

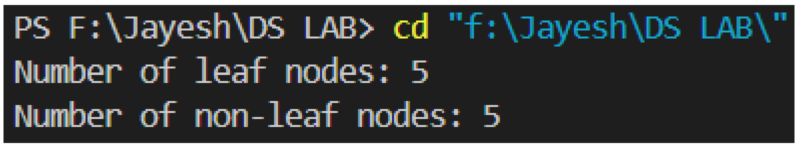


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Max Heap: 75 56 35 25 12 3 11 19 7 9
Min Heap: 3 7 11 19 9 35 12 56 25 75
```

51. Write a program to find the number of leaf and non-leaf nodes of a max heap.

```
#include <stdio.h>

int countLeafNodes(int arr[], int n) {
    int leafCount = 0;
    for (int i = 0; i < n; i++) {
        if (2 * i + 1 >= n && 2 * i + 2 >= n) {
            leafCount++;
        }
    }
    return leafCount;
}

int countNonLeafNodes(int arr[], int n) {
    int nonLeafCount = 0;
    for (int i = 0; i < n; i++) {
        if (2 * i + 1 < n || 2 * i + 2 < n) {
            nonLeafCount++;
        }
    }
    return nonLeafCount;
}

int main() {
    int arr[] = {75, 56, 35, 25, 19, 12, 11, 7, 3, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

    int leafNodes = countLeafNodes(arr, n);
    int nonLeafNodes = countNonLeafNodes(arr, n);

    printf("Number of leaf nodes: %d\n", leafNodes);
    printf("Number of non-leaf nodes: %d\n", nonLeafNodes);

    return 0;
}
```

Output:

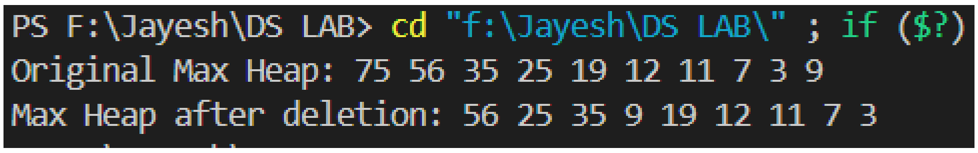


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Number of leaf nodes: 5
Number of non-leaf nodes: 5
```

52. Write a program to delete maximum value from a max heap and then reheapify.

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        heapify(arr, n, largest);
    }
}

void deleteMaxValue(int arr[], int *n) {
    if (*n == 0) {
        printf("Heap is empty.\n");
        return;
    }

    arr[0] = arr[*n - 1];
    (*n)--;

    heapify(arr, *n, 0);
}

int main() {
    int arr[] = {75, 56, 35, 25, 19, 12, 11, 7, 3, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original Max Heap: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    deleteMaxValue(arr, &n);

    printf("Max Heap after deletion: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output:

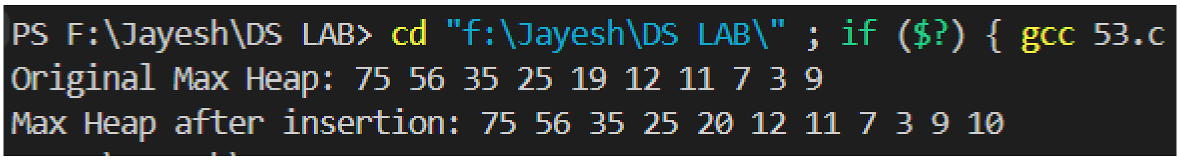


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?)
Original Max Heap: 75 56 35 25 19 12 11 7 3 9
Max Heap after deletion: 56 25 35 9 19 12 11 7 3
```

53. Write a program to insert 20 in max heap.

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        heapify(arr, n, largest);
    }
}

void insertMaxHeap(int arr[], int *n, int val) {
    arr[*n] = val;
    (*n)++;

    for (int i = (*n - 1) / 2; i >= 0; i--) {
        heapify(arr, *n, i);
    }
}

int main() {
    int arr[] = {75, 56, 35, 25, 19, 12, 11, 7, 3, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

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

    insertMaxHeap(arr, &n, 20);

    printf("Max Heap after insertion: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output:



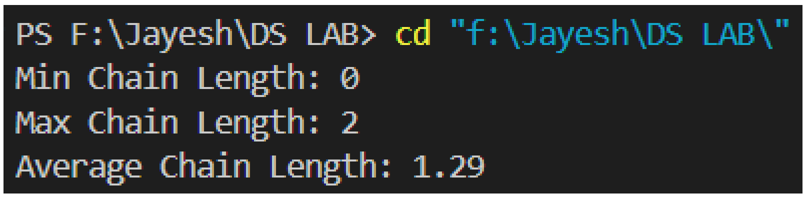
```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 53.c
Original Max Heap: 75 56 35 25 19 12 11 7 3 9
Max Heap after insertion: 75 56 35 25 20 12 11 7 3 9 10
```

54. Insert following keys 5, 28, 19, 15, 20, 33, 12, 17 and 10 in hash table using chaining hashing method and find minimum, maximum and average chain length in hash table.

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

#define TABLE_SIZE 10

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

struct HashTable {
   struct Node* table[TABLE_SIZE];
};

int hash(int key) {
   return key % TABLE_SIZE;
}

void initHashTable(struct HashTable* ht) {
   for (int i = 0; i < TABLE_SIZE; i++) {
      ht->table[i] = NULL;
   }
}

void insert(struct HashTable* ht, int key) {
   int index = hash(key);
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = key;
   newNode->next = ht->table[index];
   ht->table[index] = newNode;
}

void findChainLength(struct HashTable* ht) {
   int min = INT_MAX, max = 0, total = 0, count = 0;

   for (int i = 0; i < TABLE_SIZE; i++) {
      int chainLength = 0;
      struct Node* temp = ht->table[i];
      while (temp != NULL) {
         chainLength++;
         temp = temp->next;
      }
```

```
        if (chainLength > max) {
            max = chainLength;
        }
        if (chainLength < min) {
            min = chainLength;
        }
        total += chainLength;
        if (chainLength > 0) {
            count++;
        }
    }

    printf("Min Chain Length: %d\n", min);
    printf("Max Chain Length: %d\n", max);
    printf("Average Chain Length: %.2f\n", (float)total / count);
}

int main() {
    struct HashTable ht;
    initHashTable(&ht);

    int keys[] = {5, 28, 19, 15, 20, 33, 12, 17, 10};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++) {
        insert(&ht, keys[i]);
    }

    findChainLength(&ht);

    return 0;
}
```

