CHAPTER 5
DATA STRUCTURES: LINKED LIST

5.1 Singly Linked List

5.1.1 Implementation

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

struct sll
{
        int no;
        struct sll *next;
};

struct sll *first = NULL;
struct sll *new1 = NULL;
struct sll *temp = NULL;
struct sll *prev = NULL;

void main ( )
{
        int choice, n, pos ;

        void insert_end (int);
        void insert_front (int);
        void insert_after (int, int);
        void insert_before (int, int);
        void delete_sll (int);
        void traverse ( );

        while ( 1 )
        {
                printf ("\n\n Menu for operations of SLL...");
                printf ("\n 1. Insert at end...");
                printf ("\n 2. Insert at front...");
                printf ("\n 3. Insert after...");
                printf ("\n 4. Insert before...");
                printf ("\n 5. Delete node...");
                printf ("\n 6. Traverse list...");
                printf ("\n 7. Exit...");

                printf ("\n\n Enter Ur choice...");
                scanf ("%d", &choice);
```

Bharat Chawda

```
        switch ( choice )
        {
            case 1:
                printf ("\n Enter no:\n");
                scanf ("%d", &n);
                insert_end ( n );
                break;

            case 2:
                printf ("\n Enter no:\n");
                scanf ("%d", &n);
                insert_front ( n );
                break;

            case 3:
                printf ("\n Enter no and position:\n");
                scanf ("%d%d", &n, &pos);
                insert_after (n, pos);
                break;

            case 4:
                printf ("\n Enter no and position:\n");
                scanf ("%d%d", &n, &pos);
                insert_before (n, pos);
                break;

            case 5:
                printf ("\n Enter no to be deleted:\n");
                scanf ("%d", &n);
                delete_sll ( n );
                break;

            case 6:
                traverse ( );
                break;

            case 7:
                printf ("\n Program terminated successfully...");
                exit ( 0 );

            default:
                printf ("\n Enter valid choice...\n");

        } // switch ends...
    } // while ends...
} // main() function ends...
```

Bharat Chawda

// insert_front() starts...

```
void insert_front (int x )
{
        // prepare a new node...

        new1 = (struct sll*) malloc (sizeof (struct sll));
        new1->no = x;
        new1->next = first;

        // set first pointer to point to newly created node...

        first = new1;
}
```

// insert_end() starts...

```
void insert_end ( int x )
{
        // prepare a new node...

        new1 = (struct sll*) malloc (sizeof (struct sll) );
        new1->no = x;
        new1->next = NULL;

        // if list is not available...

        if(first = = NULL)
        {
                first = new1;
                return;
        }

        // traverse to reach to end of list...

        temp = first;
        while(temp->next != NULL)
        {
                temp = temp->next;
        }

        // insert new node by adjusting pointers...

        temp->next = new1;

}
```

Bharat Chawda

Bharat Chawda

// insert_after() starts...

```
void insert_after (int x, int pos )
{
        // prepare a new node...

        new1 = (struct sll*) malloc (sizeof (struct sll) );
        new1->no = x;
        new1->next = NULL;

        // if list is not available...

        if (first = = NULL)
        {
                first = new1;
                return;
        }

        // traverse to reach to proper position...

        temp = first;
        while (temp->next != NULL && temp->no != pos)
        {
                temp = temp->next;
        }

        // insert new node by adjusting pointers...

        new1->next = temp->next;
        temp->next = new1;
}
```

// insert_before() starts...

```
void insert_before (int x, int pos)
{
        // prepare a new node...
        new1 = (struct sll*)malloc(sizeof(struct sll));
        new1->no = x;
        new1->next = NULL;

        // if list is not available...
        if(first = = NULL)
        {
                first = new1;
                return;
        }
        // if node is to be inserted at a front position...

        if(first->no = = pos)
        {
                new1->next = first;
                first = new1;
                return;
        }

        // traverse to reach to proper position...

        temp = first;
        while (temp != NULL && temp->no != pos)
        {
                prev = temp;
                temp = temp->next;
        }

        // insert new node by adjusting pointers...

        new1->next = prev->next;
        prev->next = new1;

}
```

// delete_sll() starts...

