ASSIGNMENT

1) A program P reads in 500 integers in the range [0..100] representing the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?

A program reads 500 integers (scores of students) ranging from 0 to 100 and prints the frequency of scores above 50. The best way to store these frequencies is with an array of size 51:

1. Array Setup: Create an array named frequency with 51 elements. Each index represents scores from 51 to 100:

- frequency[0] for score 51
- frequency[1] for score 52
- ….
- frequency[49] for score 100

2. Counting Scores: As each score is read, if it's above 50, increase the count at the corresponding index using score - 51.

3. Display Results: After all scores are processed, go through the frequency array and print counts for scores from 51 to 100.

2) Consider a standard Circular Queue q; implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10]. The front and rear pointers are initialized to point at q[2] . In which position will the ninth element be added?

In a circular queue of size 11, where both front and rear start at index 2, we want to find where the ninth element will be added:

- Initial Pointers:
  - Front = 2
  - Rear = 2

As we add elements:

- 1st element: Rear moves to q[3]
- 2nd element: Rear moves to q[4]
- 3rd element: Rear moves to q[5]
- 4th element: Rear moves to q[6]
- 5th element: Rear moves to q[7]
- 6th element: Rear moves to q[8]
- 7th element: Rear moves to q[9]
- 8th element: Rear moves to q[10]
- 9th element: Rear wraps around to q[0].

Thus, the ninth element is added at position q[0].

3)Write a C Program to implement Red Black Tree ?

```
#include <stdio.h>
#include <stdlib.h>
typedef enum { RED, BLACK } Color;
typedef struct Node {
    int data;
    Color color;
    struct Node *left, *right, *parent;
} Node;
Node *root = NULL;
Node *createNode(int data);
void rotateLeft(Node *&root, Node *&pt);
void rotateRight(Node *&root, Node *&pt);

void fixViolation(Node *&root, Node *&pt);
```

```
void insert(const int &data);
void inorder(Node *root);
void printTree(Node *root, int space);
int main() {
    insert(7);
    insert(3);
    insert(18);
    insert(10);
    insert(22);
    insert(8);
    insert(11);
    insert(26);
    printf("Inorder Traversal of Created Tree:\n");
    inorder(root);
    printf("\nTree Structure:\n");
    printTree(root, 0);
    return 0;
}
Node *createNode(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->color = RED;
    newNode->left = newNode->right = newNode->parent = NULL;
    return newNode;
}
```

Explanation:

- Node Structure: Each node holds data, color (RED or BLACK), and pointers to its children and parent.

- Insertion: The insert function creates a new node, places it using bstInsert, and then calls fixViolation to maintain Red-Black properties.

- Rotations: The functions rotateLeft and rotateRight keep the tree balanced.

- Fixing Violations: The fixViolation function ensures the tree follows Red-Black rules after inserting a new node.

- Traversal: The inorder function performs an in-order traversal, and printTree visualizes the tree structure.

Submitted by,

Akhil Shaji

S1 MCA

Submitted to,

Ms.Akshara Sasidaran