2. Lists, Stacks and Queues (Implementation)

DATA STRUCTRES AND ALGORITHMS [17ECSC204]

PRAKASH HEGADE

SCHOOL OF CSE | KLE Tech

1. Stack

```
#include <stdio.h>
#include <stdlib.h>
#define STACKSIZE 5
#define TRUE 1
#define FALSE 0

struct stack
{
    int top;
    int items[STACKSIZE];
};
typedef struct stack STACK;

void push(STACK *);
void pop(STACK *);
void print(STACK *);
void peek(STACK *);
int empty(STACK *);
int full(STACK *);

int main()
{
    STACK S;
    S.top = -1;
    int choice=0;

    while(1) {
        printf("\n Menu\n");
        printf("1-PUSH\n");
        printf("2-POP\n");
        printf("3-PEEK\n");
        printf("4-PRINT\n");
        printf("5-EXIT\n");
        printf("Enter your choice\n");
        scanf("%d", &choice);
        switch(choice) {
            case 1: push(&S);
                    break;
            case 2: pop(&S);
                    break;
            case 3: peek(&S);
                    break;
            case 4: print(&S);
                    break;
            case 5: printf("Terminating\n");
                    exit(1);
        }
    }
    return 0;
}
```

```
int full(STACK *S) {
    if(S->top == STACKSIZE-1)
        return TRUE;
    else
        return FALSE;
}

void push(STACK *S) {
    if(full(S)){
        printf("Stack full\n");
        return;
    }

    int x;
    printf("Enter the item to be pushed\n");
    scanf("%d", &x);

    S->top++;
    S->items[S->top] = x;

}

int empty(STACK * S) {
   if(S->top == -1)
        return TRUE;
   else
        return FALSE;
}

void pop(STACK *S) {
    if(empty(S)){
        printf("Stack Empty\n");
        return;
    }

    int x;
    x = S->items[S->top];
    printf("Popped item is %d\n", x);
    S->top--;
}

void peek(STACK *S) {
    if(empty(S)){
        printf("Stack Empty\n");
        return;
    }

    int x;
    x = S->items[S->top];
    printf("Peeked item is %d\n", x);
}
```

```
void print(STACK *S) {
   if(empty(S)){
      printf("Stack Empty\n");
      return;
   }

   int i;
   for(i = S->top; i >= 0; i--)
      printf("| %d |\n", S->items[i]);
}
```

2. Linear Queue

```
#include <stdio.h>
#include <stdlib.h>
#define QUEUESIZE 5
#define TRUE 1
#define FALSE 0

struct queue
{
   int front;
   int rear;
   int items[QUEUESIZE];
};
typedef struct queue QUEUE;

void enqueue(QUEUE *);
void dequeue(QUEUE *);
void display(QUEUE *);
int full(QUEUE *);
int empty(QUEUE *);

int main()
{
   QUEUE q;
   q.front = 0;
   q.rear = -1;

   int choice;
```

```
   while(1){
      printf("MENU\n");
      printf("1-Enqueue\n");
      printf("2-Dequeue\n");
      printf("3-Display\n");
      printf("4-Exit\n");

      printf("\nEnter your choice\n ");
      scanf("%d", &choice);

      switch(choice){
         case 1: enqueue(&q);
               break;
         case 2: dequeue(&q);
               break;
         case 3: display(&q);
               break;
         case 4: printf("Terminating\n");
               exit(0);
      }
   }

   return 0;
}
```

```
/// Function Name: full
/// Description:   checks if rear end has reached max position
/// Input Param:   Pointer to queue
/// Return type:   TRUE if queue full, FALSE otherwise
int full(QUEUE *q) {
   if(q->rear == QUEUESIZE - 1)
      return TRUE;
   else
      return FALSE;
}
```

```
/// Function Name: enqueue
/// Description:   enqueue an item inside queue
/// Input Param:   Pointer to queue
/// Return type:   void
void enqueue(QUEUE *q) {
   if(full(q)){
      printf("Queue full\n");
      return;
   }
   int x;
   printf("Enter the enqueue item\n");
   scanf("%d", &x);

   q->rear++;
   q->items[q->rear] = x;
}

/// Function Name: empty
/// Description:
///            item                              f      r
///            ----------------------------------------------------------------
///            initial                           0      -1
///            one insertion/deletion            1      0
///            ...
///            n insertions/deletions            n      n-1`
/// Input Param:   Pointer to queue
/// Return type:   TRUE if queue empty, FALSE otherwise
int empty(QUEUE *q) {
   if(q->front > q->rear)
      return TRUE;
   else
      return FALSE;
}

/// Function Name: dequeue
/// Description:   dequeue an item from queue
/// Input Param:   Pointer to queue
/// Return type:   void
void dequeue(QUEUE *q) {
   if(empty(q)){
      printf("Empty queue\n");
      return;
   }
   int x;
   x = q->items[q->front];
   printf("Dequeued Item is %d\n", x);
   q->front++;
}
```

```
/// Function Name: display
/// Description:   display the items from queue
/// Input Param:   Pointer to queue
/// Return type:   void
void display(QUEUE *q) {
    if(empty(q)) {
        printf("Empty Queue\n");
        return;
    }
    int i;
    for(i = q->front; i<= q->rear; i++)
        printf("%d\n", q->items[i]);
}
```

