EXPERIMENT NO. 6

Author: Harsheet Chordiya

Roll No.: 37

Sem and Section: 3 CSEB

Source file: expt06.c

Date: 14/01/2021

==========

Aim: To study doubly linked linear lists [DLLs] and implement various operations on it.

Problem Statement:

Create a self-referential structure, node_dll to represent a node of a doubly linked linear list.Implement the routines to (1) create a list, (2) insert an element - at the beginning, at the end and at a specified position in the list, (3) delete an element from the front, rear, or a specified position at the list, (4) reverse the list, (5) sort the list, and (6) search the list. Create a menu-driven C program to test these routines.

==========

THEORY

==========

A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. Therefore, it consists of three parts—data, a pointer to the next node, and a pointer to the previous node. Doubly linked list or more commonly referred to as DLL is similar in node structure as that for a singly linked list except for the fact that they have an extra pointer which is used for back traversal on the list. It has two pointers namely next and previous. DLL can be considered as a variation of the singly linked list. Here one pointer (or reference) points to the next node and the other pointer points to the previous node.Thus, we see that a doubly linked list calls for more space per node and more expensive basic operations. However, a doubly linked list

provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions (forward and backward). The main advantage of using a doubly linked list is that it makes searching twice as efficient. Generally, doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion. However, we can easily manipulate the elements of the list since the list maintains pointers in both the directions (forward and backward). Every node in a doubly linked list has-

Data

Address of the next node

Address of the previous node

ALGORITHMS:

Algorithm to insert a node:

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 9

[END OF IF]

Step 2: SET = AVAIL

Step 3: SET AVAIL = AVAIL NEXT

Step 4: SET DATA = VAL

Step 5: SET PREV = NULL

Step 6: SET NEW_NODE->NEXT= START

Step 7: SET START -> PREV = NEW_NODE

Step 8: SET START =NEW_NODE

Step 9: EXIT

Algorithm to delete a node:

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 6

[END OF IF]

Step 2: SET PTR = START

Step 3: SET START = START->NEXT

Step 4: SET START->PREV = NULL

Step 5: FREE PTR

Step 6: EXIT

Algorithm to reverse a doubly link list:

1.To reverse the list we start with the first node. Say a pointer current keeps track of the current node. Now initially current points to head node.

2.Swap the previous and next pointer fields of current node.

3.Move the position of current pointer to its next node. In general, now current.prev holds the address of next node.

4.Repeat Step 2-3 until current pointer becomes NULL.

5.When, the current pointer becomes NULL then the list will look something like.

6.Finally, swap the head and last pointers and you are done.

7.Now, if you compare the above image to the below given image you will find both are similar linked list.

Algorithm to sort doubly linked list::

1.Define a node current which will point to head.

2.Define another node index which will point to node next to current.

3.Compare data of current and index node. ...

4.Current will point to current. ...

5.Continue this process till the entire list is sorted.

