Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2022-2023

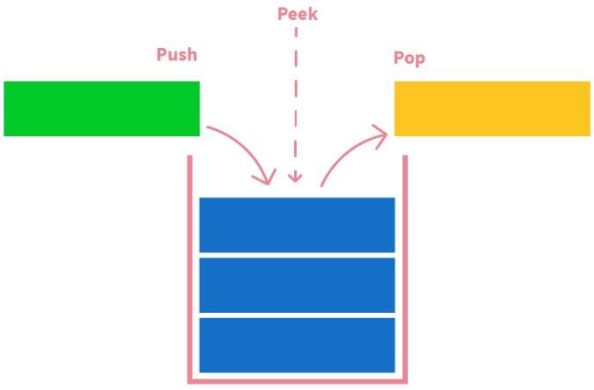
Name – Prerna Sunil Jadhav **SAP ID - 60004220127**

Experiment No – 03

AIM: Implementation of Stack and queue using doubly linked List.

STACK

Stack is an ordered list in which insertion and deletion are done at one end, where the end is generally called the top. The last inserted element is available first and is the first one to be deleted. Hence, it is known as Last In, First Out (LIFO) or First In, Last Out (FILO). In a stack, the element at the top is known as the top element.



Algorithm:

```
Push()
1.  If the stack is empty then take a new node, add data to it and assign “null” to its previous and next pointer as it is the first node of the DLL.
2.  Assign top and start as the new node. Otherwise, take a new node, add data to it and assign the “previous” pointer of the new node to the “top” node earlier and next as “null”.
3.  Further, update the “top” pointer to hold the value of the new node as that will be the top element of the stack now.

Pop()
1.  If the stack is empty, then print that stack is empty, Otherwise, assign top ->prev -> next as “null” and assign top as top->prev.

PrintStack()
1.  If the stack is empty, then print that stack is empty. Otherwise, traverse the doubly linked list from start to end and print the data of each node.
```

Program:

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

//stack by doubly linked list
typedef struct Node {
    int data;
    struct Node *prev;
    struct Node *next;
}node;
```

```
node *start = NULL;
node *top = NULL;

void push(int d)
{
    node *n;
    n = (node *) malloc(sizeof(node));
    n->data = d;
    if (start == NULL) {
        n->prev = NULL;
        n->next = NULL;
        start = n;
        top = n;
    }
    else {
        top->next = n;
        n->next = NULL;
        n->prev = top;
        top = n;
    }
}

void topelement()
{
    if (start == NULL)
        printf("Stack is empty");
    else
        printf(
            "The element at top of the stack is : %d   \n",
            top->data);
}

void stacksize()
{
    int c = 0;
    if (start == NULL)
        printf("Stack is empty");
    else {
        struct Node* ptr = start;
        while (ptr != NULL) {
            c++;
            ptr = ptr->next;
        }
    }
    printf("Size of the stack is : %d \n ", c);
}
```

```
void pop()
{
    struct Node* n;
    n = top;
    if (start == NULL)
        printf("Stack is empty");
    else if (top == start) {
        top = NULL;
        start = NULL;
        free(n);
    }
    else {
        top->prev->next = NULL;
        top = n->prev;
        free(n);
    }
}

void display()
{
    if (start == NULL)
        printf("Stack is empty");
    else {
        struct Node* ptr = start;
        printf("Stack is : ");
        while (ptr != NULL) {
            printf("%d ", ptr->data);
            ptr = ptr->next;
        }
        printf("\n");
    }
}

void main()
{
    printf("Prerna Sunil Jadhav - 60004220127\n");
    int ch,n;
    display();
    A:printf("\nEnter The operation you want to perform \n1) PUSH \n2) POP \n3) Exit \n");
    scanf("%d", &ch);
    switch (ch)
    {
        case 1:
        {
            printf("\nEnter an element:");
```

```
            scanf("%d", &n);
            push(n);
            printf("\nAfter Pushing ");
            topelement();
            stacksize();
            display();
            goto A;
        }

        case 2:
        {
            pop();
            printf("\nAfter deletion ");
            display();
            goto A;
        }
    }
}
```

OUTPUT:

```
Prerna Sunil Jadhav - 60004220127
Stack is empty
Enter The operation you want to perform
1) PUSH
2) POP
3) Exit
1

Enter an element:12

After Pushing The element at top of the stack is : 12
1) PUSH
2) POP
3) Exit
1

Enter an element:657

After Pushing The element at top of the stack is : 657
Size of the stack is : 2
 Stack is : 12 657

Enter The operation you want to perform
1) PUSH
2) POP
3) Exit
2
```

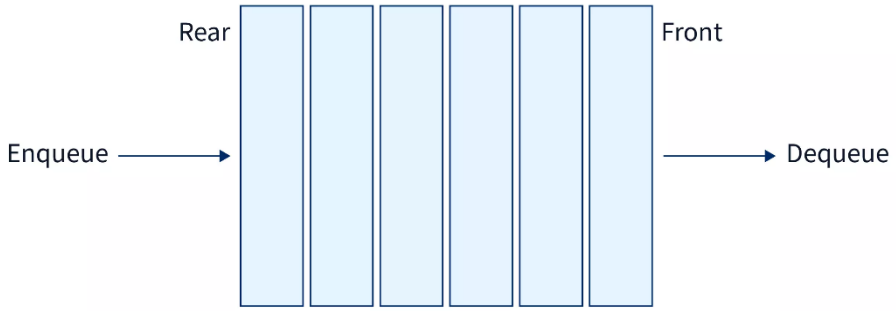
```
After deletion Stack is : 12

