Tribhuvan University

Kathmandu , Nepal

Bachelor in computer Application (BCA)

Hetauda City College

Lab report of DATA STRUCTURE AND ALGORITHM

Submitted By:

Ashok Giri

BCA,3rd semester

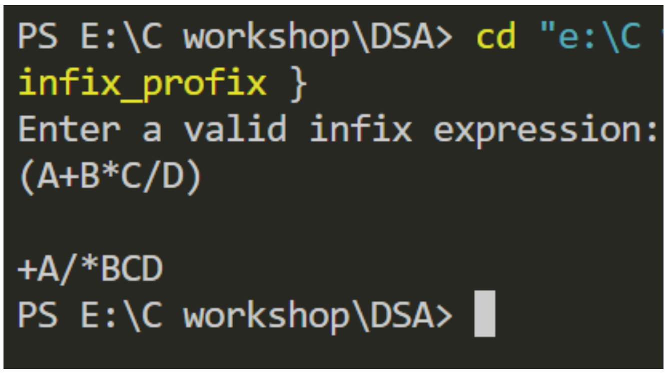
Submitted to:

Mr. Madhu Kumar Singh

1. **Write a program to convert an expression from infix to prefix Expression**

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

#define MAX_SIZE 100

int precedence(char);

int main()
{
   int i, otos = -1, ptos = -1, length;
   char infix[MAX_SIZE], prestack[MAX_SIZE], opstack[MAX_SIZE];

   printf("Enter a valid infix expression:\n");
   fgets(infix, sizeof(infix), stdin);

   length = strlen(infix);

   for (i = length - 1; i >= 0; i--)
   {
      if (isalpha(infix[i]))
      {
         prestack[++ptos] = infix[i];
      }
      else if (infix[i] == ')')
      {
         opstack[++otos] = infix[i];
      }
      else if (infix[i] == '(')
      {
         while (otos != -1 && opstack[otos] != ')')
         {
            prestack[++ptos] = opstack[otos--];
         }
         otos--; // Pop the '(' from the operator stack
      }
      else
      {
         while (otos != -1 && precedence(opstack[otos]) > precedence(infix[i]))
         {
            prestack[++ptos] = opstack[otos--];
         }
         opstack[++otos] = infix[i];
      }
```

```
    }

    while (otos != -1)
    {
        prestack[++ptos] = opstack[otos--];
    }

    // Reverse the resulting prefix expression
    for (i = ptos; i >= 0; i--)
    {
        printf("%c", prestack[i]);
    }

    return 0;
}

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



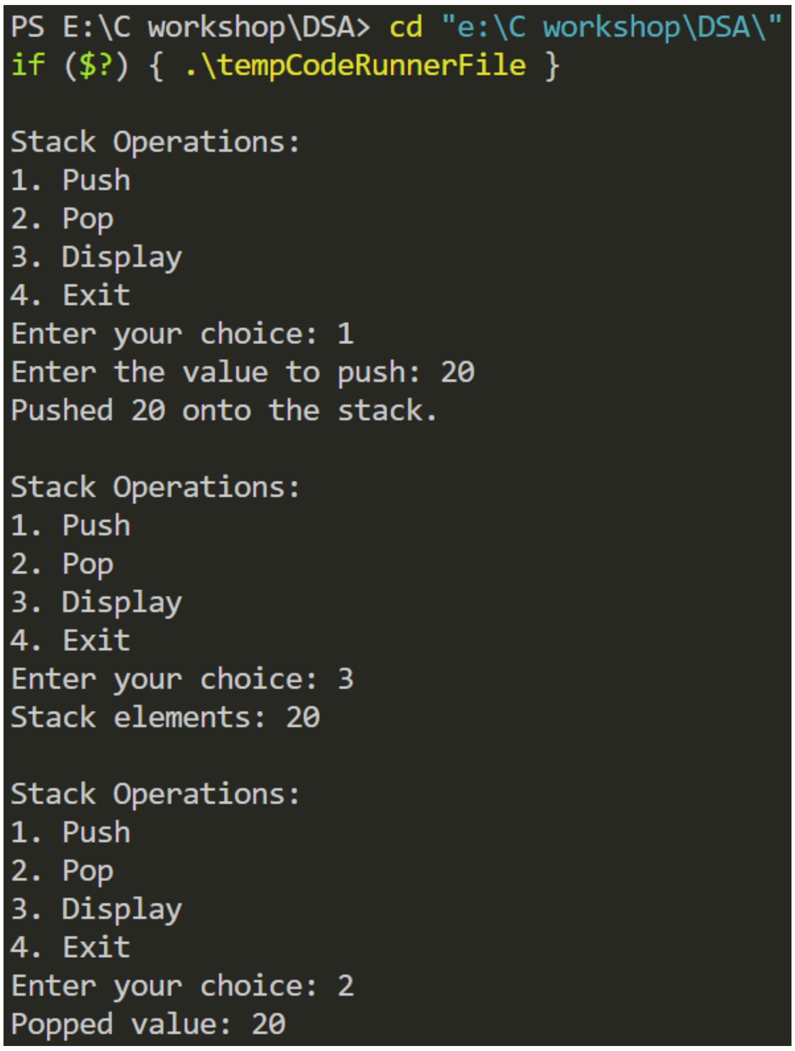
```
PS E:\C workshop\DSA> cd "e:\C
infix_profix }
Enter a valid infix expression:
(A+B*C/D)

+A/*BCD
PS E:\C workshop\DSA>
```

