Data Structures and Algorithms Practical

Practical 1: Write a program in C to display following patterns

a)
```
1
1 2
1 2 3
1 2 3 4
```

b)
```
        1
      2   2
    3   3   3
      2   2
        1
```

Code: a)

```
#include <stdio.h>
 int main() {
    int n, i, j;
    printf("Enter the number of rows: ");
    scanf("%d", &n);

    for(i = 1; i <= n; i++) {
        for(j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");    }

    return 0;
}
```

Sample Output:

Enter the number of rows: 4

1

1 2

1 2 3

1 2 3 4

b)

```
#include <stdio.h>
int main() {
```

```
    int n, i, j, space;
    printf("Enter the number of rows: ");
    scanf("%d", &n);

    for(i = 1; i <= n; i++) {
        // Print spaces
        for(space = 1; space <= n - i; space++) {
            printf(" ");
        }
        // Print asterisks
        for(j = 1; j <= (2 * i - 1); j++) {
            printf("*");
        }
        // Move to the next line
        printf("\n");
    }
  return 0;
}
```

Sample output

Enter number of rows for the upper half of the diamond: 4

```
   1
  2 2
 3 3 3
4 4 4 4
 3 3 3
  2 2
   1
```

---

**Oral question**

---

1. **What is a *nested loop*?**
   **A loop inside another loop is called a nested loop**
2. **Which loop is used for pattern printing here?**
   **The for loop is used — one outer loop for rows and one inner loop for columns.**
3. **Why `for` loop used for in C**
   A **`for` loop** in C is used when you want to **repeat a set of statements a fixed number of times**

---

**Practical 2: Write a C program to display a) pyramid of alphabets b) pyramid of Asterisks**

## A) Code for printing pyramid of alphabets:

```
#include <stdio.h>
int main()

{
    int n, i, j;
    char ch;
    printf("Enter the number of rows: ");
    scanf("%d", &n);
    for(i = 1; i <= n; i++)
    {
        // Print leading spaces
        for(j = 1; j <= n - i; j++)
        {
            printf(" ");
        }
        // Print ascending characters
        for(ch = 'A'; ch < 'A' + i; ch++)
        {
            printf("%c", ch);
        }
        // Print descending characters
        for(ch = 'A' + i - 2; ch >= 'A'; ch--)
        {
          printf("%c", ch);
        }

        printf("\n");
    }
    return 0;
}
```

**Sample Output:**

**Enter the number of rows: 4**

```
      A
     ABA
   ABCBA
  ABCDCBA
```

**B) Code for printing pyramid of asterisk pattern:**

```
#include <stdio.h>
int main()
{
    int n, i, j, space;
    printf("Enter the number of rows: ");
    scanf("%d", &n);
```

```
    for(i = 1; i <= n; i++)
    {
        // Print spaces
        for(space = 1; space <= n - i; space++)
        {
          printf(" ");
        }
        // Print asterisks
        for(j = 1; j <= (2 * i - 1); j++)
        {
          printf("*");
        }
        // Move to the next line
        printf("\n");
    }
  return 0;
}
```

**Sample Output:**

**Enter the number of rows: 5**

```
    *
   ***
  *****
 *******
*********
```

**Practical 3: Student Database Management**

Implement C program for developing a student result management system. The database should support adding new entries and sorting based on performance.

**Code:**

```
#include <stdio.h>

struct Student
{
    int roll;
    char name[50];
    float marks;
};

int main()
{
    struct Student s[100], temp;
    int n, i, j;

    printf("Enter number of students: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        printf("\nEnter roll, name, marks: ");
        scanf("%d %s %f", &s[i].roll, s[i].name, &s[i].marks);
    }

    // Sort by marks (descending)
    for(i = 0; i < n-1; i++) {
        for(j = i+1; j < n; j++) {
            if(s[i].marks < s[j].marks) {
                temp = s[i];
                s[i] = s[j];
                s[j] = temp;
            }
        }
    }

    printf("\n--- Student Result List (Sorted by Marks) ---
\n");
    printf("Roll\tName\tMarks\n");
    for(i = 0; i < n; i++)
{
   printf("%d\t%s\t%.2f\n", s[i].roll, s[i].name, s[i].marks);
}
    return 0;
}
```