3. Circular Queue

```
#include <stdio.h>
#include <stdlib.h>
#define MAXQUEUE 5
#define TRUE    1
#define FALSE   0
struct cqueue {
    int front;
    int rear;
    int items[MAXQUEUE];
};
typedef struct cqueue CQUEUE;

void Enqueue(CQUEUE *);
void Dequeue(CQUEUE *);
int empty(CQUEUE *);
int full(CQUEUE *);
void display(CQUEUE *);

int main() {
    int choice =0;
    CQUEUE cq;
    cq.front = MAXQUEUE - 1;
    cq.rear = MAXQUEUE - 1;
    while(1)  {
        printf("\n**** MENU ****\n");
        printf("1 - Enqueue\n");
        printf("2 - Dequeue\n");
        printf("3 - Display\n");
        printf("4 - Exit\n");
        printf("**************\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                Enqueue(&cq);
                break;
            case 2: Dequeue(&cq);
                break;
            case 3: display(&cq);
                break;
            case 4: printf("Program will Exit. \n");
                exit(0);
        }
    }
    return 0;
}
```

```
int empty(CQUEUE *pcq) {
    if(pcq->front == pcq->rear)
        return TRUE;
    else
        return FALSE;
}
```

```
int full(CQUEUE *pcq){
    if(pcq->front == (pcq->rear+1) % MAXQUEUE)
        return TRUE;
    else
        return FALSE;
}
```

```
void Enqueue(CQUEUE *pcq) {
  if(full(pcq)){
    printf("Queue full\n");
  }
  else {
    int x;
    printf("Enter item to insert:\t");
    scanf("%d", &x);
    pcq->rear = (pcq->rear+1) % MAXQUEUE;
    pcq->items[pcq->rear] = x;
    printf("Insertion Successful\n");
  }
}

void Dequeue(CQUEUE *pcq)
{
  if(empty(pcq)){
    printf("Queue empty\n");
  }
  else {
    int x;
    pcq->front=(pcq->front+1)% MAXQUEUE;
    x = pcq->items[pcq->front];
    printf("%d Dequeued\n", x);
  }
}

void display(CQUEUE *pcq)
{
  if(empty(pcq))
    printf("Queue Empty\n");
  else {
    int i;
    printf("Queue Contents are:\n");
    i = (pcq->front + 1) % MAXQUEUE;
    while(i != pcq->rear) {
      printf("%d\n", pcq->items[i]);
      i = (i+1) % MAXQUEUE;
    }
    printf("%d\n", pcq->items[i]);
    printf("\n");
  }
}
```