2. **Write a code to implement the stack**

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

```
#include <stdlib.h>

#define MAX_SIZE 10

struct Stack {

    int items[MAX_SIZE];

    int top;

};

void initialize(struct Stack* stack);

void push(struct Stack* stack, int value);

int pop(struct Stack* stack);

void display(struct Stack* stack);



int main() {

    struct Stack stack;

    initialize(&stack);

    int choice, value;

    do {

        printf("\nStack Operations:\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Display\n");

        printf("4. Exit\n");
printf("Enter your choice: ");

scanf("%d", &choice);



switch (choice) {

    case 1:

        printf("Enter the value to push: ");

        scanf("%d", &value);

        push(&stack, value);

        break;

    case 2:

        value = pop(&stack);

        if (value != -1) {

            printf("Popped value: %d\n", value);

        }

        break;

    case 3:

        display(&stack);

        break;

    case 4:

        printf("Exiting the program.\n");

        break;
```

```
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);
    return 0;
}
void initialize(struct Stack* stack) {
    stack->top = -1;
}
void push(struct Stack* stack, int value) {
    if (stack->top == MAX_SIZE - 1) {
        printf("Stack overflow. Cannot push more elements.\n");
    } else {
        stack->top++;
        stack->items[stack->top] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
}
int pop(struct Stack* stack) {
    int value = -1;
    if (stack->top == -1) {
        printf("Stack underflow. Cannot pop from an empty stack.\n");
    } else {
        value = stack->items[stack->top];
        stack->top--;
    }
    return value;
}
void display(struct Stack* stack) {
    if (stack->top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= stack->top; i++) {
            printf("%d ", stack->items[i]);
        }
        printf("\n");
    }
}
```

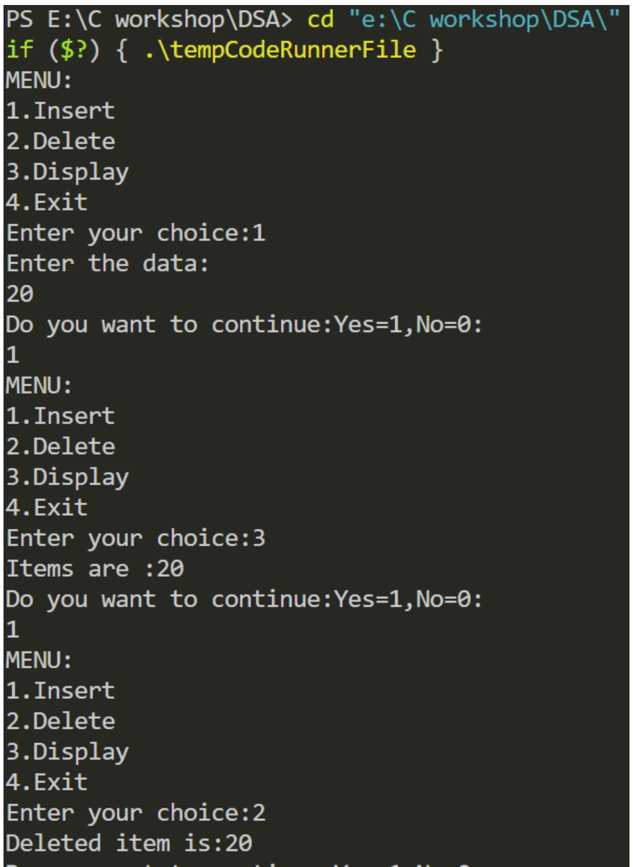


```
PS E:\C workshop\DSA> cd "e:\C workshop\DSA\"
if ($?) { .\tempCodeRunnerFile }

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 20
Pushed 20 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 20

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 20
```