Enter The operation you want to perform
1) PUSH
2) POP
3) Exit
2

After deletion Stack is empty
```

QUEUE

The Queue in data structure is an ordered, linear sequence of items. It is a FIFO (First In First Out) data structure, which means that we can insert an item to the rear end of the queue and remove from the front of the queue only. A Queue is a sequential data type, unlike an array, in an array, we can access any of its elements using indexing, but we can only access the element at the front of the queue at a time.



Algorithm:

```
Insertion at front end
1.  Allocate space for a newNode of doubly linked list.
2.  IF newNode == NULL, then
3.  print "Overflow"
4.  ELSE
5.  IF front == NULL, then
6.  rear = front = newNode
7.  ELSE
8.  newNode->next = front
9.  front->prev = newNode
10. front = newNode

Insertion at the end.
1.  IF newNode == NULL, then
2.  print "Overflow"
3.  ELSE
4.  IF rear == NULL, then
5.  front = rear = newNode
6.  ELSE
7.  newNode->prev = rear
8.  rear->next = newNode
```

```
9.  rear = newNode

Deletion from front  end : .
1.  IF front == NULL
2.  print "Underflow"
3.  ELSE
4.  Initialize temp = front
5.  front = front->next
6.  IF front == NULL
7.  rear = NULL
8.  ELSE
9.  front->prev = NULL
10. Deallocate space for temp

Deletion from Rear end :
1.  IF front == NULL
2.  print "Underflow"
3.  ELSE
4.  Initialize temp = rear
5.  rear = rear->prev
6.  IF rear == NULL
7.  front = NULL
8.  ELSE
9.  rear->next = NULL
10. Deallocate space for temp
```

Program:

```c
#include <stdio.h>
#include <stdlib.h>
//Implement queue using doubly linked list
typedef struct QNode
{
    int data;
    struct QNode * next;
    struct QNode * prev;
}QNode;

typedef struct MyQueue
{
    struct QNode * front;
    struct QNode * rear;
    int size;
}MyQueue;
```

```
QNode * getQNode(int data, QNode * prev)
{
    QNode * ref = (QNode * ) malloc(sizeof(QNode));
    if (ref == NULL)
    {
        return NULL;
    }
    ref->data = data;
    ref->next = NULL;
    ref->prev = prev;
    return ref;
}
MyQueue * getMyQueue()
{
    MyQueue * ref = (MyQueue * ) malloc(sizeof(MyQueue));
    if (ref == NULL)
    {
        return NULL;
    }
    ref->front = NULL;
    ref->rear = NULL;
    return ref;
}
void enqueue(MyQueue * ref, int data)
{
    QNode * node = getQNode(data, ref->rear);
    if (ref->front == NULL)
    {
        ref->front = node;
        ref->size = 1;
    }
    else
    {
        ref->rear->next = node;
        ref->size = ref->size + 1;
    }
    ref->rear = node;
}
int isEmpty(MyQueue * ref)
{
    if (ref->size == 0)
    {
        return 1;
    }
    else
    {
```

```
        return 0;
    }
}
int peek(MyQueue * ref)
{
    if (isEmpty(ref) == 1)
    {
        printf("\n Empty Queue");
        return -1;
    }
    else
    {
        return ref->front->data;
    }
}
int isSize(MyQueue * ref)
{
    return ref->size;
}
int dequeue(MyQueue * ref)
{
    if (isEmpty(ref) == 1)
    {
        return -1;
    }
    else
    {
        int data = peek(ref);
        QNode * temp = ref->front;
        if (ref->front == ref->rear)
        {
            ref->rear = NULL;
            ref->front = NULL;
        }
        else
        {
            ref->front = ref->front->next;
            ref->front->prev = NULL;
        }
        ref->size--;
        return data;
    }
}

void printQdata(MyQueue * ref)
{
```

```
    QNode * node = ref->front;
    printf("\n Queue Element\n");
    while (node != NULL)
    {
        printf(" %d", node->data);
        node = node->next;
    }
    printf("\n");
}
int main()
{
    printf("Prerna Sunil Jadhav - 60004220127\n");
    MyQueue * q = getMyQueue();
    enqueue(q, 1);
    enqueue(q, 2);
    enqueue(q, 3);
    enqueue(q, 4);
    enqueue(q, 5);
    printQdata(q);
    printf(" Size : %d", isSize(q));
    printf("\n Dequeue Node : %d", dequeue(q));
    printf("\n Dequeue Node : %d", dequeue(q));
    printf("\n Dequeue Node : %d", dequeue(q));
    printQdata(q);
    printf(" Size : %d\n", isSize(q));
    return 0;
}
```

OUTPUT:

```
Prerna Sunil Jadhav - 60004220127

 Queue Element
 1 2 3 4 5
 Size : 5
 Dequeue Node : 1
 Dequeue Node : 2
 Dequeue Node : 3
 Queue Element
 4 5
 Size : 2
```

Conclusion:

- Stack enables all data to operations at one end only.

- So the only element that can be removed is the element at the top of the stack, and only one item can be read or removed at a given time. The above feature makes it a LIFO (Last-In-First-Out) data structure.
- A Queue is a sequential data structure that follows the FIFO (First In First Out) approach.
- A Static stack is implemented using an array, whereas a dynamic stack is implemented using lists.
- Enqueue and Dequeue are two main operations of a queue, the others being Peek, isEmpty, isFull.
- Queues are used in BFS (Breadth first search) in tree and graph traversals.
- There can be a lot of memory wastage in static queues, as no matter what is the size of the queue, the space occupied by it remains the same.
- Traversing a queue will delete the foremost element on each iteration, eventually, emptying the queue.