CLASS - ASSIGNMENTS

Topic 1: Introduction to Data Structures

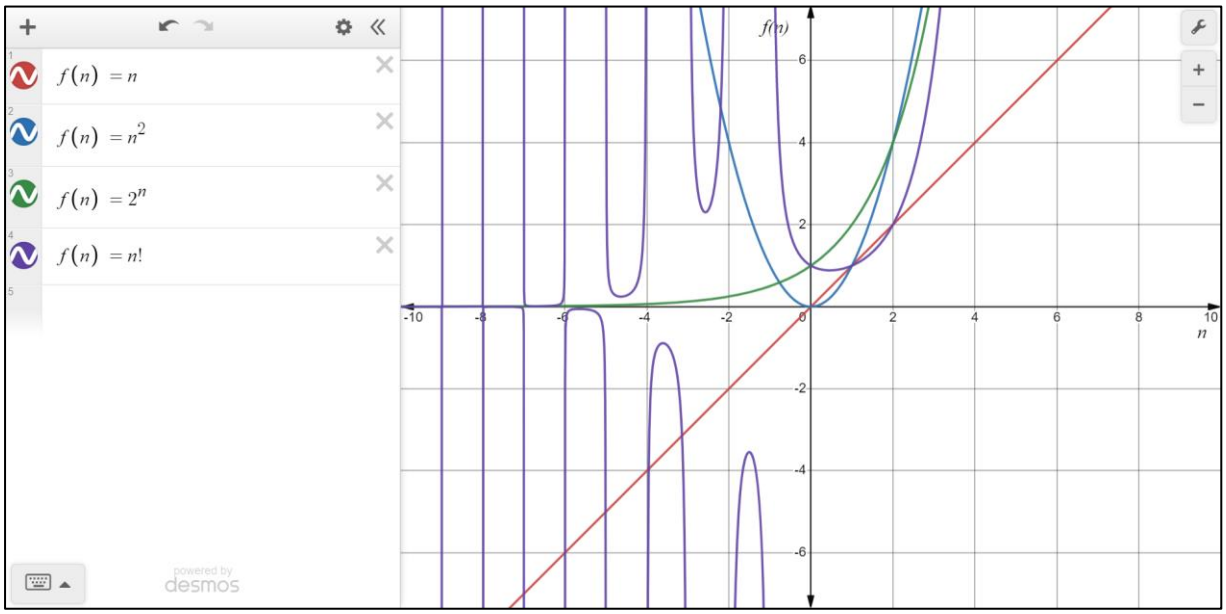
1) Draw graphs (in any tool) for comparing the following functions, where n varies from 0 to 5 (x axis representing n and y axis representing f(n) functions):

1. f(n)=n

2. f(n)= n 2

3. f(n)= 2 n

4. f(n)= n!



Topic 2: ARRAYS

1) Write pseudocode for an array of integers perform the following operations:

(1) Insert at Index

```
START
IF n = MAX
        return
ELSE
        n = n + 1
        FOR Elements from A[index] to A[n]
                Move to next adjacent location
        A[index] = new_element
STOP
```

(2) Delete first element

```
START
PRINT element to delete
SCAN key
FOR i = 0  i < size  i++
        IF(arr[i] == key)
                index = i
                break
IF index != -1
        FOR i = index  i < size - 1  i++
                arr[i] = arr[i+1]
        FOR i = 0  i < size – 1  i++
                printf("%d ",arr[i])
ELSE
        PRINT Element Not Found
```

(3) Traverse in Reverse order

```
START
DECLARE arr[size], i
FOR i = size-1   i >= 0   i- -
        PRINT array[loop]
STOP
```

2) Write pseudocode for a string perform the following operations:

(1) Check Palindrome

```
START
DECLARE char str[]
DECLARE int l = 0, r = strlen(str) - 1
WHILE r > l
        IF str[l++] != str[r--]
                PRINT a Palindrome
                return
PRINT Not a Palindrome
return
STOP
```

(2) Find occurrence of a given character

```
START
DECLARE char s[1000], c
DECLARE int i, count=0
PRINT Enter the string
gets(s)
PRINT Enter character to be searched
c = getchar()
FOR i=0  , s[i]  , i++
```

```
        IF(s[i]==c)
                count++
PRINT count
STOP
```

