DSA Lab Assessment 1

24BBS0178

Palash Shivnani

Question 1:
Algorithm for Stack Operations
Initialization:
• Initialize top = -1 (indicating that the stack is empty).
• Input the size s of the stack.
• Create an array arr[] of size s to hold the stack elements.

Push Operation (push() function):
• **Check if the stack is full**:
• If top == s - 1, the stack is full. Print "Stack is full!" and return.
• **If the stack is not full**:
• Input the data to be pushed.
• Increment top by 1 (top++).
• Store the input data in arr[top].
• Print "Data pushed!".

Pop Operation (pop() function):
• **Check if the stack is empty**:
• If top == -1, the stack is empty. Print "Stack is empty!" and return.
• **If the stack is not empty**:
• Print the element at arr[top].
• Set arr[top] = 0 (optional but clears the value in the array).
• Decrement top by 1 (top--).
• Print "Element popped!".

Display Operation (display() function):
• **Check if the stack is empty**:
• If top == -1, the stack is empty. Print "Stack is empty!" and return.
• **If the stack is not empty**:
• Start from arr[0] to arr[top] and print each element.
• For each element, display as "Element x: data", where x is the index (1-based) and data is the stack element.

Main Loop:

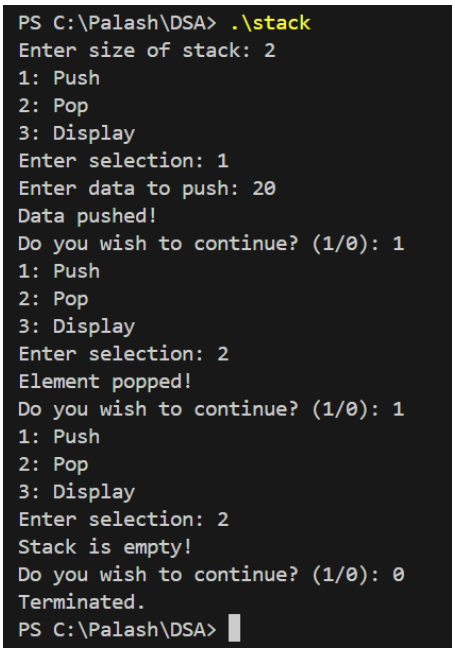
• **Input the menu selection (1, 2, or 3)**.
• If the selection is 1, call push() to push an element onto the stack.
• If the selection is 2, call pop() to pop an element from the stack.
• If the selection is 3, call display() to print the stack.
• If the selection is invalid, print "Incorrect input!" and terminate the program.
• Ask the user if they wish to continue (input 1 for yes, 0 for no).
• If the user chooses 0, terminate the program.

Termination:
• Once the user decides to exit (selects 0), print "Terminated." and exit the program.
• If an invalid menu input is provided, print "Incorrect input!" and terminate the program.

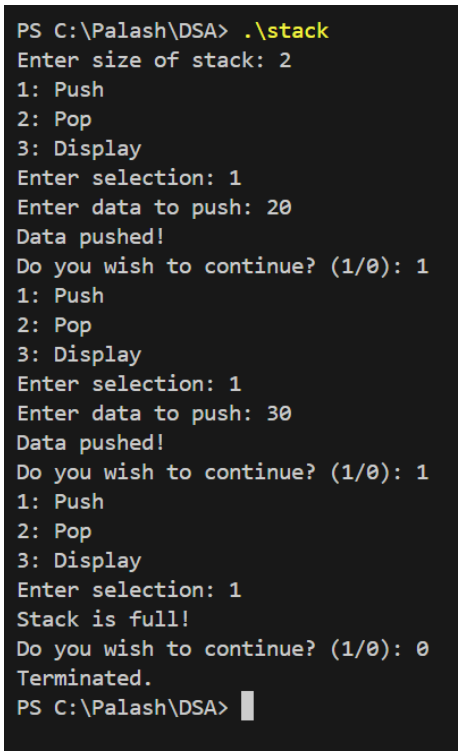
```c
#include <stdio.h>
int top=-1;
int s;
int cont=1;
void push(int arr[]){
    if (top==s-1){
        printf("Stack is full!\n");
    }
    else {
        int data;
        printf("Enter data to push: ");
        scanf("%d", &data);
        ++top;
        arr[top]=data;
        printf("Data pushed!\n");
    }
}
void pop(int arr[]){
    if (top==-1){
        printf("Stack is empty!\n");
    }
    else {
        arr[top]=0;
        --top;
        printf("Element popped!\n");
    }
}
void display(int arr[]){
    if (top==-1){
        printf("Stack is empty!\n");
    }
    else {
        for (int i=0; i<=top; i++){
            printf("Element %d: %d\n", i+1, arr[i]);
        }
    }
}
int main(){
    printf("Enter size of stack: ");
    scanf("%d", &s);
    int a[s];
    while (cont==1){
        int m=0;
        printf("1: Push \n2: Pop \n3: Display \nEnter selection: ");
        scanf("%d", &m);
        if (m==1){
            push(a);
        }
        else if (m==2){
            pop(a);
        }
        else if (m==3){
            display(a);
        }
        else {
            printf("Incorrect input!\n");
            cont=0;
            break;
        }
```

```
        printf("Do you wish to continue? (1/0): ");
        scanf("%d", &cont);
    }
    if (cont==0){
        printf("Terminated.\n");
    }
    else {
        printf("Incorrect input!\nTerminated.");
    }
}
```

