ADVANCED DATA STRUCTURE ASSIGNMENT

Submitted to:

Ms. AKSHARA SASIDHARAN

DEPARTMENT OF COMPUTER APPLICATION

Submitted from:

BESLY JOHN JACOB

29

S1 MCA

4. 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?

```
#include <stdio.h>
int main() {
int scores[500];
int frequency[101] = {0
printf("Enter 500 scores (0-100):\n");
for (int i = 0; i < 500; i++) {
scanf("%d", &scores[i]);
if (scores[i] >= 0 && scores[i] <= 100) {
frequency[scores[i]]++;
} else {
printf("Invalid score %d. Please enter scores between 0 and 100.\n", scores[i]);
i--;
}
}
printf("Frequencies of scores above 50:\n");
for (int i = 51; i <= 100; i++) {
if (frequency[i] > 0) {
printf("Score %d: %d\n", i, frequency[i]);
}}
return 0;}
```

The program initializes an array `scores` to hold 500 student scores and a frequency array `frequency` of size 101 to count occurrences of each score from 0 to 100. It prompts the user to input scores, validating each score to ensure it is within the valid range; if an invalid score is entered, it asks for re-entry without incrementing the count of valid entries. After reading all scores, the program iterates through the frequency array from index 51 to 100, printing the frequencies of scores greater than 50 only if they were recorded, thus providing a clear count of higher scores. This approach efficiently tracks score occurrences while ensuring data validity and clarity in output.

5. 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, initialized with both the front and rear pointers at `q[2]`, the ninth element will be added at position `q[10]`. As elements are added, the rear pointer moves to the next position using the formula \((\text{rear} + 1) \% \text{size}\). Therefore, the sequence of insertions will be at indices `q[2]` through `q[10]` for the first eight elements, with the ninth element occupying `q[10]`, after which the rear pointer would wrap around to `q[0]`.

6. Write a C Program to implement Red Black Tree.

```
#include <stdio.h>
#include <stdlib.h>
enum NodeColor { RED, BLACK };
struct Node {
int data;
enum NodeColor color;
struct Node *left, *right, *parent;
};
```

```
struct Node *root = NULL;
struct Node *createNode(int data) {
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode->data = data;
newNode->color = RED;
newNode->left = newNode->right = newNode->parent = NULL;
return newNode;
}

void leftRotate(struct Node *x) {
struct 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(struct Node *y) {
struct 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 fixRRConflict(struct Node *z) {
while (z->parent != NULL && z->parent->color == RED) {
if (z->parent == z->parent->parent->left) {
struct Node *y = z->parent->parent->right;
if (y == NULL || y->color == BLACK) {
if (z == z->parent->right) {
```

```
z = z->parent;
leftRotate(z);
}
z->parent->color = BLACK;
z->parent->parent->color = RED;
rightRotate(z->parent->parent);
} else {
z->parent->color = BLACK;
y->color = BLACK;
z->parent->parent->color = RED;
z = z->parent->parent;
}
} else {
struct Node *y = z->parent->parent->left;
if (y == NULL || y->color == BLACK) {
if (z == z->parent->left) {
z = z->parent;
rightRotate(z);
}
z->parent->color = BLACK;
z->parent->parent->color = RED;
leftRotate(z->parent->parent);
} else {
z->parent->color = BLACK;
y->color = BLACK;
```

```
z->parent->parent->color = RED;
z = z->parent->parent;
}
}
root->color = BLACK;
}
void insert(int data) {
struct Node *z = createNode(data);
struct Node *y = NULL;
struct 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;
} else if (z->data < y->data) {
y->left = z;
} else {
```

```
y->right = z;
}
fixRRConflict(z);
}
void inorderTraversal(struct Node *root) {
if (root != NULL) {
inorderTraversal(root->left);
printf("%d (%s) ", root->data, root->color == RED ? "RED" : "BLACK");
inorderTraversal(root->right);
}
}
int main() {
insert(10);
insert(20);
insert(30);
insert(15);
insert(25);
insert(5);
printf("Inorder Traversal of the Red-Black Tree:\n");
inorderTraversal(root);
printf("\n");
return 0;
}
```