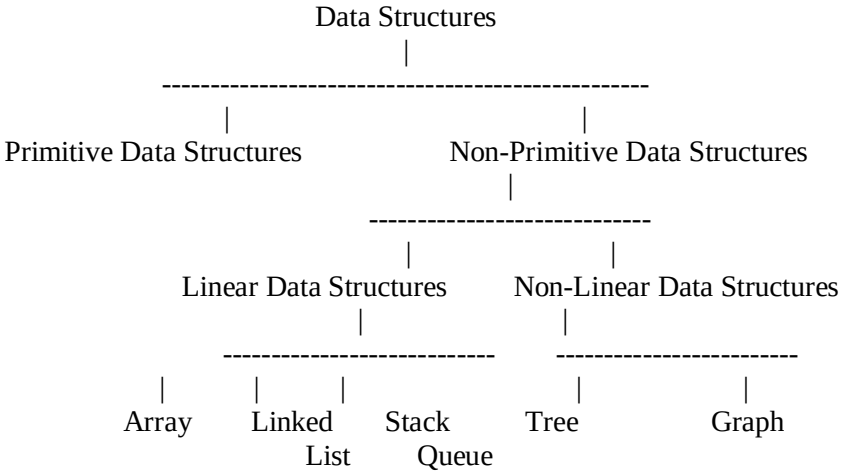
Roll No. | | | | |

Internal Assessment Test 1 – Nov 2024

Sub:	Data Structures and Applications - Solutions and Scheme					Sub code:	BCS304	Branch:	ISE
Date:	07-11-2024	Duration:	90 min's	Max Marks:	50	Sem / Sec:	III / A,B,C		OBE

	Answer any FIVE FULL QUESTIONS	MARKS	CO	RBT
1 (a)	**Define Data Structures. Classify them with a diagram and examples.** **Marks Distribution:** • Definition: 1 Mark • Classification Diagram: 2 Marks • Explanation with Examples: 2 Marks **Answer:** **Data Structures** are specialized formats for organizing, processing, retrieving, and storing data. They allow for efficient data management and support different types of data operations. **Classification of Data Structures:** Data structures can be classified into two main categories: 1. **Primitive Data Structures** 2. **Non-Primitive Data Structures**  • **Primitive Data Structures:** Examples include int, char, float, etc. • **Non-Primitive Data Structures:** Can be further classified as: o **Linear Data Structures:** Array, Linked List, Stack, Queue **Non-Linear Data Structures:** Trees, Graphs	5	CO1	L2

(b)	**Program in C for Pattern Matching: Find and Replace all occurrences of PAT in STR with REP.** **Marks Distribution:** • Logic for Pattern Matching: 2 Marks • Replace Operation: 2 Marks • Message if Pattern Not Found: 1 Mark **Answer:**	5	CO1	L3

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

void findAndReplace(char *str, char *pat, char *rep) {
    char buffer[1000];
    char *pos;
    int index = 0;
    int patLen = strlen(pat);
    int repLen = strlen(rep);
    int found = 0;

    buffer[0] = '\0'; // Initialize buffer to an empty string

    // Loop to find and replace all occurrences
    while ((pos = strstr(str, pat)) != NULL) {
        found = 1;
        // Copy characters from the start of str to the start of pattern
        strncpy(buffer + index, str, pos - str);
        index += pos - str;

        // Copy replacement into buffer
        strcpy(buffer + index, rep);
        index += repLen;

        // Move str to after the last replaced pattern
        str = pos + patLen;
    }

    if (!found) {
        printf("Pattern not found.\n");
    } else {
        strcpy(buffer + index, str); // Copy the remaining part of str
        printf("Updated String: %s\n", buffer);
    }
}

int main() {
    char str[100] = "hello world, world of programming!";
    char pat[10] = "world";
    char rep[10] = "earth";

