Date: / /

DS Lab-2

Stack Implementation.

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

int top = -1, stack [MAX];
void push ();
void pop ();
void display ();

void main ()
{
 int ch;
 while (1)
 {
 printf ("\n*** Stack Menu ***");
 printf ("\n\n1. Push \n2. pop \n3. Display
          \n 4. Exit");
 printf ("\n \n Enter your choice (1-4): ");
 scanf ("%d", &ch);

switch (ch)
{
 case 1 : push ();
          break;
```

```
case 2 : pop();
         break;
case 3 : display();
         break();
case 4 : exit(0);
default : printf("\n Wrong choice");
  }
 }
}
void push()
{
 int val;
 if (top == MAX-1)
 {
   printf("\n Stack is full");
 }
 else
 {
   printf("\n Enter element to push : ");
    scanf("%d", &val);
     top = top + 1;
     stack[top] = val;
  }
}
```

Lab-3 INFIX TO POSTFIX

```
#include <stdio.h>
#define N 100

int stack [N];

int top = -1;

void push (int item) {
        if (top == N-1)
            printf ("stack overflow! \n");
        else
            stack [++top] = item;
    }
    int pop () {
        if (top == -1)
            printf ("stack underflow \n");
        else
        return stack [top--];
    }
    int priority (char op) {
        switch (op) {
            case '*': return 2;
                      break;
            case '/': return 2;
                      break;
            Case '+': return 1;
                      break;
            case '-': return 1;
                      break;
```

```
case '(' : return 0;
           break;
  }
}
int main ()
{
   char s[50];
   char t[50];
   int i;
   int choice = 1;

   do {
      i = 0;
      printf ("Enter your infix expression:");
      scanf ("%s", s);

   for (int i=0; s[i] != '\0'; i++){
      switch (s[i]){
        case '(' : push ('(');
                   break;
        case ')' : while (stack[top] != '('){
                     t[i++] = pop();
                   }
                   pop();
                   break;
        case '*' :
        case '/' :
        case '+' :
        case '-' : while (top != -1 && priority
                   (stack[top]) >= priority (s[i])){
                     t[i++] = pop();
                   }
```

```
        push (s[i]);
          break;
    default: t[l++] = s[i]
      }
    }
    while (top != -1) {
          t[l++] = pop();
      }
      t[l] = '\0';

    printf (" Postfix expression for \"%s\" is \"%s\".\n",
            s, t);
      printf("\n :: Menu :: \n1. Try another
          infix expression. \n2. Exit. \n Enter
          your choice: ");
      scanf ("%d", &choice);
      } while (choice != 2);
      return 0;
    }
```

Date
Page No.

LAB-5

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#define que_size 3
int item, front = 0, rear = -1, q[que_size], count = 0;
void insertrear()
{
  if (count == que_size)
  {
    printf("queue underflow");
      return;
  }
  rear = (rear + 1) % que_size;
   q[rear] = item;
   count ++;
}
int deletefront()
{
  if (count == 0) return -1;
     item = q[front];
  front = (front + 1) % que_size;
     count = count - 1;
       return item;
}
void display q()
  {
    int i, f;
     if (count == 0)
    {
       printf("queue is empty");
```

```
return;
}
f = front;
printf("contents of queue \n");
for (i=0; i<= count ; i++)
{
    printf("%d\n", q[f]);
    f = (f+1) % que_size;
}
}
void main()
{
  int choice;
  for(;;)
  {
    printf("\n1.Insert rear \n2. Delete front \n3.
      Display \n4. exit \n");
    printf("Enter the choice : ");
    scanf("%d", &choice);
    switch(choice)
    {
    case 1: printf("Enter the item to be
        inserted : ");
      scanf("%d", &item);
      insertrear();
      break;
    case 2: item = deletefront();
      if(item == -1)
      printf(queue is empty\n");
      else
      printf("item is deleted is %d\n", item);
      break
```

```
case 3: display q ();
        break;
default: exit (0);
    }
  }
getch ();
}
```

Lab-6

Priority Queue

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define quesize 10
int item, p, rear = -1, q[quesize][2];
void insertrear() {
if (rear < quesize) {
q[++rear][0] = item;
q[rear][1] = p;
}
else
printf("Queue overflow\n");
}
void remove_small() {
int min = INT_MAX;
int t;
for (int i = 0; i <= rear; i++) {
if (q[i][1] < min) {
min = q[i][1];
t = i;
}
}
if (min != INT_MAX) {
printf("Element removed: %d with priority
number: %d\n", q[t][0], min);
q[t][1] = INT_MAX;
}
else
printf("Queue Underflow\n");
```