(3) Compare 2 strings

```
START
DECLARE int count1 , count2 , flag , i
DECLARE char string1[size], string2[size]
WHILE string1[count1] != '\0'
        count1 ++
WHILE string2[count2] != '\0'
        count2 ++
WHILE string1[i] == string2[i] and string1[i] != '\0'
STOP
```

3) Write pseudocode to Multiply two 3 x 3 matrices

```
START
DECLARE A, B , C n*n matrix
FOR i = 0  i < n  i++
        FOR j = 0  j < n  j++
                C[i] = 0
                C[j] = 0
FOR i=0   i < n   i++
        FOR j = 0    j < n    j++
                FOR k = 0  k < n  k++
                        C[i] = C[i] + A[i] * B[k]
                        C[j] = C[j] + A[k] * B[j]
STOP
```

Topic 3: STACK

1) Pseudocode to check a palindrome string with stack.

```
START
DECLARE int top = -1 , front = 0 , stack[MAX]
FUNCTION push(char)
FUNCTION pop()
WHILE (1)
    SWITCH (choice)
    case 1:
      PRINT Enter the String\n
      SCAN s
      FOR i = 0   s[i] != '\0'   i++
        b = s[i]
        push(b)
      FOR i = 0   i < (strlen(s) / 2)   i++
        IF (stack[top] == stack[front])
          pop()
          front++
        ELSE
          PRINT not a palindrome
          break
      IF (strlen(s) / 2)  = =  front
        PRINT palindrome
      front  =  0
      top  =  -1
      break
    case 2:
      exit(0)
```

```
    default:
        PRINT enter correct choice
STOP
```

2) Pseudocode to convert infix expression into prefix.

```
START
FOR i = 0 to lengthofinfix
        IF infix[i] is operand — prefix+= infix[i]
        ELSE IF infix[i] is ‘(’ — stack.push(infix[i])
        ELSE IF infix[i] is ‘)’ - pop and print the values of stack till the symbol ’)' is not found
        ELSE IF infix[i] is an operator(+, -, *, /, *) >
IF the stack is empty
        Push() infix[i] on the top of the stack.
ELSE
        IF precedence(infix[i] > precedence(stack.top))
                Push() infix[i] on the top of the stack
        ELSE IF (infix[i] == precedence(stack.top) && infix[i] == '*')
                Pop() and print the top values of the stack till the condition is true
                Push() infix[i] into the stackELSE IF(infix[i] == precedence(stack.top))
                Push() infix[i] on to the stack
        ELSE IF (infix[i] < precedence(stack.top))
                Pop() stack , print till stack is full and infix[i] < precedence(stack.top)
                Push() infix[i] on to the stack
Pop() and print the remaining elements of the stack
Prefix = reverse(prefix)
STOP
```

3) Pseudocode to evaluate a postfix expression.

```
DECLARE int stack[size] , top = -1 and char exp[20] , char *e  and int n1,n2,n3,num
FUNCTION push(int x)
        stack[++top] = x
FUNCTION pop()
        return stack[top--]
SCAN exp = e
WHILE *e ! = '\0'
    IF isdigit(*e)
       num = *e - 48
       push(num)
    ELSE
       n1 = n2 =  pop()
       switch(*e)
              case '+':
                 n3 = n1 + n2
                 break
              case '-':
                 n3 = n2 - n1
                 break
              case '*':
                 n3 = n1 * n2
                 break
              case '/':
                 n3 = n2 / n1
                 break
                 push(n3)
        e++
PRINT exp
```

4) Pseudocode for Fibonacci series with recursion

```
START
DECLARE int n, m= 0 , i
PRINT Enter Total terms
SCAN n
FOR i = 1   i <= n   i++
        PRINT fib(m)
        m++
return 0
int fib(int n)
        IF n == 0 || n == 1
                return n
        ELSE
        Return fib(n-1) + fib(n-2)
STOP
```