Underflow



Overflow



General

```
PS C:\Palash\DSA> .\stack
Enter size of stack: 2
1: Push
2: Pop
3: Display
Enter selection: 1
Enter data to push: 20
Data pushed!
Do you wish to continue? (1/0): 1
1: Push
2: Pop
3: Display
Enter selection: 3
Element 1: 20
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

Question 2:

Algorithm for Queue Operations:

Initialization:

• Initialize top = -1 (indicating the queue is empty).
• Input the size (s) of the queue.
• Create an array arr[] of size s to hold the queue elements.

Enqueue Operation (enq() function):

• **Check if the queue is full**:
• If top == s - 1, the queue is full. Print "Queue is full!" and return.
• **If the queue is not full**:
• Input the data to be enqueued.
• Increase top by 1 (top++).
• Store the input data in arr[top].
• Print "Data enqueued!".

Dequeue Operation (deq() function):

• **Check if the queue is empty**:
• If top == -1, the queue is empty. Print "Queue is empty!" and return.
• **If the queue is not empty**:
• Print the element at arr[0].
• Shift each element in arr[i + 1] to arr[i] for i from 0 to top - 1.
• Decrease top by 1 (top--).
• Print "Element dequeued!".

Display Operation (display() function):

• **Check if the queue is empty**:
• If top == -1, the queue is empty. Print "Queue is empty!" and return.
• **If the queue is not empty**:
• Print each element in the queue from arr[0] to arr[top].
• For each element, display as "Element x: data", where x is the index (1-based) and data is the queue element.

Main Loop:

• **Input the menu selection (1, 2, or 3)**.
• If the selection is 1, call enq() to enqueue.
• If the selection is 2, call deq() to dequeue.
• If the selection is 3, call display() to print the queue.
• If the selection is invalid, print "Incorrect input!" and terminate the program.

• Ask the user if they wish to continue (input 1 for yes, 0 for no).
• Repeat the loop if the user chooses to continue.

Termination:
• Once the user decides to exit (selects 0), print "Terminated." and exit the program.

```c
#include <stdio.h>
int top=-1;
int s;
int cont=1;
void enq(int arr[]){
    if (top==s-1){
        printf("Queue is full!\n");
    }
    else {
        int data;
        printf("Enter data to enqueue: ");
        scanf("%d", &data);
        ++top;
        arr[top]=data;
        printf("Data enqueued!\n");
    }
}
void deq(int arr[]){
    if (top==-1){
        printf("Queue is empty!\n");
    }
    else { //shifting each element forward
        for (int i=0; i<top; i++){
            arr[i]=arr[i+1];
        }
        --top;
        printf("Element dequeued!\n");
    }
}
void display(int arr[]){
    if (top==-1){
        printf("Queue is empty!\n");
    }
    else {
        for (int i=0; i<=top; i++){
            printf("Element %d: %d\n", i+1, arr[i]);
        }
    }
}
int main(){
    printf("Enter size of queue: ");
    scanf("%d", &s);
    int a[s];
    while (cont==1){
        int m=0;
        printf("1: Enqueue \n2: Dequeue \n3: Display \nEnter selection: ");
        scanf("%d", &m);
        if (m==1){
            enq(a);
        }
        else if (m==2){
```

