DOUBLE LINKED LIST

AIM: THE AIM OF THE PROGRAM IS TO IMPLEMENT THE CONCEPTS ;INSERT,DELETE,FIND AND DSPLAY ELEMENTS IN A DOUBLE LINKED LIST

ALGORITHM: Step-by-Step Algorithm

```
Step 1: Define the Structure for the Doubly Linked List Node
Define a struct node that contains:
int data for storing the value.
struct node *next for pointing to the next node.
struct node *prev for pointing to the previous node.
Step 2: Initialize Global Variables
Initialize the head pointer struct node *l = NULL.
Initialize a temporary node pointer struct node *newnode.
Step 3: Insert at the Beginning (insert_first)
Allocate memory for a new node.
Check if the allocation is successful.
Get the data for the new node from the user.
If the list is not empty:
Set newnode->next to point to the current head.
Set newnode->prev to NULL.
Update the current head's previous pointer to newnode.
Update the head pointer to newnode.
If the list is empty:
Set newnode->next and newnode->prev to NULL.
Update the head pointer to newnode.
Step 4: Insert at the End (insert_last)
Allocate memory for a new node.
Check if the allocation is successful.
Get the data for the new node from the user.
Set newnode->next to NULL.
If the list is not empty:
Traverse to the last node.
Set newnode->prev to point to the last node.
Update the last node's next pointer to newnode.
If the list is empty:
Set newnode->prev to NULL.
Update the head pointer to newnode.
Step 5: Insert After a Given Position (insert_afterp)
Allocate memory for a new node.
Check if the allocation is successful.
```

Get the data for the new node from the user.
Get the position after which to insert the new node from the user.
Traverse to the node just before the given position.
Update pointers to insert the new node after the given position:
Set newnode->next to point to the next node.
Set newnode->prev to point to the current node.
Update the next node's previous pointer to newnode.
Update the current node's next pointer to newnode.
Step 6: Delete the First Node (delete_first)
If the list is not empty:
Set a temporary pointer to the current head.
Update the head pointer to the next node.
Set the new head's previous pointer to NULL.
Free the memory of the temporary node.
Step 7: Delete the Last Node (delete_last)
If the list is not empty:
Traverse to the second last node.
Set a temporary pointer to the last node.
Update the second last node's next pointer to NULL.
Free the memory of the temporary node.
Step 8: Delete After a Given Position (delete_afterp)
Get the position after which to delete the node from the user.
Traverse to the node just before the given position.
Set a temporary pointer to the node to be deleted.
If the node to be deleted is not the last node:
Update pointers to remove the node:
Set the current node's next pointer to the node after the node to be deleted.
Update the next node's previous pointer to the current node.
Free the memory of the temporary node.
If the node to be deleted is the last node:
Update the current node's next pointer to NULL.
Free the memory of the temporary node.
Step 9: Find Element at a Given Position (find)
Get the position from the user.
Traverse to the node at the given position.
Print the data of the node at the given position.
Step 10: Find Next Element (find_next)
Get the position from the user.
Traverse to the node at the given position.
Print the data of the node next to the given position.
Step 11: Find Previous Element (find_previous)
Get the position from the user.
Traverse to the node at the given position.
Print the data of the node previous to the given position.

