

Design and Analysis of Algorithms

NAME : SHUBHAM SHIVRAJ SURYAWANSHI

ASSIGNMENT = 2

1) STACK USING LINKED LIST

```
DAA_PRACTICAL > stack_using_linked_list.cpp > Stack > push(int)
1 //SHUBHAM SHIVRAJ SURYAWANSHI
2 //REG NO 2020BIT004
3 #include <bits/stdc++.h>
4 using namespace std;
5 class Node {
6 public:
7     int data;
8     Node* link;
9     Node(int n)
10    {
11        this->data = n;
12        this->link = NULL;
13    }
14 };
15 class Stack {
16     Node* top;
17
18 public:
19     Stack() { top = NULL; }
20     void push(int data)
21     {
22         Node* temp = new Node(data);
23         if (!temp) {
24             cout << "\nStack Overflow";
25             exit(1);
26         }
27         temp->data = data;
28         temp->link = top;
29         top = temp;
30     }
31 }
```

```

30 }
31 bool isEmpty()
32 {
33     return top == NULL;
34 }
35     int peek()
36 {
37     if (!isEmpty())
38         return top->data;
39     else
40         exit(1);
41 }
42 void pop()
43 {
44     Node* temp;
45     if (top == NULL) {
46         cout << "\nStack Underflow" << endl;
47         exit(1);
48     }
49     else {
50         temp = top;
51         top = top->link;
52         free(temp);
53     }
54 }
55 void display()
56 {
57     Node* temp;
58     if (top == NULL) {
59         cout << "\nStack Underflow";
60         exit(1);
61     }

```

```

DAA_PRACTICAL > stack_using_linked_list.cpp > Stack > push(int)
59         cout << "\nStack Underflow";
60         exit(1);
61     }
62     else {
63         temp = top;
64         while (temp != NULL) {
65             cout << temp->data;
66             temp = temp->link;
67             if (temp != NULL)
68                 cout << " -> ";
69         }
70     }
71 }
72 };
73 int main()
74 {
75     Stack s;
76     s.push(11);
77     s.push(22);
78     s.push(33);
79     s.push(44);
80     s.display();
81     cout << "\nTop element is " << s.peak() << endl;
82     s.pop();
83     s.pop();
84     s.display();
85     cout << "\nTop element is " << s.peak() << endl;
86     return 0;
87 }
88

```

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Top element is 22 > cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if ($?) { g++ stack_u
st } ; if ($?) { .\stack_using_linked_list }PRACTI
44 -> 33 -> 22 -> 11
Top element is 44
22 -> 11
Top element is 22
PS D:\DSA PRACTICE\DAA_PRACTICAL>

```

2) Queue using linked list

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct QNode {
4      int data;
5      QNode* next;
6      QNode(int d)
7      {
8          data = d;
9          next = NULL;
10     }
11 };
12
13 struct Queue {
14     QNode *front, *rear;
15     Queue() { front = rear = NULL; }
16
17     void enQueue(int x)
18     {
19         QNode* temp = new QNode(x);
20         if (rear == NULL) {
21             front = rear = temp;
22             return;
23         }
24         rear->next = temp;
25         rear = temp;
26     }
27     void deQueue()
28     {
29
30         if (front == NULL)
31             return;
32         QNode* temp = front;
33         front = front->next;

```



```
DAA_PRACTICAL > G: queue_using_linkedlist.cpp > Queue > deQueue()
33     front = front->next;
34     if (front == NULL)
35         rear = NULL;
36
37     delete (temp);
38 }
39 };
40 int main()
41 {
42
43     Queue q;
44     q.enqueue(10);
45     q.enqueue(20);
46     q.dequeue();
47     q.dequeue();
48     q.enqueue(30);
49     q.enqueue(40);
50     q.enqueue(50);
51     q.dequeue();
52     cout << "Queue Front : " << ((q.front != NULL) ? (q.front)->data : -1) << endl;
53     cout << "Queue Rear : " << ((q.rear != NULL) ? (q.rear)->data : -1) << endl;
54 }
55
56
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Queue Rear : 50 > cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if
} ; if ($?) { .\queue_using_linkedlist }SA PRACTICE\DAA_PRACTI
Queue Front : 40
Queue Rear : 50
PS D:\DSA PRACTICE\DAA_PRACTICAL>
```

