Week - 3

Implementation of circular Queue

```
#include <stdio.h>
#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;
int isFull() {
  if ((front == rear+1) || (front == 0 &&
      rear == SIZE-1))
    return 1;
  return 0;
}
int isEmpty() {
  if (front == -1)
    return 1;
  return 0;
}

Void EnQueue (int element) {
  if (isFull())
  Printf ("\n Queue is full!!\n");
  Else {
  if (front == -1) {
    front = 0;
  rear = (rear + 1) % SIZE;
  Item[rear] = element;
```

```
        Printf ("\n inserted -> %d", element);
      }
    }
  }
  int deQueue (){
    int element;
    if (isEmpty()) {
      Printf ("\n Queue is Empty!! \n");
      return (-1);
    } else {
      element = items [front];
      if (front == rear) {
        front = -1;
        rear = -1;
      } else {
        front = (front + 1) % SIZE;
      }
    Printf ("\n Delete element -> %d \n",
        element);
    return (element);
    }
  }
```

```
Void display () {
int i;
if (isEmpty ())
Printf ("in empty Queue \n");
else {
Printf ("\n Front -> %d", front);
Printf ("\n Items -> ");
for (i = front; i != rear; i = (i+1) % SIZE){
Printf ("%d", items [i]);
}
Printf ("%d", items [i]);
Printf ("\n Rear -> %d \n", rear);
}
}
int main () {
En Queue (1);
En Queue (2);
En Queue (3);
En Queue (4);
En Queue (5);
display ();
```

```
dequeue(6);
dequeue(7);

display();

return 0;

}
```

Output

Inserted → 6

Inserted → 7

Front → 2

Items → 3 4 5 6 7.

Rear → 1

11/1/24

Week - 3

Insertion Operation on Linked List

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

Struct node
{
    int data;
    Struct node * next;
};

Void display();
Void insert_bugin();
Void insert_End();
Void insert_POS();
Struct node * Ptr;
if (head == NULL)
  {
    Print ("List is Empty \n");
    return;
  }
Else
  {
Ptr = head;
While (Ptr != NULL)
{
```

```
        Printf ("%d \n", Ptr -> data);
        Ptr = Ptr -> next;
        }
    }
}
void insert_begin()
{
    Struct node* temp;
    temp = (struct node*) malloc
        (sizeof (struct node));
    Printf ("Enter the value to be
            Entered \n");
    Scanf ("%d", &temp -> data);
    temp -> next = NULL;
    if (head = temp);
    if (head = NULL)
    head = temp;
    else {
        temp -> next = head;
        head = temp;
        }
    }
void insert_end()
{
    Struct node * temp, * Ptr;
    temp = (struct node*) malloc
        (sizeof (struct node));
    Printf ("Enter the value to be
            Inserted \n");
```

```
        temp -> next = NULL;
        if (head = NULL)
        {
            head = temp;
        }
        else
        {
        ptr = head;
        while (ptr -> next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> next = temp;
        }
};
void insert_pos()
{
 int pos, i;
 struct node *temp, *ptr;
 Printf("enter the position");
 Scanf("%d", &pos);
 temp = (struct node *) malloc (size of
                (struct node));
Printf("enter the value to be
        inserted \n");
```

```
switch (choice)
{
  case 1:
    insert_begine();
    break;
  case 2:
    insert_end();
    break;
  case 3;
    insert_pos();
    break;
  case 4;
    display();
    break;
  case 5:
    exit(0);
    break;
  default;
    Print f("invalid choice \n");
    break;
  }
}
}
```

output.

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Print ~~list~~ display
5. Exit

Enter your choice : 1
Enter data to insert at beginning : 35

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. display
5. Exit

Enter your choice : 3
Enter data to insert ~~at Beginning~~ : 57
Enter the position to insert at : 2

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. display ~~Exit~~
5. Exit

Enter your choice 2
Enter ~~data~~ to ~~detet~~ insert at End : 15

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. display
5. Exit

Enter your Choice : 4
linked list : 35 57 15

Enter your Choice : 5

Exit

```c
#include <stdio.h>
#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;

int isFull() {
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
        return 1;
    return 0;
}

int isEmpty() {
    if (front == -1)
        return 1;
    return 0;
}

void enQueue(int element) {
    if (isFull())
        printf("\n Queue is full!! \n");
    else {
        if (front == -1) {
            front = 0;
            rear = (rear + 1) % SIZE;
            items[rear] = element;
            printf("\n Inserted->%d", element);
        } else {
            rear = (rear + 1) % SIZE;
            items[rear] = element;
            printf("\n Inserted->%d", element);
        }
    }
}
```

