CSS2101 DATA STRUCTURES SCHEME – MID TERM EXAMINATION

Q11. You are developing a feature for a math learning app that allows users to add polynomial expressions. Each polynomial is stored as a singly linked list, where each node contains a coefficient and exponent. The terms are sorted in descending order of exponents. Based on above, answer the following questions,

1. Explain why a linked list is a suitable data structure for representing polynomials in this system.
2. Write a code logic to add two polynomials represented as linked lists.
   a. Describe how your logic handles matching and non-matching exponents.
   b. Identify one potential issue that could arise during addition and suggest how to handle it in code. (4)

Scheme:

1. why a linked list is a suitable data structure (Any two points) **(1 mark)**

- Dynamic size – can easily handle polynomials of different lengths.
- Efficient insertion/deletion – adding new terms doesn't require shifting elements (like arrays).
- Sparse polynomial representation – avoids memory wastage by storing only non-zero terms.
- Natural representation – terms can be stored in sorted order by exponents.

2. Code logic to add two polynomials **(total 2 mark)**

```
struct Node {
    int coeff, exp;
    Node* next;
};

Node* addPolynomials(Node* p1, Node* p2) {
    Node* result = NULL, *tail = NULL;

    while (p1 && p2) {                          (1 mark)
        if (p1->exp == p2->exp) {
            int sum = p1->coeff + p2->coeff;
            if (sum != 0) {
                Node* temp = new Node{sum, p1->exp, NULL};
                if (!result) result = tail = temp;
                else { tail->next = temp; tail = temp; }
            }
            p1 = p1->next;
            p2 = p2->next;
        }
        else if (p1->exp > p2->exp) {
            Node* temp = new Node{p1->coeff, p1->exp, NULL};
            if (!result) result = tail = temp;
            else { tail->next = temp; tail = temp; }
```

```
            p1 = p1->next;
        }
        else { // p2->exp > p1->exp
            Node* temp = new Node{p2->coeff, p2->exp, NULL};
            if (!result) result = tail = temp;
            else { tail->next = temp; tail = temp; }
            p2 = p2->next;
        }
    }

    // Append remaining terms of p1and p2                    (1 mark)

    while (p1) {
        Node* temp = new Node{p1->coeff, p1->exp, NULL};
        if (!result) result = tail = temp;
        else { tail->next = temp; tail = temp; }
        p1 = p1->next;
    }


    while (p2) {
        Node* temp = new Node{p2->coeff, p2->exp, NULL};
        if (!result) result = tail = temp;
        else { tail->next = temp; tail = temp; }
        p2 = p2->next;
    }

    return result;
}
```

a. Handles matching and non-matching exponents. **(0.5) mark**

- Matching exponents: coefficients are added and stored as a single term in the result (no duplicate exponents).
- Non-matching exponents: the term with the larger exponent is copied directly into result, maintaining descending order.

b. Handling issue during addition. (any one issue). **(0.5) mark**

Issue: Duplicate exponents may remain if input polynomials are not sorted → solution: ensure input is sorted before addition.

Issue: Zero coefficient terms may appear → solution: skip creating nodes with coefficient = 0.

Issue: Memory leaks when dynamically allocating nodes → solution: free unused memory.

CSS2101 DATA STRUCTURES SCHEME – MID TERM EXAMINATION

Q12.