```
            deq(a);
        }
        else if (m==3){
            display(a);
        }
        else {
            printf("Incorrect input!\n");
            cont=0;
            break;
        }
        printf("Do you wish to continue? (1/0): ");
        scanf("%d", &cont);
    }
    if (cont==0){
        printf("Terminated.\n");
    }
    else {
        printf("Incorrect input!\nTerminated.");
    }
}
```

```
PS C:\Palash\DSA> gcc -o queue queue.c
PS C:\Palash\DSA> .\queue
Enter size of queue: 3
1: Enqueue
2: Dequeue
3: Display
Enter selection: 1
Enter data to enqueue: 20
Data enqueued!
Do you wish to continue? (1/0): 1
1: Enqueue
2: Dequeue
3: Display
Enter selection: 2
Element dequeued!
Do you wish to continue? (1/0): 1
1: Enqueue
2: Dequeue
3: Display
Enter selection: 2
Queue is empty!
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

```
PS C:\Palash\DSA> .\queue
Enter size of queue: 2
1: Enqueue
2: Dequeue
3: Display
Enter selection: 1
Enter data to enqueue: 10
Data enqueued!
Do you wish to continue? (1/0): 1
1: Enqueue
2: Dequeue
3: Display
Enter selection: 1
Enter data to enqueue: 2
Data enqueued!
Do you wish to continue? (1/0): 1
1: Enqueue
2: Dequeue
3: Display
Enter selection: 1
Queue is full!
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

```
PS C:\Palash\DSA> .\queue
Enter size of queue: 2
1: Enqueue
2: Dequeue
3: Display
Enter selection: 1
Enter data to enqueue: 10
Data enqueued!
Do you wish to continue? (1/0): 1
1: Enqueue
2: Dequeue
3: Display
Enter selection: 3
Element 1: 10
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

Question 3:
Algorithm for Circular Queue

Initialization:

• Initialize front = -1 and rear = -1 (indicating that the queue is empty).
• Input the size (s) of the queue.
• Create an array arr[] of size s to hold the queue elements.

Enqueue Operation (enq() function):

• **Check if the queue is full**:
• If (rear + 1) % s == front, the queue is full. Print "Queue is full!" and return.
• **If the queue is not full**:
• Input the data to be enqueued.
• If the queue is empty (front == -1), set front = 0.
• Increment rear using the circular formula: rear = (rear + 1) % s.
• Store the input data in arr[rear].
• Print "Data enqueued!".

Dequeue Operation (deq() function):

• **Check if the queue is empty**:
• If front == -1, the queue is empty. Print "Queue is empty!" and return.
• **If the queue is not empty**:
• Print the element at front (arr[front]).
• If front == rear, the queue will be empty after this operation. Set front = -1 and rear = -1.
• Otherwise, increment front using the circular formula: front = (front + 1) % s.
• Print "Element dequeued!".

Display Operation (display() function):

• **Check if the queue is empty**:
• If front == -1, the queue is empty. Print "Queue is empty!" and return.
• **If the queue is not empty**:
• Start at the front and print elements one by one.
• Use the circular formula to move through the queue: (i + 1) % s, until i == rear.
• Print all the elements from front to rear.

Main Loop:

• **Input the menu selection (1, 2, or 3)**.
• If the selection is 1, call enq() to enqueue.
• If the selection is 2, call deq() to dequeue.
• If the selection is 3, call display() to print the queue.
• If the selection is invalid, print "Incorrect input!" and terminate the program.

• Ask the user if they wish to continue (input 1 for yes, 0 for no).
• Repeat the loop if the user chooses to continue.

