Week-5

PAGE NO :
DATE :

1. Demonstrate Min Stack

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

typedef struct {
    int val;
    int min;
} Node;

typedef struct {
    Node * stack;
    int top, capacity;
} MinStack;

MinStack* minStackCreate() {
 MinStack * stack = (MinStack*) malloc(sizeof(MinStack));
 stack -> stack = (Node*) malloc(sizeof(Node)*100);
 stack -> top = -1;
 stack -> capacity = 100;
 return stack;
}

void minStackPush(MinStack* obj, int val) {
  if(obj -> top == -1) {
  obj -> stack[++(obj->top)].val = val;
  obj -> stack[obj -> top].min = val;
  }
  else {
     obj -> stack[++(obj -> top)].val = val;
     obj -> stack[obj -> top].min = (val < obj ->stack
        (val < obj->stack[obj -> top-1].min)?
        val : obj -> stack[obj -> top-1].min;
     }
}
```

```
void minStackPop(MinStack* obj) {
    if(obj->top >= 0) {
        obj->top--;
    }
}

int minStackPop(MinStack* obj) {
    if(obj->top >= 0) {
        obj->top--;
    }
}

int minStackTop(MinStack* obj) {
    return obj->stack[obj->top].val;
}

int minStackGetMin(MinStack* obj) {
    return obj->stack[obj->top].min;
}
```

Output: [null, null, null, -3, null, 0, -2]

Solved

Lab-6 (Week-5)

~~Multiplexer~~

1. WAP to implement Single Linked List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include<stdio.h>
#include<stdlib.h>
 struct node
  { int data;
    struct node *next;
  };
 struct node *s1 = NULL;
 struct node *s2 = NULL;
 struct node *start = NULL;
 struct node *create(struct node*);

void sort();
struct node *concatenate(struct node*, struct node*);
void reverse();
 void display(struct node*);

int main()
{  int option;
   struct node *a = NULL;
   do{
       printf("\n*** Main Menu ***\n");
    printf("1.Create a linked list\n");
    printf("2. Create two linked lists for
          concatenation\n3. Sort \n4.Concatenate
\n5.Reverse\n 6. Display linked list\n7.Display
 Concatenated link list\n8. Exit\n");
```

```
printf("\n Enter an option to perform the
    following operations: ");
scanf("%d", &option);

switch(option)
{ case 1 : start = create(start);
        printf("\n Linked List created successfully");
        break;
  case 2: printf("\n Linked List 1: \n");
        s1 = create(s1);
        printf("\n Linked List 2: \n");
        s2 = create(s2);
        printf("\n Linked Lists created successfully");
        break;
  case 3: sort();
        printf("\n Linked List sorted \n");
        break;
  case 4: a = concatenate(s1, s2);
        printf("\n Linked Lists concatenated successfully");
        break;
  case 5: reverse();
        printf("\n Linked List reversed \n"); break;
  case 6: printf("\n Elements in the linked list \n");
        display(start); break;
  case 7: printf("\n Elements in the linked list
        after concatenation: \n");
        display(a); break;
}while(option != 8);

return 0;
}
```

PAGE NO :
DATE :

```
struct node *create(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf("Enter -1 to exit \n");
    printf("\n Enter the date : ");
    scanf("%d", &num);
while(num!=-1)
{
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    if(start==NULL)
    { start = new_node;
      new_node->next = NULL;
    }
    else
    { ptr = start;
      while(ptr->next!=NULL)
      ptr = ptr->next;
      ptr->next = new_node;
      new_node->next = NULL;
    }
    printf("Enter the data : ");
    scanf("%d", &num);
}
return start;
}
void sort()
{ struct node *i, *j;
  int temp;
  for(i=start; i->next!=NULL; i=i->next)
  { for(j=i->next; j!=NULL; j=j->next)
    {
      if(i->data > j->data)
      {
```

```
        temp = i->data;
        i->data = j->data;
        j->data = temp;
    }}}
}
struct node *concatenate(struct node *t1, struct
    node *t2){
struct node *ptr;
 ptr = t1;
 while(ptr->next != NULL)
 {   ptr = ptr->next;
 }
ptr->next = t2;
return t1;
}
void reverse()
{ struct node *prev = NULL;
  struct node *next = NULL;
  struct node *cur = start;
while(cur != NULL)
{   next = cur->next;
    cur->next = prev;
    prev = cur;
    cur = next;
} start = prev;
}
void display(struct node *p)
{   struct node *ptr;
    ptr = p;
    while(ptr != NULL)
  {  printf("\t %d", ptr->data);
     ptr = ptr->next;
  }
    printf("\n");
}
```