3. **Write a program to implementation of circular queue**

```
#include <stdio.h>

#include <stdlib.h>

#define size 5

int insert();

int delete();

int display();

int further();

int front = -1, rear = -1, item, flag = 1;
int cqueue[5];

int main()

{

    int ch;

    while (flag = 1)

    {

        printf("MENU:\n");

        printf("1.Insert\n");

        printf("2.Delete\n");

        printf("3.Display\n");

        printf("4.Exit\n");

        printf("Enter your choice:");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            insert();

            further();

            break;

        case 2:

            delete ();
            further();
            break;
        case 3:
            display();
            further();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("Invalid Choice.try again");
        }
    }
    return 0;
}
int insert()
{
    if (front == (rear + 1) % size)
    {
        printf("Queue is full\n");
    }
    else
    {
        rear = (rear + 1) % size;
        printf("Enter the data:\n");
        scanf("%d", &item);
        cqueue[rear] = item;
    }
}
int delete()
{
    if (rear == front)
    {
        printf("Queue is empty\n");
    }
    else
    {
        front = (front + 1) % size;
        item = cqueue[front];
        printf("Deleted item is:%d\n", item);
    }
}
```

```
int display()
{
    int i;
    printf("Items are :");
    for (i = front + 1; i <= rear; i++)
    {
        printf("%d\t", cqueue[i]);
    }
    printf("\n");
}
int further()
{
    printf("Do you want to continue:Yes=1,No=0:\n");
    scanf("%d", &flag);
```

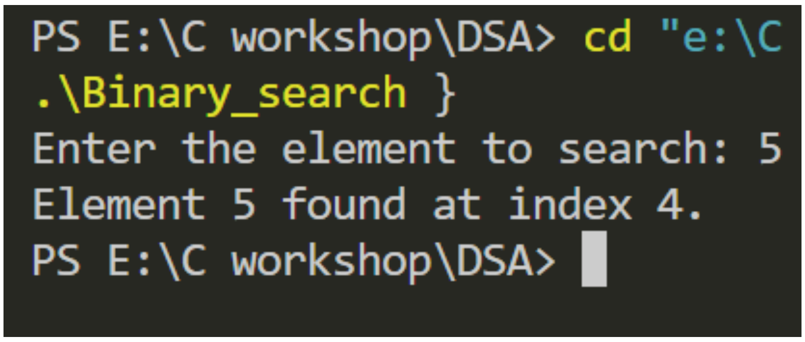


```
PS E:\C workshop\DSA> cd "e:\C workshop\DSA\"
if ($?) { .\tempCodeRunnerFile }
MENU:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the data:
20
Do you want to continue:Yes=1,No=0:
1
MENU:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
Items are :20
Do you want to continue:Yes=1,No=0:
1
MENU:
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
Deleted item is:20
```

```
}}
```

4. **Write a program for Binary Search**

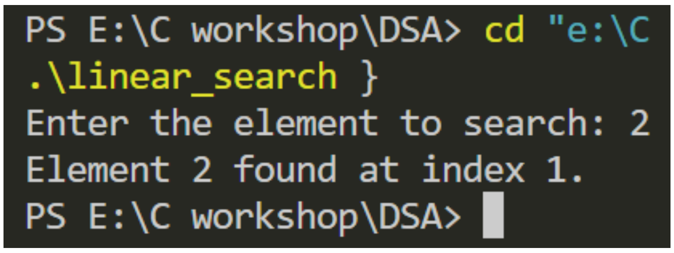
```c
#include <stdio.h>
int binarySearch(int arr[], int low, int high, int key);

int main() {
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   int n = sizeof(arr) / sizeof(arr[0]);

   int key;
   printf("Enter the element to search: ");
   scanf("%d", &key);

   int index = binarySearch(arr, 0, n - 1, key);

   if (index != -1) {
      printf("Element %d found at index %d.\n", key, index);
   } else {
      printf("Element %d not found in the array.\n", key);
   }

   return 0;
}
int binarySearch(int arr[], int low, int high, int key) {
   while (low <= high) {
      int mid = low + (high - low) / 2;

      if (arr[mid] == key) {
         return mid;
      }

      if (arr[mid] < key) {
         low = mid + 1;
      } else {
         high = mid - 1;
      }
   }
   return -1;
}
```



```
PS E:\C workshop\DSA> cd "e:\C
.\Binary_search }
Enter the element to search: 5
Element 5 found at index 4.
PS E:\C workshop\DSA> 
```