    printf("Original String: %s\n", str);
    findAndReplace(str, pat, rep);
```

	return 0; } 			

2 (a)	**Define Stack. Implement `push()`, `pop()`, and `display()` with full and empty conditions.**
	
	Marks Distribution:
	
	• Stack Definition: 1 Mark
	• `push()` Implementation: 2 Marks
	• `pop()` and `display()` Implementations: 2 Marks
	
	Answer:
	
	Definition: A **Stack** is a linear data structure that follows the LIFO (Last In First Out) principle.
	
	clude <stdio.h>
	efine MAX 5
	
	stack[MAX];
	top = -1;
	
	d push(int value) {
	f (top == MAX - 1) {
	printf("Stack Overflow\n");
	else {
	stack[++top] = value;
	}
	
	pop() {
	f (top == -1) {
	printf("Stack Underflow\n");
	return -1;
	else {
	return stack[top--];
	}
	
	d display() {
	f (top == -1) {
	printf("Stack is empty\n");
	else {
	printf("Stack elements:\n");
	for (int i = top; i >= 0; i--) {
	printf("%d\n", stack[i]);
	}
	}
	
	main() {
	ush(10);
	ush(20);
	display();
		5	CO2	L3

	printf("Popped element: %d\n", pop());
	display();
	return 0;			

(b)	**Rules to Convert Infix to Postfix & Example Conversion** **Marks Distribution:** • Conversion Rules: 3 Marks • Example Conversion: 2 Marks **Answer:** **Rules to Convert Infix to Postfix:** 1. **Operands** go directly to the output. 2. **Operators** are pushed onto a stack. 3. **Parentheses**: o '(' is pushed to the stack. o ')' causes stack pop until '(' is encountered. 4. Operators of **higher precedence** are pushed first. 5. When the expression ends, pop all operators from the stack. **Example**: For the expression: `A * (B * C + D * E) + F` The Postfix conversion is: `ABC*DE*+*F+`	5	CO2	L2
3 (a)	**C Functions to Add Two Polynomials (Using Linked Lists)** **Marks Distribution:** • Linked List Representation: 2 Marks • Code for Addition of Polynomials: 3 Marks **Answer:** For adding polynomials: • Use linked lists to represent the terms. • Traverse both lists to sum terms with the same exponents. clude <stdio.h> clude <stdlib.h> edef struct PolyNode { nt coeff; nt exp; truct PolyNode *next; olyNode;	5	CO2	L3

```
yNode* createNode(int coeff, int exp) {
PolyNode* newNode = (PolyNode*)malloc(sizeof(PolyNode));
newNode->coeff = coeff;
newNode->exp = exp;
newNode->next = NULL;
eturn newNode;


yNode* addPoly(PolyNode* p1, PolyNode* p2) {
PolyNode* result = NULL;
PolyNode** ptr = &result;

while (p1 && p2) {
  if (p1->exp > p2->exp) {
     *ptr = createNode(p1->coeff, p1->exp);
     p1 = p1->next;
  } else if (p1->exp < p2->exp) {
     *ptr = createNode(p2->coeff, p2->exp);
     p2 = p2->next;
  } else {
     *ptr = createNode(p1->coeff + p2->coeff, p1->exp);
     p1 = p1->next;
     p2 = p2->next;
  }
  ptr = &((*ptr)->next);
}

while (p1) {
  *ptr = createNode(p1->coeff, p1->exp);
  p1 = p1->next;
  ptr = &((*ptr)->next);
}

while (p2) {
  *ptr = createNode(p2->coeff, p2->exp);
  p2 = p2->next;
  ptr = &((*ptr)->next);
}

eturn result;
```

(b) **Define Abstract Data Type (ADT) and Explain Queue ADT** — 5 — CO2 — L2

Marks Distribution:

- ADT Definition: 2 Marks
- Explanation of Queue ADT: 3 Marks

Answer:

	An **Abstract Data Type (ADT)** is a model defined by a set of values and operations on those values. ADTs provide only the interface, hiding implementation details.
	
	Queue ADT:
	
	• A linear structure following FIFO (First In First Out).
	• Operations:
	  ○ **enqueue()**: Add element to the rear.
	  ○ **dequeue()**: Remove element from the front.
	  ○ **peek()**: Access front element.			

4 (a)	Develop a C program to implement insertion, deletion and display operations on linear queue. **Marks Distribution:** • Syntax – 2 marks • Functions – 3 marks (Each 1 mark) **Answer:** #include<stdio.h> #include<stdlib.h> #define MAXSIZE 10 **int** Q**[**MAXSIZE**],**front=-1**,**rear=-1**;** **void** qinsert**(int** x**)** **{** **if(**rear==MAXSIZE-1**)** printf**("\n Queue is Full.");** **else if(**front==-1**)** **{** front=0**;** rear=0**;** Q**[**front**]**=x**;** **}** **else** **{** rear++**;** Q**[**rear**]**=x**;** **}** **}** **void** qdelete**()** **{** **if(**front==-1**)** printf**("\n Queue is Empty.");** **else if(**front==rear**)** **{** printf**("\n %d is removed from Queue.",**Q**[**front**]);** front=-1**;**	5	CO2	L3