**Sample Output:**

```
Enter the number of students: 2

Enter details for student 1:
Name: q
Roll Number: 1
Marks: 34

Enter details for student 2:
Name: a
Roll Number: 2
Marks: 56

Students sorted by marks (highest to lowest):
-----------------------------------------------------------------------------------------------------------
Roll No    Name       Marks
-----------------------------------------------------------------------------------------------------------
2     a               56.00
1     q               34.00
```

---

**Oral question**

---

**Q1. What is a *structure* in C?**
**A1. A structure in C is a user-defined data type that groups different types of data under one name.**
**👉 Example: A student has roll, name, and marks, all stored together using a struct.**

---

**Q2. Why do we use a structure in this program?**
**A2. We use a structure to store multiple details (roll number, name, and marks) of each student in a single record.**

---

**Q3. How is an array of structures used here?**
**A3. An array of structures (struct Student s[100]) is used to store data of multiple students — one structure for each student.**

---

**Q7. Which sorting technique is used here?**
**A7. Bubble Sort (or simple comparison-based sorting) — it compares marks of each pair of students and swaps them if needed.**

---

**Practical 4: Write a C program for implementation of stack using array**

**Code:**

```
#include <stdio.h>
#define SIZE 100

int stack[SIZE];
int top = -1;

void push(int value) {
    if (top == SIZE - 1)
        printf("Stack Overflow!\n");
    else
        stack[++top] = value;
}

void pop() {
    if (top == -1)
        printf("Stack Underflow!\n");
    else
        printf("Popped: %d\n", stack[top--]);
}

void display() {
    if (top == -1)
        printf("Stack is empty!\n");
    else {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--)
            printf("%d ", stack[i]);
        printf("\n");
    }
}

int main() {
    int choice, value;

    while (1) {
        printf("\n1.Push  2.Pop  3.Display  4.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
```

```
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

**Sample Output:**

Enter the size of STACK[MAX=100]:5

STACK OPERATIONS USING ARRAY

---

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter the Choice:1
Enter a value to be pushed:34

Enter the Choice:1
Enter a value to be pushed:23

---

Oral Questions

---

**Q1.** What is a stack?

👉 A stack is a list where the last item added is the first to be removed.

**Q2.** What rule does a stack follow?

👉 LIFO – Last In, First Out.

**Q3.** What are the main stack operations?

👉 Push (insert), Pop (delete), and Display.

**Q4.** What is top?

👉 It shows the position of the last element in the stack.

**Q5.** What is the initial value of top?

👉 -1, means the stack is empty.

**Q6.** What is **push**?

👉 Adding (inserting) an element to the top of the stack.

**Q7.** What is **pop**?

👉 Removing (deleting) the top element from the stack.

**Q8.** What is **stack overflow**?

👉 When we try to push into a full stack.

**Q9.** What is **stack underflow**?

👉 When we try to pop from an empty stack.

**Q10.** How are elements displayed?

👉 From top to bottom.

**Q11.** Is this stack static or dynamic?

👉 Static (because it uses an array).

**Q12.** What is the time for push and pop?

👉 Constant time – O(1).

**Q13.** Can we use linked list for stack?

👉 Yes, it can be done using linked list also.

**Q14.** What does #define SIZE 100 mean?

👉 It sets the maximum size of the stack to 100.

**Q15.** What does LIFO mean?

👉 Last In, First Out – the last added comes out first.

---

**Practical 5: Decimal to binary conversion using stack**

```
#include <stdio.h>
#define SIZE 100

int stack[SIZE];
int top = -1;

void push(int n) {
    stack[++top] = n;
}

int pop() {
    return stack[top--];
}

int main()
{
    int num, rem;

    printf("Enter a decimal number: ");
    scanf("%d", &num);

    // Push remainders to stack
    while (num > 0) {
        rem = num % 2;
        push(rem);
        num = num / 2;
    }

    printf("Binary equivalent: ");
    while (top != -1)
        printf("%d", pop());

    printf("\n");
    return 0;
}
```

**Sample Output:**

```
Enter a decimal number: 5
Binary equivalent: 101
```

---

Oral Questions

---

**Q5.** What is the rule followed by stack?
👉 LIFO − Last In, First Out.

**Q6.** What is the role of push() function?
👉 To insert remainders (0 or 1) into the stack.
**Q7.** What is the role of pop() function?
👉 To remove and print elements from the stack in reverse order.

---

**Practical 6:** Implement bubble sort to reorder product prices in an online store to help customers compare them easily.

**Code:**

