USN | | | | | | | | | | |

Internal Assessment Test 2 – December 2023

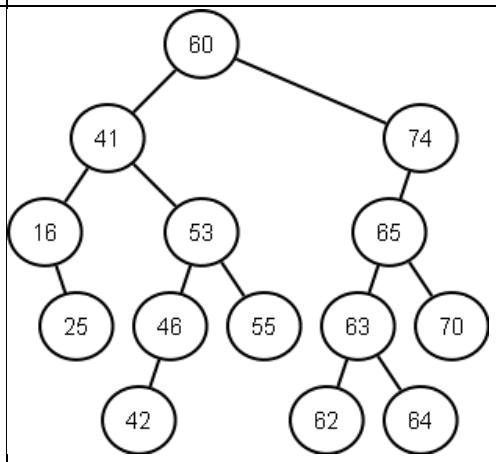
Sub:	DSA					Sub Code:	BCS304A	Branch:	AIDS & CSE(AIDS)
Date:	20/01/24	Duration:	90 minutes	Max Marks:	50	Sem/Sec:	III -A, B & C		OBE

		Answer any FIVE FULL Questions	MARKS	CO	RBT

1		Write the function to perform the following on SLL: (usually using integer data type)
		• Inverting a singly linked list. (Reversing)
		• Assume the list contains 3 nodes with data 10, 20, 30. Insert a node with data 40 at the front of the list.
		Sollution:
		Reversing:
		`static void reverse(struct Node** head_ref)`
		`{`
		`struct Node* prev = NULL;`
		`struct Node* current = *head_ref;`
		`struct Node* next = NULL;`
		`while (current != NULL) {`
		`// Store next`
		`next = current->next;`
		`// Reverse current node's pointer`
		`current->next = prev;`
		`// Move pointers one position ahead.`
		`prev = current;`
		`current = next;`
		`}`
		`*head_ref = prev;`
		`}`
		Insertion from the beginning
		`void insert_begin()`
		`{`
		`struct node *temp;`
		`temp=(struct node *)malloc(sizeof(struct node));`
		`if(temp==NULL)`
		`{`
		`printf("nOut of Memory Space:n");`
		`return;`
		`}`
		`printf("nEnter the data value for the node:t" );`
		`scanf("%d",&temp->info);`
		`temp->next =NULL;`
		`if(start==NULL)`
		`{`
		`start=temp;`
		`}`
		`else`
		`{`
			[10]	3	L3

		temp->next=start;
		start=temp;
		}}			

2	a	What is a tree? Write the routines in C language to traverse (i)post-order t (ii)In-order and (iii) Pre-order.
		I) void postorder(root)
		{ root=postorder(root->left);
		root=postorder(root->right);
		printf(root->data)
		}
		ii)
		void Inorder(root)
		{ root= Inorder (root->left);
		printf(root->data)
		root= Inorder (root->right);
		}
		II) void preorder(root)
		{ printf(root->data)
		root=preorder(root->left);
		root=preorder(root->right);
		}
			[6]	4	L3

