**Assignment Number**

**Problem Statement**

Menu driven program in C to perform all operations on a single linked list.

**Theory**

In Array we have some limitations like we have to initialize the array size at the time when program is written and its space is reserved during compilation of the program. During the execution of the program, it may so happen that the actual size is less than the defined size and so, a good amount of space is wasted. The opposite phenomena could also happen. Another problem is that during insertion operation, we have to move the elements from the end of the array by a index to the right and then insert the element at the desired position. This means the number of shifting for insertion and deletion is O(n) for as list of n elements in average case.

With using linked list we can overcome from all this problems. Basically in linked list we create a node which has two parts, one reserved for the data and other holds the address of the next node, adjacent to the former node. A basic representation of linked list is given below:

Representation of an ordered list of alphabet, L=(A,B,C,D) using linked list may be depicted as shown in figure :



**Algorithm**

**Input :** A pointer, say ‘start’ that would hold the address of the first node of a singly linked list and an element, say ‘item’ that has to be inserted or deleted or to be searched as per user instruction and a position, say ‘pos’ from where an element is to be inserted or deleted.

**Output :**

1) Item inserted or deleted Successfully or a suitable unsuccessful message.

2) Item inserted or deleted from position ‘pos’ or suitable unsuccessful message.

3) Item found a position ‘pos’ or not found message.

**Data Structure used :** A single linked list where each node contains a data part, say DATA and an address field, say LINK that contains the address of the intermediate next node of a given node.

**Steps :**

**Algorithm\_Display()**

1. Begin
2. If(start = NULL)

Then

* 1. Print “List is Empty !!”

1. Else
   1. Set ptr = start
   2. While(ptr ! = NULL)
      1. Process(ptr->data) //Process function prints the data in ptr node
      2. Set ptr = ptr->link

EndWhile

EndIf

1. End

**Algorithm\_InsertBegin()**

1. Begin
2. Set temp=Getnode() //
3. If temp = NULL

Then

* 1. Print “Memory Allocation Unsuccessful !!”
  2. Exit

1. End If
2. Set temp->data = item
3. Set temp->link = NULL
4. If start = NULL

Then

* 1. Set Start = temp

1. Else
   1. Set ptr = start
   2. While( ptr->link ! = NULL)

Begin

* + 1. Set ptr = ptr->link

End While

* 1. Set ptr->link = temp

End If

1. End

**Algorithm\_InsertEnd()**

1. Begin
2. Set temp = Getnode() //
3. If temp = NULL

Then

* 1. Print “Memory Allocation Unsuccessful !!”

End If

1. Set temp->data = item
2. Set temp->link = NULL
3. If(start = NULL)

Then

* 1. Set start = temp

1. Else
   1. Set ptr = start
   2. While(ptr->link != NULL)

Begin

* + 1. Set ptr = ptr->link

End While

* 1. Set ptr->link = temp

End If

1. End

**Algorithm\_InsertPos()**

1. Begin
2. Set temp=Getnode() // Getnode is a function which dynamically

// allocates memory for a node and returns its pointer

1. If temp = NULL

Then

* 1. Print “Memory Allocation Unsuccessful !!”

End If

1. Set temp->data = item
2. Set temp->link = NULL
3. If pos = 0

Then

* 1. Set temp->link = start
  2. Set start = temp

1. Else
   1. Set i=0
   2. Set ptr = start
   3. While(i< pos-1)

Begin

* + 1. Set ptr = ptr->link
    2. If(ptr = NULL)

Then

* + - 1. Print “Position not Found !!”
      2. Exit

End If

* + 1. Set i = i+1

End While

* 1. Set temp->link = ptr->link
  2. Set ptr->link = temp

End If

1. End

**Algorithm\_DeleteBegin()**

1. Begin
2. If (ptr = NULL)

Then

* 1. Print “List is Empty !!”

1. Else
   1. Set ptr = start
   2. Set start = start -> link
   3. Free(ptr)

End if

1. End

**Algorithm\_DeleteEnd()**

1. Begin
2. If ( start = NULL)

Then

* 1. Print “List is Empty !!”