Termination:
• Once the user decides to exit (selects 0), print "Terminated." and exit the program.

```c
#include <stdio.h>
int front=-1;
int rear=-1;
int s;
int cont=1;
void enq(int arr[]) {
    if ((rear+1) % s==front) {
        printf("Queue is full!\n");
    }
    else {
        int data;
        printf("Enter data to enqueue: ");
        scanf("%d", &data);
        if (front==-1) {
            front=0;
        }
        rear=(rear+1) % s;
        arr[rear]=data;
        printf("Data enqueued!\n");
    }
}
void deq(int arr[]) {
    if (front==-1) {
        printf("Queue is empty!\n");
    }
    else {
        printf("Element dequeued: %d\n", arr[front]);
        if (front==rear) {
            front=rear=-1;
        }
        else {
            front=(front+1) % s;
        }
    }
}
void display(int arr[]) {
    if (front==-1) {
        printf("Queue is empty!\n");
    }
    else {
        printf("Queue elements are: \n");
        int i=front;
        while (1) {
            printf("%d ", arr[i]);
            if (i==rear) {
                break;
            }
            i=(i+1) % s;
        }
        printf("\n");
    }
```

```
}
int main() {
    printf("Enter size of queue: ");
    scanf("%d", &s);
    int a[s];
    while (cont==1) {
        int m=0;
        printf("1: Enqueue \n2: Dequeue \n3: Display \nEnter selection: ");
        scanf("%d", &m);
        if (m==1) {
            enq(a);
        }
        else if (m==2) {
            deq(a);
        }
        else if (m==3) {
            display(a);
        }
        else {
            printf("Incorrect input!\n");
            cont=0;
            break;
        }
        printf("Do you wish to continue? (1/0): ");
        scanf("%d", &cont);
    }
    if (cont==0) {
        printf("Terminated.\n");
    }
    else {
        printf("Incorrect input!\nTerminated.\n");
    }
}
```

Question 4:

Algorithm for Linked List Operations

Initialization:

• Initialize head = NULL (indicating that the linked list is empty).

• The node structure contains two fields:

- data: Stores the data of the node.
- next: Points to the next node in the list.

Insert Operation (insert() function):

• **Input the user's choice**:

- 1: Insert at the beginning.
- 2: Insert at the end.
- 3: Insert at a given position.

• **If the user selects 1 (Insert at the beginning)**:

- Input the data to insert.
- Create a new node, allocate memory for it, and assign the input data.
- Set the next pointer of the new node to the current head.
- Update head to point to the new node.
- Print the inserted data and position.

• **If the user selects 2 (Insert at the end)**:

- Input the data to insert.
- Create a new node, allocate memory for it, and assign the input data.
- If the list is empty (head == NULL), set head to the new node.
- If the list is not empty, traverse the list to find the last node.
- Set the next pointer of the last node to the new node.
- Print the inserted data and position.

• **If the user selects 3 (Insert at a given position)**:

- Input the position and the data to insert.
- If the position is invalid (less than 1), print an error message.
- If the position is valid, traverse the list to the desired position.
- Insert the new node at the specified position by adjusting the next pointers.
- Print the inserted data and position.

Delete Operation (delete() function):

• **Input the user's choice**:

- 1: Delete from the beginning.
- 2: Delete from the end.
- 3: Delete from a given position.

• **If the user selects 1 (Delete from the beginning)**:

- Check if the list is empty (head == NULL), print an error message if true.
- If the list is not empty, set head to the next node.

- Free the memory of the deleted node.
- Print the deleted data and position.

• **If the user selects 2 (Delete from the end)**:
- Check if the list is empty (head == NULL), print an error message if true.
- If the list contains only one node, delete the node and set head to NULL.
- If the list contains multiple nodes, traverse the list to find the second last node.
- Set the next pointer of the second last node to NULL.
- Free the memory of the last node.
- Print the deleted data and position.

• **If the user selects 3 (Delete from a given position)**:
- Input the position to delete.
- If the position is invalid (less than 1 or greater than the list length), print an error message.
- If the position is valid, traverse the list to the desired position.
- Delete the node at the specified position by adjusting the next pointers.
- Free the memory of the deleted node.
- Print the deleted data and position.

Search Operation (search() function):

• Input the data to search in the list.
• Traverse the list from head to find a node with the matching data.
• If the data is found, print the data and its position.
• If the data is not found, print an error message.

Main Loop:

• **Input the menu selection (1, 2, or 3)**.
• If the selection is 1, call insert() to insert a node.
• If the selection is 2, call delete() to delete a node.
• If the selection is 3, call search() to search for a node.
• If the selection is invalid, print "Incorrect input!" and terminate the program.
• Ask the user if they wish to continue (input 1 for yes, 0 for no).
• Repeat the loop if the user chooses to continue.