```
}
void display () {
printf ("Elements of queue : \nele \tprior \n");
  for (int i = 0; i <= rear; i++) {
   if (q[i][1] != INT_MAX)
   printf ("%d \t %d \n", q[i][0], q[i][1]);
  }
}
int main () {
 int choice;
 for (;;) {
 printf ("Enter : \n1. Insert Element \n2. Delete
   Highest Prior \n3. Display \n4. Exit \n");
     Scanf ("%d", &choice) {
 case 1 : printf ("Enter element & priority : \n");
  Scanf ("%d %d", &item, &p);
  insertear ();
  break;
  case 2 : remove_small ();
  break
  case 3 : display ();
  break;
  case 4 : exit (0);
  default : printf ("Wrong choice \n");
  }
 }
 return 0;
}
```

Date
Page No.

DS - Lab

Singly linked list

```
#include <stdio.h>
#include <stdlib.h>
 struct node
{
int info;
 struct node *link;
};
type def struct node* NODE;
  NODE get node()
   {
   NODE x;
x=(NODE) malloc (size of (struct node));
 if (x==NULL)
{
printf ("mem full\n");
   exit(0);
}
return x;
}
void free node (NODE x)
 {
 free(x);
}
NODE insert- front (NODE first, int item)
{
NODE temp;
temp = get node ();
temp → info = item;
temp → link = NULL;
```

```
if (first == NULL)
return temp;
temp -> link = first;
first = temp;
return temfirst;
}
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);
}
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 cur, prev;
if (first == NULL)
{
printf("list is empty cannot delete \n");
 return first;
}
if (first -> link == NULL)
{
printf("list 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~~ rear end is %d", cur ->
                                                  info);
free (cur);
prev -> link = NULL;
 return first;
}
NODE insert_posth (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 (" IP \n");
  return first;
```

```
}
NODE delete_pos (int pos, NODE first)
{
if (first == NULL {
printf ("list empty \n");
return first;
}
NODE temp = first;
if (pos==1)
{
first = temp -> link;
free (temp);
return first;
}
NODE prev;
for (int i=1; temp != NULL && i<pos ; i++)
{
prev = temp;
temp = temp -> link;
}
if (temp == NULL || temp -> link == NULL){
printf ("Invalid posth \n");
return null NULL
}
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 concat (NODE first, NODE second)
{
 NODE cur;
 if (first = NULL)
  return second;
  if (second = = NULL)
   return first;
   cur = first;
   while (cur→link ! =NULL)
 }
void main()
{
 int item, choice, pos, i, n;
  NODE a, b;
  NODE first = NULL;
for(;;)
{
printf(" 1. insert_front \n2 .delete_front\n3 insert
_rear \n4. delete_rear\n5. insert at pos\n6.
delete at pos \n7. concat\n8. reverse\n9.order
 list \n10. display \n");
 printf(" Enter the choice\n");
  scanf("%d", &choice);
   switch (choice)
  {
case 1: printf(" enter the item at front-end \n");
scan f ("%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 item \n");
 scanf ("%d", &item);
printf ("enter the post^n \n");
  scanf ("%d", &pos);
 first = insert_pos (item, pos, first);
 break;
 case 6:
 printf ("Enter post^m of deletion \n");
  scanf ("%d" &pos);
  first = delete_pos (pos, first);
 break;
 case 7:
 printf ("Enter the no. of nodes in 2 \n");
   scanf ("%d" &n);
    a = NULL;
    for (i=0; i<n; i++)
   {
   printf ("Enter the item \n");
    scanf ("%d" &item);
    ab = insert_rear (b, item);
   }
   a = concat (a, b);
    display (a);
    break;
```

```
case 9:
first = order-list (first);
break;
case 10: display (first);
break;
default : exit (0);
break;
}
}
}
```

DS - Lab-8

7/12/12 linked list

```
# include <stdio.h>
# include <stdlib.h>
 struct node
{
int info;
 struct node *link*;
};
type def struct node * NODE;
  NODE getnode()
{
NODE x;
x = (NODE) malloc (size of (struct node));
if (x = NULL
{
print f ("mem full \n");
 exit (0);
}
return x;
}
NODE insert_front (NODE first, int item)
{
 NODE temp;
 temp = get node ();
```

Scanned with CamScanner

