**NAME:** AGNIV PRAMANICK **SECTION:** A **USN:** 1NT21IS017 **DATE:** 27/02/23

**Q. Write a program to implement doubly linked list using c program.**

**THEORY**

A node in a doubly linked list carries a pointer to both the preceding and following node in the sequence, making it a more complex sort of linked list. Hence, a node in a doubly linked list has three components: node data, a pointer to the node after it in the sequence (the next pointer), and a pointer to the node before it (previous pointer).

Both the previous and subsequent parts of the first and last nodes will always be null, signifying an end in both directions.

Because each node in a singly linked list has the address of the next node and lacks a record of its previous nodes, we can only traverse the list in one direction. Nevertheless, doubly linked lists get over this single linked list restriction. Each node in the list has the address of the node that came before it, therefore using the prior address stored in the previous part of each node, we can also obtain all the information about the previous node.

//////////////////////////////////////////////////////////////////////////////////////

**ALGORITHM**

**Insert at beginning**

* **Step 1:** if ptr=NULL, write overflow. Go to step 9. [END OF IF]
* **Step 2:** SET NEW\_NODE = ptr
* **Step 3:** SET ptr = ptr -> NEXT
* **Step 4:** SET NEW\_NODE -> DATA = VAL
* **Step 5:** SET NEW\_NODE -> PREV = NULL
* **Step 6:** SET NEW\_NODE -> NEXT = START
* **Step 7:** SET head -> PREV = NEW\_NODE
* **Step 8:** SET head = NEW\_NODE
* **Step 9:** EXIT

**Insert at end**

* **Step 1:** IF PTR = NULL, write overflow and go to step 11. [end of if]
* **Step 2:** SET NEW\_NODE = PTR
* **Step 3:** SET PTR = PTR -> NEXT
* **Step 4:** SET NEW\_NODE -> DATA = VAL
* **Step 5:** SET NEW\_NODE -> NEXT = NULL
* **Step 6:** SET TEMP = START
* **Step 7:** Repeat Step 8 while TEMP -> NEXT! = NULL
* **Step 8:** SET TEMP = TEMP -> NEXT

[END OF LOOP]

* **Step 9:** SET TEMP -> NEXT = NEW\_NODE
* **Step 10C:** SET NEW\_NODE -> PREV = TEMP
* **Step 11:** EXIT

**Insert random**

* **Step 1:** IF PTR = NULL write overflow and go to step 15. [end of if]
* **Step 2:** SET NEW\_NODE = PTR
* **Step 3:** SET PTR = PTR -> NEXT
* **Step 4:** SET NEW\_NODE -> DATA = VAL
* **Step 5:** SET TEMP = START
* **Step 6:** SET I = 0
* **Step 7:** REPEAT 8 to 10 until I<="" li="">
* **Step 8:** SET TEMP = TEMP -> NEXT
* **STEP 9:** IF TEMP = NULL
* **STEP 10:** WRITE "LESS THAN DESIRED NO. OF ELEMENTS", step 15 end of if.
* **Step 11:** SET NEW\_NODE -> NEXT = TEMP -> NEXT
* **Step 12:** SET NEW\_NODE -> PREV = TEMP
* **Step 13:** SET TEMP -> NEXT = NEW\_NODE
* **Step 14:** SET TEMP -> NEXT -> PREV = NEW\_NODE
* **Step 15:** EXIT

**Delete at beginning**

* **STEP 1:** IF HEAD = NULL underflow step 6.
* **STEP 2:** SET PTR = HEAD
* **STEP 3:** SET HEAD = HEAD → NEXT
* **STEP 4:** SET HEAD → PREV = NULL
* **STEP 5:** FREE PTR
* **STEP 6:** EXIT

**Delete at last**

* **Step 1:** IF HEAD = NULL, write underflow and go to step 7. [end of if]
* **Step 2:** SET TEMP = HEAD
* **Step 3:** REPEAT STEP 4 WHILE TEMP->NEXT! = NULL
* **Step 4:** SET TEMP = TEMP->NEXT

[END OF LOOP]

* **Step 5:** SET TEMP ->PREV-> NEXT = NULL
* **Step 6:** FREE TEMP
* **Step 7:** EXIT

**Delete random**

* **Step 1:** IF HEAD = NULL write underflow and go to step 9. [end of if]
* **Step 2:** SET TEMP = HEAD
* **Step 3:** Repeat Step 4 while TEMP -> DATA! = ITEM
* **Step 4:** SET TEMP = TEMP -> NEXT

   [END OF LOOP]

* **Step 5:** SET PTR = TEMP -> NEXT
* **Step 6:** SET TEMP -> NEXT = PTR -> NEXT
* **Step 7:** SET PTR -> NEXT -> PREV = TEMP
* **Step 8:** FREE PTR
* **Step 9:** EXIT

**Search**

* **Step 1:** IF HEAD == NULL write underflow and go to step 8. [end of if]
* **Step 2:** Set PTR = HEAD
* **Step 3:** Set i = 0
* **Step 4:** Repeat step 5 to 7 while PTR! = NULL
* **Step 5:** IF PTR → data = item

return i  
[END OF IF]

* **Step 6:** i = i + 1
* **Step 7:** PTR = PTR → next
* **Step 8:** Exit

**////////////////////////////////////////////////////////////////////////////////////**

**CODE**

#include<stdio.h>   //library functions activated

#include<stdlib.h>

struct node   // node structure given

{

    struct node \*prev;

    struct node \*next;

    int data;

};

struct node \*head;

//user defined functions

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main ()

{

int choice =0;

    while(choice != 9)

    {

        printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

        printf("\nChoose one option from the following list ...\n");

        printf("\n===============================================\n");