3)Doubly Linked List

```

DAA_PRACTICAL >  Doublylinkedlist.cpp >  main()
1  #include <iostream>
2  using namespace std;
3  struct Node {
4      int data;
5      struct Node* next;
6      struct Node* prev;
7  };
8  void insert_front(struct Node** head, int new_data)
9  {
10     struct Node* newNode = new Node;
11     newNode->data = new_data;
12     newNode->next = (*head);
13     newNode->prev = NULL;
14     if ((*head) != NULL)
15         (*head)->prev = newNode;
16     (*head) = newNode;
17 }
18 void insert_After(struct Node* prev_node, int new_data)
19 {
20     if (prev_node == NULL) {
21         cout<<"Previous node is required , it cannot be NULL";
22         return;
23     }
24     struct Node* newNode = new Node;
25     newNode->data = new_data;
26     newNode->next = prev_node->next;
27     prev_node->next = newNode;
28     newNode->prev = prev_node;

```

```

DAA_PRACTICAL > DoublyLinkedList.cpp > main()
30     newNode->next->prev = newNode;
31 }
32 void insert_end(struct Node** head, int new_data)
33 {
34
35     struct Node* newNode = new Node;
36
37     struct Node* last = *head;
38     newNode->data = new_data;
39     newNode->next = NULL;
40     if (*head == NULL) {
41         newNode->prev = NULL;
42         *head = newNode;
43         return;
44     }
45     while (last->next != NULL)
46         last = last->next;
47     last->next = newNode;
48     newNode->prev = last;
49     return;
50 }
51 void displayList(struct Node* node) {
52     struct Node* last;
53
54     while (node != NULL) {
55         cout<<node->data<<"<==>";
56         last = node;
57         node = node->next;

```



```
DAA_PRACTICAL > DoublyLinkedList.cpp > main()
56     last = node;
57     node = node->next;
58 }
59 if(node == NULL)
60     cout<<"NULL";
61 }
62 int main()
63     struct Node* head = NULL;
64     insert_end(&head, 40);
65     insert_front(&head, 20);
66     insert_front(&head, 10);
67     insert_end(&head, 50);
68     insert_After(head->next, 30);
69     cout<<"Doubly linked list is as follows: "<<endl;
70     displayList(head);
71     return 0;
72 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
10<==>20<==>30<==>40<==>50<==>NULL
PS D:\DSA PRACTICE\DAA_PRACTICAL>
```

4) Enqueue

DAA_PRACTICAL > enqueue.cpp > main()

```
1  #include <iostream>
2  #define SIZE 5
3  using namespace std;
4  class Queue {
5      private:
6          int items[SIZE], front, rear;
7      public:
8          Queue() {
9              front = -1;
10             rear = -1;
11         }
12
13         bool isFull() {
14             if (front == 0 && rear == SIZE - 1) {
15                 return true;
16             }
17             return false;
18         }
19
20         bool isEmpty() {
21             if (front == -1)
22                 return true;
23             else
24                 return false;
25         }
26 }
```

```

27 void enqueue(int element) {
28     if (isFull()) {
29         cout << "Queue is full";
30     } else {
31         if (front == -1) front = 0;
32         rear++;
33         items[rear] = element;
34         cout << endl
35             << "Inserted " << element << endl;
36     }
37 }
38
39 int dequeue() {
40     int element;
41     if (isEmpty()) {
42         cout << "Queue is empty" << endl;
43         return (-1);
44     } else {
45         element = items[front];
46         if (front >= rear) {
47             front = -1;
48             rear = -1;
49         }
50         else {
51             front++;