```
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
return temp;
temp -> link = first;
first = 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 (NODE first)
{
  int swapped, i;
   NODE ptr1, lptr = NULL;
  if (first == NULL
    return first;
do
{
 swapped = 0;
 ptr = first;
 while (ptr1 -> link != lptr
{
 if (ptr1 -> info > ptr1 -> link -> info)
 {
```

Sukeerthi

```
int temp = ptr1 -> info;
ptr1 -> info = ptr1 -> link -> info;
ptr1 -> link -> info = temp
  swapped = 1;
}
ptr1 = ptr1 -> link;
}
  ptr = ptr1;
}
  while (swapped);
  return first;
}
void count (NODE first)
{
  NODE temp;
  temp = first;
  int c = 0;
  while (temp != NULL)
  {
  temp = temp -> link;
  c++;
}
void list_search (NODE first, int key){
  NODE temp;
  temp = first;
```

```
int c=0, f=0;
while (temp!=NULL){
    c++;
    if(temp->info==key){
     printf("searched unsuccessful, element
     post^n : %d \n", c);
        f=1; break;
    }
    temp = temp -> link;
}
 if(f==0)
    printf("Search Unsuccessful!\n");
}
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);
}
}
```

Sulekha

```
int main (){
int item, choice, pos, i, n;
NODE first = NULL;
for (;;)
{
printf ("1. insert-front \n2. delete_rear \n
   3. display \n 4. count_items \n5. search
\n 6. order \n Any other key to exit \n");
printf ("enter the choice \n");
scanf ("%d", &choice \n");
scanf ("
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_rear (first);
   break;
   case 4: count (first);
   break
   case 5: printf ("Enter element to be
   searched : ");
   scanf ("%d", &item);
```

```
break;
case 6:
 first=order_list(first)
break;
default : exit (0);
}
}
}
```

14/12/12 Double linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int info;
 struct node * rlink;
 struct node * llink;
};
type def struct node * NODE;
NODE get node() {
   NODE x;
   x = (NODE) malloc (size of (struct
                        node));
   if (x == NULL)
   {
    printf (" Memory full \n");
    exit(0);
   }
   return x;
  }
  NODE dinsert_rear(int item, NODE
  head){
     NODE temp, cur;
```

```
temp -> info = item;
cur = head -> rlink;
head -> rlink = temp;
temp -> llink = head;
temp -> rlink = cur;
cur -> llink = temp;
return head;
}
NODE ddelete_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 item deleted is %d\n",
free(cur);
return head;
}
NODE ddelete_rear (NODE head)
```

```
{
NODE cur,prev;
if(head -> rlink == head)
{
printf("dq empty \n")
    retur head;
}
cur = head -> llink;
prev = cur -> llink;
head -> llink = prev;
prev -> rlink = head;
printf("The item deleted is %d \n",
            cur -> info);
   free(cur);
   return head;
}
NODE dlsearch(NODE head, int key,
int x){
   NODE cur, prev, temp;
int f=0, c=1;
   if(head -> rlink == head)
{
printf("list empty \n");
   return head;
}
```

Scanned with CamScanner

```
if (cur -> info == key) {
    f = 1;
    break;
}
cur = cur -> rlink;
    c++;
}
if (f == 1 && z == 0) {
printf ("Search successful, found at
index %d \n", c);
return head;
}
if (f == 1 && z == 1) {
prev = cur -> llink;
printf ("Enter towards left of %d
= ", key);
    temp = getnode ();
    scanf ("%d", &temp -> info);
    prev -> rlink = temp;
    temp -> llink = prev;
    cur -> llink = temp;
    temp -> rlink = cur;
    return head;
    }
    if (f == 1 && z == 2) {
```

```
prev = cur;
cur = cur -> link;
printf("Enter towards right of %d=key);
temp = get node();
scanf("%d", &temp -> info);
prev -> rlink = temp;
temp -> llink = prev;
cur -> llink = temp;
temp -> rlink = temp;
return head;
}
printf("Search unsuccessful \n");
}
NODE delete_all_key (int item, NODE
head)
{
NODE prev, cur, next;
int count;
if (head -> rlink == head)
{
printf("List Empty\n");
return head;
}
count++
prev = cur -> llink;
```

```
next = cur -> rlink;
prev -> rlink = next;
next -> llink = prev;
free(cur);
cur(cur);
cur = next;
}
}
if (count == 0)
printf("Key not found \n");
else
printf("Key found at %d post"
& deleted \n", count);
return head;
}
void display(NODE head)
{
NODE temp;
if (head -> rlink == head)
{
printf("dq empty \n");
return;
}
printf("contents of dq \n");
temp = head -> rlink;
```

```
while (temp1 != head)
{
  printf ("%d", temp -> info);
    temp = temp -> rlink;
}
printf("\n");
}
void main ()
{
  NODE head, last;
  int item, choice;
    head = getnode ();
    head -> rlink = head;
    head -> llink = head;
    for (;;) {
printf ("Enter choice: \n1. Insert front
    \n2. Delete front \n3. Insert rear
    \n4. Delete rear \n5. Simple
    search \n6. Insert left of
    key \n7. Insert right of
key \n8. Delete all occurences
of key \n9. Display \n --
    Any other key to exit --- \n")
```

