Single linked list

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

void insert_Front();
void insert_End();
void insert_Any();
void delete_Front();
void delete_End();
void delete_Any();
void print();
void main(){
   int choice = 1;
   while(choice<9){
         printf("SINGLE LINKED LIST OPERATIONS\n");
         for(int k=0;k<29;k++){
                  printf("%c",'-');
         }printf("\n");
         printf("1.insert at beginning\n2.insert at end\n3.insert at any position\n4.delete from
beginning\n5.delete from end\n6.delete from any position\n7.print\n8.exit\n\n");
         printf("Enter your choice = ");
         scanf("%d",&choice);
       switch(choice){
         case 1: insert_Front();
                  break;
         case 2: insert_End();
                  break;
         case 3: insert_Any();
                  break;
         case 4: delete_Front();
                  break;
         case 5: delete_End();
                  break;
         case 6: delete_Any();
                  break;
         case 7: print();
                  break;
         case 8: exit(0);
                  break;
         default:printf("invalid key...program terminated !!!\n");
                  break;
       }
   }
}
void insert_Front(){
   struct node *ptr;
   int item;
```

```
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL){
        printf("overflow\n");
    }else{
        printf("\nEnter value = ");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("Node inserted successfully\n\n");
    }
}
void insert_End(){
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL){
        printf("overflow\n");
    }else{
        printf("Enter value = ");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL){
            ptr -> next = NULL;
            head = ptr;
            printf("Node inserted successfully\n\n");
        }else{
            temp = head;
            while (temp -> next != NULL){
                temp = temp -> next;
            }
            temp->next = ptr;
            ptr->next = NULL;
            printf("Node inserted successfully\n\n");

        }
    }
}
void insert_Any(){
    int i,l,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL){
        printf("overflow\n");
    }else{
        printf("\nEnter element value = ");
        scanf("%d",&item);
        ptr->data = item;
        printf("Enter the location after which you want to insert = ");
        scanf("%d",&l);
        temp=head;
        for(i=0;i<l;i++){
```

```
            temp = temp->next;
            if(temp == NULL){
               printf("insertion failed\n\n");
               return;
            }
         }
         ptr ->next = temp ->next;
         temp ->next = ptr;
         printf("Node inserted successfully\n\n");
      }
   }
   void delete_Front(){
      struct node *ptr;
      if(head == NULL){
         printf("underflow\n\n");
      }else{
         ptr = head;
         head = ptr->next;
         free(ptr);
         printf("Node deleted from the begining\n\n");
      }
   }
   void delete_End(){
      struct node *ptr,*ptr1;
      if(head == NULL){
         printf("underflow\n\n");
      }
      else if(head -> next == NULL){
         head = NULL;
         free(head);
         printf("Only node of the list deleted\n\n");
      }else{
         ptr = head;
         while(ptr->next != NULL){
            ptr1 = ptr;
            ptr = ptr ->next;
         }
         ptr1->next = NULL;
         free(ptr);
         printf("Deleted Node from the last ...\n\n ");
      }
   }
   void delete_Any(){
      struct node *ptr,*ptr1;
      int l,i;
      printf("Enter the location to delete node = ");
      scanf("%d",&l);
      ptr=head;
      for(i=0;i<l;i++){
         ptr1 = ptr;
         ptr = ptr->next;
         if(ptr == NULL){
```

```
            printf("deletion failed\n\n");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("Deleted node %d ",l+1);
}
void print(){
    struct node *ptr;
    ptr = head;
    if(ptr == NULL){
        printf("underflow\n");
    }else{
        printf("The current Linked List\n\n");
        while (ptr!=NULL){
            printf("%d\n",ptr->data);
            ptr = ptr -> next;
        }
        printf("\n");
    }
}
```

Queue

```
#include<stdio.h>
#include<stdlib.h>
int main(){
            int i,max;
            int Q[100];
            int front = -1;
            int rear = -1;
            int item;
            int ch=1;
            printf("QUEUE OPERATIONS\n");
            for(i=0;i<16;i++){
                        printf("%c",'-');
            }
            printf("\n");
            printf("Enter size of the Queue = ");
            scanf("%d",&max);
            while(ch<5){
                        printf("1.Enqueue\n2.Dequeue\n3.Print the Queue\n4.exit\n\n");
                        printf("Enter your choice = ");
                        scanf("%d",&ch);
                        switch(ch){
                                    case 1 : if(rear == max-1){
                                                            printf("Queue overflow/Queue is full\nredirecting to
main menu...\n\n");
                                                            break;
                                                }
                                                else if(front==-1 && rear==-1){
                                                            front=rear=0;
```