```

```

DAA_PRACTICAL > enqueue.cpp > main()
51     front++;
52 }
53 cout << endl
54     << "Deleted -> " << element << endl;
55     return (element);
56 }
57 }
58
59 void display() {
60     int i;
61     if (isEmpty()) {
62         cout << endl
63             << "Empty Queue" << endl;
64     } else {
65         cout << endl
66             << "Front index-> " << front;
67         cout << endl
68             << "Items -> ";
69         for (i = front; i <= rear; i++)
70             cout << items[i] << " ";
71         cout << endl
72             << "Rear index-> " << rear << endl;
73     }
74 }
75 };
76

```

```
DAA_PRACTICAL > enqueue.cpp > main()
75 };
76
77 int main() {
78     Queue q;
79     q.dequeue();
80     q.enqueue(1);
81     q.enqueue(2);
82     q.enqueue(3);
83     q.enqueue(4);
84     q.enqueue(5);
85     q.enqueue(6);
86     q.display();
87     q.dequeue();
88     q.display();
89
90     return 0;
91 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Items -> 2 3 4 5
Rear index-> 4
PS D:\DSA PRACTICE\DAA_PRACTICAL>

5) Dequeue

```

DAA_PRACTICAL > G+ dequeue.cpp > main()
1  // SHUBHAM SHIVRAJ SURYAWANSHI
2
3  #include <iostream>
4  using namespace std;
5  #define MAX 10
6  class Deque {
7      int arr[MAX];
8      int front;
9      int rear;
10     int size;
11     public:
12     Deque(int size) {
13         front = -1;
14         rear = 0;
15         this->size = size;
16     }
17
18     void insertfront(int key);
19     void insertrear(int key);
20     void deletefront();
21     void deleterear();
22     bool isFull();
23     bool isEmpty();
24     int getFront();
25     int getRear();
26 };
27
28 bool Deque::isFull() {
29     return ((front == 0 && rear == size - 1) ||
30         front == rear + 1);
31 }
32

```

```

28 bool Deque::isFull() {
29     return ((front == 0 && rear == size - 1) ||
30         front == rear + 1);
31 }
32
33 bool Deque::isEmpty() {
34     return (front == -1);
35 }
36
37 void Deque::insertfront(int key) {
38     if (isFull()) {
39         cout << "Overflow\n"
40             << endl;
41         return;
42     }
43
44     if (front == -1) {
45         front = 0;
46         rear = 0;
47     }
48
49     else if (front == 0)
50         front = size - 1;
51
52     else
53         front = front - 1;
54
55     arr[front] = key;
56 }
57
58 void Deque ::insertrear(int key) {
59     if (isFull()) {

```

```

58 void Deque ::insertrear(int key) {
59     if (isFull()) {
60         cout << " Overflow\n " << endl;
61         return;
62     }
63
64     if (front == -1) {
65         front = 0;
66         rear = 0;
67     }
68
69     else if (rear == size - 1)
70         rear = 0;
71
72     else
73         rear = rear + 1;
74
75     arr[rear] = key;
76 }
77
78 void Deque ::deletefront() {
79     if (isEmpty()) {
80         cout << "Queue Underflow\n"
81             << endl;
82         return;
83     }
84
85     if (front == rear) {
86         front = -1;
87         rear = -1;
88     } else if (front == size - 1)

```



```

132     cout << "insert element at rear end \n";
133     dq.insertrear(5);
134     dq.insertrear(11);
135
136     cout << "rear element: "
137     << dq.getRear() << endl;
138
139     dq.deleterear();
140     cout << "after deletion of the rear element, the new rear element: " << dq.getRear() << endl;
141
142     cout << "insert element at front end \n";
143
144     dq.insertfront(8);
145
146     cout << "front element: " << dq.getFront() << endl;
147
148     dq.deletefront();
149
150     cout << "after deletion of front element new front element: " << dq.getFront() << endl;
151

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```

> cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if ($?) { g++ dequeue.cpp -o dequeue } ; if ($?)
insert element at rear end
rear element: 11
after deletion of the rear element, the new rear element: 5
insert element at front end
front element: 8
after deletion of front element new front element: 5
PS D:\DSA PRACTICE\DAA_PRACTICAL>

```