1. **Read details of N students** (name, roll number, and marks in 3 subjects). --->**0.5 marks**
2. **Calculate the average marks** of each student. --->1 marks
3. Identify and display the **topper** (student with the highest average).--->**1 marks+ display(0.5 marks)**

```
 #include <stdio.h>

struct Student {
   char name[51];
   int  roll;
   int  marks[3
   float average;     };

void readStudents(struct Student s[], int n);
void computeAverages(struct Student s[], int n);
void displayStudents(struct Student s[], int n);
float findMaxAverage(struct Student s[], int n);
void printToppers(struct Student s[], int n, float maxAvg);

int main(void) {
   int n, i;

   printf("Enter number of students: ");
   if (scanf("%d", &n) != 1 || n <= 0 || n > 200) {
      printf("Invalid number of students.\n");
      return 1;
   }

      struct Student students[200];

   readStudents(students, n);
   computeAverages(students, n);

   printf("\n---- Student Records ----\n");
   displayStudents(students, n);

   float maxAvg = findMaxAverage(students, n);
   printf("\nTopper(s) with Average = %.2f:\n", maxAvg);
   printToppers(students, n, maxAvg);

   return 0;
}

/* 1) Read details of N students */
void readStudents(struct Student s[], int n) {
```

```
    int i, j;
    for (i = 0; i < n; i++) {
        printf("\nEnter details for student %d\n", i + 1);

        printf("Name: ");
        scanf("%s", s[i].name);   // single word name

        printf("Roll Number: ");
        scanf("%d", &s[i].roll);

        printf("Enter marks in 3 subjects: ");
        for (j = 0; j < 3; j++) {
            scanf("%d", &s[i].marks[j]);
        }
    }
}
/* 2) Calculate the average marks of each student */
void computeAverages(struct Student s[], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        int sum = 0;
        for (j = 0; j < 3; j++) {
            sum += s[i].marks[j];
        }
        s[i].average = sum / 3.0f;
    }
}

/* 3) Display all student details */ // Not mandatory
void displayStudents(struct Student s[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("Name: %-20s | Roll: %-6d | Marks: %3d %3d %3d | Average: %6.2f\n",
               s[i].name, s[i].roll,
               s[i].marks[0], s[i].marks[1], s[i].marks[2],
               s[i].average);
    }
}

/* find highest average */
float findMaxAverage(struct Student s[], int n) {
    int i;
    float maxAvg = s[0].average;
    for (i = 1; i < n; i++) {
        if (s[i].average > maxAvg) {
            maxAvg = s[i].average;
        }
    }
    return maxAvg;
}
```

```
/* 4) Identify and display topper(s) (handles ties) */
void printToppers(struct Student s[], int n, float maxAvg) {
   int i;
   for (i = 0; i < n; i++) {
      if (s[i].average == maxAvg) {
         printf("  %s (Roll: %d)\n", s[i].name, s[i].roll);
      }
   }
}
```

Q13. A movie streaming app allows users to manage their watchlist. Users can: Move to the next or previous movie, Insert a new movie between two existing ones, Remove any movie from the list. The app must maintain smooth navigation and quick updates. Answer below question based on above scenario:

(a) Suggest a suitable data structure for this system.
(b) Justify your choice based on the operations.
(c) Write a C function to handle the deletion of a movie from the middle of the watchlist.

Component	Expected Response	**Marks**
1. Suitable Data Structure	Doubly Linked List (DLL)	**0.5 mark**
2. Justification	DLL allows bidirectional traversal (next/previous), easy insertion between nodes, and efficient deletion from any position.	**0.5 mark**
3. Deletion Find middle – 1 M Delete middle-1 M	void deleteMiddle(struct Movie **head) { if (*head == NULL) return; // Step 1: Count nodes int count = 0; struct Movie *temp = *head; while (temp != NULL) { count++; temp = temp->next; } if (count == 1) { // Only one node free(*head); *head = NULL; return; }	**2 mark**

	// Step 2: Find middle position int mid = count / 2; // If even, deletes (n/2 + 1)-th node temp = *head; for (int i = 0; i < mid; i++) { temp = temp->next; } **// middle can be identified using other logic like slow and fast pointer approach.** // Step 3: Delete 'temp' (middle node) if (temp->prev != NULL) temp->prev->next = temp->next; if (temp->next != NULL) temp->next->prev = temp->prev; // If head is middle if (temp == *head) *head = temp->next; printf("Deleted movie: %s\n", temp->title); free(temp); }		

(3)

Q14. A circular linked list contains the elements: 10 → 20 → 30 → 40 → 50 → (back to 10)

You are asked to rotate this list clockwise by 2 positions.

Answer the following:

1. Illustrate with a diagram of the list after rotation, showing how the elements are linked.
2. Write a function to implement the given task using the pointer to the last node .

