PROGRAM-6: WAP to Implement Singly Linked List with following operations a).Create a linked list b).Deletion of first element, specified element and last element in the list c).Display the contents of the linked list.

Date / / Page SPLASH

LAB PROGRAM-6

```
#include <stdio.h>
#include <malloc.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;
}
int freenode (NODE x)
{
free (x);
return 0;
}
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\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;
```

Date / /
Page
SPLASH

```
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("The 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("The item deleted at rear-end is %d", cur->info);
free(cur);
prev->link = NULL;
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);
  }
}
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;
free(cur);
return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur != NULL)
{
if (count == pos) break;
prev = cur;
cur = cur->link;
```

```
        count++;
    }
    prev->link = cur->link;
    free node(cur);
    return first;
}
int main()
{
    int item, choice, pos;
    NODE first = NULL;
    system("CLS");
    for(;;)
    {
        printf("\n 1: Insert front \n 2: Delete front \n
3. Insert rear \n 4. Delete rear \n 5. Delete at specified
position \n 6: Display list \n 7. 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 5: printf("enter the position of the item to be deleted \n");
```

```
        scanf("%d", &pos);
        first = delete_pos(pos, first);
        break;
    case 6: display(first);
        break;
    default: exit(0);
        break;
    }
}
getch();
return 0;
}
```

PROGRAM-7: WAP to Implement Single Link List with following operations a).Sort the linked list b).Reverse the linked list c).Concatenation of two linked list.

Classmate
Date
Page

LAB PROGRAM - 7

```
#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 IF (NODE second, int item)
  { NODE temp;
   temp = get node();
   temp -> info = item;
   temp -> link = NULL;
    if (second == NULL)
      return temp;
```

```
temp -> link = Second;
Second = temp;
return Second; }
NODE delete_front (NODE first)
{ NODE temp;
  if (first == NULL)
{ printf ("List is empty cannot delete");
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 IR (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)
```

Classmate
Date
Page

```
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"); }
if (first -> link == NULL)
{ printf ("item deleted is %d \n", first -> info);
  free (first); }
  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 -> link = NULL;
  if (first == NULL && pos == 1)
    return temp;
  { printf (" invalid pos \n");
    return first; }
    if (pos == 1)
    { temp -> link = first;
      return temp; }
    count = 1;
    prev = NULL;
```

```
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"); }
NODE delete_pos(int pos, NODE first)
{ NODE cur;
  NODE prev;
  int count;
  if (first == NULL || pos <= 0)
{ printf("invalid position\n");
return NULL; }
  { cur = first;
    first = first -> link;
    freenode(cur);
    return first; }
  prev = NULL;
  cur = first;
  count = 1;  return first;
{ 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;
   { 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;
    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); }}
void main()
{ int item, choice, pos, second, option, choice2, item1, num;
    NODE first = NULL;
    NODE second = NULL;
    for (;;)
    { printf (" 1: Insert_front \n 2: Delete_front \n
3: Insert_rear \n 4: Delete_rear \n 5: random_position \n
6: traverse \n 7: Sort \n 8: concate \n 9: display_list \n
9: 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);
  case 4; first = delete_rear(first);
  Case 5: printf("press 1 to insert or 2 to delete at any
  desired position\n");
    Scanf("%d", &demand);
    if (demand == 1){
  printf("enter the position to insert\n");
    Scanf("%d", &pos);
    printf("enter the item to insert\n");
    first = insert_pos(item, pos, first); }
  if (demand == 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 = del (first);
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 ("1 in press 1 to insert front and 2 to insert rear\n");
    scanf ("%d", &choice2);
      if (choice2 == 1)
        printf ("enter the item at front end \n");
    scanf ("%d", &item1);
      Second = IF (Second, item1); }
    if (choice2 == 2)
    printf ("enter the item at rear end \n");
      scanf ("%d", &item1);
      Second = IR (Second, item1); }}
      first = concate (first, Second);
        break;
Case 9: display (first);
      break;
    default: exit (0);
      break;
  }
}
  getch ();
}
```

PROGRAM-8: WAP to implement Stack and Queues using Linked Representation.