```
                                        printf("Enter the number to insert first = ");
                                        scanf("%d",&item);
                                        Q[rear]=item;
                                }
                                else{
                rear = rear+1;
                printf("Enter the number to insert = ");
                scanf("%d",&item);
                Q[rear] = item;
                                }
                                break;
                        case 2 : if(front==-1 && rear==-1){
                                        printf("Queue underflow/Queue is empty\nredirecting
to main menu...\n\n");
                                }
                else if(front==rear){
                                        item = Q[front];
                                        printf("The last element deleted = %d\n",Q[front]);
                                        front=rear=-1;
                }
                                else{
                                        item = Q[front];
                                        printf("The element that deleted  = %d\n",Q[front]);
                                        front = front+1;
                                }
                                break;
                        case 3 : if(front == -1){
                                        printf("Queue is empty\nredirecting to main menu...\n\
n");
                                        break;
                                }
                                else{
                                        printf("The current queue\n");
                                        for(i=front;i<=rear;i++){
                                                printf("%d\n",Q[i]);
                                        }
                                }
                                printf("\n");
                                break;
                        case 4 : printf("exiting the program...\n");
                                exit(0);
                        default: printf("Something went wrong !!!\nprogram terminated...\n");
                                exit(0);

                }
        }
        return 0;
}
```

circular Q

```
#include<stdio.h>
#include<stdlib.h>
int main(){
	int i,max;
	int op=1;
	int front=-1;
	int rear=-1;
	int item;
	int Q[max];
	printf("CIRCULAR QUEUE OPERATIONS\n");
	for(i=0;i<25;i++){
		printf("%c",'-');
	}
	printf("\n");
	printf("Enter size of circular Queue = ");
	scanf("%d",&max);
	while(op<4){
		printf("1.Enqueue\n2.Dequeue\n3.Print CircularQ\n4.exit\n\n");
		printf("Choice = ");
		scanf("%d",&op);
		switch(op){
			case 1 : if((rear+1)%max == front){
					printf("Queue overflow/full\n");
					break;
				}
				else if(front==-1){
					front=rear=0;
					printf("Enter the number to insert first = ");
					scanf("%d",&item);
					Q[rear]=item;
				}
				else{
					rear = (rear+1)%max;
					printf("Enter the element to insert = ");
					scanf("%d",&item);
					Q[rear]=item;
				}
				break;
			case 2 : if(front==-1){
					printf("Queue underflow/empty\n");
				}
				else if(front==rear){
					item=Q[front];
					printf("The last element deleted = %d\n",Q[front]);
					front=rear=-1;
				}
				else{
					item=Q[front];
					printf("The element deleted = %d\n",Q[front]);
					front=(front+1)%max;

				}
```

```
                                        break;
                        case 3 : if(front==-1){
                                                printf("Queue is empty\nredirecting to main menu...\
n");
                                        }
                                        else{
                                                /*for(i=front;i<=rear;i++){
                                                        printf("%d\n",Q[i]);
                                                }*/
                                                while(front != rear){
                                                        printf("%d\n",Q[front]);
                                                        front=(front+1)%max;
                                                }
                                                printf("%d\n",Q[rear]);
                                                printf("\n");
                                        }
                                        break;
                        case 4 : printf("exiting the program...\n");
                                        exit(0);
                        default: printf("Something went wrong !!!\nterminating the program...\n");
                                        exit(0);
                }
        }
        printf("\n");
        return 0;
}
```

DEQ

```
#include <stdio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
//  insert_front function will insert the value from the front
void insert_front(int x)
{
   if((f==0 && r==size-1) || (f==r+1))
   {
      printf("Overflow");
   }
   else if((f==-1) && (r==-1))
   {
      f=r=0;
      deque[f]=x;
   }
   else if(f==0)
   {
      f=size-1;
      deque[f]=x;
   }
   else
   {
```

```
        f=f-1;
        deque[f]=x;
    }
}

// insert_rear function will insert the value from the rear
void insert_rear(int x)
{
    if((f==0 && r==size-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
        r=0;
        deque[r]=x;
    }
    else if(r==size-1)
    {
        r=0;
        deque[r]=x;
    }
    else
    {
        r++;
        deque[r]=x;
    }

}

// display function prints all the value of deque.
void display()
{
    int i=f;
    printf("\nElements in a deque are: ");

    while(i!=r)
    {
        printf("%d ",deque[i]);
        i=(i+1)%size;
    }
     printf("%d",deque[r]);
}

// getfront function retrieves the first value of the deque.
void getfront()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else
```

```
    {
        printf("\nThe value of the element at front is: %d", deque[f]);
    }

}

// getrear function retrieves the last value of the deque.
void getrear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at rear is %d", deque[r]);
    }

}

// delete_front() function deletes the element from the front
void delete_front()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d", deque[f]);
        f=-1;
        r=-1;

    }
     else if(f==(size-1))
     {
         printf("\nThe deleted element is %d", deque[f]);
         f=0;
     }
     else
     {
          printf("\nThe deleted element is %d", deque[f]);
          f=f+1;
     }
}

// delete_rear() function deletes the element from the rear
void delete_rear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
```

```
    }
    else if(f==r)
    {
       printf("\nThe deleted element is %d", deque[r]);
       f=-1;
       r=-1;