```
 rear=-1;
}
else
{
 printf("\n %d is deleted from Queue.",Q[front]);
 front++;
}
}
void display()
{
 int i;
 printf("\n The Queue elements are...\n");
 if(front==-1)
  printf("\n No elements in Queue.");
 else
 {
  for(i=front;i<=rear;i++)
   printf(" %d ",Q[i]);
 }
}
int main()
{
 int choice,x;
 while(1)
 {
  printf("\n 1.Data insert\n 2.Data Delete\n 3.Data Display\n 4.Exit");
  printf("\n Please, Enter your choice : ");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: printf("\n Please, Enter the element : ");
     scanf("%d",&x);
     qinsert(x);
     break;
   case 2: qdelete();
     break;
   case 3: display();
     break;
   case 4: exit(0);
   default : printf("\n wrong Choice.");
  }
 }
}
```

(b)	What is Linked list? Explain the Different types of Linked list with a neat diagram. **Marks Distribution:** • Definition – 2 marks • Types with diagram – 3 marks	5	CO3	L2

Answer:

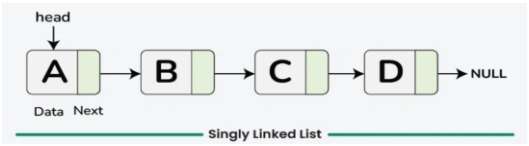
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.

Types Of Linked Lists:

1. Singly Linked List
2. Doubly Linked List
3. Singly Circular Linked List
4. Doubly Circular linked list

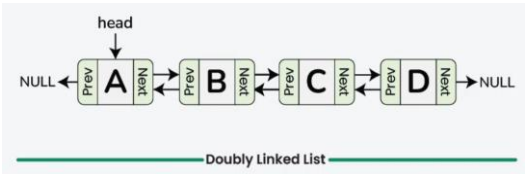
Singly Linked List

Singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way.



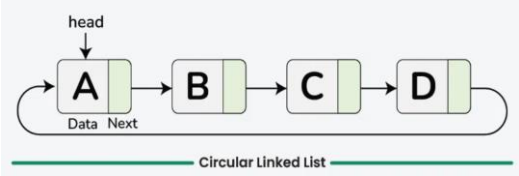
Doubly Linked List

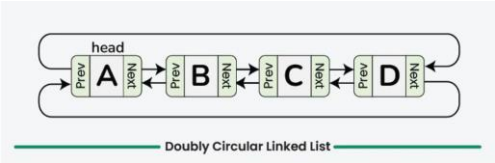
A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence. Therefore, it contains three parts of data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well.



Singly Circular Linked List

A Singly circular linked list is a type of linked list in which the last node's next pointer points back to the first node of the list, creating a circular structure. This design allows for continuous traversal of the list, as there is no null to end the list.



	Doubly Circular linked list Doubly Circular linked list or a circular two-way linked list is a complex type of linked list that contains a pointer to the next as well as the previous node in the sequence  			

5 (a)	Give the Structure representation of Doubly linked list and write the c function for following:
	i) Insert an element at an end of DLL
	ii) Delete a node at the beginning of DLL
	
	Mark Distribution:
	• Description and representation: 3 marks
	• Function : 2 marks(each one)
	
	Structure of a Doubly Linked List Node:
	
	Each node in a doubly linked list includes:
	• **Data:** The actual information held within the node, which could be numbers, strings, or any other data type.
	• **Next Pointer:** A reference to the next node in the list, which helps in traversing the list forward.
	• **Prev Pointer:** A reference to the previous node in the list, which facilitates backward traversal.
	
	i) Insert an element at an end of DLL
	
	void insert_end()
	{
	EMPLOYEE node, temp;
	node = create();
	if (start == NULL) /*If the list is empty.*/
	{
	start = node;
	}
	else
	{
	temp = start;
	while (temp->rlink != NULL) /*Traverse till the end of the list.*/
	{
	temp = temp->rlink;
	}
	temp->rlink = node; /*Temp's right link is assigned the address of node.*/
	node->llink = temp; /*Node's left link is assigned the address of temp.*/
	}
	}
		5	CO3	L3

ii) Delete a node at the beginning of DLL

```
void delete_front()
{
EMPLOYEE temp;
temp = start;
if ( temp == NULL ) /*If the list is empty.*/
{
printf ( "\nList is Empty" );
}
else if ( temp->rlink == NULL ) /*If there is one node in the list.*/
{
printf ( "\nThe deleted employee ssn is %s", temp->ssn );
free ( temp );
start = NULL;
}
else /*If there are many nodes.*/
{
start = temp->rlink; /*Assign the address of next node which is present in start's right link to