5. **Write a program to implementation of linear search**

```
#include <stdio.h>
int linearSearch(int arr[], int size, int key);

int main() {
   int arr[] = {10, 2, 45, 32, 17, 8, 22, 14};
   int size = sizeof(arr) / sizeof(arr[0]);

   int key;
   printf("Enter the element to search: ");
   scanf("%d", &key);

   int index = linearSearch(arr, size, key);

   if (index != -1) {
      printf("Element %d found at index %d.\n", key, index);
   } else {
      printf("Element %d not found in the array.\n", key);
   }

   return 0;
}
int linearSearch(int arr[], int size, int key) {
   for (int i = 0; i < size; i++) {
      if (arr[i] == key) {
         return i;
      }
   }
   return -1;
}
```

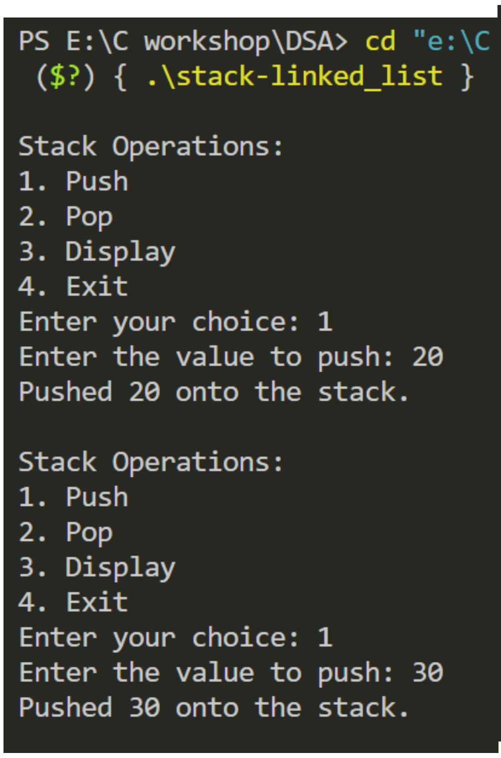


```
PS E:\C workshop\DSA> cd "e:\C
.\linear_search }
Enter the element to search: 2
Element 2 found at index 1.
PS E:\C workshop\DSA> 
```

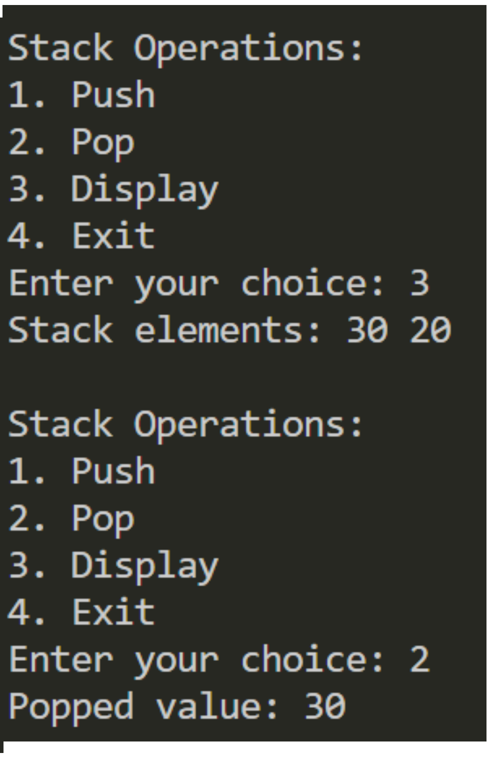
6. **Write a program to implement stack using linked list**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Stack {
   struct Node* top;
};

void initialize(struct Stack* stack);
void push(struct Stack* stack, int value);
int pop(struct Stack* stack);
void display(struct Stack* stack);

int main() {
   struct Stack stack;
   initialize(&stack);

   int choice, value;

