Experiment 12
Implementation Of Linked List

Date: 10-11-2020

Aim: Implementation of linked list

Data Structure Used: Linked List

Operation Used: Comparisons

Algorithm:

Algorithm for InsertFront

Input: Header Node of a linked list (LL) and the ITEM to be inserted
Output: Linked List with the new node inserted after the Header node
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1: new→data = ITEM
    Step 2: new→link = Header→link
    Step 3: Header→link = new
Step 5: endif
Step 6: Stop

Description of the Algorithm: This Algorithm inserts a node just after the header node

Algorithm for InsertBack

Input: Header Node of a linked list (LL) and the ITEM to be inserted
Output: Linked List with the new node inserted at the end of the List
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1;ptr = Header
    Step 2: while(ptr→ link !=NULL) do
        Step 1: ptr=ptr→link
    Step 3: endWhile
    Step 4:new→data = ITEM
    Step 5: new→link = NULL
    Step 6: ptr→link = new
Step 5: endif
Step 6: Stop

Description of the Algorithm: This algorithm goes to the end of the List and inserts a node after the last node

Algorithm for InsertFront

Input: Header Node of a linked list (LL), the ITEM to be inserted and the position (POS)

Output: Linked List with the new node inserted at the corresponding position
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1: i=-1
    Step 2: ptr = Header
    Step 3: while(i<pos-1 and ptr!=NULL) then
        Step 1: i++
        Step 2: ptr=ptr→link
    Step 4: endwhile
    Step 5: if(ptr!=NULL) then
        Step 1: new→data = ITEM
        Step 2: new→link = ptr→link
        Step 3: ptr→link = new
    Step 6: else
        Step 1: print("Given position is not found")
        Step 2: Stop
    Step 7: endif
Step 5: endif
Step 6: Stop

Description of the Algorithm: This algorithm traversed the List, on reaching the node at the index position passed it inserts a new node at that position. Eg: if the List is "34 21 56 12" and assume the elements are indexed from 0 (even though it is a linked list and indexing of elements don't make any sense) if I want to insert 23 at position 2. The resulting Linked list will be
"34 21 23 56 12".

Algorithm for DeleteFront
Input: Header Node of a linked list (LL)
Output: The item removed from the list
Data Structure: Linked List

Step 1 : Start
Step 2: if(Header→link ==NULL) then
    Step 1: print("Linked List is empty")
    Step 2: Stop
Step 3: else
    Step 1: ptr = Header→link
    Step 2: Header→link = ptr→ link
    Step 3: ITEM = ptr→data
    Step 4: ReturnNode(ptr)
    Step 5: return ITEM
Step 4: endif
Step 5: Stop

Description of the Algorithm: This algorithm deletes the node just after the header node

Algorithm for DeleteRear
Input: Header Node of a linked list (LL)
Output: The item removed from the end of the list
Data Structure: Linked List

```
Step 1 : Start
Step 2: if(Header→link ==NULL) then
        Step 1: print("Linked List is empty")
        Step 2: Stop
Step 3: else
        Step 1: ptr = Header →link
        Step 2: ptr1 = Header
        Step 3: while(ptr→link!=NULL) do
                Step 1: ptr1=ptr
                Step 2: ptr = ptr→link
        Step 4: EndWhile
        Step 5: ITEM = ptr→data
        Step 6: ptr1→link = ptr→link
        Step 7: ReturnNode(ptr)
        Step 8: return ITEM
Step 4: EndIf
Step 5  : Stop
```

Description of the Algorithm: This algorithm deletes the Node at the end of the linked list

Algorithm for Delete from a position

Input: Header Node of a linked list (LL) and the position of the node to be removed
Output: The item removed from the specified position of the list
Data Structure: Linked List

```
Step 1 : Start
Step 2: if(Header→link == NULL)
        Step 1: Print("The List Is Empty")
        Step 2: Stop
Step 3: else
        Step 1: i=-1
        Step 2: ptr = Header
        Step 3: while(i<pos-1 and ptr!=NULL) then
                Step 1: i++
                Step 2: ptr=ptr→link
        Step 4: endwhile
        Step 5:if(ptr→link ==NULL)
                Step 1: ITEM = ptr->link→ data
                Step 2: ptr1 = ptr→link
                Step 3: ptr→link = ptr1→link
                Step 4: ReturnNode(ptr1)
                Step 5:return(ITEM)
        Step 6: else
                Step 1: Print("Index Out Of Bounds")
                Step 2: Stop
        Step 7:endif
Step 4: endif
Step 5: Stop
```

Description of the Algorithm: Just like the insertion at any position algorithm passing the position of the element to be deleted will remove the element. It takes a pointer (ptr) to the element right before the one to be deleted and then links the link part of ptr to the link of the element to be deleted.

