Problem Statement:- C program to perform all operations of a single linked list.

SOURCE CODE :-

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    int data;
    struct node *next;
}st;
st *head = NULL;
void insert_at_beg()
  {
      st *temp;
      temp = (st*) malloc(sizeof(st));
      printf("\t Node data:");
      scanf("%d", &temp->data);
      if(head==NULL)
       {
           temp->next = NULL;
           head= temp;
       }
      else
        {
           temp->next= head;
           head = temp;
        }
  }
void insert_at_end()
 {
      st *temp, *cur;
      temp = (st*) malloc(sizeof(st));
      Printf("Node data:");
      Scanf("%d", &temp->data);
```

```
if (head == NULL)
 { printf ("Empty");
   }
else
  {
     curr = head;
     while (cur != NULL)
      {
         cur = cur -> next;
       }
     cur -> next = temp;
      for temp -> next = NULL;
   }
}
void insert_after_a_node()
{
    st *temp, *curr;
    int a;
    temp = (st *) malloc (sizeof (st));
    printf ("Node data : ");
    scanf ("%d", & temp -> data);
    printf ("The node after which insertion to be done : ");
     scanf ("%d", a);
     if (head == NULL)
     {
         printf ("Empty");
     }
   else
    { curr = head;
      while (cur -> next != NULL)
      {
          if (curr -> data == a)
            { break; }
```

```
        curr = curr -> next;
      }
    temp -> next = curr -> next;
    curr -> next = temp;
  }
 }
void insert_before_a_node ( )
{
    st *temp, *curr, *prev;
    int a;
    temp = (st*) malloc (sizeof(st));
    Printf ("Node data:");
    scanf ("%d", &temp->data);
    Printf ("The node before which insertion is to be
             done:");
    scanf ("%d", &a);
    if(head == NULL)
    {
       Printf ("Empty");
    }
else
 {
    curr = head; prev = head;
    while (curr -> next != NULL)
     { if(curr -> data == a)
       { break;
       }
       curr = curr -> next;
     }
  temp -> next = curr;
prev -> next = temp;
if (prev == curr)
```

```
    {
        temp -> next = head;
        head = temp;
    }
  }
 }
}
void insert_in_between()
{
    st *temp, *curr, *prev;
    int a, b;
    temp = (st *) malloc (sizeof (st));
    Printf ("Node data:");
    scanf ("%d", &temp -> data);
    Printf ("The node before which insertion is to be done:");
    scanf ("%d", &a);
    Printf ("The node after which insertion is to be
             done:");
     scanf ("%d", &b);
    if (head == NULL)
    {
       Printf ("Empty");
    }
else
{
    curr = head; prev = head;
    while (curr != NULL)
    {
        if (prev -> data == a && cur -> data == b)
            break;
        prev = curr;
    } cur = curr -> next;
    if (curr != prev->next)
    { prev -> next = temp;
```

```
    temp-> next = curr;
    }
     else
      printf("Operation not possible");
    }
  }
Void count()
  {
      int c =0;
      St *curr;
      if (head == NULL)
     {  printf("Empty");
        }
      else
        {
          curr = head;
          while (cur != NULL)
           {
              cur = curr -> next;
              c++;
            }
        }
      printf("The no. of nodes are %d", c);
    }
Void search()
  {
      int a;
      Printf("The node to be searched:");
      scanf("%d", &a);
      St *curr;
      if (head == NULL)
      { printf("Empty");
        }
```

```
else
  {   curr= head;
      while(curr->next!=NULL)
      {   if(curr->data==a)
          break;
         Printf("Node found");
         curr= curr->next;
      }
   }
}
void delete_at_beg()
 {
     st *temp, *curr;
     if(head==NULL)
     {
          Printf("Empty");
      }
      else
         {   curr=head;
             temp=head;
             head= curr->next;
             free(temp);
             Printf("Deleted Node");
     }
  }
void delete-at-end()
  {
       st *curr, *prev;
        if(head==NULL)
        {  printf("\nEmpty");
          return;
        }
     else
      { curr=head; prev=head;
```