4. Linear Queue as Double Ended Queue

```
#include <stdio.h>
#include <stdlib.h>
#define MAXQUEUE 5
#define TRUE   1
#define FALSE  0
struct dqueue {
  int front;
  int rear;
  int items[MAXQUEUE];
};
typedef struct dqueue DQUEUE;

void EnqueueRear(DQUEUE *);
void DequeueFront(DQUEUE *);
void EnqueueFront(DQUEUE *);
void DequeueRear(DQUEUE *);
void Display(DQUEUE *);
int empty(DQUEUE *);
int full(DQUEUE *);

int main() {
    int choice =0, x = 0;
    DQUEUE q;
    q.front=0;
    q.rear=-1;
    while(1) {
      printf("\n**** MENU ****\n");
      printf("1 - Enqueue Rear\n");
      printf("2 - Enqueue Front\n");
      printf("3 - Dequeue Rear\n");
      printf("4 - Dequeue Front\n");
      printf("5 - Display\n");
      printf("6 - Exit\n");
      printf("**************\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

        switch(choice) {
          case 1: EnqueueRear(&q);
              break;
          case 2: EnqueueFront(&q);
              break;
          case 3: DequeueRear(&q);
              break;
          case 4: DequeueFront(&q);
              break;
          case 5: Display(&q);
              break;
          case 6: printf("Program will Exit. \n");
              exit(0);
        }
    }
    return 0;
}
```

```
// Increment Rear and Insert
void EnqueueRear(DQUEUE *pdq)
{
  if(full(pdq))
    printf("Queue full\n");
  else {
    int x;
    printf("Enter Enqueue Item\n");
    scanf("%d", &x);
    (pdq->rear)++;
    pdq->items[pdq->rear]=x;
  }
}

// Decrement front and Insert
void EnqueueFront(DQUEUE *pdq)
{
 // We can insert at front only if front is not equal to zero
  if (pdq->front != 0){
    int x;
    printf("Enter Enqueue Item\n");
    scanf("%d", &x);
    (pdq->front)--;
    pdq->items[pdq->front]=x;
  }
  else
    printf("Enqueue Invalid\n");
}

// Remove and Increment front
void DequeueFront(DQUEUE *pdq)
{
  if(empty(pdq))
    printf("Empty Queue\n");
  else {
    int x;
    x = pdq->items[pdq->front];
    (pdq->front)++;
    printf("%d Dequeued\n", x);
  }
}

// Remove and Decrement rear
void DequeueRear(DQUEUE *pdq)
{
  if(empty(pdq))
    printf("Empty Queue\n");
  else
  {
    int x;
    x = pdq->items[pdq->rear];
    (pdq->rear)--;
    printf("%d Dequeued\n", x);
  }
}

int empty(DQUEUE *pdq)
{
  if(pdq->front > pdq->rear)
    return TRUE;
  else
    return FALSE;
}

int full(DQUEUE *pdq)
{
  if(pdq->rear == MAXQUEUE-1)
    return TRUE;
  else
    return FALSE;
}

void Display(DQUEUE *pdq)
{
  if(empty(pdq))
    printf("Empty Queue\n");
  else{
    int i= 0;
    printf("Queue Items are:\n");
    for(i=pdq->front; i<=pdq->rear; i++)
      printf("%d\n", pdq->items[i]);
  }
}
```

6. Circular Queue as DECK

```
#include <stdio.h>
#include <stdlib.h>
#define MAXQUEUE 5
#define TRUE 1
#define FALSE 0
struct dequeue {
  int items[MAXQUEUE];
  int front;
  int rear;
};
typedef struct dequeue DEQUEUE;
```

```
void EnqueueFront(DEQUEUE *);
void DequeueFront(DEQUEUE *);
void EnqueueRear(DEQUEUE *);
void DequeueRear(DEQUEUE *);
int empty(DEQUEUE *);
int full(DEQUEUE *);
void Display(DEQUEUE *);
int main() {
   int choice =0;
   DEQUEUE dq;
   dq.front=MAXQUEUE-1;
   dq.rear=MAXQUEUE-1;
   while(1)
   {
      printf("\n **** MENU ****\n");
      printf("1 - Enqueue Front\n");
      printf("2 - Enqueue Rear\n");
      printf("3 - Dequeue Front\n");
      printf("4 - Dequeue Rear\n");
      printf("5 - Display\n");
      printf("6 - Exit\n");
      printf("***************\n");
      printf("Enter your choice:\t");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1: EnqueueFront(&dq);
               break;
         case 2: EnqueueRear(&dq);
               break;
         case 3: DequeueFront(&dq);
               break;
         case 4: DequeueRear(&dq);
               break;
         case 5: Display(&dq);
               break;
         case 6: printf("Program will exit\n");
               exit(0);
         }
   }
   return 0;
}


int empty(DEQUEUE *pdq) {
   if(pdq->front == pdq->rear)
      return TRUE;
   else
      return FALSE;
}

int full(DEQUEUE *pdq){
   if(pdq->front == (pdq->rear+1)% MAXQUEUE)
      return TRUE;
   else
      return FALSE;
}

void EnqueueFront(DEQUEUE *pdq) {
   if(full(pdq))
      printf("Queue Full\n");
   else {
      int x;
      printf("Enter Enqueue Item\n");
      scanf("%d", &x);
      pdq->items[pdq->front] = x;
      pdq->front = (pdq->front - 1 + MAXQUEUE)% MAXQUEUE;
   }
}

void EnqueueRear(DEQUEUE *pdq) {
   if(full(pdq))
      printf("Queue Full\n");
   else {
        int x;
        printf("Enter Enqueue Item\n");
        scanf("%d", &x);
        pdq->rear = (pdq->rear + 1) % MAXQUEUE;
        pdq->items[pdq->rear] = x;
    }
}

void DequeueFront(DEQUEUE *pdq) {
    if(empty(pdq))
        printf("Queue Empty\n");
    else {
        int x;
        pdq->front = (pdq->front+1) % MAXQUEUE;
        x = pdq->items[pdq->front];
        printf("%d Dequeued\n", x);
    }
}

void DequeueRear(DEQUEUE *pdq) {
    if(empty(pdq))
        printf("Queue Empty\n");
    else {
        int x;
        x= pdq->items[pdq->rear];
        pdq->rear = (pdq->rear - 1 + MAXQUEUE) % MAXQUEUE;
        printf("%d Dequeued\n", x);
    }
}

void Display(DEQUEUE *pdq) {
    if(empty(pdq))
        printf("Queue Empty\n");
    else {
        int i;
        printf("Queue Contents are:\n");
        i = (pdq->front + 1) % MAXQUEUE;
        while(i != pdq->rear) {
            printf("%d\n", pdq->items[i]);
            i = (i+1) % MAXQUEUE;
        }
        printf("%d\n", pdq->items[i]);
    }
}
```