```
void delete_sll (int x)
{
        // if list is not available...

        if(first = = NULL)
        {
                printf("\n SLL is empty...");
                return;
        }

        // set temp to point to front node in a list...
        temp = first;

        // if node to be deleted is front node...
        if (first->no = = x)
        {
                first = first->next;
                free (temp);
                return;
        }
```

Bharat Chawda

```
        // traverse to reach to proper position...

        while (temp != NULL && temp->no != x)
        {
                prev = temp;
                temp = temp->next;
        }

        // if node to be deleted is not available...

        if (temp = = NULL)
        {
                printf ("\n Node to be deleted is not available...\n");
                return;
        }

        // delete node by freeing memory...

        prev->next = temp->next;
        free ( temp );

}
```

// traverse ( ) starts...

```
void traverse ( )
{
        // if list is not available...
        if(first = = NULL)
        {
                printf ("\n The SLL is empty...");
                return;
        }

        // traverse to end of list...

        temp = first;

        printf ("\n The SLL is : ");

        while (temp != NULL)
        {
                printf ("%d     ", temp->no);
                temp = temp->next;
        }

}
```

Bharat Chawda

5.1.2 SLL: Insert at End / Append

- **Algorithm:**

❖ **INSERT_END (X)**

- [Inserts a node having element 'X' at end of Singly Linked List.]

❖ **Variables:**

i)	**FIRST**	:	Pointer to point to first node in a list.
ii)	**TEMP**	:	Pointer to point to nodes while traversing a list.
iii)	**NEW1**	:	Pointer to point a new node which is to be inserted.
iv)	**NO**	:	Data / Information part of a node.
v)	**NEXT**	:	Pointer / Address part of a node.
vi)	**X**	:	Element to be inserted in a list.

❖ **Steps:**

- **Step-1:** [Prepare a new node.]

i) Create a new node by allocating memory.

```
NEW1 ← Get new node.
```

ii) Assign X to data part of a node.

```
NO (NEW1) ← X
```

iii) Assign NULL to pointer part of a node.

```
NEXT (NEW1) ← NULL
```

- **Step-2:** [If there is no list, or, list is empty.]

```
IF ( FIRST = NULL ) THEN
        FIRST ← NEW1
        RETURN
END IF
```

- **Step-3:** [Traverse to reach to end of list.]

```
TEMP ← FIRST
WHILE ( NEXT ( TEMP ) <> NULL ) DO
        TEMP ← NEXT (TEMP)
END WHILE
```

- **Step-4:** [Insert new node by adjusting pointers.]

```
NEXT ( TEMP ) ← NEW1
```

- **Step-5:** [Finish]

```
RETURN
```

Bharat Chawda

5.1.3 SLL: Insert After

- **Algorithm:**

  ❖ **INSERT_AFTER (X, POS)**

  - [Inserts a node having element ‘X’ in a list **after** a node having value ‘POS’.]

  ❖ **Variables:**

  i) **FIRST** : Pointer to point to first node in a list.

  ii) **TEMP** : Pointer to point to nodes while traversing a list.

  iii) **NEW1** : Pointer to point a new node which is to be inserted.

  iv) **NO** : Data / Information part of a node.

  v) **NEXT** : Pointer / Address part of a node.

  vi) **X** : Element to be inserted in a list.

  vii) **POS** : Element showing reference position.

  ❖ **Steps:**

  - **Step-1:** [Prepare a new node.]

    i) Create a new node by allocating memory.

    ```
    NEW1 ← Get new node.
    ```

    ii) Assign X to data part of a node.

    ```
    NO (NEW1) ← X
    ```

    iii) Assign NULL to pointer part of a node.

    ```
    NEXT (NEW1) ← NULL
    ```

  - **Step-2:** [If there is no list, or, if list is empty.]

    ```
    IF ( FIRST = NULL ) THEN
            FIRST ← NEW1
            RETURN
    END IF
    ```

  - **Step-3:** [Traverse to reach to proper position in a list.]

    ```
    TEMP ← FIRST
    WHILE (NEXT (TEMP) <> NULL   AND   NO (TEMP) <> POS) DO
            TEMP ← NEXT (TEMP)
    END WHILE
    ```

  - **Step-4:** [Insert new node by adjusting pointers.]

    ```
    NEXT ( NEW1 ) ← NEXT ( TEMP )
    NEXT ( TEMP ) ← NEW1
    ```

  - **Step-5:** [Finish]

    ```
    RETURN
    ```

Bharat Chawda

5.1.4 SLL: Insert Before

- **Algorithm:**

❖ **INSERT_BEFORE (X, POS)**

- [Inserts a node having element 'X' in a list **before** a node having value 'POS'.]

❖ **Variables:**

i)	**FIRST**	:	Pointer to point to first node in a list.
ii)	**TEMP**	:	Pointer to point to nodes while traversing a list.
iii)	**PREV**	:	Pointer to point to previous node while traversing.
iv)	**NEW1**	:	Pointer to point a new node which is to be inserted.
v)	**NO**	:	Data / Information part of a node.
vi)	**NEXT**	:	Pointer / Address part of a node.
vii)	**X**	:	Element to be inserted in a list.
viii)	**POS**	:	Element showing reference position.

❖ **Steps:**

- **Step-1:** [Prepare a new node.]

i) Create a new node by allocating memory.