==========

PROGRAM

==========

```
//Header file delecrations
#include<stdio.h>
#include<stdlib.h>
struct dblist{
    int data;
    struct dblist *pre;
    struct dblist *next;

};
typedef struct dblist* list;
//Function Declaration/Signatures/Prototypes
list createNode(int key);
int search(list, int);
list insertBeg(list,int);
list insertEn(list,int);
list deleteBeg(list,int*);
list deleteEn(list,int*);
list deletePos(list, int);
list insertPos(list, int,int);
void displayList(list);
void reverse(list*);
int length(list);
list sort(list);
```

```
//Driver Functions
int main(){
    list neww, first = NULL;
    int key, choice, Pos, len;
    do
    {
        printf("\nLL...\n");
        printf("\t1. Create Newnode \t2. Insert node at Begin \t3. Insert
node at End \t4. Insert Position\n");
        printf("\t5. Delete node at Begin \t6. Delete node at End \t7.
Delete Position\t8. Diplay \n");
        printf("\t9. Length   \t10. Search");
        printf("\t11. Sort \t 12.Reverse\t0. Exit\t");
        printf("\tChoice?\t");
        scanf("%d", &choice);
        switch (choice)
        {
        case 0:
            printf("\tYou opted to exit\n");
            break;
        case 1:
            neww = createNode(key);
            printf("Node created Successfully...");
            break;
        case 2:
            printf("Enter Data :");
            scanf("%d", &key);
            first = insertBeg(first, key);
            break;
```

```
        case 3:
            printf("Enter Data :");
            scanf("%d", &key);
            first = insertEn(first, key);
            break;
        case 4:
            printf("Enter the position at which node is to be inserted:
\n");
            scanf("%d", &Pos);
            printf("Enter Data :");
            scanf("%d", &key);
            first = insertPos(first, key, Pos);
            break;
        case 5:
            first = deleteBeg(first, &key);
            printf("Node deleted Successfully...");
            break;
        case 6:
            first = deleteEn(first, &key);
            printf("\nNode deleted Successfully...\n");
            break;
        case 7:
            printf("Enter the position at which node is to be deleted:
\n");
            scanf("%d", &Pos);
            first = deletePos(first, Pos);
            printf("\nNode deleted Successfully...\n");
            break;
        case 8:
```

```
        displayList(first);
        break;
    case 9:
        len = length(first);
        printf("The length is %d", len);
        break;
    case 10:
        printf("Enter element to search.\n");
        scanf("%d", &key);
        key = searchDLL(first, key);
        if (key == -1)
        {
            printf("\nElement not found.\n");
        }
        else
        {
            printf("\nElement found at %d location.\n", key);
        }
        break;
    case 11:
        first = sort(first);
        displayList(first);
        break;
    case 12:
        reverse(&first);
        break;
    }
```

```
    } while (choice != 0);
    return 0;
}
//function defination
list createNode(int key){
    list neww;
    neww = (list)malloc(sizeof(struct dblist));
    neww->data = key;
    neww->next = NULL;
    neww->pre = NULL;
    return neww;
}
list insertBeg(list first, int key){
    list neww;
    neww = createNode(key);
    //assume that neww node will be always created..
    if(first == NULL)
        return neww;
    neww->next = first;
    first->pre = neww;
    return neww;
}
list insertEn(list first,int key){
    list neww;

    neww = createNode(key);

    if(neww == NULL){
```

```
        return neww;
    }
    list temp;
    temp = first;

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

    neww->pre = temp;
    temp->next = neww;
    return first;

}
void displayList(list first) {
   if(first==NULL){
        printf("List is empty...\n");
    }
    while(first != NULL){
        printf("-->%d[%lu] ", first->data, (unsigned long) first);
        first = first->next;
    }
    printf("\n");
   }


list deleteEn(list first,int* key){
        if(first == NULL){
```

```
            return first;
        }
        list temp = first;
        while(temp->next!=NULL){
            temp =temp->next;
        }
        if(temp->pre!=NULL){
            temp->pre->next = NULL;
        }else{
            first = NULL;
        }
        *key = temp->data;
        free(temp);
        return first;
}
list deleteBeg(list first,int* key){
    list temp;
   if(first==NULL){
        return first;
   }
   temp = first;
   first = first->next;
   if (first != NULL)
   {
        first->pre = NULL;
   }
   *key = temp->data;
   free(temp);
```

```
    return first;
}
void reverse(list* first)
{
     list temp = NULL;
     list current = *first;
     while (current !=  NULL)
     {
       temp = current->pre;
       current->pre = current->next;
       current->next = temp;
       current = current->pre;
     }
     if(temp != NULL )
        *first = temp->pre;
}
list deletePos(list first, int Pos)
{
    if (Pos > length(first) || Pos < 0)
    {
        printf("\nError deletion...Wrong Position...\n");
        return (first);
    }
    printf("\nDeletion successfull...\n");
    list ptr = first;
    list temp = first->next;
    int i = 0;
    while (i < Pos - 1)
```

```
    {
        ptr = ptr->next;
        temp = temp->next;
        i++;
    }
    ptr->next = temp->next;
    ptr->pre = temp->pre;
    free(temp);
    return (first);
}
int length(list first)
{
    int length = 0;
    if (first == NULL)
    {
        printf("\nEmpty List...\n");
        return length;
    }
    while (first != NULL)
    {
        length = length + 1;
        first = first->next;
    }
    return length;
}
list sort(list first)
{
    list ptr = first;
```

```
    list ptr1 = first->next;
    if (first == NULL)
    {
        printf("List Empty.\n");
        return (first);
    }
    printf("Sorted List : \n");
    int temp;
    int i, j, n = length(first);
    for (i = 0; i < n; i++)
    {
        for (j = 0; ptr->next != NULL; j++)
        {
            if (ptr->data > ptr1->data)
            {
                temp = ptr->data;
                ptr->data = ptr1->data;
                ptr1->data = temp;
            }
            ptr = ptr->next;
            ptr1 = ptr1->next;
        }
        ptr = first;
        ptr1 = first->next;
    }
    return (first);
    }
int searchDLL(list first, int key)
```

```
{
    list ptr = first;
    int Pos = 0;
    while (ptr != NULL)
    {
        if (ptr->data == key)
        {
            break;
        }
        ptr = ptr->next;
        Pos++;
    }
    if (Pos == length(first))
    {
        return (-1);
    }
    return (Pos + 1);
}
list insertPos(list first, int key, int Pos)
{
    list ptr = createNode(key);
    list temp;
    int i, loc;
    if (ptr == NULL)
    {
        printf("\nOVERFLOW...\n");
        printf("\nInsertion Failed...\n");
        return (first);
```

```
    }
    if (Pos > length(first) || Pos < 1)
    {
        printf("\nError deletion...Wrong Position...\n");
        return (first);
    }
    else
    {
        temp = first;
        for (i = 0; i < Pos; i++)
        {
            temp = temp->next;
            if (temp == NULL)
            {
                printf("\nInsertion Failed...\n");
                return (first);
            }
        }
        printf("\nInsertion Sucessfull...\n");
        ptr->next = temp->next;
        ptr->pre = temp;
        temp->next = ptr;
        temp->next->pre = ptr;
    }
    return (first);
}
```