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

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

```
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);

    return 0;
}
void initialize(struct Stack* stack) {
    stack->top = NULL;
}
void push(struct Stack* stack, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed. Cannot push the value onto the stack.\n");
        return;
    }

    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;

    printf("Pushed %d onto the stack.\n", value);
}
int pop(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Stack underflow. Cannot pop from an empty stack.\n");
        return -1;
    }

    struct Node* poppedNode = stack->top;
    int value = poppedNode->data;

    stack->top = poppedNode->next;
    free(poppedNode);

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


}
```



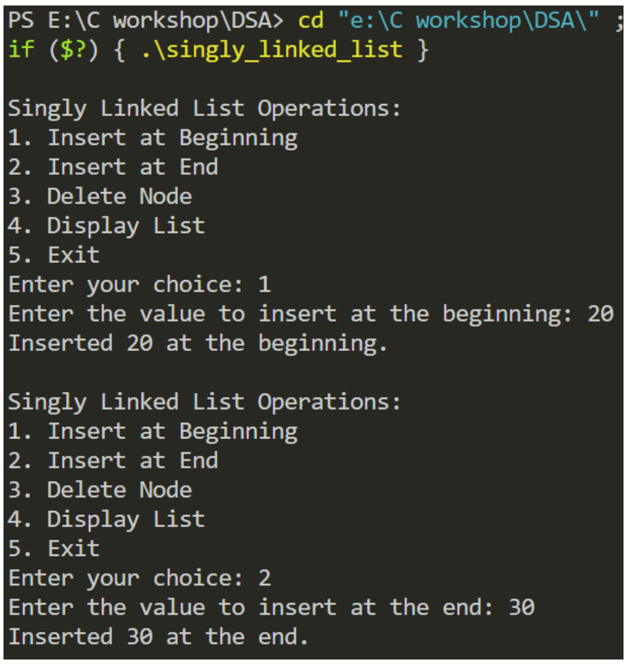
```
PS E:\C workshop\DSA> cd "e:\C
 ($?) { .\stack-linked_list }

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 20
Pushed 20 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 30
Pushed 30 onto the stack.
```



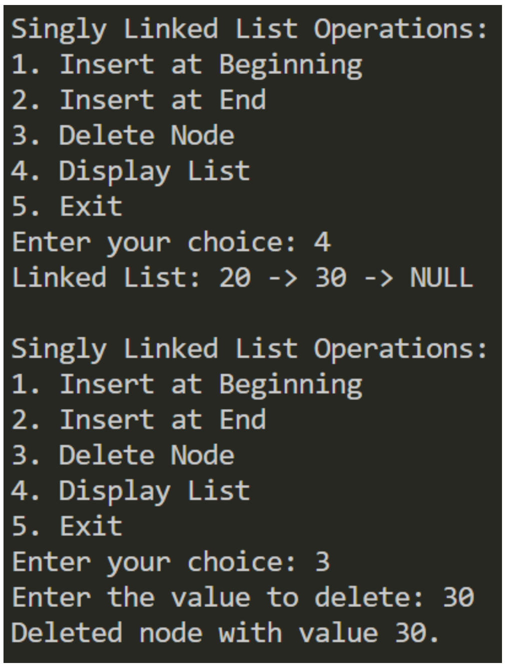
```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 30 20

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped value: 30
```

7. Write a program to implement singly linked list

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int value);
void insertAtBeginning(struct Node** head, int value);
void insertAtEnd(struct Node** head, int value);
void deleteNode(struct Node** head, int value);
void displayList(struct Node* head);

int main() {
    struct Node* head = NULL;
    int choice, value;

    do {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Delete Node\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtBeginning(&head, value);
                break;
            case 2:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;
            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &value);
                deleteNode(&head, value);
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 5);

    return 0;
}
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
    printf("Inserted %d at the beginning.\n", value);
}
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
    printf("Inserted %d at the end.\n", value);
}
void deleteNode(struct Node** head, int value) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

    struct Node* current = *head;
    struct Node* previous = NULL;
    if (current->data == value) {
        *head = current->next;
        free(current);
        printf("Deleted node with value %d.\n", value);
        return;
    }
    while (current != NULL && current->data != value) {
        previous = current;
        current = current->next;
    }
    if (current != NULL) {
        previous->next = current->next;
        free(current);
        printf("Deleted node with value %d.\n", value);
    } else {
        printf("Node with value %d not found.\n", value);
    }
}
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
    } else {
        printf("Linked List: ");
        struct Node* current = head;
        while (current != NULL) {
            printf("%d -> ", current->data);
            current = current->next;
        }
        printf("NULL\n");
    }
}
```



```
PS E:\C workshop\DSA> cd "e:\C workshop\DSA\" ;
if ($?) { .\singly_linked_list }

Singly Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display List
5. Exit
Enter your choice: 1
Enter the value to insert at the beginning: 20
Inserted 20 at the beginning.