Output:

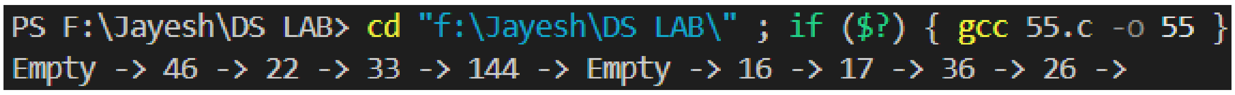


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\"
Min Chain Length: 0
Max Chain Length: 2
Average Chain Length: 1.29
```

55. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using Double hashing, where h(x)= x mod 10, h2(x) = x mod 6 + 1.

```
#include <stdio.h>
#define TABLE_SIZE 10

int hash1(int key) {
    return key % TABLE_SIZE;
}

int hash2(int key) {
    return (key % 6) + 1;
}

void insert(int hashTable[], int key) {
    int index = hash1(key);
    int i = 1;
    while (hashTable[index] != -1) {
        index = (hash1(key) + i * hash2(key)) % TABLE_SIZE;
        i++;
    }
    hashTable[index] = key;
}

void displayHashTable(int hashTable[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != -1) {
            printf("%d -> ", hashTable[i]);
        } else {
            printf("Empty -> ");
        }
    }
    printf("\n");
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }

    int keys[] = {17, 16, 22, 36, 33, 46, 26, 144};
    int n = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < n; i++) {
        insert(hashTable, keys[i]);
    }

    displayHashTable(hashTable);

    return 0;
}
```

Output:

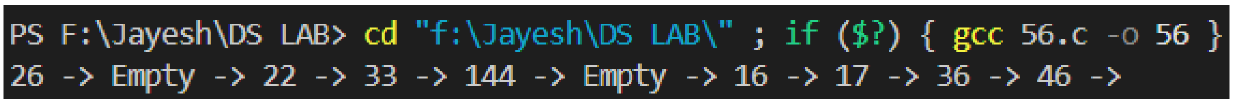


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 55.c -o 55 }
Empty -> 46 -> 22 -> 33 -> 144 -> Empty -> 16 -> 17 -> 36 -> 26 ->
```

56. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using linear probing, where h(x)= x mod 10.

```c
#include <stdio.h>
#define TABLE_SIZE 10
int hash(int key) {
    return key % TABLE_SIZE;
}
void insert(int hashTable[], int key) {
    int index = hash(key);
    while (hashTable[index] != -1) {
        index = (index + 1) % TABLE_SIZE;
    }
    hashTable[index] = key;
}
void displayHashTable(int hashTable[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != -1) {
            printf("%d -> ", hashTable[i]);
        } else {
            printf("Empty -> ");
        }
    }
    printf("\n");
}
int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }
    int keys[] = {17, 16, 22, 36, 33, 46, 26, 144};
    int n = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < n; i++) {
        insert(hashTable, keys[i]);
    }
    displayHashTable(hashTable);
    return 0;
}
```

Output:

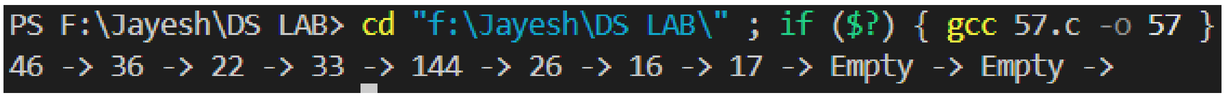


```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 56.c -o 56 }
26 -> Empty -> 22 -> 33 -> 144 -> Empty -> 16 -> 17 -> 36 -> 46 ->
```

57. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using quadratic probing, where h(x)= x mod 10.

```c
#include <stdio.h>
#define TABLE_SIZE 10
int hash(int key) {
    return key % TABLE_SIZE;
}
void insert(int hashTable[], int key) {
    int index = hash(key);
    int i = 1;
    while (hashTable[index] != -1) {
        index = (index + i * i) % TABLE_SIZE;
        i++;
    }
    hashTable[index] = key;
}
void displayHashTable(int hashTable[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != -1) {
            printf("%d -> ", hashTable[i]);
        } else {
            printf("Empty -> ");
        }
    }
    printf("\n");
}
int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }
    int keys[] = {17, 16, 22, 36, 33, 46, 26, 144};
    int n = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < n; i++) {
        insert(hashTable, keys[i]);
    }
    displayHashTable(hashTable);
    return 0;
}
```

Output:



```
PS F:\Jayesh\DS LAB> cd "f:\Jayesh\DS LAB\" ; if ($?) { gcc 57.c -o 57 }
46 -> 36 -> 22 -> 33 -> 144 -> 26 -> 16 -> 17 -> Empty -> Empty ->
```