7. Singly Linked List Implementation

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

```
typedef struct node NODE;

// Maintain the number of nodes in the list in a global variable
int currnodes = 0;

NODE *  insert_at_start(NODE * start);
NODE *  insert_at_end(NODE * start);
NODE *  insert_at_position(NODE * start);
NODE *  delete_from_start(NODE * start);
NODE *  delete_from_end(NODE * start);
NODE *  delete_from_position(NODE * start);
NODE * getnode();
void getdata(NODE *);
void display_list(NODE *start);

int main() {
  NODE * start=NULL;
  int choice = 0;
  while(1) {
    printf("\n\n* * * * * *   Menu  * * * * * *\n");
    printf("1. Insert node at start\n");
    printf("2. Insert node at End\n");
    printf("3. Insert node at a Position\n");
    printf("4. Delete node from start\n");
    printf("5. Delete node from end\n");
    printf("6. Delete node from a Position\n");
    printf("7. Display List\n");
    printf("8. Exit\n");
    printf("* * * * * * ********* * * * * * *\n");
    printf("Enter your choice:\n");
    scanf("%d", &choice);
    switch (choice){
      case 1: start = insert_at_start(start);
          break;
      case 2: start = insert_at_end(start);
          break;
      case 3: start = insert_at_position(start);
          break;
      case 4: start = delete_from_start(start);
          break;
      case 5: start = delete_from_end(start);
          break;
      case 6: start = delete_from_position(start);
          break;
      case 7: display_list(start);
          break;
      case 8: printf("Exiting program\n\n");
          exit(0);
    }
 }
  return 0;
}
```

```
// Function to allocate the memory for the struct node
NODE * getnode() {
  NODE * newnode;
  newnode = (NODE *) malloc(sizeof(NODE));
  // If the memory allocation fails malloc will return NULL
  if(newnode == NULL)
    printf("Memory allocation failed.\n");

  // Return the newnode at any case
  return newnode;
}

void getdata(NODE * newnode) {
  // Get the information from the user, Initialize the next pointer to NULL
  printf("Enter the information of node:\n");
  scanf("%d", &newnode->data);
  newnode->next = NULL;
}

NODE *  insert_at_start(NODE * start) {
  NODE * newnode;
  newnode = getnode();
  // Memory allocation failed
  if(newnode == NULL)
    return start;
  // Get the data from the user
  getdata(newnode);

  // If the list is empty, newnode is the start of the list
  if(start == NULL)
    start = newnode;
  else {
    // Add the newnode at the beginning and update the start
    newnode->next = start;
    start = newnode;
  }
  // Increment currnodes, print a message and return updated start
  currnodes++;
  printf("%d is inserted at front of the list\n\n", newnode->data);
  return start;
}

NODE *  delete_from_start(NODE * start) {
  if(start == NULL)
    printf("List is Empty!\n");
  else  {
    NODE * tempnode = start;
    start=start->next;
    printf("%d is deleted from front of the list\n", tempnode->data);
    free(tempnode);     currnodes--;
  }
  return start;
}
```

```
NODE *  insert_at_end(NODE * start) {
   NODE * newnode, * nextnode;
   newnode = getnode();
   if(newnode == NULL)
      return start;
   getdata(newnode);

   if(start == NULL)
      start = newnode;
   else {
      nextnode = start;
      while(nextnode->next != NULL)
            nextnode = nextnode->next;

      nextnode->next = newnode;
   }
   currnodes++;
   printf("%d is inserted at the end of the list\n\n", newnode->data);
   return start;
}

NODE *  delete_from_end(NODE * start) {
   NODE *prevnode, *nextnode;
   if(start == NULL)
      printf("List is Empty!\n");
   else {
      if(currnodes == 1)  { // or start->next == NULL
         nextnode = start;
         start=NULL;
      }
      else {
         nextnode = start;
         prevnode = NULL;
         while(nextnode->next!=NULL)  {
            prevnode = nextnode;
            nextnode = nextnode->next;
         }
         prevnode->next = NULL;
      }
      printf("%d is deleted from end of the list.\n", nextnode->data);
      free(nextnode);
      currnodes--;
   }
   return start;
}

NODE *  insert_at_position(NODE * start) {
            // Refer Activity Book
}

NODE *  delete_from_position(NODE * start) {
            // Refer Activity Book
}
```

```
void display_list(NODE *start) {
   NODE * tempnode;
   if(start == NULL)
      printf("List is Empty!\n");
   else  {
      tempnode = start;
      printf("The list contents are:\n");
      while(tempnode != NULL) {
         printf("%d  -->  ", tempnode->data);
         tempnode = tempnode->next;
      }
      printf("NULL\n");
   }
}
```