PAGE NO :
DATE :

Output : **** Main Menu ****

1. Create a Linked List
2. Create two linked lists for concatenation
3. Sort
4. Concatenate
5. Reverse
6. Display Linked list
7. Display concatenated linked list

Enter an option to perform the following operations: 1
Enter -1 to exit
Enter the data : 2
Enter the data : 4
" " " : 6
" " " : 8
" " " : 10
Enter the data : -1
Linked List create successfully

**** Main Menu ****

Enter an option to perform the following operation: 5
Linked List reversed

**** Main Menu ****

Enter an option to perform the following operation: 6
Element in the linked list
10 8 6 4 2

**** Main Menu ****

Enter an option to perform the following operation: 4
Linked Lists concatenated successfully

**** Main Menu ****

Enter an option to perform the following operation : 7
Elements in the linked list after concatination:
18 78 64 24 84

**** Main Menu ****

Enter an option to perform the following operation: 8

2. Queue implementation using singly linked list

```
#include<stdio.h>
#include<stdlib.h>
 struct node
 { int data;
   struct node *next;
 };
 struct node *start = NULL;
 void enqueue();
 void dequeue();
 void display();
 int main()
  {  int val, option;
    do {
     printf("\n Enter the number to perform following
          operations \n 1. Enqueue \n 2. Dequeue\n
  3. Display \n 4. Exit \n");
 scanf("%d", &option);
  switch(option)
   {  case 1: enqueue(); break;
      case 2: dequeue(); break;
       case 3: display(); break;
    }
 } while(option != 4);
 return 0;
 }
 void enqueue()
 { struct node *new_node;
   int num;
   printf("Enter the data\n");
   scanf("%d", &num);
   new_node = (struct node*) malloc(sizeof(struct node
```

```
new_node -> data = num;
new_node -> next = start;
start = new_node;
}
void dequeue()
{ struct node *ptr, *preptr;
  ptr = start;
  if(start == NULL)
  { printf("Stack is empty");
    exit(0);
  }
  else if(start -> next == NULL)
  { start = start -> next;
    Printf("\n Element popped from the
stack is: %d\n", ptr -> data);
    free(ptr);
  }
  else {
    while(ptr -> next != NULL)
  { preptr = ptr;
    ptr = ptr -> next;
  }
  preptr -> next = NULL;
  printf("\n Element popped from the stack is: %d\n",
                ptr -> data);
  free ptr;
  }
}
void display()
{ struct node *ptr;
  ptr = start;
  while(ptr != NULL)
  { printf("\t %d", ptr -> data);
    ptr = ptr -> next;
```

printf("\n");
}

Output: Enter the number to perform following operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the data
18
Enter the number to perform following operations.
1
Enter the data
78
Enter the number the perform following operations:
3
78 18
Enter the number the perform following operations:
2
Element popped from the stack is : 18

PAGE NO :
DATE :

3. Stack Implementation using Linked list

```
#include <stdio.h>
#include <stdlib.h>
 struct node
 { int data;
  struct node * next;
 };
 struct node *start = NULL;
 void push();
 void pop();
 void display();
int main()
 {      int val, option;
     do {
       printf("\nEnter the number to perform
     following operations\n1. Push\n2. Pop\n3. Display
 \n4. Exit\n");
 scanf("%d", &option);
 switch(option)
 { case 1: push(); break;
     case 2: pop(); break;
     case 3: display(); break;
   }
 } while(option != 4);
return 0;
}
void push()
 { struct node * new_node;
   int num;
  printf("Enter the data\n")
   scanf("%d", &num);
   new_node = (struct node*) malloc(sizeof(struct node));
   new_node -> data = num;
   new_node -> next = start;
 }  start = new_node;
```

```
void pop()
{ struct node *ptr;
    ptr = start;
  if(start == NULL)
  { printf("Stack is empty\n");
    exit(0);
  }
  else {
      ptr = start;
      start = ptr -> next;
      printf("\nElement popped from the stack
      is : %d\n", ptr->data);
      free(ptr);
  }
}
void display()
 { struct node *ptr;
     ptr = start;
   while(ptr != NULL)
   {
      printf("\t%d", ptr->data);
       ptr = ptr -> next;
   }
   printf("\n");
 }
```

