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?

To store the frequencies of each score above 50 efficiently, the best way for program P would be to use an **array** where the index represents the score and the value at each index represents the frequency of that score.

Approach:

1. **Declare an array of size 101** (since the score range is [0..100]), indexed from 0 to 100. This array will store the frequency of all scores, but the program will only print frequencies of scores greater than 50.
2. **Initialize all values of the array to 0**.
3. **For each input score**, increment the value at the corresponding index in the array. For example:
    - If the score is 55, increment frequency[55].
    - If the score is 78, increment frequency[78].
4. **After processing all scores**, iterate over the array from index 51 to 100 and print the frequencies for scores greater than 50.

Example:

- **Declare the frequency array:**

```
int frequency[101] = {0}; // Array to store frequencies of scores 0 to 100
```

- **Processing the input:** For each input score x:

```
frequency[x]++;
```

- **Printing frequencies of scores above 50:**

```
for (int i = 51; i <= 100; i++) {
    if (frequency[i] > 0) {
        printf("Score %d: %d students\n", i, frequency[i]);
    }
}
```

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 implementation, the way elements are added and where the pointers are located determines where new elements will be stored. Given that:

- The size of the queue is 11.
- The front and rear pointers are both initialized to point at q[2].

In a circular queue, the rear pointer is used to determine where the next element will be added, and the front pointer indicates where the next element will be removed. Here’s how you can calculate the position of

the ninth element:

1. **Initial State**:
    - Front (f) = 2
    - Rear (r) = 2
2. **Adding Elements**:
    - When the first element is added, r will move to (2 + 1) % 11 = 3.
    - When the second element is added, r will move to (3 + 1) % 11 = 4.
    - Continue this until the ninth element.
3. **Calculating Positions**:
    - 1st element: r = 3
    - 2nd element: r = 4
    - 3rd element: r = 5
    - 4th element: r = 6
    - 5th element: r = 7
    - 6th element: r = 8
    - 7th element: r = 9
    - 8th element: r = 10
    - 9th element: r = 0

So, the position where the ninth element will be added is 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* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = newNode->parent = NULL;
    newNode->color = RED;  // New node must be red
    return newNode;
}
void leftRotate(Node** root, Node* x) {
    Node* y = x->right;
    x->right = y->left;
    if (y->left != NULL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
```

```
}
void rightRotate(Node** root, Node* y) {
    Node* x = y->left;
    y->left = x->right;
    if (x->right != NULL)
        x->right->parent = y;
    x->parent = y->parent;
    if (y->parent == NULL)
        *root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;
}
void fixViolation(Node** root, Node* z) {
    while (z->parent != NULL && z->parent->color == RED) {
        if (z->parent == z->parent->parent->left) {
            Node* y = z->parent->parent->right;
            if (y != NULL && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
```

```
                z = z->parent;
                leftRotate(root, z);
            }
            z->parent->color = BLACK;
            z->parent->parent->color = RED;
            rightRotate(root, z->parent->parent);
        }
    } else {
        Node* y = z->parent->parent->left;
        if (y != NULL && y->color == RED) {
            z->parent->color = BLACK;
            y->color = BLACK;
            z->parent->parent->color = RED;
            z = z->parent->parent;
        } else {
            if (z == z->parent->left) {
                z = z->parent;
                rightRotate(root, z);
            }
            z->parent->color = BLACK;
            z->parent->parent->color = RED;
            leftRotate(root, z->parent->parent);
        }
    }
  }
  (*root)->color = BLACK;
}
```

```
// Function to insert a new node
void insert(Node** root, int data) {
    Node* z = createNode(data);
    Node* y = NULL;
    Node* x = *root;
    while (x != NULL) {
        y = x;
        if (z->data < x->data)
            x = x->left;
        else
            x = x->right;
    }
    z->parent = y;
    if (y == NULL) {
        *root = z;  // Tree was empty
    } else if (z->data < y->data) {
        y->left = z;
    } else {
        y->right = z;
    }
    fixViolation(root, z);
}
void inOrder(Node* root) {
    if (root == NULL)
        return;
    inOrder(root->left);
    printf("%d ", root->data);
```

```
    inOrder(root->right);
}
int main() {
    Node* root = NULL;
    // Inserting nodes
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 30);
    insert(&root, 15);
    insert(&root, 25);
    insert(&root, 5);
    insert(&root, 1);
    printf("In-order Traversal of Red-Black Tree: ");
    inOrder(root);
    printf("\n");
    return 0;
}
```

Submitted By

Lakshmi H

S1 MCA

Submitted To

Ms. Akshara Sasidaran