Date :   Page No :

Assignment Number

Problem Statement

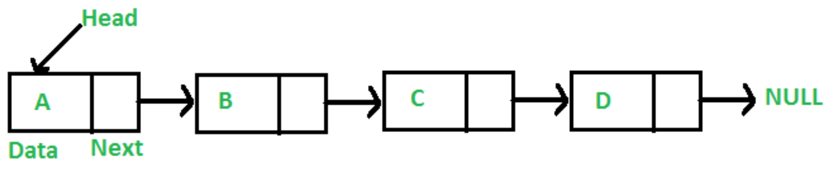
Menu driven program in C to perform all operations on a single linked list.

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()

```
Step 1 :   Begin
Step 2 :   If(start = NULL)
           Then
    a)  Print “List is Empty !!”
Step 3 :   Else
    a)  Set ptr = start
    b)  While(ptr ! = NULL)
        i.   Process(ptr->data) //Process function prints the data in ptr node
        ii.  Set ptr = ptr->link
        EndWhile
    EndIf
Step 4 :   End
```

Algorithm_InsertBegin()

```
Step 1 :   Begin
Step 2 :   Set temp=Getnode() //
Step 3 :   If temp = NULL
    Then
```

```
     a) Print “Memory Allocation Unsuccessful !!”
     b) Exit
Step 4 :  End If
Step 5 :  Set temp->data = item
Step 6 :  Set temp->link = NULL
Step 7 :  If start = NULL
     Then
     a) Set Start = temp
Step 8 :  Else
     a) Set ptr = start
     b) While( ptr->link ! = NULL)
        Begin
        i.  Set ptr = ptr->link
        End While
     c) Set ptr->link = temp
     End If
Step 9 :  End
```

Algorithm_InsertEnd()

```
Step 1 :  Begin
Step 2 :  Set temp = Getnode() //
Step 3 :  If temp = NULL
     Then
     a) Print “Memory Allocation Unsuccessful !!”
     End If
Step 4 :  Set temp->data = item
Step 5 :  Set temp->link = NULL
Step 6 :  If(start = NULL)
     Then
     a) Set start = temp
Step 7 :  Else
     a) Set ptr = start
     b) While(ptr->link != NULL)
        Begin
        i.  Set ptr = ptr->link
        End While
     c) Set ptr->link = temp
     End If
```

```
Step 8 :   End
```

Algorithm_InsertPos()

```
Step 1 :   Begin
Step 2 :   Set temp=Getnode() // Getnode is a function which dynamically
                              // allocates memory for a node and returns its pointer
Step 3 :   If temp = NULL
    Then
    a)  Print “Memory Allocation Unsuccessful !!”
    End If
Step 4 :   Set temp->data = item
Step 5 :   Set temp->link = NULL
Step 6 :   If pos = 0
    Then
    a)  Set temp->link = start
    b)  Set start = temp
Step 7 :   Else
    a)  Set i=0
    b)  Set ptr = start
    c)  While(i< pos-1)
        Begin
        i.   Set ptr = ptr->link
        ii.  If(ptr = NULL)
             Then
             I.   Print “Position not Found !!”
             II.  Exit
             End If
        iii. Set i = i+1
        End While
    d)  Set temp->link = ptr->link
    e)  Set ptr->link = temp
    End If
Step 8 :   End
```

Algorithm_DeleteBegin()

```
Step 1 :   Begin
Step 2 :   If (ptr = NULL)
    Then
```

```
     a) Print "List is Empty !!"
Step 3 :  Else
     a) Set ptr = start
     b) Set start = start -> link
     c) Free(ptr)
     End if
Step 4 :  End
```

Algorithm_DeleteEnd()

```
Step 1 :  Begin
Step 2 :  If ( start = NULL)
     Then
     a) Print "List is Empty !!"
Step 3 :  Else If( start -> link = NULL)
          Then
     a) Set ptr = start
     b) Set start = NULL
     c) Free(ptr)
Step 4 :  Else
     a) Set ptr = start
     b) While(ptr->link ! = NULL)
        Begin
        i.   Set temp= ptr
        ii.  Set ptr = ptr ->link
        End While
     c) Set temp->link = NULL
     d) Free(ptr)
     End If
```

Algorithm _DeletePos()

```
Step 1 :  Begin
Step 2 :  If( start = NULL)
     Then
     a) Print "List is Empty !!"
Step 3 :  Else
     a) If( pos = 0)
        Then
```

```
            i.   Set ptr = start
            ii.  Set start = start -> link
            iii. Free(ptr)
        b)  Else
            i.   Set ptr = start
            ii.  Set i = 0
            iii. While(i<pos)
                 Begin
                 A)  Set temp=ptr
                 B)  Set ptr = ptr->link
                 C)  If (ptr = NULL)
                     Then
                     I.   Print "Position not found !!"
                     II.  Exit
                     EndIf
                 D)  Set i= i+1
                 EndWhile
            iv.  Set temp->link = ptr ->link
            v.   Free(ptr)
            EndIf
        EndIf
    Step 4 :    End
```

Algorithm_Search()

```
    Step 1 :    Begin
    Step 2 :    If (start = NULL)
        Then
        a)  Print "List is Empty !!"
    Step 3 :    Else
        a)  Set ptr = start
        b)  While(ptr ! = NULL)
            Begin
            i.   If(ptr->data = item)
                 Then
                 A)  Set flag = 1
                 B)  Exit While Loop
            ii.  Else
                 A)  Set ptr = ptr->link
                 B)  Set count = count +1
                 End If
            EndWhile
        c)  If(flag = 1)
            Then
            i.   Print “Item found at position pos”
        d)  Else
            i.   Print “Item not found”
            End If
        End If
    Step 4 :   End
```

Algorithm_Main()

```
Step 1 :   While(True)
    a)  Print "1.Display”
    b)  Print “2.Insert at Beginning”
    c)  Print “3.Insert at end”
    d)  Print “4.Insert at any position”
    e)  Print “5.Delete at Beginning”
    f)  Print “6.Delete at End”
    g)  Print “7.Delete from any position”
    h)  Print “8.Searching an Item”
    i)  Print “9.Exit"
    j)  Input choice
    k)  If(choice = 1)
        i.   Call Display()
    l)  Else If(choice = 2)
        i.   Call InsertBegin()
    m)  Else If(choice = 3)
        i.   Call InsertEnd()
    n)  Else If(choice = 4)
        i.   Call InsertPos()
    o)  Else If(choice = 5)
        i.   Call DeleteBegin()
    p)  Else If(choice = 6)
        i.   Call DeleteEnd()
    q)  Else If(choice = 7)
        i.   Call DeletePos()
```

```
r) Else If(choice = 8)
    i. Call Search()
s) Else
    i. Print “Wrong choice”
    ii. 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.