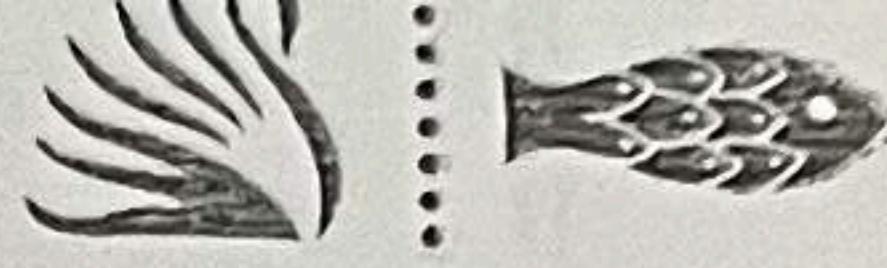


```
#include <stdio.h>
#define MAX 10
int q[MAX], front = -1, rear = -1;
void insert();
int delete_element();
int peek();
void display();
void insert()
{
    int num;
    printf("In enter the no. to be inserted into queue");
    scanf("%d", &num);
    if (rear == MAX - 1)
        printf("Overflow");
    else if (front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
        rear++;
    q[rear] = num;
}
int delete_element()
{
    int val;
    if (front == -1 || front > rear)
    {
        printf("In underflow");
    }
}
```

```
return -1 ;  
}  
else {  
    val = q[front];  
    front++;  
    if (front > rear)  
    {  
        rear = -1;  
        front = -1;  
    }  
    return val;  
}  
}  
  
int peek ()  
{  
    if (front == -1 || front > rear)  
    {  
        printf ("In queue is empty");  
        return -1;  
    }  
    else  
    return q[front];  
}  
  
void display ()  
{  
    int i;  
    printf ("In ");  
    if (front == -1 || front > rear)  
        printf ("In queue is empty");  
    else {  
        for (i = front; i <= rear; i++)
```

```
    printf("It : %d\n", q[i]);  
}  
}  
  
void main() {  
    int val, choice;  
    while (1)  
    {  
        printf("Enter your choice In 1.Insert\n2.Delete In 3.Peek\n");  
        scanf("%d", &choice);  
        switch (choice) {  
            case 1: insert();  
                break;  
            case 2: val = delete_element();  
                if (val != 1)  
                    printf("The deleted num is : %d", val);  
                break;  
            case 3: val = peek();  
                if (val != -1)  
                    printf("The first value in queue is : %d", val);  
                break;  
            case 4: display();  
                break;  
            case 5: exit(0);  
            default: printf("Default, wrong choice");  
        }  
    }  
}
```



## CIRCULAR QUEUE:

,

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

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

```
#define SIZE 5
```

```
int items[SIZE], rear = -1, front = -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("In queue is full");
```

```
}
```

```
else {
```

```
if (front == -1)
```

```
front = 0;
```

~~rear = (rear + 1) % SIZE;~~~~items[rear] = element;~~

```
printf("%d is inserted", element);
```

```
}
```

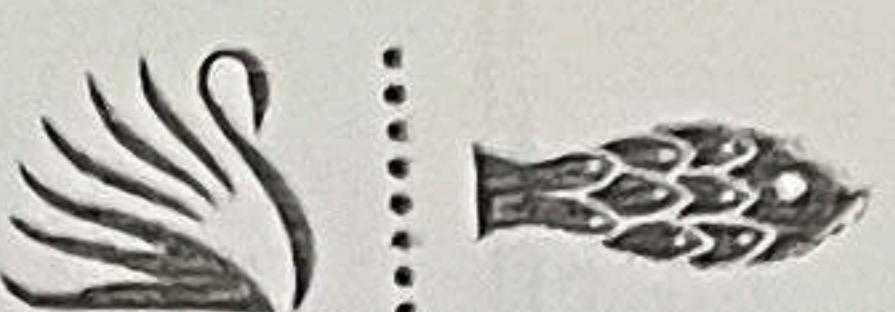
```
}
```

```
int dequeue ()  
{  
    int value;  
    if (isEmpty ())  
    {  
        printf ("In queue is empty");  
        return -1;  
    }  
    else  
    {  
        front = (front + 1) % size  
    }  
    return (value);  
}
```

```
}  
void display ()  
{  
    int i;  
    if (isEmpty ())  
        printf ("queue is empty");  
    else  
    {  
        printf ("In front position = %d \n", front);  
        for (i=front; i!=rear; i=(i+1)%size)  
        {  
            printf ("%d \t", items[i]);  
        }  
        printf ("%d \t", items[i]);  
    }
```

```
}
```

```
void main ()  
{  
    int choice, element;
```