```c
int deQueue() {
    int element;
    if (isEmpty()) {
        printf("in Queue is empty !! \n");
        return (-1);
    } else {
        element = items[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % SIZE;
        }

        printf("\n Deleted elements ->%d \n", element);
        return (element);
    }
}

void display() {
    int i;
    if (isEmpty())
        printf(" in Empty Queue \n");
    else {
        printf("\n Front->%d", front);
        printf("\n Items->");
        for (i = front; i != rear; i = (i + 1) % SIZE) {
            printf("%d ", items[i]);
        }

        printf("%d", items[i]);
        printf("\n Rear->%d\n", rear);
    }
}
```

```c
    else {
        printf("\n Front->%d", front);
        printf("\n Items->");
        for (i = front; i != rear; i = (i + 1) % SIZE) {
            printf("%d ", items[i]);
        }

        printf("%d", items[i]);
        printf("\n Rear->%d\n", rear);
    }
}

int main() {
    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);

    display();

    deQueue();
    deQueue();

    display();

    enQueue(6);
    enQueue(7);

    display();

    return 0;
}
```

```
Inserted->1
Inserted->2
Inserted->3
Inserted->4
Inserted->5
Front->0
Items->1 2 3 4 5
Rear->4

Deleted elements ->1

Deleted elements ->2

Front->2
Items->3 4 5
Rear->4

Inserted->6
Inserted->7
Front->2
Items->3 4 5 6 7
Rear->1
```

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


struct Node {
    int data;
    struct Node* next;
};


void insertAtBeginning(struct Node** head, int newData) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));


    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }


    newNode->data = newData;
    newNode->next = *head;


    *head = newNode;
}


void insertAfter(struct Node* prevNode, int newData) {

    if (prevNode == NULL) {
        printf("Previous node cannot be NULL\n");
        return;
    }
```

```
        return;
    }


    struct Node* lastNode = *head;
    while (lastNode->next != NULL) {
        lastNode = lastNode->next;
    }


    lastNode->next = newNode;
}


void insertAtPosition(struct Node** head, int newData, int position) {

    if (position < 1) {
        printf("Invalid position\n");
        return;
    }


    if (position == 1) {
        insertAtBeginning(head, newData);
        return;
    }


    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));


    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
```

```c
    newNode->data = newData;


    struct Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }


    if (temp == NULL) {
        printf("Position exceeds the length of the list\n");
        free(newNode);
        return;
    }


    newNode->next = temp->next;


    temp->next = newNode;
}


void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));


    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }


    newNode->data = newData;
    newNode->next = prevNode->next;


    prevNode->next = newNode;
}


void insertAtEnd(struct Node** head, int newData) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));


    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }


    newNode->data = newData;
    newNode->next = NULL;


    if (*head == NULL) {
        *head = newNode;
```

```c
int choice, data, position;

while (1) {
    printf("1. Insert at Beginning\n");
    printf("2. Insert at End\n");
    printf("3. Insert at Position\n");
    printf("4. Print List\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to insert at the beginning: ");
            scanf("%d", &data);
            insertAtBeginning(&head, data);
            break;
        case 2:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;
        case 3:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position to insert at: ");
            scanf("%d", &position);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            printf("Linked list: ");
            printList(head);
            break;
        case 5:
```

```
            break;
        case 2:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;
        case 3:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position to insert at: ");
            scanf("%d", &position);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            printf("Linked list: ");
            printList(head);
            break;
        case 5:

            while (head != NULL) {
                struct Node* temp = head;
                head = head->next;
                free(temp);
            }
            return 0;
        default:
            printf("Invalid choice\n");
    }
}

return 0;
}
```

```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Print List
5. Exit
Enter your choice: 1
Enter data to insert at the beginning: 35
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Print List
5. Exit
Enter your choice: 3
Enter data to insert: 57
Enter position to insert at: 2
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Print List
5. Exit
Enter your choice: 2
Enter data to insert at the end: 15
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Print List
5. Exit
Enter your choice: 4
Linked list: 35 57 15
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Print List
5. Exit
Enter your choice: 5
```