ASSIGNMENT

1. **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?

ANS:To store the frequencies of scores above 50 from the range [0..100], the best approach for program P is to use an array of size 51 (indices 0 to 50). This array will represent the counts of scores from 51 to 100, where the index corresponds to the score itself.

- **Array Initialization**: Create an array frequency of size 51, initialized to zero. The index 0 will correspond to the score 51, index 1 to score 52, and so forth, up to index 49 corresponding to score 100.
- **Reading Scores**: As you read each of the 500 scores, if a score is greater than 50, increment the corresponding index in the frequency array. For example, if the score is 75, increment frequency[24] (since 75 - 51 = 24).
- **Printing Frequencies**: After processing all scores, iterate through the frequency array and print the counts for scores 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?

ANS: Given that the queue has a size of 11 and both the front and rear pointers start at q[2], let's track the positions as elements are added: Initially:

Front = 2 Rear = 2

When the first element is added, the rear pointer moves to q[3].

For the second element, the rear pointer moves to q[4].

For the third element, it moves to q[5].

For the fourth element, it moves to q[6].

For the fifth element, it moves to q[7].

For the sixth element, it moves to q[8].

For the seventh element, it moves to q[9].

For the eighth element, it moves to q[10].

For the ninth element, it will wrap around to q[0] since q[10] is the last position.

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

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

ANS:

```
#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);
Node *bstInsert(Node *root, Node *pt);
void insert(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;
}

void rotateLeft(Node *&root, Node *&pt) {
    Node *pt_y = pt->right;
    pt->right = pt_y->left;
    if (pt->right != NULL) pt->right->parent = pt;

    pt_y->parent = pt->parent;
    if (pt->parent == NULL) {
        root = pt_y;
    } else if (pt == pt->parent->left) {
        pt->parent->left = pt_y;
    } else {
        pt->parent->right = pt_y;
    }
    pt_y->left = pt;
    pt->parent = pt_y;
}

void rotateRight(Node *&root, Node *&pt) {
    Node *pt_y = pt->left;
    pt->left = pt_y->right;
    if (pt->left != NULL) pt->left->parent = pt;

    pt_y->parent = pt->parent;
    if (pt->parent == NULL) {
        root = pt_y;
    } else if (pt == pt->parent->left) {
        pt->parent->left = pt_y;
    } else {
        pt->parent->right = pt_y;
    }
    pt_y->right = pt;
    pt->parent = pt_y;
```

```
}

void fixViolation(Node *&root, Node *&pt) {
  Node *pt_parent, *pt_grandparent;
  while ((pt != root) && (pt->color == RED) && (pt->parent->color == RED)) {
    pt_parent = pt->parent;
    pt_grandparent = pt->parent->parent;

    if (pt_parent == pt_grandparent->left) {
      Node *pt_uncle = pt_grandparent->right;
      if (pt_uncle != NULL && pt_uncle->color == RED) {
        pt_grandparent->color = RED;
        pt_parent->color = BLACK;
        pt_uncle->color = BLACK;
        pt = pt_grandparent;
      } else {
        if (pt == pt_parent->right) {
          rotateLeft(root, pt_parent);
          pt = pt_parent;
          pt_parent = pt->parent;
        }
        rotateRight(root, pt_grandparent);
        Color temp = pt_parent->color;
        pt_parent->color = pt_grandparent->color;
        pt_grandparent->color = temp;
        pt = pt_parent;
      }
    } else {
      Node *pt_uncle = pt_grandparent->left;
      if ((pt_uncle != NULL) && (pt_uncle->color == RED)) {
        pt_grandparent->color = RED;
        pt_parent->color = BLACK;
        pt_uncle->color = BLACK;
        pt = pt_grandparent;
      } else {
        if (pt == pt_parent->left) {
          rotateRight(root, pt_parent);
          pt = pt_parent;
          pt_parent = pt->parent;
        }
        rotateLeft(root, pt_grandparent);
```

```
            Color temp = pt_parent->color;
            pt_parent->color = pt_grandparent->color;
            pt_grandparent->color = temp;
            pt = pt_parent;
        }
      }
   }
   root->color = BLACK;
}

Node *bstInsert(Node *root, Node *pt) {
   if (root == NULL) return pt;

   if (pt->data < root->data) {
      root->left = bstInsert(root->left, pt);
      root->left->parent = root;
   } else if (pt->data > root->data) {
      root->right = bstInsert(root->right, pt);
      root->right->parent = root;
   }

   return root;
}

void insert(int data) {
   Node *pt = createNode(data);
   root = bstInsert(root, pt);
   fixViolation(root, pt);
}

void inorder(Node *root) {
   if (root == NULL) return;
   inorder(root->left);
   printf("%d ", root->data);
   inorder(root->right);
}

void printTree(Node *root, int space) {
   if (root == NULL) return;
   space += 10;
   printTree(root->right, space);
```

```
    printf("\n");
    for (int i = 10; i < space; i++) printf(" ");
    printf("%d(%s)\n", root->data, root->color == RED ? "RED" : "BLACK");
    printTree(root->left, space);
}
```

EXPLANATION

- Node Creation: Each new node is initialized as RED.
- Rotations: Functions for left and right rotations.
- Fix Violations: Adjusts tree properties after insertion.
- BST Insert: Standard insertion method for a binary search tree.
- In-order Traversal: Displays elements in sorted order.
- Tree Printing: Visually represents the tree structure.

SUBMITTED BY:

MILOSH SONY
S1 MCA
ROLL NO: 48

SUBMITTED TO:

MS.AKSHARA SASIDARAN