1. Else If( start -> link = NULL)

Then

* 1. Set ptr = start
  2. Set start = NULL
  3. Free(ptr)

1. Else
   1. Set ptr = start
   2. While(ptr->link ! = NULL)

Begin

* + 1. Set temp= ptr
    2. Set ptr = ptr ->link

End While

* 1. Set temp->link = NULL
  2. Free(ptr)

End If

Algorithm \_DeletePos()

1. Begin
2. If( start = NULL)

Then

* 1. Print “List is Empty !!”

1. Else
   1. If( pos = 0)

Then

* + 1. Set ptr = start
    2. Set start = start -> link
    3. Free(ptr)
  1. Else
     1. Set ptr = start
     2. Set i = 0
     3. While(i<pos)

Begin

* + - 1. Set temp=ptr
      2. Set ptr = ptr->link
      3. If (ptr = NULL)

Then

* + - * 1. Print “Position not found !!”
        2. Exit

EndIf

* + - 1. Set i= i+1

EndWhile

* + 1. Set temp->link = ptr ->link
    2. Free(ptr)

EndIf

EndIf

1. End

**Algorithm\_Search()**

1. Begin
2. If (start = NULL)

Then

* 1. Print “List is Empty !!”

1. Else
   1. Set ptr = start
   2. While(ptr ! = NULL)

Begin

* + 1. If(ptr->data = item)

Then

* + - 1. Set flag = 1
      2. Exit While Loop
    1. Else
       1. Set ptr = ptr->link
       2. Set count = count +1

End If

EndWhile

* 1. If(flag = 1)

Then

* + 1. Print “Item found at position pos”
  1. Else
     1. Print “Item not found”

End If

End If

1. End

**Algorithm\_Main()**

1. While(True)
   1. Print "1.Display”
   2. Print “2.Insert at Beginning”
   3. Print “3.Insert at end”
   4. Print “4.Insert at any position”
   5. Print “5.Delete at Beginning”
   6. Print “6.Delete at End”
   7. Print “7.Delete from any position”
   8. Print “8.Searching an Item”
   9. Print “9.Exit"
   10. Input choice
   11. If(choice = 1)
       1. Call Display()
   12. Else If(choice = 2)
       1. Call InsertBegin()
   13. Else If(choice = 3)
       1. Call InsertEnd()
   14. Else If(choice = 4)
       1. Call InsertPos()
   15. Else If(choice = 5)
       1. Call DeleteBegin()
   16. Else If(choice = 6)
       1. Call DeleteEnd()
   17. Else If(choice = 7)
       1. Call DeletePos()
   18. Else If(choice = 8)
       1. Call Search()
   19. Else
       1. Print “Wrong choice”
       2. Exit

EndWhile

**Source Code**

#include<stdlib.h>

#include<stdio.h>

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

void search();

struct node

{

int data;

struct node \*link;

};

struct node \*start=NULL;

int main()

{

int ch;

while(1){

printf("\n1.Display\n2.Insert at Beginning\n3.Insert at end\n4.Insert at any position\n5.Delete at Beginning\n6.Delete at End\n7.Delete from any position\n8.Searching an Item\n9.Exit");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

display();

break;

case 2:

insert\_begin();

break;

case 3:

insert\_end();

break;

case 4:

insert\_pos();

break;

case 5:

delete\_begin();

break;

case 6:

delete\_end();

break;

case 7:

delete\_pos();

break;

case 8:

search();

break;

case 9: exit(0);

break;

default:

printf("\n Wrong choice:\n");

break;

}

}

return 0;

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

printf("\nList is empty!!");

return;

}

else

{

ptr=start;

printf("\nThe List elements are:");

while(ptr!=NULL)

{

printf("%d ",ptr->data );

ptr=ptr->link ;

}

}

}

void insert\_begin()

{

struct node \*temp;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nMemory allocation unsuccessfull:\n");

