

Name: KUSUM.M.R

09/12/2020

USN: IBM19CS077

### Lab Program - 9

> WAP to implement doubly link list with primitive operations

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

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

```
struct node
```

```
{
```

```
    int info;
```

```
    struct node *rlink;
```

```
    struct node *llink;
```

```
};
```

```
typedef struct node *NODE;
```

```
NODE getnode()
```

```
{
```

```
    NODE x;
```

```
    x = (NODE) malloc (sizeof (struct node));
```

```
    if (x == NULL)
```

```
    { printf("Memory full\n");
```

```
        exit (0);
```

```
    }
```

```
    return x;
```

```
}
```

```
void freenode (NODE x)
```

```
{ free(x);
```

```
}
```

```
NODE insert-front (int item, NODE head)
```

```
{ NODE temp, cur;
```

```
temp = getnode();
```

```
temp->info = item;
```

```
temp->llink = NULL;
```

```
temp->rlink = NULL;
```

```
cur = head->rlink;
```

```
head->rlink = temp;
```

```
temp->llink = head;
```

```
temp->rlink = cur;
```

```
cur->llink = temp;
```

```
return head;
```

```
}
```

```
NODE insert-rear (int item, NODE head)
```

```
{ NODE temp, cur;
```

```
temp = getnode();
```

```
temp->info = item;
```

```
temp->llink = NULL;
```

```
temp->rlink = NULL;
```

```
cur = head->llink;
```

```
head->llink = temp;
```

```
temp->rlink = head;
```

```
cur->rlink = temp;
```

```
temp->llink = cur;
```

```
return head;
```

```
}
```



NODE ddelete-front (NODE head)

{

NODE cur, next;

if (head → rlink == head)

{ printf ("List is empty \n");  
return head;

}

cur = head → rlink;

next = cur → rlink;

head → rlink = next;

next → llink = head;

printf ("Item deleted at the front end is: %d \n",  
cur → info);

free (cur);

return head;

}

NODE ddelete-rear (NODE head)

{

NODE cur, prev;

if (head → rlink == head)

{ printf ("List is empty \n");  
return head;

}

cur = head → llink;

prev = cur → llink;

prev → rlink = head;

head → llink = prev;

printf ("Item deleted at the rear end is %d \n",  
cur → info);

free (cur);

return head;

}

```
void display (NODE head)
```

```
{  
    NODE temp;  
    if (head → rlink == head)  
    {  
        printf("List is empty\n");  
    }  
    printf("The contents of the list are :\n");  
    temp = head → rlink;  
    while (temp != head)  
    {  
        printf("%d\n", temp → info);  
        temp = temp → rlink;  
    }  
}
```

```
void search (int key, NODE head)
```

```
{  
    NODE cur;  
    int count;  
    if (head → rlink == head)  
    {  
        printf("List is empty\n");  
    }  
    cur = head → rlink;  
    count = 1;  
    while (cur != head && cur → info != key)  
    {  
        cur = cur → rlink;  
        count++;  
    }  
    if (cur == head)  
    {  
        printf("Search unsuccessful\n");  
    }  
    else  
    {  
        printf("Key element found at the position %d\n", count);  
    }  
}
```



NODE insert-leftpos(int item, NODE head)

```
{  
    NODE cur, prev, temp;  
    if (head → rlink == head)  
    {  
        printf("List is empty\n");  
        return head;  
    }
```

```
    cur = head → rlink;  
    while (cur != head)
```

```
{  
    if (cur → info == item)  
    {  
        break;  
    }  
    cur = cur → rlink;
```

```
}
```

```
if (cur == head)
```

```
{  
    printf("No such item found in the list\n");  
    return head;
```

```
}
```

```
prev = cur → link;
```

```
temp = getnode();
```

```
temp → link = NULL;
```

```
temp → rlink = NULL;
```

```
printf("Enter the item to be inserted at the left of  
the given item:\n");
```

```
scanf("%d", &temp → info);
```

```
prev → rlink = temp;
```

```
temp → link = prev;
```

```
temp → rlink = cur;
```

```
cur → link = temp;
```

```
return head;
```

```
}
```