CODE:

#include<stdio.h>

Date __/__/__
Page
SPLASH

LAB PROGRAM 7 - 8

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <process.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_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_front (NODE first)?
```

Date / /
Page
SPLASH

```
{ 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; }
Void display (NODE first)
{ NODE temp;
if (first == NULL)
printf(" list is empty cannot display item \n");
for(temp = first; temp != NULL; temp = temp->link)
{printf("%d \n", temp->info); } }
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_S(NODE first)
{ NODE temp;
 if (first == NULL)
{printf(" Stack is empty cannot delete \n");
 return first; }
temp = first;
temp = temp->link;
```

Date __/__/__
Page
SPLASH

printf("item deleted at front and is = %d\n", front->info);
free(first);
return item;
void display - 3 (NODE first)
{ NODE temp;
if (first == NULL)
printf("Stack 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;
System("cls");
for(;;)
{ printf("\n Queue Operation : \n 1: Insert rear \n 2: Delete front \n 3: Display list (Queue) \n \n Stack operations \n 4: Insert front \n 5: Delete front \n 6: Display list (Stack) \n 7: Exit \n\n");
printf("enter the choice \n");
scanf("%d", &choice);
switch(choice)
{ case 1: printf("enter the item at rear end \n");
scanf("%d", &item);
first = insert_rear(first, item);
break;
case 2: first = delete_front(first);
break;
case 3: display(first);
break;
case 4: printf("enter the item at front end \n");
scanf("%d", &item);
first = insert_front(first, item);

```
    break;
case 5: first = delete_front s(first);
    break;
case 6: display s(first);
    break;
default: exit(0);
break;
}
}
getch();
return 0;
}
```

PROGRAM-9: WAP to implement doubly link 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.

Date / /
Page
SPLASH

LAB PROGRAM - 9

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
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 dinsert_rear(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 insert_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->rlink==head)
{
printf("dq empty\n");
return head;
}
cur=head->rlink;
next=cur->rlink;
head->rlink=next;
next->llink=head;
printf("the node deleted is %d", cur->info);
free(cur);
```

Date __/__/__
Page
SPLASH

return head; }

NODE ddelete_rear(NODE head)
{ NODE cur, prev;
if (head → rlink == head)
{ printf("dq empty\n");
return head;
}
cur = head → llink;
prev = cur → llink;
head → llink = prev;
prev → rlink = head;
printf("The node deleted is %d", cur → info);
freeNode(cur);
return head;
}

NODE insert_leftpos(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)
{

```
cprintf("Key not found\n");
    return head;
}
cprev = cur -> llink;
cprintf("enter location left of %d = ", itav);
temp = getnode();
scanf("%d", &temp->info);
cprev -> rlink = temp;
temp -> llink = cprev;
cur -> llink = temp;
temp -> rlink = cur;
return head;
}
NODE insert_rightpos(int itav, NODE head)
{
NODE temp, cur, next;
if (head -> rlink == head)
{
cprintf("List empty\n");
return head;
}
cur = head -> rlink;
while (cur != head)
{
if (item == cur -> info) break;
cur = cur -> rlink;
}
if (cur == head)
{
cprintf("Key not found\n");
return head;
}
```

```
next = cur -> rlink;
printf("node to be inserted right of :d=", item);
temp = getnode();
scanf("%d", &temp->info);
cur -> rlink = temp;
temp -> llink = cur;
next -> llink = temp;
temp -> rlink = next;
return head;
}

NODE Search(NODE head, int item)
{
NODE temp, cur;
int flag=0;
if (head->rlink == head)
{
cur = cur -> rlink;
}
if (cur == head)
printf("Search unsuccessfull \n");
if (flag=1)
printf("Search successfull \n");
}

NODE deleteallKey(int item, NODE head)
{
NODE prev, cur, next;
int count;
if (head->rlink == head)
{
printf("list empty \n");
return head;
}
```

```
count = 0;
cur = head -> right;
while(cur != head)
{
if (item != cur -> info)
  cur = cur -> right;
else
{ count++;
prev = cur -> left;
next = cur -> right;
prev -> right = next;
next -> left = prev;
free node(cur);
 cur = next;
if (count == 0)
  printf("not found\n");
else {
  printf("found at %d position and deleted", count);
  return head;
 }
}
void display(node head)
{
node = temp;
if(head -> right == head)
{
printf(" dq empty\n");
return;
}
printf(" contents of dq\n");
  temp = head -> right;
  while (temp != head)
```

```
{
printf("%d", temp->info);
temp = temp->link;
}
printf("\n");
}
void main()
{
NODEPTR last;
int item, choice;
char ch;
ch = getch();
ch = tolower(ch);
clrscr();
while(1)
{ printf("\n1: insert front\n2: insert rear\n3: delete
front\n4: delete rear\n5: insert left of key
element\n6: insert right of key element\n7: search\n
8: delete a particular occurrence\n9: display\n10: exit\n");
Case 1: printf("enter the item at front end\n");
scanf("%d", &item);
last = insert_front(item, ch);
break;
Case 2: printf("enter the item at rear end\n");
scanf("%d", &item);
last = insert_rear(item, ch);
break;
Case 3: last = delete_front(ch);
break;
Case 4: last = delete_rear(ch);
break;
```

```
Case 5:
printf("enter the key element\n");
  scanf("%d", &item);
  last = insert_leftpos(item, head);
  break;
Case 6:
printf("enter the key element\n");
  scanf("%d", &item);
  last = insert_rightpos(item, head);
  break;
Case 7:
  printf("enter the search element\n");
  scanf("%d", &item);
  search(head, item);
  break;
Case 8: printf("enter element to be delete\n");
    scanf("%d", &item);
    last = delete_all_key(item, head);
Case 9: display(head);
    break;
  default: exit(0);
  }
}
getch();
}
```

