25/11/2020

LP #6 : WAP to implement singly linked list with the following operations:
a) create a linked list b) insertion of node at first position, specific position and last position c) display contents of linked list
d) deletion of first element, specified element and last element from the list.

```
#include <stdio.h>
#include <stdlib>
struct node
{ int info;
 struct node *link; };
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 insert_front(NODE first, int item) {
NODE temp;
temp=getnode();
temp->info = item;
temp-> link = NULL;
if (first == NULL)
 return temp;
temp->link = first;
 first = temp;
 return first;
  }

NODE delete_front(NODE first) {
 NODE temp;
 if (first == NULL)
```

```
{ pf("list is empty cannot delete\n");
 return first;
}
 temp=first;
 temp = temp → link;
 free(first);
 return temp;
}

NODE insert_rear (NODE first, int item) {
 NODE temp, cur;
 temp = getnode();
 temp → info = item;
 temp → link = NULL;
 if (first == NULL)
 return temp;
 cur = first;
 while (cur → link) = NULL)
 cur = cur → link;
 cur → link = temp;
 return first;
}
```

```
NODE delete_rear(NODE first) {
NODE cur, prev;
if (first == NULL)
{printf("list is empty cannot delete \n");
return first;
}
if (first -> link == NULL)
{
printf("item deleted is %d\n", first -> info);
free(first);
return NULL;
}
prev = NULL;
cur = first;
while(cur -> link) = NULL)
{
prev = cur;
cur = cur -> link;
}
printf("item deleted at rear end is %d", cur -> info);
free(cur);
prev -> link = NULL;
return first;
}
```

```
NODE insert_pos(int item, int pos, NODE first) {
 NODE temp, cur, prev;
 int count;
 temp = getnode();
 temp -> info = item;
 temp -> link = NULL;
 if (first == NULL && pos == 1)
 {
  return temp;
 }
 if (first == NULL)
 {
  printf ("invalid position\n");
  return NULL;
 }
 if (pos == 1)
 {
  temp -> link = first;
  return temp;
 }
  count = 1;
  prev = NULL;
  cur = first;
 while (cur != NULL && count != pos)
```

```
{
  prev = cur;
  cur = cur -> link;
  count++;
}
 if (count == pos) {
  prev -> link = temp;
  temp -> link = cur;
  return first;
 }
 printf ("invalid position\n");
  return first;
 }

NODE delete_info ( int item, NODE first) {
 NODE prev, cur;
 if (first == NULL) {
 printf ("list is empty \n");
  return NULL;
   }
 if (item == first -> info)
{
  cur = first;
  first = first -> link;
```

```
freenode(cur);
return first;
 }

prev=NULL;
cur = first;
while(cur) = NULL)
{ if(item == cur->info) break;
  prev = cur;
  cur = cur -> link;
 }
if (cur == NULL) {
printf(" search is unsuccessfull\n");
return first;
}
prev -> link = cur -> link;
printf(" item deleted is %d ", cur -> info);
freenode(cur);
return first;
 }

void display (NODE first) {
NODE temp;
if ( first == NULL)
printf(" list empty cannot display items\n");
```

```
for( temp = first ; temp != NULL ; temp = temp -> link ) {
   printf ("%d\n", temp -> info); }
}


int main() {
int item, choice, pos;
NODE first = NULL;
printf (" \n 1: Insert_front \n2 : Delete_front \n 3 : Insert_rear \n 4 : Delete_
rear \n 5 : Insert at specified position \n 6 : delete specified element \n
7 : display_list \n 8 : Exit\n ");
do
{
printf (" enter the choice \n ");
scanf ("%d", &choice);
switch (choice)
{
case 1: printf (" enter the item at front-end \n ");
        scanf ("%d", &item);
        first = insert_front (first, item);
        break;
case 2 : first = delete_front (first);
        break;
case 3 : printf (" enter the item at the rear-end \n ");
        scanf ("%d", &item);
```

```
            first = insert_rear (first, item);
            break;
case 4 : first = delete_rear (first);
            break;
case 5 : printf (" Enter the item and position : \n");
            scanf (" %d %d ", &item, &pos);
            first = insert_pos (item, pos, first);
            break;
case 6 : printf (" enter the element to be deleted \n");
            scanf (" %d ", &item);
            first = delete_info (item, first);
            break;
case 7 : display (first);
            break;
case 8 : break;
default : break;
}
} while (choice != 8);
return 0;
}
```

classmate
Date 25/11/2020
Page

LP#7: WAP to implement singly linked list with the following operations:
a) Sort the linked list b) Reverse the linked list c) Concatination of 2 linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int info;
struct node *link; };

typedef struct node *NODE;
NODE getnode() {
NODE x;
x=(NODE) malloc (sizeof(struct node));
if (x==NULL) {
printf("mem full\n");
exit(0);
}
return x;
}

