classmate
Date
Page

LAB 7

SINGLE LINK LIST CONCATINATION

```
#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 SF (NODE record, int item)
{
   NODE temp;
   temp = getnode ();
   temp -> info = item;
   temp -> link = NULL;
   if (record == NULL)
      return temp;
   temp -> link = record;
   record = temp;
   return record;
}
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\n", 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 = first;
    while (cur -> link != NULL)
     cur = cur -> link;
     cur -> link = temp;
     return first;
}
NODE RR (NODE second, int item)
{
    NODE temp, cur;
    temp = getnode ();
    temp -> info = item;
```

```
    temp->link = NULL;
    if (second == NULL)
      return temp;
    cur = second;
    while (cur->link != NULL)
    cur = cur->link;
    cur->link = temp;
    return second;
}
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;
    NODE prev, cur;
    int count;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL && pos==1)
    return temp;
    if(first==NULL)
    {
        printf("invalid pos\n");
        return first;
    }
    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_pos(int pos, NODE first)
{
  NODE cur;
  NODE prev;
  int count;
  if(first==NULL || pos<=0)
  {
    printf("invalid position\n");
```

```
    return NULL;
}
if (pos==1)
{
    cur=first;
    first = first->link;
    freenode (cur);
    return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur!=NULL)
{
    if (count==pos)
    break;
    prev = cur;
    cur = cur->link;
    count++;
}
if (count!=pos)
{
    printf ("Invalid position \n");
    return first;
}
```

```
    if (count != pos)
    {
      printf("Invalid position specified\n");
      return first;
    }
    prev->link = cur->link;
    freenode(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 asc (NODE first)
{
    NODE prev = first;
    NODE cur = NULL;
    int temp;
```

```
if (first = NULL)
{
    return 0;
}
else
{
    while (prev != NULL)
    {
        cur = prev -> link;
        while (cur != NULL)
        {
            if (prev -> info > cur -> info)
            {
                temp = prev -> info;
                prev -> info = cur -> info;
                cur -> info = temp;
            }
            cur = cur -> link;
        }
        prev = prev -> link;
    }
}
return first;
}
NODE des (NODE first)
```

```
{
    NODE prev = first;
    NODE cur = NULL;
    int temp;
    if (first == NULL)
    {
        return 0;
    }
    else
    {
        while (prev != NULL)
        {
            cur = prev -> link;
            while (cur != NULL)
            {
                if (prev -> info < cur -> info)
                {
                    temp = prev -> info;
                    prev -> info = cur -> info;
                    cur -> info = temp;
                }
                cur = cur -> link;
            }
            prev = prev -> link;
        }
    }
} return first;
```

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

```
int main()
{
int item, choice, pos, element, option, choice2, item2, num;
NODE first = NULL;
NODE second = NULL;
for(;;)
{
printf("\n1. Insert_front \n2. Delete_front \n3. Insert_rear \n4.
      Delete_rear \n5. Random_pos \n6. Reverse \n7. Sort \n8
      Concate \n9. display \n10. 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);
        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("press 1 to insert or 2 to delete at any
        desired position \n");
        scanf("%d", &element);
        if (element == 1)
        {
        printf("enter the position to insert \n");
        scanf("%d", &pos);
        printf("enter the item to insert \n");
        scanf("%d", &item);
        first = insert_pos(item, pos, first);
        }
        if (element == 2)
        {
        printf("enter the position to delete \n");
        scanf("%d", &pos);
        first = delete_pos(pos, first);
        }
        break;
case 6: first = reverse(first);
        break;
case 7: printf("press 1 for ascending sort and 2 for
        descending sort: \n");
        scanf("%d", &option);
        if (option == 1)
        first = asc(first);
        if (option == 2)
        first = des(first);
```

```
        break;
case 8: printf("create a second list\n");
        printf("enter the number of elements in
        second list\n");
        scanf("%d", &num);
        for(int i=1; i<=num; i++)
        {
          printf("\n press 1 to insert front and 2
                 to insert rear\n");
          scanf("%d", &choice2);
          if(choice2==1)
          {
            printf("enter the item at front end");
            scanf("%d", &item1);
            second=INF(second, item1);
          }
          if(choice2==2)
          {
            printf("enter the item at back end");
            scanf("%d", &item1);
            second = IR(second, item1);
          }
        }
        first = concate(first, second);
        break;
```

```
        case 9: display (first);
                break;
        default: exit (0);
                break;
            }
        }
        return 0;
}
```