Answer:

1. Clockwise rotation means the last 2 nodes (40, 50) move to the front. **(1 Mark)**

40 → 50 → 10 → 20 → 30 → (back to 40)

2. void rotateClockwise(struct Node **last, int k) { **(2 Mark)**

```
if (*last == NULL || k == 0) return;
```

```
    // Count number of nodes
    int count = 1;
    struct Node *temp = (*last)->next;  // head
    while (temp != *last) {
        count++;
        temp = temp->next;
    }

  // rotation
    k = k % count;
    if (k == 0) return;

    // Move last forward (count - k) times
    int steps = count - k;
    for (int i = 0; i < steps; i++) {
        *last = (*last)->next;
    }
}
```

Q15. You are required to read and process a sequence of numbers (both positive and negative). Whenever a negative number is encountered, output the five numbers that appeared immediately before it **in reverse order** (most recent first), then discard the negative number and continue processing the remaining input.

- If fewer than five numbers exist before a negative number, display error message and terminate.
- Repeat the same process for every negative number encountered until the input ends.

Design and implement a solution to this problem using **most suitable data structure**. Give all necessary functions.

Sample Input/Output:

Input: 5 10 20 30 40 50 -1 60 70 80 90 100 -2 110 120 -3

Output:

50 40 30 20 10 100 90 80 70 60

Error: fewer than 5 numbers before the negative number: –3

Ans:

```
#include <stdio.h>

#include <stdlib.h>
```

CSS2101 DATA STRUCTURES SCHEME – MID TERM EXAMINATION

```
#define MAX 1000


int stack[MAX];

int top = -1;


// push function

void push(int x) {

    if (top == MAX - 1) {

        printf("Stack Overflow\n");

        exit(1);

    }

    stack[++top] = x;

}


// pop function

int pop() {

    if (top == -1) {

        printf("Stack Underflow\n");

        exit(1);

    }

    return stack[top--];

}


int main() {

    int x;
```

```
    while (scanf("%d", &x) == 1) {
        if (x >= 0) {
            push(x);
        } else {
            if (top < 4) {
                printf("Error: fewer than 5 numbers before the negative number: %d\n",x);
                return 0; // terminate
            }
            // print last 5 numbers in reverse order using pop
            for (int i = 0; i < 5; i++) {
                printf("%d ", pop());
            }
            printf("\n");
        }
    }

    return 0;
}
```

Push + Pop : 1Mark

Main fn : 2Mark

Q16. Write the algorithm to convert a given **infix expression** into its equivalent **prefix expression** using stack. Note: The infix expression may include ONLY the following operators (listed in the order of their precedence):

- ^ (exponentiation) → **right associative**
- * (multiplication), / (division) → **left associative**
- + (addition), - (subtraction) → **left associative**

Also, convert the infix expression, A ^ B ^ C * D / E to prefix expression showing the step-by-step conversion using the table given below:

Token	Stack [0] [1] [2] …	Top	Output

Ans:

1. Reverse the given infix expression

2. Initialize an empty stack for operators.

   Initialize an empty string for the output (prefix).

3. Scan the reversed infix expression from left to right:

   a) If the symbol is an operand → Append it to output.

   b) If the symbol is an operator:

      i. If stack is empty → Push operator.

      ii. If operator has higher precedence than top of stack → Push operator.

      iii. If operator has same precedence →

         - Push operator (for left-associative ops).

         - Special case: If operator is '^' (right-associative), pop stack top and then push '^'.

      iv. If operator has lower precedence than top of stack →

         Pop from stack to output until condition is satisfied,

         then push the operator.

4. After scanning the expression:

   Pop and append all remaining operators from stack to output.

5. Reverse the output string to get the Prefix expression.

--1.5Marks

CSS2101 DATA STRUCTURES SCHEME – MID TERM EXAMINATION

Token	Stack (bottom → top)	Top Index	Output (so far)
E	[ ]	-	E
/	[/]	0	E
D	[/]	0	E D
*	[/, *]	1	E D
C	[/, *]	1	E D C
^	[/, *, ^]	2	E D C
B	[/, *, ^]	2	E D C B
^	[/, *, ^] (old ^ popped → output, new ^ pushed)	2	E D C B ^
A	[/, *, ^]	2	E D C B ^ A
(end)	[/, *] → pop ^	1	E D C B ^ A ^
(end)	[/] → pop *	0	E D C B ^ A ^ *
(end)	[ ] → pop /	-	E D C B ^ A ^ * /

Prefix Expression: / * ^ A ^ B C D E

-1.5Marks

Q17.