```
NEW1 ← Get new node.
```

ii) Assign X to data part of a node.

```
NO (NEW1) ← X
```

iii) Assign NULL to pointer part of a node.

```
NEXT (NEW1) ← NULL
```

- **Step-2:** [If there is no list, or, if list is empty.]

```
IF ( FIRST = NULL ) THEN
        FIRST ← NEW1
        RETURN
END IF
```

- **Step-3:** [If node is to be inserted at front position in a list.]

```
IF ( NO (FIRST) = POS ) THEN
        NEXT ( NEW1 ) ← FIRST
        FIRST ← NEW1
        RETURN
END IF
```

Bharat Chawda

- **Step-4:** [Traverse to reach to proper position in a list.]

```
TEMP  ← FIRST
WHILE (TEMP <> NULL   AND   NO (TEMP) <> POS ) DO
        PREV  ←  TEMP
        TEMP  ←  NEXT (TEMP)
END WHILE
```

- **Step-5:** [Insert new node by adjusting pointers.]

```
NEXT ( NEW1 )  ←  NEXT ( PREV )
NEXT ( PREV )  ←  NEW1
```

- **Step-6:** [Finish]

```
RETURN
```

5.1.5 SLL: Delete Node

- **Algorithm:**

❖ **DELETE_SLL (X)**

- [Deletes a node having element ‘X’ from a singly linked list.]

❖ **Variables:**

i) **FIRST** : Pointer to point to first node in a list.

ii) **TEMP** : Pointer to point to nodes while traversing a list.

iii) **PREV** : Pointer to point to previous node while traversing.

iv) **NEW1** : Pointer to point a new node which is to be inserted.

v) **NO** : Data / Information part of a node.

vi) **NEXT** : Pointer / Address part of a node.

vii) **X** : Element to be deleted from a list.

❖ **Steps:**

- **Step-1:** [If there is no list, or, if list is empty.]

```
IF ( FIRST = NULL ) THEN
        WRITE ( ‘ List is not available. ’ )
        RETURN
END IF
```

- **Step-2:** [Set TEMP to point to front node in a list.]

```
TEMP  ←  FIRST
```

Bharat Chawda

- **Step-3:** [If node to be deleted is front node.]

```
IF ( NO (FIRST) = X ) THEN
        FIRST  ←  NEXT (FIRST)
        FREE ( TEMP )
        RETURN
END IF
```

- **Step-4:** [Traverse to reach to proper position in a list.]

```
WHILE (TEMP <> NULL   AND   NO (TEMP) <> X) DO
        PREV  ←  TEMP
        TEMP  ←  NEXT (TEMP)
END WHILE
```

- **Step-5:** [If node to be deleted is not available.]

```
IF ( TEMP  =  NULL ) THEN
        WRITE ( ' Node to be deleted is not available.' )
        RETURN
END IF
```

- **Step-6:** [Delete node by freeing memory.]

```
NEXT ( PREV )  ←  NEXT ( TEMP )
FREE ( TEMP )
```

- **Step-7:** [Finish]

```
RETURN
```

● ● ●

Bharat Chawda