Design and Analysis of Algorithms

NAME : SHUBHAM SHIVRAJ SURYAWANSHI

ASSIGNMENT = 2

1) STACK USING LINKED LIST

DAA_PRACTICAL > stack_useing_linked_list.cpp > Stack > push(int)

```cpp
//SHUBHAM SHIVRAJ SURYAWANSHI
//REG NO 2020BIT004
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* link;
    Node(int n)
    {
        this->data = n;
        this->link = NULL;
    }
};
class Stack {
    Node* top;

public:
    Stack() { top = NULL; }
    void push(int data)
    {
        Node* temp = new Node(data);
        if (!temp) {
            cout << "\nStack Overflow";
            exit(1);
        }
        temp->data = data;
        temp->link = top;
        top = temp;
    }
```

```
    }
    bool isEmpty()
    {
        return top == NULL;
    }
          int peek()
    {
        if (!isEmpty())
            return top->data;
        else
            exit(1);
    }
    void pop()
    {
        Node* temp;
        if (top == NULL) {
            cout << "\nStack Underflow" << endl;
            exit(1);
        }
        else {
            temp = top;
            top = top->link;
            free(temp);
        }
    }
    void display()
    {
        Node* temp;
        if (top == NULL) {
            cout << "\nStack Underflow";
            exit(1);
        }
```

DAA_PRACTICAL > stack_useing_linked_list.cpp > Stack > push(int)

```
            cout << "\nStack Underflow";
            exit(1);
        }
        else {
            temp = top;
            while (temp != NULL) {
                cout << temp->data;
                temp = temp->link;
                if (temp != NULL)
                    cout << " -> ";
            }
        }
    }
};
int main()
{
    Stack s;
    s.push(11);
    s.push(22);
    s.push(33);
    s.push(44);
    s.display();
    cout << "\nTop element is " << s.peek() << endl;
    s.pop();
    s.pop();
    s.display();
    cout << "\nTop element is " << s.peek() << endl;
    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Top element is 22                > cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if ($?) { g++ stack_u
st } ; if ($?) { .\stack_useing_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

```cpp
#include <bits/stdc++.h>
using namespace std;
struct QNode {
    int data;
    QNode* next;
    QNode(int d)
    {
        data = d;
        next = NULL;
    }
};

struct Queue {
    QNode *front, *rear;
    Queue() { front = rear = NULL; }

    void enQueue(int x)
    {
        QNode* temp = new QNode(x);
        if (rear == NULL) {
            front = rear = temp;
            return;
        }
        rear->next = temp;
        rear = temp;
    }
    void deQueue()
    {

        if (front == NULL)
            return;
        QNode* temp = front;
        front = front->next;
```

DAA_PRACTICAL > queqe_useing_linkedlist.cpp > Queue > deQueue()

```cpp
        front = front->next;
        if (front == NULL)
            rear = NULL;

        delete (temp);
    }
};
int main()
{

    Queue q;
    q.enQueue(10);
    q.enQueue(20);
    q.deQueue();
    q.deQueue();
    q.enQueue(30);
    q.enQueue(40);
    q.enQueue(50);
    q.deQueue();
    cout << "Queue Front : " << ((q.front != NULL) ? (q.front)
    cout << "Queue Rear : " << ((q.rear != NULL) ? (q.rear)->d
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

3)Doubly Linked List

DAA_PRACTICAL > Doublylinkeqdlist.cpp > main()

```cpp
#include <iostream>
using namespace std;
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};
void insert_front(struct Node** head, int new_data)
{
    struct Node* newNode = new Node;
    newNode->data = new_data;
    newNode->next = (*head);
    newNode->prev = NULL;
    if ((*head) != NULL)
    (*head)->prev = newNode;
    (*head) = newNode;
}
void insert_After(struct Node* prev_node, int new_data)
{
    if (prev_node == NULL) {
    cout<<"Previous node is required , it cannot be NULL";
    return;
}
    struct Node* newNode = new Node;
    newNode->data = new_data;
    newNode->next = prev_node->next;
    prev_node->next = newNode;
    newNode->prev = prev_node;
```

