classmate
Date 9/12/2020
Page

LP#9

WAP to implement doubly linked list with primitive operations a) create a doubly linked list b) insert a new node to the left of the node c) delete the node based on a specific value d) Display the contents of the list e) delete the duplicates [plus other functions]

```
#include <stdio.h>
#include <stdlib.h>
 struct node {
int info;
struct node *link;
struct node *rlink;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x = (NODE) malloc (size of (struct node));
if (x == NULL) {
printf ("mem full\n");
exit (0);
}
return x;
}
```

```
void freenode (NODE x) {
  free (x); }

NODE dinsert front (int item, NODE head) {
NODE temp, cur;
 temp = getnode();
temp -> info = item;
cur = head -> rlink
head -> rlink = temp;
temp -> llink = head;
temp -> rlink = cur;
cur -> llink = temp;
return head;
}

NODE dinsert_rear (int item, NODE head) {
 NODE temp, cur;
 temp = getnode();
 temp -> info = item;
 cur = head -> llink;
 head -> llink = temp;
 temp -> rlink = head;
 temp -> llink = cur;
 cur -> rlink = temp;
```

```
return head;
}

NODE delete_front (NODE head) {
 NODE cur, next;
 if (head -> link = head) {
  printf ("list empty \n");
  return head;
 }
 cur = head -> link;
 next = cur -> link;
 head -> link = next;
 next -> llink = head;
 printf ("the node deleted is %d \n", cur -> info);
 freenode (cur);
 return head;
  }

NODE delete_rear (NODE head) {
 NODE cur, prev;
 if (head -> link = head) {
 printf ("list empty \n");
 return head;
 }
```

```
cur = head -> llink;
prev = cur -> llink;
head -> llink = prev;
prev -> rlink = head;
printf ("the node deleted is %d\n", cur -> info);
freenode (cur);
return head;
}

void display (NODE head) {
NODE temp;
if (head -> rlink == head) {
printf ("list empty \n");
return;
}
printf ("contents of list \n");
temp = head -> rlink;
while (temp != head) {
printf ("%d", temp -> info);
temp = temp -> rlink;
}
printf ("\n");
}
```

```
NODE insert_lftpos (int item, NODE head) {
NODE temp, cur, prev;
if (head -> rlink == head) {
printf ("list empty \n");
return head;
}
cur = head -> rlink;
while (cur != head) {
if (item == cur -> info) break;
cur = cur -> rlink;
}
if (cur == head) {
printf ("key not found \n");
return head;
}
prev = cur -> llink;
printf ("enter towards lft of %d = ", item);
temp = getnode ();
scanf ("%d", &temp -> info);
prev -> rlink = temp;
temp -> llink = prev;
cur -> llink = temp;
temp -> rlink = cur;
return head; }
```

```
NODE insert_rightpos (int item, NODE head) {
NODE temp, cur, next;
if (head -> rlink == head) {
printf ("list empty\n");
return head; }
cur = head -> rlink;
while (cur != head)
{
if (item == cur -> info) break;
   cur = cur -> rlink;
}
if (cur == head) {
printf ("key not found\n");
return head;
}
next = cur -> rlink;
printf ("enter towards right of %d =", item);
temp = getnode();
scanf ("%d", &temp -> info);
next -> llink = temp;
temp -> rlink = next;
cur -> rlink = temp;
temp -> llink = cur;
return head; }
```

```
NODE delete_dup_key (int item, NODE head) {
NODE prev, cur, next;
int count;
if (head -> rlink == head) {
printf ("List is empty !");
return head;
}
count = 0
cur = head -> rlink;
while (cur != head) {
if (item != cur -> info)
  cur = cur -> rlink;
else
{  count ++;
   if (count == 1) {
   cur = cur -> rlink; }
if (count != 1) {
prev = cur -> llink;
next = cur -> rlink;
prev -> rlink = next;
next -> llink = prev;
freenode (cur);
cur = next;
}
```

```
}
}
if (count ==0)
printf ("Key not found");
else
printf ("Duplicates are deleted \n");
return head;
}

void search (int item, NODE head) {
NODE cur;
if (head -> rlink == head) {
printf ("List Empty");
return;
}
cur = head -> rlink;
while (cur != head) {
if (item == cur -> info) break;
cur = cur -> rlink;
}
if (cur == head) {
printf ("search unsuccessful \n");
return;
}
```

```
printf("search successfull\n");
}

NODE delete_value (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;
  freenode(cur);
  }
}
```

```
if (count == 0)
printf ("Key not found \n");
else
printf ("Key found at %d positions and are deleted \n", count);
return head;
}

int main () {
NODE head, last;
int item, choice;
head = getnode ();
head -> rlink = head;
head -> llink = head;
printf ("\n1: insert front \n2: insert rear \n3: delete front \n4: delete
rear \n 5: insert key towards left \n 6: insert key towards right
\n 7: delete duplicate \n 8: delete all specified node \n 9: search item
\n 10: display \n 11: exit \n");
do
{
printf ("enter the choice \n");
scanf ("%d", &choice);
switch (choice)
{
case 1: printf ("enter the item at the front end \n");
```

```
        scanf("%d", &item);
        last = dinsert_front(item, head);
        break;
case 2: printf("enter the item at rear end\n");
        scanf("%d", &item);
        last = dinsert_rear(item, head);
        break;
case 3: last = ddelete_front(head);
        break;
case 4: last = ddelete_rear(head);
        break;
case 5: printf("enter the key item\n");
        scanf("%d", &item);
        last = insert_leftpos(item, head);
        break;
case 6: printf("enter the key item\n");
        scanf("%d", &item);
        last = insert_rightpos(item, head);
        break;
case 7: printf("enter key whose duplicates to be deleted\n");
        scanf("%d", &item);
        last = delete_dup_key(item, head);
        break;
```

```
case 8 : printf ("enter key which has to be deleted \n");
         scanf ("%d", &item);
         last = delete_value (item, head);
         break;
case 9 : printf ("enter item \n");
         scanf ("%d", &item);
         search (item, head);
         break;
case 10 : display (head);
          break;
case 11 : break;
default : printf (" invalid choice ");
          break;
  }
} while (choice != 11);
return 0;
}
```