Topic 4: QUEUE

1) Pseudocode for Linear Queue

```
START
Initialize (Queue, Front, Rear)
        Front=Rear=-1
IsEmpty (Queue, Front, Rear)
        IF Front==-1 OR Front>Rear
                Return True
        ELSE:
                Return False
IsFull (Queue, Rear, MAX)
        IF Rear==MAX-1:
                Return True
        ELSE:
                Return False
Enqueue (Queue, Rear, Item)
        IF !IsFull()
                Rear++
                Queue[Rear]=Item
                IF(Front=-1)
                        Front++
Dequeue (Queue, Front)
        IF !IsEmpty()
                Remove Queue[Front]
                Front++
Peek (Queue, Front)
        IF !IsEmpty()
                Display Queue[Front]
Traverse (Queue, Front, Rear)
```

```
        IF !IsEmpty()
                For i=Front to Rear
                        Display Queue[i]
STOP
```

2) Pseudocode for Double-Ended Queue

```
START
Initialize (Queue, front, rear)
        front=rear=-1
IsEmpty (Queue, front, rear)
        IF front==-1 OR front>rear
                Return True
        ELSE:
                Return False
IsFull (Queue, rear, MAX)
        IF rear==MAX-1:
                Return True
        ELSE:
                Return False
Insertion_at_front()
        IF !IsFull()
                IF front > 1
                        front=front-1
                        q[front]=no
Insertion_at_rear()
        IF !IsFull()
                rear=rear+1
                        q[rear]=no
                IF rear=0
```

```
                rear=1
        IF front=0
                front=1
Deletion_from_front()
    IF !IsEmpty()
        no=q[front]
        IF front=rear
                front=0
                rear=0
        ELSE
                front=front+1
Deletion_from_end()
    IF !IsEmpty()
        no=q[rear]
        IF front= rear
                front=0
                rear=0
        ELSE
                rear=rear-1
STOP
```

Topic 5: LINKED LIST

1) Pseudocode for singly linked list operations

```
START
Initialize struct Node
        int data
        struct Node *next
Insert_at_beginning(int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode->data = value
        IF head == NULL
                newNode->next = NULL
                head = newNode
        ELSE
                newNode->next = head
                head = newNode
Insert_at_end(int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode->data = value
        newNode->next = NULL
        IF head == NULL
                head = newNode
        ELSE
                struct Node *temp = head
                while(temp->next != NULL)
                temp = temp->next
                temp->next = newNode
Insert_after(int value, int loc1, int loc2)
```

```
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode->data = value
        IF head == NULL
                newNode->next = NULL
                head = newNode
        ELSE
                struct Node *temp = head
                while(temp->data != loc1 && temp->data != loc2)
                        temp = temp->next
                newNode->next = temp->next
                temp->next = newNode
Delete_at_beginning()
        IF head != NULL
                struct Node *temp = head
                IF head->next == NULL
                        head = NULL
                        free(temp)
                ELSE
                        head = temp->next
                free(temp)
Delete_at_end()
        IF head != NULL
                struct Node *temp1 = head,*temp2
                IF head->next == NULL
                        head = NULL
                ELSE
                        WHILE temp1->next != NULL
                        temp2 = temp1
```

```
                    temp1 = temp1->next
            temp2->next = NULL
        free(temp1)
Traverse ()
        IF head != NULL
            struct Node *temp = head
            while(temp->next != NULL)
                PRINT temp->data
                temp = temp->next
            PRINT temp->data
STOP
```

2) Pseudocode for stack using linked list

```
START
Initialize struct Node
        int data
        struct Node *next
Push(int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode->data = value
        IF top == NULL
            newNode->next = NULL
        ELSE
             newNode->next = top
        top = newNode
Pop()
        IF top != NULL
             struct Node *temp = top
```

```
                PRINT temp->data
                top = temp->next
                free(temp)
Traverse()
        IF top != NULL
                struct Node *temp = top
                WHILE temp->next != NULL
                        PRINT temp->data
                        temp = temp -> next
                PRINT temp->data
STOP
```

3) Pseudocode for queue using linked list

```
START
Initialize struct Node
        int data
        struct Node *next
Enqueue (int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode->data = value
        newNode -> next = NULL
        IF front == NULL
                front = rear = newNode
        ELSE
                rear -> next = newNode
                rear = newNode
Dequeue()
        IF front != NULL
                struct Node *temp = front
```

```
            front = front -> next
            PRINT temp->data)
            free(temp)

Traverse()
        IF front        != NULL
            struct Node *temp = front
            WHILE temp->next != NULL
                PRINT temp->data
                temp = temp -> next
            PRINT temp->data
STOP
```