Output:

Enter the number to perform following operations

1. Push
2. Pop
3. Display
4. Exit

1

Enter the data

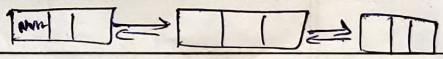
2

PAGE NO :
DATE :

Enter the number to perform following operations :
1
Enter the data
4
Enter the number to perform following operations :
3
4 2
Enter the number to perform following operations :
2
Element popped from the stack is : 4.

8/4/24
29/4/24

Deletion by searching the value in doubly linked list



```
ptr = head;

while (ptr != NULL)
{   if (ptr->data == val)
    {   ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        free(ptr);
        ptr = NULL;
    }
    ptr = ptr->next;
}
```

Also learn reverse, sort, concat, merge
how to delete duplicate elements

sort & concat

DS-Lab (Week 7)

2. WAP to implement doubly link list with primitive operations

(a) Create a node

```
#include<stdio.h>
#include<stdlib.h>
struct node{ struct node *prev; struct node *next; int data;};

struct node* addToEmpty(struct node* head, int d)
{ head = malloc(sizeof(struct node));
  head->data = d; head->prev = NULL;
  head->next = NULL;
  return head;
};
struct node* addAtEnd(struct node *head, int d
{ struct node *p = malloc(sizeof(struct node));
  p->prev = NULL;
  p->data = data; p->next = NULL;
  struct node *temp = head;
  while(temp->next != NULL)
  { temp = temp->next;
  }
  temp->next = p;
  p->prev = temp;
  return head;
};
struct node* createList(struct node *head)
{ int n, ele;
  printf("\nEnter the number of elements");
  scanf("%d", &n);
  if(n==0)
  { printf("\n No node is created");
    scanf("%d", &ele);
  head = malloc(sizeof(struct node));
  head->data = ele;
  head->prev = NULL; head->next = NULL;
  return head; }
```

```
printf("\n enter the element");
scanf("%d", &ele);
head = addToEmpty(head, ele);
for(int i=2; i<=n; i++)
{ printf("\n enter the element");
  scanf("%d", &ele);
  head = addAtEnd(head, ele);
}
printf("\n DL created"); return head; };

struct node * addBeforepos( struct node * head, int
pos, int data)
{ struct node * temp=head;
  struct node * temp2=NULL;
  struct node * p= malloc(sizeof(struct node));
  p->prev =NULL;
  p->data = data;
  p->next = NULL;
  while (pos>1)
  { temp = temp->next;
    pos--;
  }
temp2 = temp->prev;
temp2->next = p;
p->prev = temp2;
temp->prev = p;
p->next = temp;
printf("\n Insertion Success");
return head;
};
struct node * delete_val( struct node * head, int val)
{ int flag=0;
  struct node *ptr = head;
  while (ptr != NULL)
```

```
if(ptr->data==val)
  { flag==1;
    ptr->prev->next=ptr->next;
    ptr->next->prev=ptr->prev;
    free(ptr);
    ptr=NULL; printf("\n Deleted Value %d",val);
return head;
}
ptr=ptr->next; }
if(flag==0)
 { printf("\n No Deletion"); }
return head;
};
int main()
{ int pos; int choice, val;
struct node *head=NULL;
 do{ printf("\n 1.Create \n 2.Insert before \n
      3.Delete \n Enter the choice \n");
scanf("%d", &choice);
   switch(choice)
    { case 1: head=createList(head); break;
      case 2: printf("\n enter pos:");
              scanf("%d", &pos);
       head=addbeforepos(head, pos, 32); break;
 case 3: printf("\n enter value to be deleted");
          scanf("%d", &val);
          head=delete_val(head, val); break;
case4: printf("\n Exciting"); break;
}
}while(choice!=4);

 struct node *ptr=head;
 while(ptr!=NULL)
 { print("\n %d", ptr->data);
    ptr=ptr->next;
 }
return 0;
```