while(1)

{

printf ("1. Insert 2 Delete 3. Display 4. Exit );

printf ("Enter choice );

scanf ("%d", &choice);

switch (choice) :

{

case 1 : printf ("In Enter the element to insert");

scanf ("%d", &element);

enqueue (element);

break;

case 2 : element = deQueue ();

if (element != -1)

printf ("Element is deleted ", element);

break;

case 3 : ~~display ()~~

break;

case 4 : exit (0);

default : printf ("In Invalid choice ");

}

}

## OUTPUT :

1. The operation to be performed

1. Insert

2. Delete

3. Display

choice : 1

5 deleted

\* LINKED LIST (SINGLT)  
INSERTION at (i) Beginning (ii) specified (iii) and positions

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

```
struct Node* createNode (int data) {
    struct Node* newNode = (struct Node*) malloc (sizeof(struct Node));
    newNode->next = NULL;
    newNode->data = data;
    return newNode;
}
```

```
struct Node* insertAtBeginning (struct Node* head, int data) {
```

```
    struct Node* newNode = createNode (data);
    newNode->next = head;
    return newNode;
}
```

```
struct Node* insertAtPosition (struct Node* head, int data,
                               int position) {
```

```
    struct Node* newNode = createNode (data);
```

```
    if (position == 1 || head == NULL) {
        newNode->next = head;
        return newNode;
    }
```

```
struct Node* current = head;
for (int i=1; i< position - 1 && current != NULL; i++) {
    current = current->next;
}
if (current == NULL) {
    printf("Invalid position!\n");
    return head;
}
newNode->next = current->next;
current->next = newNode;
return head;
}
```

```
struct Node* insertAtEnd (struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
}
```

```
struct Node* current = head;
while (current->next != NULL) {
    current = current->next;
}
current->next = newNode;
return head;
}
```

```
void displayList (struct Node* head) {
    printf("Linked list:");
    while (head != NULL) {
        printf("\t%d", head->data);
        head = head->next;
    }
}
```

```
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position;
    do {
        printf("In Menu:\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at any position\n");
        printf("3. Insert at the end\n");
        printf("4. Display");
        printf("\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter data to insert at beginning:");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 2: printf("Enter data to insert at the end");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 3: printf("Enter data to insert");
                scanf("%d", &data);
                printf("Enter position to insert at");
                scanf("%d", &position);
                head = insertAtPosition(head, data, position);
                break;
            case 4: displayList(head);
                break;
            case 5: printf("Exiting the program");
                break;
        }
    } while (choice != 5);
}
```

print("Invalid choice! Please enter valid option"),  
}

} while (choice != 5);  
return 0;  
}

O/P: Menu:

- 1. Insert at Beginning
- 2. Insert at Any Position
- 3. Insert at End
- 4. Display
- 5. Exit

Enter your choice : 1

Enter data to insert at Beginning : 3

→ Menu

- 1. Insert at beginning
- 2. Insert at any position
- 3. Insert at end
- 4. Display
- 5. Exit

Enter your choice : 4

Linked list : 3 4 5

→

Menu

- 1. Insert beginning
- 2. Insert any position
- 3. Insert end
- 4. Display
- 5. Exit

Enter your choice : 2

Enter data to insert : 4

Enter position to insert at : 2

→ Menu

- 1. Insert at beginning
- 2. Insert at any position
- 3. Insert at end
- 4. Display
- 5. Exit

Enter your choice : 5

Exiting the program.