```
#include <stdio.h>
void bubbleSort(float arr[], int n) {
    int i, j;
    float temp;
    for(i = 0; i < n - 1; i++) {
        for(j = 0; j < n - i - 1; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

```
int main() {
    float price[100];
    int n, i;
    printf("Enter number of products: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        printf("Enter price %d: ", i + 1);
        scanf("%f", &price[i]);
    }

    bubbleSort(price, n);

    printf("\nSorted prices (Low → High):\n");
    for(i = 0; i < n; i++) {
        printf("₹%.2f\n", price[i]);
    }

    return 0;
}
```

**Sample Output:**

```
Enter the number of products: 4
Enter the prices of the products:
Price 1: ₹300
Price 2: ₹327
Price 3: ₹543
Price 4: ₹567
Sorted prices:
₹300.00
₹327.00
₹543.00
₹567.00
```

---

**Oral Questions**

---

**Q2.** What is bubble sort?

👉 It is a sorting method where adjacent elements are compared and swapped if they are in the wrong order.

**Q4.** What type of sorting is done here?

👉 Ascending order (Low → High).

**Q5.** What is the time complexity of bubble sort?

👉 O(n²).

**Q6.** How many loops are used in bubble sort?

👉 Two nested loops.

**Q7.** What does the inner loop do?

👉 It compares and swaps adjacent elements.

**Q8.** What does the outer loop do?

👉 It controls the number of passes (rounds).

**Q9.** What happens if the array is already sorted?

👉 No swaps are made; it remains the same.

---

**Practical 7:** Implement selection sort to reorder product prices in an online store to help customers compare them easily.

Code:

```
#include <stdio.h>

int main() {

    float price[100], temp;

    int n, i, j, minIndex;

    printf("Enter number of products: ");

    scanf("%d", &n);

    for(i = 0; i < n; i++) {

        printf("Enter price %d: ", i + 1);

        scanf("%f", &price[i]);

    }


    // Selection Sort (ascending order)

    for(i = 0; i < n - 1; i++) {

        minIndex = i;

        for(j = i + 1; j < n; j++) {

            if(price[j] < price[minIndex]) {

                minIndex = j;

            }
```

```
        }

        // Swap

        temp = price[i];

        price[i] = price[minIndex];

        price[minIndex] = temp;

    }

    printf("\nSorted prices (Low → High):\n");

    for(i = 0; i < n; i++) {

        printf("₹%.2f\n", price[i]);

    }

    return 0;

}
```

**Sample output:**

**Enter the number of products: 3**

**Enter the prices of the products:**

**Price 1: ₹230**

**Price 2: ₹5000**

**Price 3: ₹29**

**Sorted prices:**

**₹230.00**

**₹2900.00**

**₹5000.00**

---

**Oral Questions**

---

**Q2. What is selection sort?**

**👉 It is a sorting method that selects the smallest element and places it in the correct position one by one.**

**Q3. How does selection sort work?**

**👉 It finds the minimum value in the unsorted part and swaps it with the first element of that part.**

**Q4. How many loops are used in selection sort?**

**👉 Two loops – one for selecting position and another for finding the smallest element.**

**Q7. What is the time complexity of selection sort?**

👉 **O(n²).**

**Q8. What type of sorting is done here?**

👉 **Ascending order (Low → High).**

---

**Practical 8:** Implement Insertion sort to reorder product prices in an online store to help customers compare them easily.

**Code:**

```
#include <stdio.h>

int main() {
    float price[100], key;
    int n, i, j;

    printf("Enter number of products: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        printf("Enter price %d: ", i + 1);
        scanf("%f", &price[i]);
    }

    // Insertion Sort (ascending order)
    for(i = 1; i < n; i++) {
        key = price[i];
        j = i - 1;

        // Move larger elements one position ahead
        while(j >= 0 && price[j] > key) {
            price[j + 1] = price[j];
            j--;
        }
        price[j + 1] = key;
    }

    printf("\nSorted prices (Low → High):\n");
    for(i = 0; i < n; i++) {
        printf("₹%.2f\n", price[i]);
    }

    return 0;
}
```

**Sample Output:**
Enter the number of products: 2
Enter the prices of the products:
Price 1: ₹5000
Price 2: ₹2344
Sorted prices:
₹2344.00
₹5000.00

