**ASSIGNMENT NO. :**

**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()

Begin

If(start = NULL)

Then

Print “List is Empty !!”

Else

Set ptr = start

While(ptr ! = NULL)

Begin

Process(ptr->data) //Process function prints the data in ptr node

Set ptr = ptr->link

End While

End If

End

Algorithm\_Insert\_beginning()

Begin

Set temp=Getnode() //

If temp = NULL

Then

Print “Memory Allocation Unsuccessful !!”

End If

Set temp->data = item

Set temp->link = NULL

If start = NULL

Then

Set Start = temp

Else

Set ptr = start

While( ptr->link ! = NULL)

Begin

Set ptr = ptr->link

End While

Set ptr->link = temp

End If

End

Algorithm\_insert\_end()

Begin

Set temp = Getnode() //

If temp = NULL

Then

Print “Memory Allocation Unsuccessful !!”

End If

Set temp->data = item

Set temp->link = NULL

If(start = NULL)

Then

Set start = temp

Else

Set ptr = start

While(ptr->link != NULL)

Begin

Set ptr = ptr->link

End While

Set ptr->link = temp

End If

End

Algorithm\_Insert\_pos()

Begin

Set temp=Getnode() //

If temp = NULL

Then

Print “Memory Allocation Unsuccessful !!”

End If

Set temp->data = item

Set temp->link = NULL

If pos = 0

Then

Set temp->link = start

Set start = temp

Else

Set i=0

Set ptr = start

While(i< pos-1)

Begin

Set ptr = ptr->link

If(ptr = NULL)

Then

Print “Position not Found !!”

End If

Set i = i+1

End While

Set temp->link = ptr->link

Set ptr->link = temp

End If

End

Algorithm\_delete\_begin()

Begin

If (ptr = NULL)

Then

Print “List is Empty !!”

Else

Set ptr = start

Set start = start -> link

Free(ptr)

End if

End

Algorithm\_delete\_end()

Begin

If ( start = NULL)

Then

Print “List is Empty !!”

Else If( start -> link = NULL)

Then

Set ptr = start

Set start = NULL

Free(ptr)

Else

Set ptr = start

While(ptr->link ! = NULL)

Begin

Set temp= ptr

Set ptr = ptr ->link

End While

Set temp->link = NULL

Free(ptr)

End If

End If

Algorithm \_delete\_pos()

Begin

If( start = NULL)

Then

Print “List is Empty !!”

Else

If( pos = 0)

Then

Set ptr = start

Set start = start -> link

Free(ptr)

Else

Set ptr = start

Set i = 0

While(i<pos)

Begin

Set temp=ptr

Set ptr = ptr->link

If (ptr = NULL)

Then

Print “Position not found !!”

End

Set i= i+1

End While

Set temp->link = ptr ->link

Free(ptr)

End If

End If

End

Algorithm\_search()

Begin

If (start = NULL)

Then

Print “List is Empty !!”

Else

Set ptr = start

While(ptr ! = NULL)

Begin

If(ptr->data = item)

Then

Set flag = 1

Break

Else

Set ptr = ptr->link

Set count = count +1

End If

End While

If(flag = 1)

Then

Print “Item found at position pos”

Else

Print “Item not found”

End If

End If

End

Algorithm\_main()

// write yourself. Too lazy to write

**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 :-**

**SET 1:** 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:40

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: 50

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:40 50

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) If we would have used realloc function then it would preserve the contents of the memory as by using free(ptr) we deleted the content of temporary variable.

2) To sort the elements, its easier to manipulate in array compared to linked list. Secondly, it requires more space as pointers are also stored with informations.

3) Traversal to a particular element takes time as we have to go through all the elements which came before it.