NODE insert\_rightpos(int item, NODE head)

```
{ NODE temp, cur, next;  
  if (head → rlink == head)  
  {  
    printf("List is empty \n");  
    return head;  
  }
```

```
  cur = head → rlink;
```

```
  while (cur != head)
```

```
  {  
    if (cur → info == item)
```

```
    {  
      break;
```

```
    }
```

```
    cur = cur → rlink;
```

```
  }
```

```
  if (cur == head)
```

```
  {  
    printf("No such item found in the list \n");  
    return head;
```

```
  }
```

```
  next = cur → rlink;
```

```
  temp = getnode();
```

```
  temp → llink = NULL;
```

```
  temp → rlink = NULL;
```

```
  printf("Enter the item to be inserted at the right  
  of the given item: \n");
```

```
  scanf("%d", &temp → info);
```

```
  cur → rlink = temp;
```

```
  temp → llink = cur;
```

```
  next → llink = temp;
```

```
  temp → rlink = next;
```

```
  return head;
```

```
}
```



NODE delete\_duplicates (int item, NODE head)

{  
    NODE prev, cur, next;

    int count = 0;

    if (head → rlink == head)

    {  
        printf("List is empty\n");

        return head;

    }  
    cur = head → rlink;

    while (cur != head)

    {  
        if (cur → info != item)

        {  
            cur = cur → rlink;

        }

    else

    {  
        count ++;

        if (count == 1)

        {  
            cur = cur → rlink;

            continue;

        }

    else

    {  
        prev = cur → link;

        next = cur → rlink;

        prev → rlink = next;

        next → link = prev;

        free(cur);

        cur = next;

    }

    }

if (count == 0)

{  
    printf("No such item found in the list\n");

}

else

{  
    printf("All the duplicate elements of the given  
    item are removed successfully\n");

}  
return head; }

NODE delete-all-key (int item, NODE head)

```
{
    NODE prev, cur, next;
    int count;
    if (head → rlink == head)
    {
        printf ("List empty");
        return head;
    }
    count = 0;
    cur = head → rlink;
    while (cur != head)
    {
        if (item != cur → info)
            cur = cur → rlink;
        else
        {
            count++;
            prev = cur → llink;
            next = cur → rlink;
            prev → rlink = next;
            next → llink = prev;
            free node (cur);
            cur = next;
        }
    }
    if (count == 0)
        printf ("Key not found");
    else
        printf ("Key found at %d positions and are deleted\n", count);

    return head;
}
```



```
int main()
```

```
{  
    NODE head;
```

```
    int item, choice, key;
```

```
    head = getnode();
```

```
    head → llink = head;
```

```
    head → rlink = head;
```

```
    for (;;) 
```

```
    {
```

```
        printf("1: insert-front\n2: insert-rear\n3: delete-front\n4: delete-rear\n5: display\n6: search\n7: insert-leftpos\n8: insert-rightpos\n9: delete duplicates\n10: delete-based on specified value\n11: exit\n");
```

```
        printf("Enter the choice\n");
```

```
        scanf("%d", &choice);
```

```
        switch (choice)
```

```
        {
```

```
            case 1: printf("Enter the item at front end:\n");
```

```
                    scanf("%d", &item);
```

```
                    head = insert-front(item, head);
```

```
                    break;
```

```
            case 2: printf("Enter the item at rear end:\n");
```

```
                    scanf("%d", &item);
```

```
                    head = insert-rear(item, head);
```

```
                    break;
```

```
            case 3: head = delete-front(head);
```

```
                    break;
```

```
            case 4: head = delete-rear(head);
```

```
                    break;
```

```
            case 5: display(head);
```

```
                    break;
```

```
            case 6: printf("Enter the key element to be searched:\n");
```

```
                    scanf("%d", &key);
```

```
                    dsearch(key, head);
```

```
                    break;
```

```
case 7: printf("Enter the key element :\n");  
scanf("%d", &key);  
head = dinsert-leftpos(key, head);  
break;
```

```
case 8: printf("Enter the key element :\n");  
scanf("%d", &key);  
head = dinsert-rightpos(key, head);  
break;
```

```
case 9: printf("Enter the key element whose duplicates  
should be removed :\n");  
scanf("%d", &key);  
head = ddelete-duplicates(key, head);  
break;
```

```
case 10: printf("Enter the key value :\n");  
scanf("%d", &item);  
head = delete-all-key(item, head);  
break;
```

```
case 11: exit(0);  
default: printf("Invalid choice\n");  
}  
}  
return 0;
```

```
}
```