---

**Oral Questions**

---

**Q2. What is insertion sort?**
**👉 It is a sorting method where elements are picked one by one and placed in the correct position among already sorted elements.**
**Q3. How does insertion sort work?**
**👉 It compares the current element with previous ones and shifts larger values to the right, then inserts the element at the right place.**
**Q6. What type of sorting is done here?**
**👉 Ascending order (Low → High).**
**Q8. What is the time complexity of insertion sort?**
**👉 O(n²).**
**Q10. Is insertion sort better than bubble sort?**
**👉 Yes, for small or nearly sorted data, insertion sort is faster.**

---

**Practical 9:** To implement a singly linked list ,insert/ delete node at the end of linked list

**Code:**

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

// Define structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to insert at the end
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
```

```
    temp->next = newNode;
}

// Function to delete from the end
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct Node* temp = *head;

    if (temp->next == NULL) {
        printf("Deleted: %d\n", temp->data);
        free(temp);
        *head = NULL;
        return;
    }

    struct Node* prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }

    printf("Deleted: %d\n", temp->data);
    prev->next = NULL;
    free(temp);
}

// Function to display the list
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Main function
int main() {
    struct Node* head = NULL;
    int choice, value;

    while (1) {
        printf("\n--- Singly Linked List Menu ---\n");
        printf("1. Insert at End\n");
        printf("2. Delete from End\n");
```

```
        printf("3. Display List\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;
            case 2:
                deleteFromEnd(&head);
                break;
            case 3:
                display(head);
                break;
            case 4:
                printf("Exiting program...\n");
                exit(0);
            default:
                printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

**Sample Output:**

**--- Singly Linked List Menu ---**

**1. Insert at End**

**2. Delete from End**

**3. Display List**

**4. Exit**

**Enter your choice: 1**

**Enter value to insert: 24**

**--- Singly Linked List Menu ---**

**1. Insert at End**

**2. Delete from End**

**3. Display List**

**4. Exit**

**Enter your choice: 1**

**Enter value to insert: 67**

**--- Singly Linked List Menu ---**

**1. Insert at End**

**2. Delete from End**

**3. Display List**

**4. Exit**

**Enter your choice: 3**

**Linked List: 24 -> 67 -> NULL**

---

Oral Questions

---

Q2. What is a linked list?

👉 A linked list is a collection of nodes where each node contains data and a pointer to the next node.

Q3. What are the parts of a node?

👉 1️⃣ Data part — stores the value

👉 2️⃣ Next part — stores the address of the next node

Q4. What is the difference between an array and a linked list?

👉 In arrays, elements are stored in continuous memory; in linked lists, they are connected using pointers.

Q5. What is the use of malloc() function?

👉 It dynamically allocates memory for a new node at runtime.

Q8. How do we check if the list is empty?

👉 If head == NULL, the list is empty.

**Practical 10: To create binary search tree and In order traversal**

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

// Structure for a node
struct Node {
    int data;
    struct Node *left, *right;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

// Function for Inorder Traversal (Left → Root → Right)
void inorder(struct Node* root) {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

// Main function
int main() {
    struct Node* root = NULL;
    int n, val;

    printf("How many numbers? ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter number %d: ", i + 1);
        scanf("%d", &val);
```

```
        root = insert(root, val);
    }

    printf("\nInorder Traversal: ");
    inorder(root);
    printf("\n");