        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

        printf("\nEnter your choice?\n");

        scanf("\n%d",&choice);

        switch(choice)

        {

            case 1:

            insertion\_beginning();

            break;

            case 2:

                    insertion\_last();

            break;

            case 3:

            insertion\_specified();

            break;

            case 4:

            deletion\_beginning();

            break;

            case 5:

            deletion\_last();

            break;

            case 6:

            deletion\_specified();

            break;

            case 7:

            search();

            break;

            case 8:

            display();

            break;

            case 9:

            exit(0);

            break;

            default:

            printf("Please enter valid choice..");

        }

    }

}

// to insert at the beginning

void insertion\_beginning()

{

   struct node \*ptr;

   int item;

   ptr = (struct node \*)malloc(sizeof(struct node));  //memory allocate

   if(ptr == NULL)

   {

       printf("\nOVERFLOW");  //overflow as pt is null

   }

   else

   {

    printf("\nEnter Item value");

    scanf("%d",&item);   //item value stored

   if(head==NULL)  //for head equal to null

   {

       ptr->next = NULL;

       ptr->prev=NULL;

       ptr->data=item;

       head=ptr;

   }

   else

   {

       ptr->data=item;

       ptr->prev=NULL;

       ptr->next = head;

       head->prev=ptr;

       head=ptr;

   }

   printf("\nNode inserted\n");  ///node inserted

}

}

void insertion\_last()   //insert at last

{

   struct node \*ptr,\*temp;

   int item;

   ptr = (struct node \*) malloc(sizeof(struct node));  //size allocated

   if(ptr == NULL)

   {

       printf("\nOVERFLOW");

   }

   else

   {

       printf("\nEnter value");

       scanf("%d",&item);   //item stored

        ptr->data=item;

       if(head == NULL)

       {

           ptr->next = NULL;

           ptr->prev = NULL;

           head = ptr;

       }

       else

       {

          temp = head;   //temp is equal to head

          while(temp->next!=NULL)

          {

              temp = temp->next;

          }

          temp->next = ptr;

          ptr ->prev=temp;

          ptr->next = NULL;

          }

       }

     printf("\nnode inserted\n");

    }

void insertion\_specified()  //insert specifically

{

   struct node \*ptr,\*temp;

   int item,loc,i;

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

   if(ptr == NULL)

   {

       printf("\n OVERFLOW");   //overflow statement

   }

   else

   {

       temp=head;

       printf("Enter the location");  //location to be given

       scanf("%d",&loc);

       for(i=0;i<loc;i++)

       {

           temp = temp->next;

           if(temp == NULL)

           {

               printf("\n There are less than %d elements", loc);

               return;

           }

       }

       printf("Enter value");

       scanf("%d",&item);   //item value stored

       ptr->data = item;

       ptr->next = temp->next;

       ptr -> prev = temp;

       temp->next = ptr;

       temp->next->prev=ptr;

       printf("\nnode inserted\n");

   }

}

void deletion\_beginning()   //delete from beginning

{

    struct node \*ptr;

    if(head == NULL)

    {

        printf("\n UNDERFLOW");   //underflow statement

    }

    else if(head->next == NULL)

    {

        head = NULL;

        free(head);

        printf("\nnode deleted\n");   //node deleted

    }

    else

    {

        ptr = head;

        head = head -> next;

        head -> prev = NULL;

        free(ptr);

        printf("\nnode deleted\n");

    }

}

void deletion\_last()   //delete from last

{

    struct node \*ptr;

    if(head == NULL)

    {

        printf("\n UNDERFLOW");   //underflow statement

    }

    else if(head->next == NULL)

    {

        head = NULL;

        free(head);    //head is free

        printf("\nnode deleted\n");

    }

    else

    {

        ptr = head;

        if(ptr->next != NULL)

        {

            ptr = ptr -> next;

        }

        ptr -> prev -> next = NULL;

        free(ptr);

        printf("\nnode deleted\n");

    }

}

void deletion\_specified()   //delete random

{

    struct node \*ptr, \*temp;

    int val;

    printf("\n Enter the data after which the node is to be deleted : ");

    scanf("%d", &val);   //val is stored

    ptr = head;

    while(ptr -> data != val)

    ptr = ptr -> next;

    if(ptr -> next == NULL)

    {

        printf("\nCan't delete\n");   //print delete statement

    }

    else if(ptr -> next -> next == NULL)

    {

        ptr ->next = NULL;

    }

    else

    {

        temp = ptr -> next;

        ptr -> next = temp -> next;

        temp -> next -> prev = ptr;

        free(temp);

        printf("\nnode deleted\n");

    }

}

void display()   //display function

{

    struct node \*ptr;

    printf("\n printing values...\n");   //values printed

    ptr = head;

    while(ptr != NULL)

    {

        printf("%d\n",ptr->data);

        ptr=ptr->next;

    }

}

void search()   //to search elements

{

    struct node \*ptr;

    int item,i=0,flag;

    ptr = head;

    if(ptr == NULL)

    {

        printf("\nEmpty List\n");   //empty list

    }

    else

    {

        printf("\nEnter item which you want to search?\n");

        scanf("%d",&item);   //item is stored

        while (ptr!=NULL)

        {

            if(ptr->data == item)

            {

                printf("\nitem found at location %d ",i+1);

                flag=0;

                break;

            }

            else

            {

                flag=1;

            }

            i++;

            ptr = ptr -> next;

        }

        if(flag==1)

        {

            printf("\nItem not found\n");   //item not found if flag is 1

        }

    }

}

////////////////////////////////////////////////////////////////////////////////////////////////

**OUTPUT**