DAA_PRACTICAL > Doublylinkedlist.cpp > main()

```
    newNode->next->prev = newNode;
}
void insert_end(struct Node** head, int new_data)
{

    struct Node* newNode = new Node;

    struct Node* last = *head;
    newNode->data = new_data;
    newNode->next = NULL;
    if (*head == NULL) {
    newNode->prev = NULL;
    *head = newNode;
     return;
}
while (last->next != NULL)
last = last->next;
last->next = newNode;
newNode->prev = last;
return;
}
void displayList(struct Node* node) {
    struct Node* last;

    while (node != NULL) {
        cout<<node->data<<"<==>";
        last = node;
        node = node->next;
```

```
DAA_PRACTICAL > Doublylinkedlist.cpp > main()
        last = node;
        node = node->next;
    }
    if(node == NULL)
    cout<<"NULL";
    }
int main() {
    struct Node* head = NULL;
    insert_end(&head, 40);
    insert_front(&head, 20);
    insert_front(&head, 10);
    insert_end(&head, 50);
    insert_After(head->next, 30);
    cout<<"Doubly linked list is as follows: "<<endl;
    displayList(head);
    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

4)Enqueue

DAA_PRACTICAL > enqueue.cpp > main()

```cpp
#include <iostream>
#define SIZE 5
using namespace std;
class Queue {
   private:
  int items[SIZE], front, rear;
  public:
  Queue() {
    front = -1;
    rear = -1;
  }

  bool isFull() {
    if (front == 0 && rear == SIZE - 1) {
      return true;
    }
    return false;
  }

  bool isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }

```

```
void enQueue(int element) {
  if (isFull()) {
    cout << "Queue is full";
  } else {
    if (front == -1) front = 0;
    rear++;
    items[rear] = element;
    cout << endl
       << "Inserted " << element << endl;
  }
}

int deQueue() {
  int element;
  if (isEmpty()) {
    cout << "Queue is empty" << endl;
    return (-1);
  } else {
    element = items[front];
    if (front >= rear) {
      front = -1;
      rear = -1;
    }
    else {
      front++;
```

DAA_PRACTICAL > enqueue.cpp > main()

```
        front++;
      }
      cout << endl
         << "Deleted -> " << element << endl;
      return (element);
    }
  }

  void display() {
    int i;
    if (isEmpty()) {
      cout << endl
         << "Empty Queue" << endl;
    } else {
      cout << endl
         << "Front index-> " << front;
      cout << endl
         << "Items -> ";
      for (i = front; i <= rear; i++)
        cout << items[i] << "  ";
      cout << endl
         << "Rear index-> " << rear << endl;
    }
  }
};

```

DAA_PRACTICAL > enqueue.cpp > main()

```cpp
};

int main() {
    Queue q;
    q.deQueue();
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);
    q.enQueue(4);
    q.enQueue(5);
    q.enQueue(6);
    q.display();
    q.deQueue();
    q.display();

    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

5)Dequeue

DAA_PRACTICAL > dequeue.cpp > main()

```cpp
// SHUBHAM SHIVRAJ SURYAWANSHI

#include <iostream>
using namespace std;
#define MAX 10
class Deque {
  int arr[MAX];
  int front;
  int rear;
  int size;
  public:
  Deque(int size) {
    front = -1;
    rear = 0;
    this->size = size;
  }

  void insertfront(int key);
  void insertrear(int key);
  void deletefront();
  void deleterear();
  bool isFull();
  bool isEmpty();
  int getFront();
  int getRear();
};

bool Deque::isFull() {
  return ((front == 0 && rear == size - 1) ||
      front == rear + 1);
}

```

```
bool Deque::isFull() {
  return ((front == 0 && rear == size - 1) ||
      front == rear + 1);
}

bool Deque::isEmpty() {
  return (front == -1);
}

void Deque::insertfront(int key) {
  if (isFull()) {
    cout << "Overflow\n"
      << endl;
    return;
  }

  if (front == -1) {
    front = 0;
    rear = 0;
  }

  else if (front == 0)
    front = size - 1;

  else
    front = front - 1;

  arr[front] = key;
}

void Deque ::insertrear(int key) {
  if (isFull()) {
```

```
void Deque ::insertrear(int key) {
  if (isFull()) {
    cout << " Overflow\n " << endl;
    return;
  }

  if (front == -1) {
    front = 0;
    rear = 0;
  }

  else if (rear == size - 1)
    rear = 0;

  else
    rear = rear + 1;

  arr[rear] = key;
}

void Deque ::deletefront() {
  if (isEmpty()) {
    cout << "Queue Underflow\n"
        << endl;
    return;
  }

  if (front == rear) {
    front = -1;
    rear = -1;
  } else if (front == size - 1)
```

```cpp
    dq.insertrear(5);
    dq.insertrear(11);

    cout << "rear element: "
        << dq.getRear() << endl;

    dq.deleterear();
    cout << "after deletion of the rear element, the new rear element: " << dq.getRear() << endl;

    cout << "insert element at front end \n";

    dq.insertfront(8);

    cout << "front element: " << dq.getFront() << endl;

    dq.deletefront();

    cout << "after deletion of front element new front element: " << dq.getFront() << endl;
}
```

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>
```