PROGRAM-10: Write a program a). To construct a binary search tree b). To traverse the tree using all the methods i.e, in-order, pre-order and post-order.

Date __/__/__
Page
SPLASH

TREES

LAB PROGRAM-10 BINARY SEARCH

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{int info;
struct node *llink;
struct node *rlink; };
typedef struct node *NODE;
NODE getnode()
{ NODE x;
  x=(NODE) malloc(sizeof(struct node));
if(x==NULL)
{printf("Memory not available!");
exit(0); }
return x; }
void freenode(NODE x)
{free(x); }
NODE insert(int item, NODE root)
{NODE temp, cur, prev;
char direction[10];
int i;
temp=getnode();
temp->info=item;
temp->llink=NULL;
temp->rlink=NULL;
if(root==NULL)
  return temp;
printf("Give direction to insert...\n");
scanf("%s", direction);
prev=NULL;
```

```
cur = root;
for(i=0; i<strlen(direction) && (cur!=NULL); i++)
{prev = cur;
if(direct[i]=='l')
cur = cur->llink;
else
cur = cur->rlink;}
if(cur != NULL || i != strlen(direction))
{ printf("Insertion not possible\n");
  free(temp);
  return(root);}
if(cur==NULL)
{ if(direction[i-1]=='l')
prev->llink = temp;
else
prev->rlink = temp;}
  void preorder(NODE root)
{ if(root!=NULL)
{ printf("the item is %d\n", root->info);
  preorder(root->llink);
  preorder(root->rlink);}}
void inorder(NODE root)
{ if(root!=NULL
{ inorder(root->llink);
  printf("the item is %d\n", root->info);
  inorder(root->rlink);}}
void postorder(NODE root)
 { if(root!=NULL)
 { postorder(root->llink);
   postorder(root->rlink);
    printf("the item is %d\n", root->info);}}
 void display(NODE root, int i)
```

```
{ int j;
if (root != NULL)
{ display (root -> rlink, i+1);
for (j=1; j<=i; j++)
printf (" ");
Printf ("%d \n", root -> info);
display (root -> llink, i+1); }}
int main()
{
NODE root = NULL;
int choice, i, item;
for(;;)
{printf ("1. Insert \n2 Preorder \n3. Inorder \n4. Postorder
\n5. Display \n");
printf ("Enter the choice: \n");
Scanf ("%d", &choice);
Switch (choice)
{ case1 : printf ("Enter the item: \n");
Scanf ("%d", &item);
root = insert (item, root);
break;
case 2: if (root == NULL)
{printf ("Tree is empty !"); }
else
{printf ("Given tree is..");
display (root, 1);
printf ("The preorder traversal is: \n");
preorder (root); }
break;
case 3: if (root == NULL)
{ printf ("Tree is empty"); }
else
```

```
{ printf("Given tree is..");
  display(root,1);
  printf("The inorder traversal is\n");
  inorder(root); }
  break;
case 4: if(root==NULL)
  { printf("Tree is empty"); }
  else
  { printf("Given tree is..");
    display(root,1);
    printf("The postorder traversal is\n");
    postorder(root); }
    break;
case 5: display(root,1);
    break;
default: printf("Invalid choice\n");
    exit(0);
}
}
return 0;
}
```