    }
     else if(r==0)
     {
        printf("\nThe deleted element is %d", deque[r]);
        r=size-1;
     }
     else
     {
         printf("\nThe deleted element is %d", deque[r]);
         r=r-1;
     }
}

int main()
{
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);
    display();  // Calling the display function to retrieve the values of deque
    getfront();  // Retrieve the value at front-end
    getrear();  // Retrieve the value at rear-end
    delete_front();
    delete_rear();
    display(); // calling display function to retrieve values after deletion
    return 0;
}
```

priorityQ

```
    #include <stdio.h>
    #include <stdio.h>
    int heap[40];
    int size=-1;

    // retrieving the parent node of the child node
    int parent(int i)
    {

       return (i - 1) / 2;
    }

    // retrieving the left child of the parent node.
    int left_child(int i)
```

```
{
 return i+1;
}
// retrieving the right child of the parent
int right_child(int i)
{
 return i+2;
}
// Returning the element having the highest priority
int get_Max()
{
   return heap[0];
}
//Returning the element having the minimum priority
int get_Min()
{
   return heap[size];
}
// function to move the node up the tree in order to restore the heap property.
void moveUp(int i)
{
   while (i > 0)
   {
       // swapping parent node with a child node
       if(heap[parent(i)] < heap[i]) {

       int temp;
       temp=heap[parent(i)];
       heap[parent(i)]=heap[i];
       heap[i]=temp;


   }
      // updating the value of i to i/2
      i=i/2;
   }
}

//function to move the node down the tree in order to restore the heap property.
void moveDown(int k)
{
   int index = k;

   // getting the location of the Left Child
   int left = left_child(k);

   if (left <= size && heap[left] > heap[index]) {
       index = left;
   }

   // getting the location of the Right Child
   int right = right_child(k);
```

```
    if (right <= size && heap[right] > heap[index]) {
       index = right;
    }

    // If k is not equal to index
    if (k != index) {
      int temp;
      temp=heap[index];
      heap[index]=heap[k];
      heap[k]=temp;
        moveDown(index);
    }
}

// Removing the element of maximum priority
void removeMax()
{
    int r= heap[0];
    heap[0]=heap[size];
    size=size-1;
    moveDown(0);
}
//inserting the element in a priority queue
void insert(int p)
{
    size = size + 1;
    heap[size] = p;

    // move Up to maintain heap property
    moveUp(size);
}

//Removing the element from the priority queue at a given index i.
void delete(int i)
{
    heap[i] = heap[0] + 1;

   // move the node stored at ith location is shifted to the root node
    moveUp(i);

    // Removing the node having maximum priority
    removeMax();
}
int main()
{
    // Inserting the elements in a priority queue

    insert(20);
    insert(19);
    insert(21);
    insert(18);
```

```
    insert(12);
    insert(17);
    insert(15);
    insert(16);
    insert(14);
    int i=0;

  printf("Elements in a priority queue are : ");
  for(int i=0;i<=size;i++)
    {
      printf("%d ",heap[i]);
    }
    delete(2); // deleting the element whose index is 2.
    printf("\nElements in a priority queue after deleting the element are : ");
    for(int i=0;i<=size;i++)
    {
      printf("%d ",heap[i]);
    }
 int max=get_Max();
    printf("\nThe element which is having the highest priority is %d: ",max);


    int min=get_Min();
       printf("\nThe element which is having the minimum priority is : %d",min);
    return 0;
 }
```

stack

```
#include<stdio.h>
#include<stdlib.h>
int main(){
        int stack[10];
        int top=-1;
        int item;
        int max;
        int op=1;
        printf("STACK OPERATIONS\n");
        for(int i=0;i<16;i++){
                printf("%c",'-');
        }
        printf("\n");
        printf("Enter the size of stack = ");
        scanf("%d",&max);
        while(op<5){
                printf("1.push operation\n2.pop operation\n3.print the stack\n4.exit\n\n");
                printf("Choice = ");
                scanf("%d",&op);
                switch(op){
                case 1 : if(top == max-1){
              printf("stack overflow/stack is full\nredirecting to main menu...\n\n");
              break;
```

```
            }
            else{
              printf("Enter the number to push = ");
              scanf("%d",&item);
              top=top+1;
              stack[top]=item;
            }
            break;
                case 2 : if(top == -1){
              printf("stack underflow/stack is empty\nredirecting to main menu...\n\n");
              break;
            }
            else{
              item=stack[top];
              top=top-1;
              printf("The element that poped = %d\n",item);
              break;
            }
        case 3 : if(top == -1){
              printf("stack underflow/stack is empty\nredirecting to main menu...\n\n");
              break;
            }
            else{
              printf("The current stack\n");
                 for(int i=0;i<=top;i++){
                    printf("%d \n",stack[i]);
                 }
                 printf("\n");
            }
            break;
        case 4 : printf("exiting the program\n\n");
            exit(0);
                 default: printf("Something wrong!!!\nexiting the program\n");
                          exit(0);
          }}
          return 0;
}
```