```
while(curr ->next != NULL)
  {
     prev = curr;
     curr = curr->next;
  }
    free(curr);
     prev -> next = NULL;
     printf("Node deleted");
}
void delete_before_a_node()
{
    st *curr, *prev, *next;
    int a;
    printf("The node after before which deletion
            is to be done :");
    scanf("%d", &a);
      if(head == NULL)
      {
          printf("\n Empty");
        return;
      }
    else
     {
       curr = head; prev = head; prev1 = head;
       while (curr -> next != NULL)
        {
          ff(curr ->data =a)
            { break;
            }
          prev1 = prev;
          prev = curr;
          curr = curr -> next;
        }
       prev1 -> next = curr;
        free(prev);
       Printf("Deleted Node").
     }
}
```

```
void sort()
{
    st *ptr1, *ptr2;
    int t;
    ptr1 = head;
    while(ptr1->next != NULL)
    {
        ptr2 = ptr1->next;
        while(ptr2 != NULL)
        {
            if(ptr1->data > ptr2->dat)
            {
                t = ptr1->data;
                ptr1->data = ptr2->data;
                ptr2->data = t;
            }
            ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
void display()
{
    st *curr;
    if(head == NULL)
    {   printf("\n Empty");
        return;
    }
    else
    {
        curr = head;
        printf("\n List elements:");
        while(curr != NULL)
        {   printf("%d \t", curr->data)
            curr = curr->next;
        }
    }
}
```

```
int main()
{
  int x, choice;
  while(1)
  {
    printf("\n Single Linked List Operations:");
    printf("\n1. display \t2. Insert at begin \t3. Insert
            at end \t4. Insert after \t5. Insert before
            \t6. In between \t7. Delete at begin
            \t 8. Search \t 9. Count \t 10. Delete at
            end \t 11. Delete before \t 12. Sort
            \t 13. Exit");
    printf("\n Enter your choice:");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1: display();
              break;
      case 2: insert_at_beginning();
              break;
      case 3: insert_at_end()
              break;
      case 4: insert_after_a_node();
              break;
      case 5: insert_before_a node();
              break;
      case 6: insert_in_between();
              break;
      case 7: delete_at_beg();
              break;
      case 8: search();
              break;
      case 9: count();
              break;
      case 10: delete_at_end();
               break;
```

```
default : printf("Wrong Choice");
}
}
return 0;
}
```

OUTPUT :-

```
Single linked list operations
1.display       2.insert at begin       3.insert at end ,4insert after  5.insert before 6.inbetween     7.delete at begin       8.search
        9.count 10.delete at end        11.delete before        12.sort 13.Exit
Enter your choice2
Node data10

Single linked list operations
1.display       2.insert at begin       3.insert at end ,4insert after  5.insert before 6.inbetween     7.delete at begin       8.search
        9.count 10.delete at end        11.delete before        12.sort 13.Exit
Enter your choice3
Node data12

Single linked list operations
1.display       2.insert at begin       3.insert at end ,4insert after  5.insert before 6.inbetween     7.delete at begin       8.search
        9.count 10.delete at end        11.delete before        12.sort 13.Exit
Enter your choice1

List elements10         12
```

PROBLEM STATEMENT:- C Program to perform all ~~Der~~ operations of circular linked list.

SOURCE CODE :-

```
#include <stdio.h>
Struct node
{
    Struct node *next;
    int idata;
    Struct node *prev;
};
Struct node *start = NULL;
Void idisplay();
Void insert_beg();
Void intert_end();
Void insert_before();
Void insert_after();
Void idelete_beg();
Void idelete_after();
Void idelete_end();
Void idelete_before();
int main()
{
    int opt;
    while(1){
    printf("1. Display");
    printf("2. Add at beg");
    printf("3. Add at end");
    printf("4. Add after a node");
    printf("5. Add before a node");
    printf("6. ~~Ad~~ Delete at beg");
    printf("7. Delete at end");
    printf("8. Delete after a node");
    printf("9. Delete before a node");
```

```
scanf("%d", &option);
switch(opt)
{
    case 1: display();
            break;
    case 2: insert_beg();
            break;
    case 3: insert_end();
            break;
    case 4: insert_after(); break;
    case 5: insert_before();
            break
    case 6: delete_beg();
            break;
    case 7: delete_end();
            break;
    case 8: delete_after();
            break;
    case 9: delete_before();
            break;
    case 10: exit(0);
            break;
    default: printf("\n Invalid option");
    }
}
return 0;
void display()
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        printf("\t%d", ptr->data);
        ptr = ptr->next;
}  }
```