	b	 Write the output of the Post-order traversal ,In-order traversal and Pre-order traversal for the binary tree shown here. Preorder: 60,41,16,25,53,46,42,55,74,65,63,62,64,70 Inorder: 16,25,41,42,46,53,55,60,62,63,64,65,70,74 Post order: 25,16,42,46,55,53,41,62,64,63,70,65,74,60	[4]	4	L3
3		With the C-statements, explain how do you create a node, add and delete on Doubly linked list(DLL) with proper message where each node is containing the details of employee in the form of EmpId, EmpName, Empaddr and Empsalary as a data fields. #include <stdio.h> #include <stdlib.h>	[10]	3	L3

```
// Define the structure for an employee node
struct Employee {
    int EmpId;
    char EmpName[50];
    char EmpAddr[100];
    float EmpSalary;

    struct Employee* prev;
    struct Employee* next;
};

// Function to create a new employee node
struct Employee* createNode(int empId, const char* empName, const
char* empAddr, float empSalary) {
    struct Employee* newNode = (struct Employee*)malloc(sizeof(struct
Employee));
    if (newNode == NULL) {
        printf("Memory allocation failed. Exiting...\n");
        exit(1);
    }

    newNode->EmpId = empId;
    strcpy(newNode->EmpName, empName);
    strcpy(newNode->EmpAddr, empAddr);
    newNode->EmpSalary = empSalary;

    newNode->prev = NULL;
    newNode->next = NULL;

    return newNode;
}

// Function to add a new node to the end of the DLL
void addNode(struct Employee** head, struct Employee* newNode) {
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Employee* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }

    printf("Employee added successfully.\n");
}

// Function to delete a node with a specific EmpId from the DLL
void deleteNode(struct Employee** head, int empId) {
```

```
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

    struct Employee* temp = *head;

    // Search for the node with the specified EmpId
    while (temp != NULL && temp->EmpId != empId) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Employee with EmpId %d not found. Cannot delete.\n",
empId);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Employee with EmpId %d deleted successfully.\n", empId);
}

// Function to display the details of all employees in the DLL
void displayList(struct Employee* head) {
    printf("Employee Details:\n");
    while (head != NULL) {
        printf("EmpId: %d, EmpName: %s, EmpAddr: %s, EmpSalary:
%.2f\n", head->EmpId, head->EmpName, head->EmpAddr, head-
>EmpSalary);
        head = head->next;
    }
}

// Function to free the memory allocated for the DLL
void freeList(struct Employee* head) {
    struct Employee* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
```

```
    }
}

int main() {
    struct Employee* head = NULL;

    // Example: Adding nodes to the DLL
    struct Employee* emp1 = createNode(1, "John Doe", "123 Main St",
50000.00);
    struct Employee* emp2 = createNode(2, "Jane Smith", "456 Oak St",
60000.00);
    struct Employee* emp3 = createNode(3, "Bob Johnson", "789 Pine St",
70000.00);

    addNode(&head, emp1);
    addNode(&head, emp2);
    addNode(&head, emp3);

    // Displaying the list
    displayList(head);

    // Example: Deleting a node from the DLL
    deleteNode(&head, 2);

    // Displaying the updated list
    displayList(head);

    // Freeing the memory allocated for the DLL
    freeList(head);

    return 0;
}
```

4		Write an algorithm to add 2 polynomials using circular singly linked list . And also evaluate P (x, y ,z) = 6x2y2z-4yz5+3x2yz-2xyz3
		
		Ans:
		`NODE poly_add(NODE head1, NODE head2, NODE head3)`
		`{`
		`NODE a,b;`
		`int coeff;`
		`a = head1->link;`
		`b = head2->link;`
		`while(a != head1 && b != head2)`
		`{`
		`if(a->expon == b->expon)`
		`{`
		`coeff = a->coeff + b->coeff;`
		`if(coeff != 0)`
		`head3 = attach(coeff, a->expon, head3);  a = a->link;`
		`b = b->link;`
			[10]	1	L3

```
 }
 else if(a->expon > b->expon)
 {
 head3 = attach(a->coeff, a->expon, head3);  a = a->link;
 }
 else
 {
 head3 = attach(b->coeff, b->expon, head3);  b = b->link;
 }
   }
 while(a != head1)
 {
 head3 = attach(a->coeff, a->expon, head3);  a = a-
>link;
 }
 while(b != head2)
 {
 head3 = attach(b->coeff, b->expon, head3);  b = b-
>link;
 }
 return head3;
}
```

5		Implement the linkedlist representation for the given Sparse matrices. $\begin{bmatrix} 0 & 9 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	[10]	3	L3

6	a	Write QINSERT, QDELETE and Display function in c for queues using arrays.
		
		void enqueue()
		{
		int insert_item;
		if (Rear == SIZE - 1)
		printf("Overflow \n");
		else
		{
		if (Front == - 1)
		
		Front = 0;
		printf("Element to be inserted in the Queue\n : ");
		scanf("%d", &insert_item);
		Rear = Rear + 1;
		inp_arr[Rear] = insert_item;
		}
		}
		
		void dequeue()
		{
			[6]	3	L3

```
    if (Front == - 1 || Front > Rear)
    {
       printf("Underflow \n");
       return ;
    }
    else
    {
       printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
       Front = Front + 1;
    }
}
```

b

Differentiate between arrays and linked lists.

[4] | 2 | L2

ARRAY	LINKED LISTS
1. Arrays are stored in contiguous location.	1. Linked lists are not stored in contiguous location.
2. Fixed in size.	2. Dynamic in size.
3. Memory is allocated at compile time.	3. Memory is allocated at run time.
4. Uses less memory than linked lists.	4. Uses more memory because it stores both data and the address of next node.
5. Elements can be accessed easily.	5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operation takes time.	6. Insertion and deletion operation is faster.

CI **CCI** **HoD**

----------All the Best----------