Scanned with CamScanner

```
switch (choice) {
  case 1: printf("Enter the item at
front end \n");
  scanf("%d", &item);
head = dinsert_front(item, head);
  break;
case 3: printf("enter the item at
  rear end \n");
  scanf("%d", &item);
  head = dinsert_rear(item, head);
  break;
  case 2:
    head = ddelete_front(head);
break;
case 4:
  head = ddelete_rare(head);
break
case 5: printf("Enter key \n");
  scanf("%d", &item);
head = Isearch(head, item 1);
  break;
case 7: printf("Enter key \n");
scanf("%d", &item);
head = Isearch(head, item; 2);
  break.
}
```

21/12/12 LAB-10

Binary Tree

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

struct node
{
 int info;
 struct node *rlink;
 struct node *llink;
};
type def 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 (NODE root, int item)
{
  NODE temp, cur, prev;
   temp = getnode();
```

```
temp->rlink = NULL;
temp->llink = NULL;
temp->info = item;
if (root == NULL)
return temp;
prev = NULL
cur = root;
while (cur != NULL)
{
prev = cur;
 cur = (item < cur->info)?
  cur->llink : cur->rlink;
}
if (item < prev->info)
 prev->llink = temp;
 else
prev->rlink = temp
 return root;
}
void display (NODE root, int i)
{
  int j;
 if (root != NULL)
  {
   display (root->rlink, i+1);
   for (j=0; j<i; j++)
   printf ("  ");
   printf ("%d\n", root->info);
   display (root->llink, i+1);
  }
}
NODE delete (NODE root, int item)
```

```
{
NODE cur, parent, q, suc;
if (root == NULL)
{
printf("empty \n");
return root;
}
Parent = NULL;
cur = root;
while (cur != NULL && item != cur -> info)
{
parent = cur;
cur = (item < cur -> info) ?
cur -> llink : cur -> r link;
}
if (cur == NULL)
{
printf ("not found\n");
return root;
}
if cur -> llink == NULL)
q = cur -> r link;
else if (cur -> r link == NULL)
q = cur -> llink;
else
{
suc = cur -> r link;
while (suc -> l link != NULL)
suc = suc -> l link;
suc -> l link = cur -> llink;
q = cu -> r link
}
```

```
if (parent == NULL)
    return q;
if (cur == parent -> llink)
   parent -> llink = q;
else
parent -> rlink = q
  freenode(cur)
   return root;
}
void  preorder (NODE root)
{
  if (root != NULL)
  {
   printf ("%d\n", root -> info)
   preorder (root -> llink);
   pre order (root -> rlink)
}
}
void  Postorder (NODE root)
  {
  if (root != NULL)
   {
   post order (root -> llink);
   post order (root -> rlink);
   printf ("%d\n", root -> info);
  }
}
void inorder (NODE root)
{
 if (root != NULL)
 {
```

```
inorder (root → llink);
printf("%d \n", root → info);
 inorder (root → rlink);
 }
}
void main ():
{
 int item, choice;
 NODE root = NULL;
  for (;;)
  {
printf("\n1. insert \n2. display \n3. pre \n4.
      post\n5. in \n6. delete \n7. exit \n");
  printf ("Enter the choice \n");
   scanf ("%d", & choice);
   switch (choice)
 {
  case 1 : printf (" enter the item \n");
       scanf ("%d", & item);
       root = insert (root, item);
       break;
  case 2 : display (root, 0);
       break;
  case 3 : preorder (root);
   break;
  case 4 : post order (root);
       break;
  case 5 : inorder (root);
   break;
  case 6 : printf ("Enter the item \n");
       scanf ("%d", & item);
       root = delete (root, item);
```

```
        break;
    default : exit(0);
        break;
    }
  }
}
```