}

printf("\nEnter the item to be inserted:" );

scanf("%d",&temp->data);

temp->link =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->link=start;

start=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nMemory allocation unsuccessfull!!");

}

printf("\nEnter the item to be inserted:\t" );

scanf("%d",&temp->data );

temp->link =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->link !=NULL)

{

ptr=ptr->link ;

}

ptr->link =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nMemory allocation unsuccessfull!!");

return;

}

printf("\nEnter the position for the new node to be inserted:\t");

scanf("%d",&pos);

printf("\nEnter the data value of the node:\t");

scanf("%d",&temp->data) ;

temp->link=NULL;

if(pos==0)

{

temp->link=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++)

{

ptr=ptr->link;

if(ptr==NULL)

{

printf("\nPosition not found!!");

return;

}

}

temp->link =ptr->link ;

ptr->link=temp;

}

}

void delete\_begin()

{

struct node \*ptr;

if(ptr==NULL)

{

printf("\nList is Empty!!");

return;

}

else

{

ptr=start;

start=start->link ;

printf("\nThe deleted element is :%d",ptr->data);

free(ptr);

}

}

void delete\_end()

{

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\nList is Empty:");

exit(0);

}

else if(start->link ==NULL)

{

ptr=start;

start=NULL;

printf("\nThe deleted element is:%d\t",ptr->data);

free(ptr);

}

else

{

ptr=start;

while(ptr->link!=NULL)

{

temp=ptr;

ptr=ptr->link;

}

temp->link=NULL;

printf("\nThe deleted element is:%d\t",ptr->data);

free(ptr);

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\nThe List is Empty!!");

exit(0);

}

else

{

printf("\nEnter the position of the node to be deleted:");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->link ;

printf("\nThe deleted element is:%d",ptr->data );

free(ptr);

}

else

{

ptr=start;

for(i=0;i<pos;i++)

{

temp=ptr;

ptr=ptr->link ;

if(ptr==NULL)

{

printf("\nPosition not Found!!");

return;

}

}

temp->link =ptr->link ;

printf("\nThe deleted element is:%d",ptr->data );

free(ptr);

}

}

}

void search()

{

int item,count=1,flag=0;

struct node \*ptr;

if(start==NULL)

{

printf("\nList is empty!!");

return;

}

else

{

printf("\nEnter the item to be searched:");

scanf("%d",&item);

ptr=start;

while(ptr!=NULL)

{

if(ptr->data==item)

{

flag=1;

break;

}

else{

ptr=ptr->link;

count=count+1;

}

}

if(flag==1)

{

printf("\nItem found at position %d !!",count);

}

else

{

printf("\nItem not found!!");

}

}

}

**Input and Output**

1.Display

2.Insert at Beginning

3.Insert at end

4.Insert at any position

5.Delete at Beginning

6.Delete at End

7.Delete from any position

8.Searching an Item

9.Exit

Enter your choice:2

Enter the item to be inserted: 69

1.Display

2.Insert at Beginning

3.Insert at end

4.Insert at any position

5.Delete at Beginning

6.Delete at End

7.Delete from any position

8.Searching an Item

9.Exit

Enter your choice: 3

Enter the item to be inserted: 100

1.Display

2.Insert at Beginning

3.Insert at end

4.Insert at any position

5.Delete at Beginning

6.Delete at End

7.Delete from any position

8.Searching an Item

9.Exit

Enter your choice:1

The List elements are: 69 100

1.Display

2.Insert at Beginning

3.Insert at end

4.Insert at any position

5.Delete at Beginning

6.Delete at End

7.Delete from any position

8.Searching an Item

9.Exit

Enter your choice:9

**Discussion**

1. This program dynamically allocates memory for node. Hence it will fail if memory is insufficient.
2. Dynamic allocation is flexible enough, but it is painfully slow for good performance. Hence, some kind of caching might be used for good performance.
3. Linked list is very memory efficient, but insertion, deletion and searching is O(n) at the worst case. There are special type of lists each for one specific usecase, which has much better speciailized performance.