```
void insert_beg()
{
    struct node *new_node;
    int num;
    printf("\n Enter the data:");
    scanf("%d", &num);
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node -> data = num;
    start -> prev = new_node;
    new_node -> next = start;
    new_node -> prev = NULL;
    start = new_node;
}
void insert_end()
{
    struct node *ptr, *new_node;
    int num
    printf("\n Enter the data:");
    scanf("%d", &num);
    new_node = (struct node*) malloc(sizeof(struct node));
    new_node -> data = num;
    ptr = start;
    while (ptr -> next != NULL)
        ptr = ptr -> next;
    ptr -> next = new_node;
    new_node -> prev = ptr;
    new_node -> next = NULL;
}
```

```
void insert_before()
{
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data:");
    scanf("%d", &num);
    printf("\n Enter the value before which data has to
            inserted:");
    scanf("%d", &val);
    new_node = (struct node *) malloc(sizeof(struct
                                        node));
    new_node->data = num;
    ptr = start;
    while(ptr->data != val)
        ptr = ptr->next;
    new_node->next = ptr;
    new_node->prev = ptr->prev;
    ptr->prev->next = new node;
    ptr->prev = new_node;
}

void insert_after()
{
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data:");
    scanf("%d", &num);
    printf("\n Enter the value after which the data is to
            be inserted:");
    scanf("%d", &val);
    new_node = (struct node) malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
```

```
while (ptr->data != val)
        ptr = ptr->next;
new_node->prev = ptr;
new_node->next = ptr->next;
ptr->next->prev = new_node;
ptr->next = new_node;
}
void idelete_beg()
{
    struct node *ptr;
    ptr = start;
    start = start->next;
    start->prev = NULL;
    free(ptr);
}
void idelete_end()
{
    struct node *ptr;
    ptr = start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->prev->next = NULL;
    free(ptr);
}
void idelete_after()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the value after which node has
            to be deleted:");
```

```
scanf("%d", &val);
ptr = start;
while(ptr -> data != val)
        ptr = ptr -> next;
temp = ptr -> next;
ptr -> next = temp -> next;
temp -> next -> prev = ptr;
free(temp);
}

void idelete_after()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the value before which node has to
            be deleted :");
    scanf("%d", &val);
    ptr = start;
    while(ptr -> data != val)
        ptr = ptr -> next;
    temp = ptr -> prev;
    if(temp == start)
      start = delete_beg(start);
    else
      {
        ptr -> prev = temp -> prev;
        temp -> prev -> next = ptr;
      }
    free(temp);
}
```

```
void delete_end ( )
{
    struct node * ptr;
    ptr = start;
    while (ptr -> next != start)
        ptr = ptr -> next;
    ptr -> prev -> next = start;
    start -> prev = ptr -> next;
    free (ptr);
}
```

OUTPUT :-

PROBLEM STATEMENT 10:- C program to perform all operations of Circular double linked list.

SOURCE CODE :-

```
#include <stdio.h>
struct node
{
    struct node *next;
    int data;
    struct node *prev;
};
struct node *start = NULL
Struct n  void display();
          void insert_beg();
          void insert_end();
          void delete_beg();
          void delete_end;
int main()
{
    int opt;
    while(1)
    {
        Pf("1 Display");
        printf("2. Insert at beginning");
        printf("3. Insert at end");
        printf("4. Delete at beginning");
        printf("5. Delete at end");
        printf("6. Exit");
        Scanf("%d", &opt);
        switch(opt)
        {
            case 1: display();
                    break;
```

```
case 2: insert_beg();
        break;
case 3: insert_end();
        break;
case 4: delete_beg();
        break;
case 5: delete_end();
        break;
case 6: Exit(0);
        break;
default: printf("Wrong choice");
}
}
return 0;
}

void display
{
    struct node *ptr;
    ptr = start;
    while(ptr->next != start)
    {
        printf("\t %d", ptr->data);
        ptr = ptr->next;
    }
    printf("\t %d", ptr->data);
}

void insert_beg()
{
    struct node *new_node, *ptr;
    int num;
    pf("\n Enter the data:");
    scanf("%d", &num);
    new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
```

```
while (ptr->next != start)
        ptr = ptr->next;
new_node->prev = ptr;
ptr->next = new_node;
new_node->next = start;
start->prev = new_node;
start = new_node;
}

void insert_end()
{
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data:");
    scanf("%d", &num);
    new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = new_node;
    new_node->prev = ptr;
    new_node->next = start;
    start->prev = new_node;
}

void delete_beg()
{
    struct node *ptr;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = start->next;
    temp = start;
    start = start->next;
    start->prev = ptr;
    free(temp);
}
```