→

Menu

- 1. Insert at beginning
- 2. Insert at any position
- 3. Insert at end
- 4. Display
- 5. Exit

Enter your choice : 3

Enter data to insert : 5

## Deletion

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

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

struct Node* createNode (int data) {
    struct Node* newNode = (struct Node*) malloc (sizeof (struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
struct Node* insertAtBeginning (struct Node* head, int data) {
    struct Node* newNode = createNode (data);
    newNode->next = head;
    return newNode;
}
```

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

struct Node\* deleteFirstElement in the list

struct Node\* deleteFirstElement ( struct Node\* head ) {

if (head == NULL) {

printf ("List is empty. Cannot delete\n");

return NULL;

}

struct Node\* newHead = head -> next;

free (head);

return newHead;

}

); struct Node\* deleteSpecifiedElement ( struct Node\* head, int data ) {

if (head == NULL) {

printf ("List is empty. Cannot delete\n");

return NULL;

}

if (head -> data == data) {

struct Node\* newHead = head -> next

free (head);

return newHead;

}

struct Node\* current = head;

while (current -> next != NULL & current -> next -> data != dat

}{

current = current -> next;

{

if (current -> next == NULL) {

printf ("Element not found. Cannot delete.\n");

return head;

}

struct Node\* temp = current -> next;

current -> next = temp -> next;

free (temp);

return head;

```
struct Node* deleteLastElement (struct Node* head) {  
    if (head == NULL) {  
        printf ("List is empty. Cannot delete in");  
        return NULL;  
    }  
    if (head->next == NULL) {  
        free(head)  
        return NULL;  
    }  
    struct Node* current = head;  
    while (current->next->next != NULL) {  
        current = current->next;  
    }  
    free(current->next);  
    current->next = NULL;  
    return head;  
}  
  
int main () {  
    struct Node* head = NULL;  
    int choice, data;  
    do {  
        printf ("1. Insert in Menu In ");  
        printf ("2. Insert at the beginning In ");  
        printf ("3. Delete first element ");  
        printf ("4. Delete specified element ");  
        printf ("5. Delete last element ");  
        printf ("6. Display In ");  
        printf ("7. Exit In ");  
        printf ("Enter your choice ");  
        scanf ("%d", &choice);  
        switch (choice) {
```



```
case 1 : printf ("Enter data to insert at beginning ");
scanf ("%d", &data);
head = insertAtBeginning (head, data);
break;

case 2 : head = deleteFirst Element (head );
break;

case 3 : printf ("Enter element to delete ");
scanf ("%d", &data);
head = delete specified Element (head, data);
break;

case 4 : head = delete Last Element (head );
break;

case 5 : display list (head );
break;

case 6 : printf ("Exiting program");
break;

default : printf ("Invalid choice");
}

} while (choice != 6);
return 0;
}
```

O/P: Linked List : 3 4 5 6 7 8

Menu

1. Insert at Beginning
2. Delete First element
3. Delete specified element
4. Delete last element
5. Display
6. Exit

Enter your choice : 2

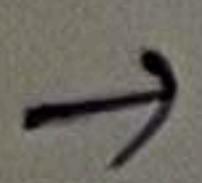
Menu

1. Insert at beginning
2. Delete first
3. Delete specific
4. Delete last
5. Display exit
6. exit

Enter your choice : 3

Enter element to delete : 2

76 1 3



Menu

1. Insert
2. Delete first
3. Delete specific
4. Delete last
5. Display
6. exit

Enter your choice : 4

Enter 06

Menu

1. Insert
2. ~~Display~~ Delete first
3. Delete specific
4. Delete last
5. Display
6. ~~exit~~

PUSH

Enter your choice : 5

linked list : 7 6 1

POP

22/01/24

29/01/22

⑧