Main function ----> 1 mark

Recursive function ----> 1 mark

```
#include <stdio.h>
// Recursive function to calculate sum of book numbers
int sumBooks(int n) {
    if (n == 0)   // Base case: no books
        return 0;
    else
        return n + sumBooks(n - 1); // Recursive call
}
int main() {
    int books;
    printf("Enter number of books on the shelf: ");
    scanf("%d", &books);

    int total = sumBooks(books);

    printf("The total sum of book numbers from 1 to %d is: %d\n", books, total);
```

```
    return 0;
}
```

Q18.

1. **Search for a student by roll number and display their details. -----> 1 mark**
2. **Delete a student by roll number if present in the list. -----> 1 mark**

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

struct Student {
    int roll;
    char name[50];
    struct Student *next;
};

/* Function to search a student by roll number */
void searchStudent(struct Student *head, int rollNo) {
    struct Student *temp = head;
    while (temp != NULL) {
        if (temp->roll == rollNo) {
            printf("Student Found!\n");
            printf("Roll: %d | Name: %s\n", temp->roll, temp->name);
            return;
        }
        temp = temp->next;
    }
    printf("Student with Roll %d not found.\n", rollNo);
}

/* Function to delete a student by roll number */
void deleteStudent(struct Student **head, int rollNo) {
    struct Student *temp = *head, *prev = NULL;

    // Case 1: head itself is to be deleted
    if (temp != NULL && temp->roll == rollNo) {
        *head = temp->next;
        free(temp);
        printf("Student with Roll %d deleted.\n", rollNo);
        return;
    }

    // Case 2: search the node
    while (temp != NULL && temp->roll != rollNo) {
        prev = temp;
```

```
        temp = temp->next;
    }

    // If not found
    if (temp == NULL) {
        printf("Student with Roll %d not found.\n", rollNo);
        return;
    }

    // Unlink the node and free memory
    prev->next = temp->next;
    free(temp);
    printf("Student with Roll %d deleted.\n", rollNo);
}

void display(struct Student *head) {
    struct Student *temp = head;
    if (head == NULL) {
        printf("No students in the list.\n");
        return;
    }
    printf("\n--- Student Records ---\n");
    while (temp != NULL) {
        printf("Roll: %d | Name: %s\n", temp->roll, temp->name);
        temp = temp->next;
    }
}

int main() {
    // Create a sample list manually
    struct Student *head = NULL, *s1, *s2, *s3;

    s1 = (struct Student*)malloc(sizeof(struct Student));
    s1->roll = 101; strcpy(s1->name, "Alice");
    s1->next = NULL;
    head = s1;

    s2 = (struct Student*)malloc(sizeof(struct Student));
    s2->roll = 102; strcpy(s2->name, "Bob");
    s2->next = NULL;
    s1->next = s2;

    s3 = (struct Student*)malloc(sizeof(struct Student));
    s3->roll = 103; strcpy(s3->name, "Charlie");
    s3->next = NULL;
    s2->next = s3;

    display(head);

    // Search operation
```

```
    searchStudent(head, 102);
    searchStudent(head, 110);

    // Delete operation
    deleteStudent(&head, 101);
    display(head);

    deleteStudent(&head, 110);  // not found case
    display(head);

    return 0;
}
```

Q19. Implement stack using Singly Linked List. Consider the following node structure definitions and function prototypes:

```
typedef struct node *Nodeptr;

        struct node{

          int data;

          Nodeptr next;

        };

        void Push(Nodeptr *top, int item);

        int Pop(Nodeptr *top);
```

Ans:

```
void Push(Nodeptr *top, int item){

Nodeptr temp;

temp = (Nodeptr) malloc(sizeof(struct node));

temp->data = item;

temp->next = top;

top = temp;

}                                                    --1Mark

int Pop(Nodeptr *top){

Nodeptr temp;

int item;

if (top == NULL) {printf("Stack Underflow"); return ERROR; }

temp = top;

top = top->next;

item = top->data; free(temp); }--1Mark
```