start.*/

start->llink = NULL;
printf ( "\nThe deleted employee ssn is %s", temp->ssn );
free ( temp );
}
}
```

(b) Define sparse matrix. For the given sparse matrix, give the linked list representation: 5 CO3 L2

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Mark Distribution:

- Definition : 2 Marks
- Linked list representation : 3 Marks

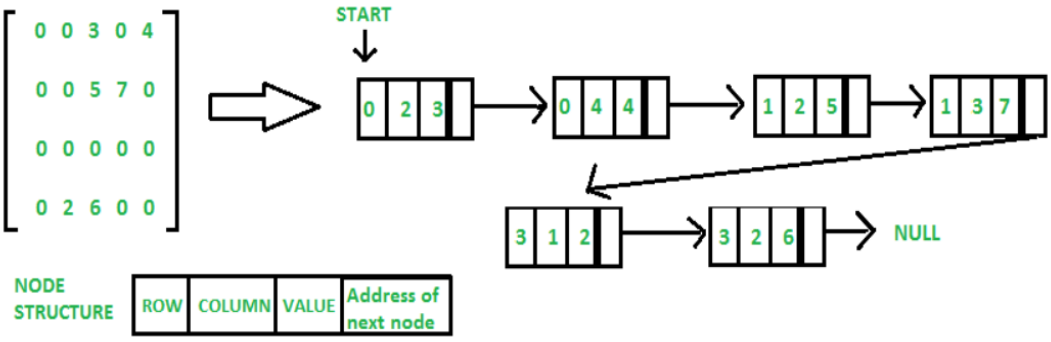
Answers:

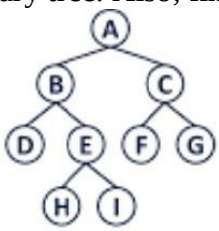
Sparse Matrix:

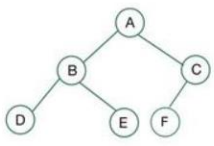
A **sparse matrix** is a matrix in which most of the elements are zero. This type of matrix is useful in scenarios where storage and computational efficiency are critical, as we can store only the non-zero elements rather than all elements, including the zeros.

In linked list, each node has four fields. These four fields are defined as:

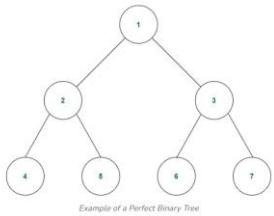
- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non-zero element located at index – (row,column)
- **Next node:** Address of the next node



6 (a)	Write recursive C functions for in-order, pre-order and post-order traversals of a binary tree. Also, find all the traversals for the given tree.
	 
	
	Marks Distribution:
	• Recursive Function : 3 marks (1 mark each)
	• Traversal : 2 marks
	Answers:
	INORDER:
	`void inorder(struct Node* root) {`
	`  if (root == NULL)`
	`  return;`
	`  inorder (root->left);`
	`  printf("%d ", root->data);`
	`  inorder (root->right);`
	`}`
	PRE-ORDER:
	`void preorder(struct Node* root) {`
	`  if (root == NULL)`
	`  return;`
	`  printf("%d ", root->data);`
	`  preorder (root->left);`
	`  preorder (root->right);`
	`}`
	POST-ORDER:
	`void postorder(struct Node* root) {`
	`  if (root == NULL)`
	`  return;`
	`  postorder (root->left);`
	`  postorder (root->right);`
	`  printf("%d ", root->data);`
	`}`
		5	CO4	L3

	inorder - D B H E I A F C G **preorder** - A B D E H I C F G **postorder** - D H I E B F G C A			
(b)	Define Binary Tree. Explain the Types of Binary Tree with a neat diagram. **Marks Distribution:** • Definition: 2 marks • Types : 3 marks **Answer:** A binary tree is a tree-type non-linear data structure with a maximum of two children for each parent. Every node in a binary tree has a left and right reference along with the data element. The node at the top of the hierarchy of a tree is called the root node. The nodes that hold other sub-nodes are the parent nodes. A parent node has two child nodes: the left child and right child. **Types of Binary Tree** 1. Skewed Binary Trees 2. Complete Binary Trees 3. Full Binary Tree 4. Extended Binary tree **Skewed Binary Trees** A skewed binary tree is a type of binary tree in which all the nodes have only either one child or no child. There are two types of skewed binary tree: left skewed and right skewed binary tree.  **Complete Binary Trees** A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.  *Complete Binary Tree* **Full Binary Tree** A **perfect binary tree** or Full Binary tree is a special type of binary tree in which all the leaf nodes are at the same depth, and all non-leaf nodes have two children. In simple terms,	5	CO4	L2

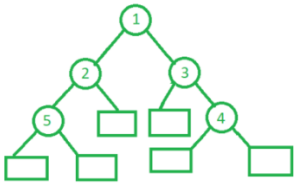
this means that all leaf nodes are at the maximum depth of the tree, and the tree is completely filled with no gaps.



Example of a Perfect Binary Tree

Extended binary tree

Extended binary tree is a type of binary tree in which all the null sub tree of the original tree are replaced with special nodes called **external nodes** whereas other nodes are called **internal nodes**



Extended Binary Tree

CI **CCI** **HOD**