8. Doubly Linked List Implementation

```
#include <stdio.h>
struct node {
  int data;
  struct node *next;
  struct node *prev;
};
typedef struct node NODE;
int currnodes = 0;

// Function prototypes same as singly linked list

int main() {
   NODE * start;
   start = NULL;
   int choice  = 0;
     .....
    // Will be same as that of singly Linked List
     .....
   return 0;
}

NODE * getnode() {
   NODE * newnode;
   newnode = (NODE *)malloc(sizeof(NODE));

   if(newnode == NULL)
      printf("Memory Allocation Failed\n");
   return newnode;
}

void getdata(NODE * newnode) {
   printf("Enetr the information for linked list\n");
   scanf("%d", &newnode->data);
   newnode->next = NULL;
   newnode->prev = NULL;
}
```

```
NODE * insert_at_front(NODE * start) {
   NODE * newnode;
   newnode = getnode();
   if(newnode == NULL)
      return start;
   getdata(newnode);

   if(start == NULL)
      start = newnode;
   else  {
      newnode->next= start;
      start->prev=newnode;
      start = newnode;
   }
   currnodes++;
   printf("%d info is inserted at the start of the doubly linked list\n", newnode->data);
   return start;
}

NODE * insert_at_end(NODE * start) {
   NODE * newnode, *tempnode;
   newnode = getnode();
   if(newnode == NULL)
      return start;
   getdata(newnode);

   if(start == NULL)
      start = newnode;
   else {
      tempnode = start;
      while(tempnode->next != NULL)
         tempnode = tempnode->next;

      tempnode->next = newnode;
      newnode->prev = tempnode;
   }
   currnodes++;
   printf("%d info is inserted at the End of the doubly linked list\n", newnode->data);
   return start;
}

NODE * insert_at_position(NODE * start) {
            // Refer Activity Book
}

NODE * delete_from_start(NODE * start) {
   NODE * tempnode;
   if(start == NULL)
      printf("List is empty\n");
   else {
      if(currnodes == 1) // or start->next == NULL
      {
         tempnode = start;
         start = NULL;
      }
      else {
         tempnode = start;
         start = start->next;
         start->prev = NULL;
      }
      printf("Node %d deleted from the start of the Doubly linked list\n", tempnode->data);
      free(tempnode);
      currnodes--;
   }
   return start;
}

NODE * delete_from_end( NODE * start) {
   NODE * tempnode, *prevnode;
   if(start == NULL)
      printf("List is empty\n");
   else {
      if(currnodes == 1)
      {
         tempnode = start;
         start = NULL;
      }
      else
      {
         tempnode = start;
         while(tempnode->next != NULL)
            tempnode = tempnode->next;

         prevnode = tempnode;
         prevnode = prevnode->prev;
         prevnode->next = NULL;
      }
      printf("Node %d deleted from the end of the Doubly linked list\n", tempnode->data);
      free(tempnode);
      currnodes--;
   }
   return start;
}

NODE * delete_from_position(NODE * start) {
            // Refer Activity Book
}

void display_list(NODE * start) {
   NODE * tempnode;
   if(currnodes == 0)
      printf("List Empty\n");
   else {
      tempnode = start;
      printf("The list contents are:\n");
      printf("\nNULL <--> ");
```

```
    while(tempnode != NULL) {
      printf(" %d <--> ", tempnode->data);
      tempnode = tempnode->next;
    }
    printf("NULL\n");
  }
}
```