==========

EXECUTION-TRAIL

==========

```
LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 1

Node created Successfully...

LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 2

Enter Data :23


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 2

Enter Data :25


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position
```

```
        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 3

Enter Data :54


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 4

Enter the position at which node is to be inserted:

3

Enter Data :75


Insertion Failed...


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 4

Enter the position at which node is to be inserted:

2

Enter Data :66


Insertion Sucessfull...
```

```
LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8

-->25[7736680] -->23[7736656] -->54[7736704] -->66[7736752]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 2

Enter Data :39


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8

-->39[7736776] -->25[7736680] -->23[7736656] -->54[7736704] --
>66[7736752]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position
```

```
        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 5

Node deleted Successfully...

LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8

-->25[7736680] -->23[7736656] -->54[7736704] -->66[7736752]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 7

Enter the position at which node is to be deleted:

3


Deletion successfull...


Node deleted Successfully...


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position
```

```
        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8

-->25[7736680] -->23[7736656] -->54[7736704]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 2

Enter Data :24


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 9

The length is 4

LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 10

Enter element to search.

25
```

```
Element found at 2 location.


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8

-->24[7736752] -->25[7736680] -->23[7736656] -->54[7736704]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 12


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 8



-->54[7736704] -->23[7736656] -->25[7736680] -->24[7736752]
```

```
LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 11

Sorted List :

-->23[7736656] -->24[7736752] -->25[7736680] -->54[7736704]


LL...

        1. Create Newnode       2. Insert node at Begin         3. Insert
node at End   4. Insert Position

        5. Delete node at Begin         6. Delete node at End   7. Delete
Position      8. Diplay

        9. Length       10. Search      11. Sort         12.Reverse
0. Exit         Choice? 0

        You opted to exit
```

==========

VIVA-VOICE

==========

1. Compare doubly linked list woth singly linked list. When would you choose to use one over other?

Ans: • Complexity-

Singly linked list : In singly linked list the complexity of insertion and deletion at a known position is O(n).

Doubly linked list : In case of doubly linked list the complexity of insertion and deletion at a known position is O(1)

•Order of elements-

Singly linked list : Singly linked list allows traversal elements only in one way.

Doubly linked list : Doubly linked list allows element two way traversal.

•Usage-

Singly linked list : Singly linked list are generally used for implementation of stacks

Doubly linked list : On other hand doubly linked list can be used to implement stacks as well as heaps and binary trees.

If you need to be able to quickly get both the previous as well as the next element from a given element then a doubly linked list is best. If you only need to get the next element from a given element, then a singly linked list is good enough . Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored. If we need better performance while searching and memory is not a limitation in this case doubly linked list is more preferred.

2. Is it possible with a doubly linked list to leave backward links broken (at times). Write an algorithm of C function to detect such dangling links in doubly linked lists.

Ans: Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Function dangling Links (FIRST) Given a DLL pointer FIRST referencing the first node of the list, this function detects the dangling links present in the code. TEMP is a local dlist variable.

1. Is the list empty?? If FIRST = NULL RETURN NULL

2. Initialize pointers TEMP = FIRST

3. Traverse remainder of the list & reassign pointers Repeat While NEXT (FIRST) <> NULL FIRST = NEXT (FIRST) IF (PREV(FIRST) <> TEMP) Write('Dangling Link detected') Return TEMP. TEMP = NEXT (TEMP)

3. What advantages you will gain by converting doubly linked list to circular doubly linked list? Comment.

Ans: Circular linked list over doubly linked list:

No requirement for a NULL assignment in the code. The circular list never points to a NULL pointer unless fully deallocated. Circular linked lists are advantageous for end operations since beginning and end coincide. Circular linked list also performs all regular functions of a singly linked list. In fact, circular doubly linked lists discussed below can even eliminate the need for a full-length traversal to locate an element. That element would at most only be exactly opposite to the start, completing just half the linked list.

Disadvantages Of DLL:

It uses extra memory when compared to the array and singly linked list. Since elements in memory are stored randomly, therefore the elements are accessed sequentially no direct access is allowed.

==========

CONCLUSION

==========

In this experiment we have studied what is the doubly linked list and its various operations. We have also learnt how to write the algorithms for operations of doubly linked list. Sir has explained us in detail the use of doubly linked list over singly linked list i.e. where to use use doubly linked list and where to use doubly linked list. Doubly linked list is all about node. So, sir also explained us node insertion , deletion in all cases.