Termination:

• Once the user decides to exit (selects 0), print "Terminated." and exit the program.
• If an invalid menu input is provided, print "Incorrect input!" and terminate the program.

```
#include <stdio.h>
#include <stdlib.h>
int cont=1;
typedef struct node{
    int data;
    node *next;
}node;
node *head=NULL;
int insert(){
    int m, data;
    struct node* newNode = (struct Node*)malloc(sizeof(node));
    printf("1=Beggining \n2=End \n3=Given Position \nWhere do you wish to insert
node: ");
    scanf("%d", &m);
    if (m==1){
        printf("Enter data: ");
        scanf("%d", &data);
        (*newNode).data=data;
        (*newNode).next=head;
        head=newNode;
    }
    else if (m==2){
        printf("Enter data: ");
        scanf("%d", &data);
        node *newNode=(node*)malloc(sizeof(node));
        (*newNode).data=data;
        (*newNode).next=NULL;
        if (head==NULL) {
            head=newNode;
        }
        else {
            struct node *temp=head;
        while ((*temp).next!=NULL) {
            temp=(*temp).next;
        }
            (*temp).next=newNode;
        }
    }
    else if (m==3){
        int pos;
        printf("Enter position to insert data: ");
        scanf("%d", &pos);
        printf("Enter data: ");
        scanf("%d", &data);
        if (pos<1) {
            printf("Invalid position.\n");
            return;
        }
        node* newNode=(node*)malloc(sizeof(node));
        (*newNode).data=data;
        if (pos==1) {
```

```
            (*newNode).next=head;
            head=newNode;
            printf("%d inserted at position %d.\n", data, pos);
            return;
        }
        node* temp=head;
        for (int i=1; i<pos-1 && temp!=NULL; i++) {
            temp = (*temp).next;
        }

        if (temp==NULL) {
            printf("Position out of range.\n");
            free(newNode);
            return;
        }
        (*newNode).next=(*temp).next;
        (*temp).next=newNode;
        printf("%d inserted at position %d.\n", data, pos);
    }
}
int delete(){
    int m, data;
    printf("1=Beggining \n2=End \n3=Given Position \nWhere do you wish to delete
node: ");
    scanf("%d", &m);
    if (m==1){
        if(head==NULL) {
        printf("List is empty.\n");
        return;
        }
        node *temp=head;
        head=(*head).next;
        printf("%d deleted from the beginning.\n", (*temp).data);
        free(temp);
    }
    else if (m==2){
        if(head==NULL) {
            printf("List is empty.\n");
            return;
        }
        if((*head).next==NULL) {
            printf("%d deleted from the end.\n", (*head).data);
            free(head);
            head=NULL;
            return;
        }
        node *temp=head;
        while((*(*temp).next).next!=NULL) {
        temp=(*temp).next;
        }
        printf("%d deleted from the end.\n", (*(*temp).next).data);
```

```
        free((*temp).next);
        (*temp).next=NULL;
    }
    else if (m==3){
        int pos;
        printf("Enter position to delete data: ");
        scanf("%d", &pos);
        if(pos<1||head==NULL) {
        printf("Invalid position or list is empty.\n");
        return;
        }
        if(pos==1) {
            node *temp=head;
            head=(*head).next;
            printf("%d deleted from position %d.\n", (*temp).data, pos);
            free(temp);
            return;
        }
        node *temp=head;
        for(int i=1;i<pos-1&&temp!=NULL;i++) {
            temp=(*temp).next;
        }
        if(temp==NULL||(*temp).next==NULL) {
            printf("Position out of range.\n");
            return;
        }
        node *delNode=(*temp).next;
        (*temp).next=(*delNode).next;
        printf("%d deleted from position %d.\n", (*delNode).data, pos);
        free(delNode);
    }
}
int search(){
    int data;
    printf("Enter data to search: ");
    scanf("%d", &data);
    node *temp=head;
    int pos=1;
    while(temp!=NULL) {
        if((*temp).data==data) {
            printf("%d found at position %d.\n", data, pos);
            return;
        }
        temp=(*temp).next;
        pos++;
    }
    printf("%d not found in the list.\n", data);
}
int main(){
    while (cont==1){
        int m=0;
```

```
        printf("1: Insert \n2: Delete \n3: Search \nEnter selection: ");
        scanf("%d", &m);
        if (m==1){
            insert();
        }
        else if (m==2){
            delete();
        }
        else if (m==3){
            search();
        }
        else {
            printf("Incorrect input!\n");
            cont=0;
            break;
        }
        printf("Do you wish to continue? (1/0): ");
        scanf("%d", &cont);
    }
    if (cont==0){
        printf("Terminated.\n");
    }
    else {
        printf("Incorrect input!\nTerminated.");
    }
    return 0;
}
```

```
PS C:\Palash\DSA> ./sll
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beggining
2=End
3=Given Position
Where do you wish to insert node: 1
Enter data: 20
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 2
1=Beggining
2=End
3=Given Position
Where do you wish to delete node: 1
20 deleted from the beginning.
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 2
1=Beggining
2=End
3=Given Position
Where do you wish to delete node: 1
List is empty.
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

```
PS C:\Palash\DSA> ./sll
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beggining
2=End
3=Given Position
Where do you wish to insert node: 2
Enter data: 30
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beggining
2=End
3=Given Position
Where do you wish to insert node: 1
Enter data: 40
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 3
Enter data to search: 40
40 found at position 1.
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

Question 5:

Algorithm for doubly linked list operations:

Initialization:

- **Input:** None
- **Action:** Initialize head = NULL (indicating the list is empty).