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

```
NODE insert_front (NODE first, int item) {
 NODE temp;
 temp = getnode();
 temp → info = item;
 temp → link = NULL;
 if (first = NULL);
 return temp;
 temp → link = first;
 first = temp;
  return first;
 }


NODE delete_front (NODE first) {
 NODE temp;
 if (first == NULL)
 {
printf (" list is empty cannot delete \n");
return first;
 }

temp = first;
 temp = temp → link;
 printf (" item deleted at front end is = %d", first → info);
 free (first);
 return temp;
}
```

```
NODE insert_rear (NODE first, int item) {
 NODE temp, cur;
 temp = getnode();
 temp → info = item;
 temp → link = NULL;
 if (first == NULL)
 return temp;
 cur = temp;
 while (cur → link != NULL)
 cur = cur → link;
 cur → link = temp;
 return first;
}

NODE delete_rear (NODE first) {
 NODE cur, prev;
 if (first == NULL) {
 printf ("list is empty cannot delete \n");
 return first;
 }
 if (first → link = NULL)
 {
 printf ("item deleted is %d \n", first → info);
 free (first);
```

```
return NULL;
}

prev = NULL;
cur = first;
while (cur -> link != NULL) {
 prev = cur;
 cur = cur -> link; }
printf("item deleted at rear-end is %d", cur -> info);
free(cur);
 prev -> link = NULL;
 return first;
 }


NODE order_list (int item, NODE first) {
 NODE temp, prev, cur;
 temp = getnode();
 temp -> info = item;
 temp -> link = NULL;
 if (first == NULL) return temp;
 if (item < first -> info) {
  temp -> link = first;
  return temp;
 }

  prev = NULL;
```

```
cur = first;
while (cur != NULL && item > cur->info)
{ prev = cur;
  cur = cur->link; }
prev->link = temp;
temp->link = cur
return first;
}

NODE reverse (NODE first) {
NODE cur, temp;
cur = NULL;
while (first != NULL) {
 temp = first;
 first = first->link;
 temp->link = cur;
 cur = temp;
}
 return cur;
}

NODE concat (NODE first, NODE second) {
NODE cur;
if (first == NULL)
```

```
return second;
if (second = NULL)
return first;
cur = first;
while ( cur -> link ) = NULL)
 cur = cur -> link;
 cur -> link = second;
 return first; }

void display ( NODE first ) {
 NODE temp;
 if (first == NULL)
 printf (" list empty cannot display item \n ");
 for ( temp = first; temp ! = NULL ; temp = temp -> link ) {
 printf ("%d \n", temp -> info); }
}

int main () {
int item, choice, pos, n;
NODE first = NULL, a, b;
printf (" \n1: Insert front \n2 : Delete front \n3 : Insert rear \n
4: Delete rear \n 5: Sorted list \n 6 : Reverse list \n 7: Concatinate
2 strings \n 8: Display list \n 9: Exit \n ");
do
```

```
{
printf("\n enter the choice \n");
scanf("%d", &choice);
switch (choice)
{
case 1: printf("enter the item at the front end \n");
        scanf("%d", &item);
        first = insert_front (first, item);
        break;
case 2 : first = delete_front (first);
         break;
case 3 : printf("enter the item at rear-end \n");
         scanf("%d", &item);
         first = insert_rear (first, item);
         break;
case 4 : first = delete_rear (first);
         break;
case 5 : printf("enter the item to be inserted in ordered list \n");
         scanf("%d", &item);
         first = order_list (item, first);
         break;
case 6 : first = reverse (first);
         display (first);
         break;
```

```
case 7: printf ("enter no. of nodes in 1\n");
        scanf("%d", &n);
        a = NULL;
        for(int i=0; i<n; i++
      { printf("enter the item\n");
        scanf("%d", &item);
          a = insert_rear(a, item);
        }
      printf("enter no. of nodes in 2\n");
       scanf("%d", &n);
        b = NULL;
       for(int i=0; i<n; i++)
     { printf("enter the item\n");
       scanf("%d", &item);
         b = insert_rear(b, item);
       }
       a = concat(a, b);
       display(a);
       break;

case 8: display(first);
        break;
case 9 : break;
default : break;
```

```
}
} while (choice != 9);
return 0;
}
```