9. Circular Linked List Implementation

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node *next;
};
typedef struct node NODE;
int currnodes = 0;

NODE *  insert_at_start(NODE * last);
NODE *  insert_at_end(NODE * last);
NODE *  delete_from_start(NODE * last);
NODE *  delete_from_end(NODE * last);
NODE * getnode();
void getdata(NODE *);
void display_list(NODE *last);

int main( ) {
  NODE * last=NULL;
  int choice = 0;
  while(1) {
    printf("\n\n* * * * * * *   Menu  * * * * * * *\n");
    printf("1. Insert node at start\n");
    printf("2. Insert node at End\n");
    printf("3. Delete node from start\n");
    printf("4. Delete node from end\n");
    printf("5. Display List\n");
    printf("6. Exit\n");
    printf("* * * * * * * ********* * * * * * * *\n");
    printf("Enter your choice:\n");
    scanf("%d", &choice);

    switch (choice) {
      case 1: last = insert_at_start(last);
          break;
      case 2: last = insert_at_end(last);
          break;
      case 3: last = delete_from_start(last);
          break;
      case 4: last = delete_from_end(last);
          break;
      case 5: display_list(last);
          break;
```

```
        case 6: printf("Exiting program\n\n");
            exit(0);
      }
   }
   return 0;
}

NODE * getnode() {
   NODE * newnode;
   newnode = (NODE *) malloc(sizeof(NODE));
   if(newnode == NULL)
      printf("Memory allocation failed.\n");
   return newnode;
}

void getdata(NODE * newnode) {
   printf("Enter the information of node:\n");
   scanf("%d", &newnode->data);
   newnode->next = NULL;
}

NODE *  insert_at_start(NODE * last) {
   NODE * newnode;
   newnode = getnode();
   if(newnode == NULL)
      return last;
   getdata(newnode);

   if(last == NULL)
      last = newnode;
   else
      newnode->next = last->next;

   last->next = newnode;
   currnodes++;
   printf("%d is inserted at front of the circular list\n\n", newnode->data);
   return last;
}

NODE *  insert_at_end(NODE * last) {
   NODE * newnode;
   newnode = getnode();
   if(newnode == NULL)
      return last;
   getdata(newnode);

   if(last == NULL)
      last = newnode;
   else
      newnode->next = last->next;

   last->next = newnode;
   currnodes++;
   printf("%d is inserted at the end of the list\n\n", newnode->data);
   return newnode;
}

NODE *  delete_from_start(NODE * last) {
   NODE * tempnode;

   if(last == NULL)
      printf("List is Empty!\n");
   else  {
      if(last->next == last) {
         tempnode = last;
         last = NULL;
      }
      else {
         tempnode = last->next;
         last->next = tempnode->next;
      }
      printf("%d is deleted from front of the list\n\n", tempnode->data);
      free(tempnode);
      currnodes--;
   }
   return last;
}

NODE *  delete_from_end(NODE * last) {
   NODE *prevnode = NULL;
   if(last == NULL){
      printf("List is Empty!\n");
      return last;
   }
   else {
      if(currnodes == 1) {
         printf("%d is deleted from end of the list.\n", last->data);
         free(last);
         currnodes--;
         return NULL;
      }
      else {
         prevnode = last->next;
         while(prevnode->next != last)
            prevnode = prevnode->next;

         prevnode->next = last->next;
         printf("%d is deleted from end of the list.\n", last->data);
         free(last);
         currnodes--;
         return prevnode;
      }
   }
}
```

```
void display_list(NODE *last)
{
   NODE * tempnode;
   if(last == NULL)
      printf("List is Empty!\n");
   else
   {
      tempnode = last->next;
      printf("The list contents are:\n");
      while(tempnode != last)
      {
         printf("%d  -->  ", tempnode->data);
         tempnode = tempnode->next;
      }
      printf("%d  -->  ", tempnode->data);
   }
}
```

~*~*~*~*~*~*~