Insert Operation (insert()):

- **Input:** Data and position.
- **Action:** Insert a node based on user input:
    - If m == 1 (Insert at beginning):
        - Create a new node.
        - Set its next pointer to the current head and its prev to NULL.
        - If the list is not empty, set the prev pointer of the old head to the new node.
        - Update the head to the new node.
    - If m == 2 (Insert at end):
        - Create a new node.
        - If the list is empty, set head to the new node.
        - Otherwise, traverse the list until the last node and insert the new node at the end.
    - If m == 3 (Insert at a given position):
        - Input the position.
        - Traverse the list to the specified position and insert the new node there, adjusting the prev and next pointers as needed.

Delete Operation (delete()):

- **Input:** Position of the node to delete.
- **Action:** Delete a node based on user input:
    - If m == 1 (Delete from beginning):
        - If the list is empty, print an error message.
        - Otherwise, update head to the next node and free the old head.
    - If m == 2 (Delete from end):
        - If the list is empty or has only one node, handle accordingly.
        - Otherwise, traverse to the last node and delete it.
    - If m == 3 (Delete from a given position):
        - Input the position.
        - Traverse the list to the specified position and remove the node, updating the surrounding pointers.

Search Operation (search()):

- **Input:** Data to search for.
- **Action:** Search through the list for the specified data:
    - Traverse the list from the beginning, comparing the data field of each node with the target data.

- If found, print the position of the node.
- If not found, print an appropriate message.

Main Program Flow:

- **Loop** until the user chooses to terminate.
    1. **Display the menu options:**
        - 1: Insert
        - 2: Delete
        - 3: Search
    2. **Input:** The user's menu selection (m).
    3. **If** the user selects 1 (Insert):
        - Call insert() to add a node.
    4. **If** the user selects 2 (Delete):
        - Call delete() to remove a node.
    5. **If** the user selects 3 (Search):
        - Call search() to search for a node.
    6. **If** the user selects an invalid option:
        - Print "Incorrect input!" and terminate the program.
    7. **Prompt:** Ask the user if they wish to continue (input 1 for yes, 0 for no).
    8. **Repeat:** Continue the loop if the user chooses to continue.

Termination:

- **Condition:** If the user selects 0 to exit or an invalid input is entered:
    - Print "Terminated."

```
#include <stdio.h>
#include <stdlib.h>
int cont=1;
typedef struct node{
    int data;
    struct node* next;
    struct node* prev;
}node;
node* head=NULL;
int insert(){
    int m,data;
    struct node* newNode=(struct node*)malloc(sizeof(node));
    (*newNode).prev=NULL;
    (*newNode).next=NULL;
    printf("1=Beginning \n2=End \n3=Given Position \nWhere do you wish to insert
node: ");
    scanf("%d",&m);
    if(m==1){
        printf("Enter data: ");
        scanf("%d",&data);
        (*newNode).data=data;
        (*newNode).next=head;
        if(head!=NULL){
            (*head).prev=newNode;
        }
        head=newNode;
    }
    else if(m==2){
        printf("Enter data: ");
        scanf("%d",&data);
        (*newNode).data=data;
        if(head==NULL){
            head=newNode;
        }
        else{
            node* temp=head;
            while((*temp).next!=NULL){
                temp=(*temp).next;
            }
            (*temp).next=newNode;
            (*newNode).prev=temp;
        }
    }
    else if(m==3){
        int pos;
        printf("Enter position to insert data: ");
        scanf("%d",&pos);
        printf("Enter data: ");
        scanf("%d",&data);
        if(pos<1){
            printf("Invalid position.\n");
```

```
            return 0;
        }
        (*newNode).data=data;
        if(pos==1){
            (*newNode).next=head;
            if(head!=NULL){
                (*head).prev=newNode;
            }
            head=newNode;
            printf("%d inserted at position %d.\n",data,pos);
            return 0;
        }
        node* temp=head;
        for(int i=1;i<pos-1&&temp!=NULL;i++){
            temp=(*temp).next;
        }
        if(temp==NULL){
            printf("Position out of range.\n");
            free(newNode);
            return 0;
        }
        (*newNode).next=(*temp).next;
        (*newNode).prev=temp;
        if((*temp).next!=NULL){
            (*(*temp).next).prev=newNode;
        }
        (*temp).next=newNode;
        printf("%d inserted at position %d.\n",data,pos);
    }
    return 0;
}
int delete(){
    int m;
    printf("1=Beginning \n2=End \n3=Given Position \nWhere do you wish to delete
node: ");
    scanf("%d",&m);
    if(m==1){
        if(head==NULL){
            printf("List is empty.\n");
            return 0;
        }
        node* temp=head;
        head=(*head).next;
        if(head!=NULL){
            (*head).prev=NULL;
        }
        printf("%d deleted from the beginning.\n",(*temp).data);
        free(temp);
    }
    else if(m==2){
        if(head==NULL){
```

```
        printf("List is empty.\n");
        return 0;
    }
    if((*head).next==NULL){
        printf("%d deleted from the end.\n",(*head).data);
        free(head);
        head=NULL;
        return 0;
    }
    node* temp=head;
    while((*temp).next!=NULL){
        temp=(*temp).next;
    }
    (*(*temp).prev).next=NULL;
    printf("%d deleted from the end.\n",(*temp).data);
    free(temp);
}
else if(m==3){
    int pos;
    printf("Enter position to delete data: ");
    scanf("%d",&pos);
    if(pos<1||head==NULL){
        printf("Invalid position or list is empty.\n");
        return 0;
    }
    if(pos==1){
        node* temp=head;
        head=(*head).next;
        if(head!=NULL){
            (*head).prev=NULL;
        }
        printf("%d deleted from position %d.\n",(*temp).data,pos);
        free(temp);
        return 0;
    }
    node* temp=head;
    for(int i=1;i<pos&&temp!=NULL;i++){
        temp=(*temp).next;
    }
    if(temp==NULL){
        printf("Position out of range.\n");
        return 0;
    }
    (*(*temp).prev).next=(*temp).next;
    if((*temp).next!=NULL){
        (*(*temp).next).prev=(*temp).prev;
    }
    printf("%d deleted from position %d.\n",(*temp).data,pos);
    free(temp);
}
return 0;
```

```
}
int search(){
    int data;
    printf("Enter data to search: ");
    scanf("%d",&data);
    node* temp=head;
    int pos=1;
    while(temp!=NULL){
        if((*temp).data==data){
            printf("%d found at position %d.\n",data,pos);
            return 0;
        }
        temp=(*temp).next;
        pos++;
    }
    printf("%d not found in the list.\n",data);
    return 0;
}
int main(){
    while(cont==1){
        int m=0;
        printf("1: Insert \n2: Delete \n3: Search \nEnter selection: ");
        scanf("%d",&m);
        if(m==1){
            insert();
        }
        else if(m==2){
            delete();
        }
        else if(m==3){
            search();
        }
        else{
            printf("Incorrect input!\n");
            cont=0;
            break;
        }
        printf("Do you wish to continue? (1/0): ");
        scanf("%d",&cont);
    }
    if(cont==0){
        printf("Terminated.\n");
    }
    else{
        printf("Incorrect input!\nTerminated.\n");
    }
    return 0;
}
```

```
PS C:\Palash\DSA> ./dll
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beginning
2=End
3=Given Position
Where do you wish to insert node: 1
Enter data: 35
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 2
1=Beginning
2=End
3=Given Position
Where do you wish to delete node: 1
35 deleted from the beginning.
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 2
1=Beginning
2=End
3=Given Position
Where do you wish to delete node: 1
List is empty.
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```

```
PS C:\Palash\DSA> ./dll
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beginning
2=End
3=Given Position
Where do you wish to insert node: 1
Enter data: 30
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 1
1=Beginning
2=End
3=Given Position
Where do you wish to insert node: 1
Enter data: 40
Do you wish to continue? (1/0): 1
1: Insert
2: Delete
3: Search
Enter selection: 3
Enter data to search: 40
40 found at position 1.
Do you wish to continue? (1/0): 0
Terminated.
PS C:\Palash\DSA>
```