    return 0;
}
```

**Sample Output:**

How many numbers? 4
Enter number 1: 23
Enter number 2: 21
Enter number 3: 12
Enter number 4: 56

Inorder Traversal: 12 21 23 56

---

**Oral Questions**

---

**Q2. What is a Binary Search Tree?**

**👉 A BST is a tree where each node has at most two children –**
**Left child < Root < Right child.**

**Q3. What is a node in a tree?**

**👉 A node is a structure that contains data and links to left and right sub-nodes.**

**Q8. What is inorder traversal?**

**👉 It visits nodes in the order: Left → Root → Right.**

**Q9. What will inorder traversal of a BST produce?**

**👉 Sorted (ascending) order of elements.**

**Q15. What are the three types of tree traversals?**

**👉 Inorder, Preorder, and Postorder.**

---

**Practical 11: Implement binary search technique to search a name in contact list**

**Code:**

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

// Function to sort names alphabetically (Bubble Sort)
void sortContacts(char names[][50], int n) {
    char temp[50];
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (strcmp(names[j], names[j + 1]) > 0) {
                strcpy(temp, names[j]);
                strcpy(names[j], names[j + 1]);
                strcpy(names[j + 1], temp);
            }
        }
    }
}
```

```
// Function for Binary Search
int binarySearch(char names[][50], int n, char target[]) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        int cmp = strcmp(names[mid], target);

        if (cmp == 0)
            return mid;
        else if (cmp < 0)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

int main() {
    char contacts[10][50] = {"ASK", "CRK", "SPT", "PRJ",
                             "ESY", "AAR", "GDR", "SHC", "VIT",
"NID"};
    int n = 10;
    char target[50];

    printf("Enter the name to search: ");
    scanf("%s", target);

    // Sort before searching
    sortContacts(contacts, n);

    int pos = binarySearch(contacts, n, target);

    if (pos != -1)
        printf("Binary Search: %s found at sorted index %d\n",
target, pos);
    else
        printf("Binary Search: %s not found.\n", target);

    return 0;
}
```

**Sample Output:**

```
Enter the name to search: SHC
Binary Search: SHC found at sorted index 8
```

**Q2. What is binary search?**

👉 **Binary search is a searching method that divides the sorted list into halves to find an element quickly.**

**Q9. What is the time complexity of binary search?**

👉 **O(log n).**

**Q10. What is the time complexity of bubble sort?**
**👉 O(n²).**

---
-

**Practical 12:Write a c program to implement queue using array**
**Code:**

```
#include <stdio.h>
#define SIZE 5   // maximum size of the queue

int main() {
   int queue[SIZE];
   int front = 0, rear = 0;
   int choice, item, i;

   printf("Queue using Array\n");
   printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit\n");

   while (1) {
      printf("\nEnter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1: // Insert
            if (rear == SIZE)
               printf("Queue is Full!\n");
            else {
               printf("Enter element: ");
               scanf("%d", &item);
               queue[rear++] = item;
               printf("%d inserted into queue.\n", item);
            }
            break;

         case 2: // Delete
            if (front == rear)
               printf("Queue is Empty!\n");
            else {
               printf("Deleted element: %d\n", queue[front]);
               front++;
            }
            break;

         case 3: // Display
            if (front == rear)
               printf("Queue is Empty!\n");
            else {
               printf("Queue elements are:\n");
               for (i = front; i < rear; i++)
```

```
            printf("%d\n", queue[i]);
        }
        break;

    case 4: // Exit
        printf("Exiting program...\n");
        return 0;

    default:
        printf("Invalid choice! Try again.\n");
  }
}

  return 0;
```

**Sample Output:**

```
Queue using Array
1.Insertion
2.Deletion
3.Display
4.Exit
Enter the Choice:1

Enter no 1:45

Enter the Choice:1

Enter no 2:78

Enter the Choice:3

Queue Elements are:
 45
78
```

Q2. What is a queue?

👉 A queue is a linear data structure that follows the FIFO (First In, First Out) principle.

Q3. What does FIFO mean?

👉 The element inserted first is removed first.

Q4. What are front and rear used for?

👉

- front → points to the first element (for deletion)
- rear → points to the last position (for insertion)

Q5. What happens when the queue is full?

👉 It shows "Queue is Full" (Overflow condition).

**Q6. What happens when the queue is empty?**

👉 **It shows "Queue is Empty" (Underflow condition).**

**Q7. Which operations are performed on a queue?**

👉 **Insertion (enqueue) and Deletion (dequeue).**

**Q8. What is the role of the SIZE constant?**

👉 **It defines the maximum capacity of the queue.**

**Q9. In which end is insertion done?**

👉 **At the rear end.**

**Q10. From which end is deletion done?**

👉 **From the front end.**

**Q11. What is the time complexity of insertion and deletion in a queue?**

👉 **Both are O(1).**

**Q12. What data structure is used here to store the queue elements?**

👉 **An array.**

**Q13. What is displayed when user selects "Display"?**

👉 **All elements of the queue from front to rear.**

**Q14. What is a real-life example of a queue?**

👉 **Line of people waiting for a bus or ATM.**

**Q15. What happens when front == rear?**

👉 **The queue is empty.**