Singly Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display List
5. Exit
Enter your choice: 2
Enter the value to insert at the end: 30
Inserted 30 at the end.
```

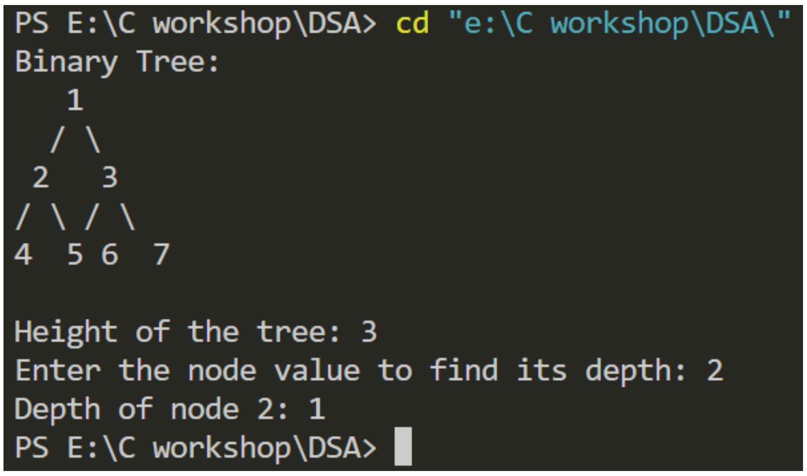


```
Singly Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display List
5. Exit
Enter your choice: 4
Linked List: 20 -> 30 -> NULL

Singly Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Display List
5. Exit
Enter your choice: 3
Enter the value to delete: 30
Deleted node with value 30.
```

8. Write a program to find the height and depth of given tree

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int value);
int findHeight(struct TreeNode* root);
int findDepth(struct TreeNode* root, int key, int depth);
int main() {
   struct TreeNode* root = createNode(1);
   root->left = createNode(2);
   root->right = createNode(3);
   root->left->left = createNode(4);
   root->left->right = createNode(5);
   root->right->left = createNode(6);
   root->right->right = createNode(7);
   int key, height, depth;
   printf("Binary Tree:\n");
   printf("   1\n");
   printf("  / \\\n");
   printf(" 2   3\n");
   printf("/ \\ / \\\n");
   printf("4  5 6  7\n");
   height = findHeight(root);
   printf("\nHeight of the tree: %d\n", height);
   printf("Enter the node value to find its depth: ");
   scanf("%d", &key);
   depth = findDepth(root, key, 0);

   if (depth != -1) {
      printf("Depth of node %d: %d\n", key, depth);
   } else {
      printf("Node %d not found in the tree.\n", key);
   }

   return 0;
}
struct TreeNode* createNode(int value) {
   struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
   if (newNode == NULL) {
      printf("Memory allocation failed.\n");
      exit(EXIT_FAILURE);
   }
   newNode->data = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}
int findHeight(struct TreeNode* root) {
   if (root == NULL) {
      return 0;
   } else {
      int leftHeight = findHeight(root->left);
      int rightHeight = findHeight(root->right);
```

```
        return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
    }
}
int findDepth(struct TreeNode* root, int key, int depth) {
    if (root == NULL) {
        return -1;
    }

    if (root->data == key) {
        return depth;
    }

    int leftDepth = findDepth(root->left, key, depth + 1);
    if (leftDepth != -1) {
        return leftDepth;
    }

    int rightDepth = findDepth(root->right, key, depth + 1);
    return rightDepth;
}
```



```
PS E:\C workshop\DSA> cd "e:\C workshop\DSA\"
Binary Tree:
   1
  / \
 2   3
/ \ / \
4  5 6  7

Height of the tree: 3
Enter the node value to find its depth: 2
Depth of node 2: 1
PS E:\C workshop\DSA>
```