4) Pseudocode for singly circular linked list operations:

(1) Insert first

```
Insert_at_first(int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode -> data = value
        IF(head == NULL)
            head = newNode
            newNode -> next = head
        ELSE
            struct Node *temp = head
            while(temp -> next != head)
                temp = temp -> next
            newNode -> next = head
            head = newNode
            temp -> next = head
```

(2) Insert last

```
Insert_at_last(int value)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode -> data = value
        IF(head == NULL)
                head = newNode
                newNode -> next = head
        ELSE
                struct Node *temp = head
                WHILE(temp -> next != head)
                        temp = temp -> next
                temp -> next = newNode
                newNode -> next = head
```

(3) Insert after

```
Insert_after(int value, int location)
        struct Node *newNode
        newNode = (struct Node*)malloc(sizeof(struct Node))
        newNode -> data = value
        IF(head == NULL)
                head = newNode
                newNode -> next = head
        ELSE
                struct Node *temp = head
                WHILE(temp -> data != location)
                        IF(temp -> next == head)
                                PRINT node is not found
                                return
                        ELSE
```

```
                    temp = temp -> next
        newNode -> next = temp -> next
        temp -> next = newNode
```

(4) Delete first

```
    IF(head == NULL)
        PRINT List is Empty
    ELSE
        struct Node *temp = head
        IF(temp -> next == head)
            head = NULL
            free(temp)
        ELSE
            head = head -> next
            free(temp)
```

(5) Delete last

```
Delete_at_last()
    IF(head == NULL)
        printf("List is Empty!!! Deletion not possible!!!")
     ELSE
        struct Node *temp1 = head, temp2
        IF(temp1 -> next == head)
            head = NULL
            free(temp1)
        ELSE
            WHILE(temp1 -> next != head)
                temp2 = temp1
                temp1 = temp1 -> next
            temp2 -> next = head
            free(temp1)
```

(6) Delete after

```
Delete _after()
        IF(head == NULL)
                PRINT List is Empty
         ELSE
                struct Node *temp1 = head, temp2
                WHILE(temp1 -> data != delValue)
                        IF(temp1 -> next == head)
                                PRINT Given node is not found
                  ELSE
                        temp2 = temp1
                        temp1 = temp1 -> next
           IF(temp1 -> next == head)
                  head = NULL
                  free(temp1)
           ELSE
                  IF(temp1 == head)
                        temp2 = head
                        WHILE(temp2 -> next != head)
                                temp2 = temp2 -> next
                        head = head -> next
                        temp2 -> next = head
                        free(temp1)
                  ELSE
                        IF(temp1 -> next == head)
                                temp2 -> next = head
                        ELSE
                                temp2 -> next = temp1 -> next
                        free(temp1)
```

5) Pseudocode for doubly circular linked list operations:

(1) Insert first

```
Insert_first(struct Node** start, int value)
        struct node* new_node = create(data)
          IF (new_node)
                IF (head == NULL)
                        new_node->next = new_node
                        new_node->prev = new_node
                        head = new_node
                        return
           head->prev->next = new_node
           new_node->prev = head->prev
           new_node->next = head
           head->prev = new_node
           head = new_node
```

(2) Insert last

```
Insert_last(struct Node** start, int value)
        struct node* new_node = create(data)
            IF (new_node)
                IF (head == NULL)
                        new_node->next = new_node
                        new_node->prev = new_node
                        head = new_node
                        return
           head->prev->next = new_node
           new_node->prev = head->prev
           new_node->next = head
           head->prev = new_node
```

(3) Delete first

```
Delete_at_first ()
        IF (head == NULL)
                printf("\nList is Empty\n")
                return
        ELSE  IF (head->next == head)
                free(head)
                head = NULL
                return
        struct node* temp = head
        head->prev->next = head->next
        head->next->prev = head->prev
        head = head->next
        free(temp)
```

(4) Delete last

```
Delete_at_last()
        IF (head == NULL)
                printf("\nList is Empty\n")
                return
        ELSE IF (head->next == head)
                free(head)
                head = NULL
                return
        struct node* last_node = head->prev
        last_node->prev->next = head
        head->prev = last_node->prev
        free(last_node)
        last_node = NULL
```