Program Code:

```
/*******************************
 * Linked List Implementation
 * Done By: Rohit Karunakaran
 * ****************************/

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

typedef struct Linked_List_Node
{
    struct Linked_List_Node *link;
    int data;
}Node;

void initList(Node* Header)
{
    //Header = (Node*) malloc (sizeof(Node));
    Header->link = NULL;
    Header->data = 0;
}


//Insertion Algorithms
void insertStart(Node *Header,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->data = val;
        new_node->link = NULL;
        Node* ptr = Header->link;
        Header->link = new_node;
        new_node->link=ptr;
    }
    else
    {
        printf("Insertion Not Possible\n");
        exit(1);
    }
    return ;
}

void insertAt(Node *Header,int val,int pos) //Insert at a specified position from
the header node
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        Node* ptr = Header;
        int index = -1;
        while(index<pos-1 && ptr!=NULL)
        {
```

```
            ptr=ptr->link;
            index ++;
        }
        if(ptr !=NULL)
        {
            new_node->link = ptr->link;
            new_node->data = val;
            ptr->link =new_node;
        }
        else
        {
            printf("Given position is not found \nExiting......\n");
            exit(1);
        }
    }
    else
    {
        printf("Insertion Not Possible");
        exit(1);
    }
    return ;

}

void insertEnd(Node *Header,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->data = val;
        new_node->link = NULL;
        Node* ptr=Header;

        while(ptr->link != NULL)
        {
            ptr = ptr->link;
        }

        ptr->link = new_node;
    }

    else
    {
        printf("Insertion not possible");
        exit(1);
    }

    return;
}

//Deletion Algorithms
int deletionBegin(Node *Header)
{
    if(Header->link == NULL)
    {
```

```
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
    else
    {
        Node* ptr = Header->link;
        Header->link = ptr->link;
        int elem = ptr->data;
        free(ptr);
        return elem;
    }
}

int deletionAt(Node* Header, int pos)
{
    if(Header->link == NULL)
    {
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
    else
    {
        int index = -1;
        Node* ptr = Header;
        while(index<pos-1&&ptr!=NULL)
        {
            ptr=ptr->link;
            index++;
        }
        if(ptr->link!=NULL)
        {
            int elem = ptr->link->data;
            Node* red = ptr->link;
            ptr->link = ptr->link->link;
            free(red);
            return elem;
        }
        else
        {
            printf("Index Is out of Bounds \n");
            exit(1);
            return 0;
        }
    }
}

int deletionEnd(Node* Header)
{
    if(Header->link == NULL)
    {
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
```

```
    else
    {
        Node* ptr=Header->link;
        Node* ptr1=Header;
        while(ptr->link!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->link;
        }
        int elem = ptr->data;
        ptr1->link = NULL;
        free(ptr);
        return elem;
    }
}

void displayList(Node* Header)
{
    Node* ptr = Header->link;
    if(ptr!=NULL)
    {
        printf("The List is : ");
        while(ptr!=NULL)
        {
            printf("%d ",ptr->data);
            ptr=ptr->link;
        }
        printf("\n");
    }
    else
    {
        printf("The Linked list is empty\n");
    }
}

int menu(Node* Header)
{
    int RUN = 1;
    while(RUN)
    {
        printf("\n");
        printf("============================\n");
        printf("            MENU            \n");
        printf("============================\n");
        printf("1.Insert At Begining\n");
        printf("2.Insert At End\n");
        printf("3.Insert At Position\n");

        printf("4.Delete From Begining\n");
        printf("5.Delete From End\n");
        printf("6.Delete From Position\n");
        printf("7.Display the linked List\n");
        printf("8.Exit\n");
        printf("Enter Choice: ");
        int choice;
        int elem;
```

```
    int pos;
    scanf("%d%*c",&choice);

    switch(choice)
    {
        case 1: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                insertStart(Header,elem);
                printf("\n");
                break;

        case 2: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                insertEnd(Header,elem);
                printf("\n");
                break;

        case 3: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                printf("Enter the postion to insert %d : ",elem);
                scanf("%d%*c",&pos);
                insertAt(Header,elem,pos);
                printf("\n");
                break;

        case 4: elem = deletionBegin(Header);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;

        case 5: elem = deletionEnd(Header);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;


        case 6: printf("Enter the postion of the element to be deleted : ");
                scanf("%d%*c",&pos);
                elem = deletionAt(Header,pos);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;

        case 7: displayList(Header);
                break;

        case 8: RUN=0;
                break;
        default: printf("Enter a valid choice\n");
                printf("\n");
                 break;

    }

}
printf("Exiting........\n");
```

```
    return RUN;
}

int main()
{
    Node *Header = (Node*)malloc(sizeof(Node));
    initList(Header);
    return menu(Header);
}
```

Result: The Program is successfully compiled and the desired result is obtained

Sample Input and output

```
============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 7
The List is : 39 93 72

============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 6
Enter the postion of the element to be deleted2
The Element removed is 72

============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 5
The Element removed is 93
```

```
============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 5
The Element removed is 93

============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 4
The Element removed is 39

============================
           MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 7
The Linked list is empty
```

```
============================
            MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 8
Exiting........%
```