Step 12: Display the List (display)
If the list is not empty:
Traverse through the list from the head to the last node.
Print the data of each node.
If the list is empty, print "List is empty".
Step 13: Main Function (main)
Display the menu options.
Repeatedly:
Get the user's choice.
Call the appropriate function based on the user's choice.
Exit the loop if the user selects the exit option.
PROGRAM:

```
#include <stdio.h>
#include<stdlib.h>

struct node {
   int data;
   struct node *next;
   struct node *prev;
 };
struct node *l=NULL;
struct node *newnode;

void insert_first(){
   newnode=(struct node*)malloc(sizeof(struct node));
   if(newnode!=NULL){
      printf("Enter the element : ");
      scanf("%d",&newnode->data);
      if (l!=NULL){
      newnode->next=l;
      newnode->prev=NULL;
      l->prev=newnode;
      l=newnode;
      }
      else
      {
         newnode->next=NULL;
         newnode->prev=NULL;
         l=newnode;
      }

   }

}
```

```
void insert_last(){
    newnode=(struct node*)malloc(sizeof(struct node));
    struct node*p;
    if(newnode!=NULL){
        printf("enter no");
        scanf("%d",&newnode->data);
        newnode->next=NULL;
        if (l!=NULL){
            p=l;
            while(p->next!=NULL){
                p=p->next;
            }
            newnode->prev=p;
            p->next=newnode;

        }
        else
        {
            newnode->prev=NULL;
            l=newnode;
        }
    }

}

void insert_afterp(){
    newnode=(struct node*)malloc(sizeof(struct node));
    struct node *p=l;
    int pos,c=1;
    if(newnode!=NULL){
        printf("Enter the element : ");
        scanf("%d",&newnode->data);
        printf("enter the positon");
        scanf("%d",&pos);
        while(c<pos-1){
            p=p->next;
            c++;
        }
        newnode->next=p->next;
        newnode->prev=p;
        p->next=newnode;
        p->next->prev=newnode;
    }
```

```
}

void delete_first(){
    struct node*p=l;
    l=l->next;
    l->prev=NULL;
    free(p);
}

void delete_last(){
    struct node*p=l;
    struct node*temp;
    while(p->next->next!=NULL){
        p=p->next;
    }
    temp=p->next;
    p->next=NULL;
    free(temp);

}

void delete_afterp(){
    struct node*p=l;
    struct node *temp;
    int pos,c=1;
    printf("Enter position : ");
    scanf("%d",&pos);
    while(c<pos){
        p=p->next;
        c++;
    }
    temp=p->next;
    if (temp->next==NULL){
        free(temp);
        p->next=NULL;
    }
    else{
        p->next=temp->next;
        temp->next->prev=p;
        free(temp);
    }
}

void find(){
    struct node *p=l;
    int pos,c=1;
    printf("enter position ");
    scanf("%d",&pos);
    while(c<pos){
        p=p->next;
        c++;
    }
    printf("The element present at the position %d is %d",pos,p->data);
}

void find_next(){
    struct node *p=l;
    int pos,c=1;
    printf("Enter the position : ");
    scanf("%d",&pos);
    while(c<pos){
        p=p->next;
        c++;
    }
    printf("The element present next to the position %d is %d",pos,p->next->data);
}

void find_previous(){
    struct node *p=l;
    int pos,c=1;
    printf("enter position ");
    scanf("%d",&pos);
    while(c<pos){
        p=p->next;
        c++;
    }
    printf("The element present previous to the position %d is %d",pos,p->prev->data);
}


void display(){
    if (l!=NULL){
        struct node*p;
        p=l;
        while(p!=NULL){
            printf("%d",p->data);
            p=p->next;
        }
    }
    else
        printf("List is empty");
}



void main(){
    int n;
    struct node *l=NULL;
    printf("options\n1.enter at first\n2.enter at last\n3.insert after p\n4.delete first
element\n5.delete last\n6.delete after p\n7.find\n8.find next\n9.find
previous\n10.display\n11.exit\n");
    do{
        printf("\nEnter your option:");
        scanf("%d",&n);
        switch(n){
            case 1:
                insert_first();
                break;
            case 2:
                insert_last();
                break;
            case 3:
                insert_afterp();
                break;
            case 4:
                delete_first();
                break;
            case 5:
                delete_last();
                break;
            case 6:
                delete_afterp();
                break;
            case 7:
                find();
                break;
            case 8:
                find_next();
                break;
            case 9:
                find_previous();
                break;
            case 10:
                display();
                break;
        }
    }while(n!=11);
}
```

OUTPUT:

```
options
1.enter at first
2.enter at last
3.insert after p
4.delete first element
5.delete last
6.delete after p
7.find
8.find next
9.find previous
10.display
11.exit

Enter your option:1
Enter the element : 10

Enter your option:2
enter no20

Enter your option:10
10 20
Enter your option:4

Enter your option:10
20
Enter your option:1
Enter the element : 15

Enter your option:7
enter position 1
The element present at the position 1 is 15
Enter your option:6
Enter position : 1

Enter your option:10
15
Enter your option:2
```

RESULT: THE PROGRAM IMPLEMENTS THE CONCEPTS ;INSERT,DELETE,FIND AND DSPLAY ELEMENTS IN A DOUBLE LINKED LIST SUCCESSFULLY