**ASSIGNMENT**

**Submitted by,**

**Abel John Jacob**

**S1 MCA**

**Roll No.1**

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

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

** Declare an array of size 101 (since the scores range from 0 to 100), initialized to 0.**

* **In C: int frequencies[101] = {0};**

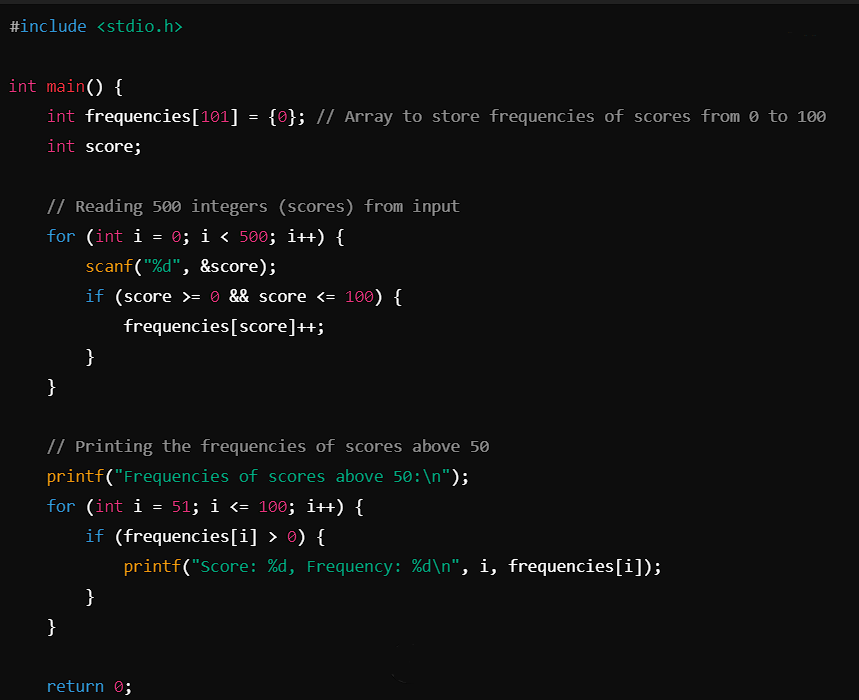
** Iterate over the input scores:**

* **For each score, increment the value in the corresponding index of the array.**

** Print the frequencies of scores above 50:**

* **Loop through the array from index 51 to 100 and print the non-zero frequencies.**

** An example is shown below.**

****

**Steps**

** Array Declaration: The array frequencies[101] is initialized to 0. Each index of this array corresponds to a score, and the value at that index holds the frequency of that score.**

** Input Handling: The program reads 500 scores, and for each valid score (in the range 0 to 100), it increments the respective index in the array.**

** Printing Scores Above 50: After processing the input, the program prints the frequencies of scores from 51 to 100, ignoring scores 50 and below.**

**2.Consider a standard Circular Queue \&#39;q\&#39; implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue areq[0], q[1], q[2].....,q[10]. The front and rear pointers areinitialized to point at q[2] . In which position will the ninthelement be added?**

** Queue size: 11 (q[0] to q[10]).**

** Initial positions: Both front and rear pointers are initialized to q[2].**

** The rear pointer moves forward as elements are added: Every time an element is added, the rear pointer is incremented by one. If it exceeds the last index (q[10]), it wraps around to q[0].**

**Working of the process**

** Initially,**

* **front = 2, rear = 2.**

** First element added:**

* **The rear pointer moves to the next position: rear = 3.**

** Second element added:**

* **rear = 4.**

** Third element added:**

* **rear = 5.**

** Fourth element added:**

* **rear = 6.**

** Fifth element added:**

* **rear = 7.**

** Sixth element added:**

* **rear = 8.**

** Seventh element added:**

* **rear = 9.**

** Eighth element added:**

* **rear = 10.**

** Ninth element added:**

* **rear = 0 (wraps around to the beginning as the size of queue is 11).**

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

**#include <stdio.h>**

**#include <stdlib.h>**

**enum Color { RED, BLACK };**

**struct Node**

**{**

**int data;**

**enum Color color;**

**struct Node \*left, \*right, \*parent;**

**};**

**struct Node\* createNode(int data)**

**{**

**struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));**

**node->data = data;**

**node->color = RED;**

**node->left = node->right = node->parent = NULL;**

**return node;**

**}**

**void leftRotate(struct Node \*\*root, 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 \*\*root, 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 fixViolation(struct Node \*\*root, struct Node \*z)**

**{**

**while (z != \*root && z->parent->color == RED)**

**{**

**struct Node \*parent\_z = z->parent;**

**struct Node \*grand\_parent\_z = z->parent->parent;**

**if (parent\_z == grand\_parent\_z->left)**

**{**

**struct Node \*uncle\_z = grand\_parent\_z->right;**

**if (uncle\_z && uncle\_z->color == RED)**

**{**

**grand\_parent\_z->color = RED;**

**parent\_z->color = BLACK;**

**uncle\_z->color = BLACK;**

**z = grand\_parent\_z;**

**}**

**else**

**{**

**if (z == parent\_z->right)**

**{**

**leftRotate(root, parent\_z);**

**z = parent\_z;**

**parent\_z = z->parent;**

**}**

**rightRotate(root, grand\_parent\_z);**

**parent\_z->color = BLACK;**

**grand\_parent\_z->color = RED;**

**z = parent\_z;**

**}**

**}**

**else**

**{**

**struct Node \*uncle\_z = grand\_parent\_z->left;**

**if (uncle\_z && uncle\_z->color == RED)**

**{**

**grand\_parent\_z->color = RED;**

**parent\_z->color = BLACK;**

**uncle\_z->color = BLACK;**

**z = grand\_parent\_z;**

**}**

**else**

**{**

**if (z == parent\_z->left)**

**{**

**rightRotate(root, parent\_z);**

**z = parent\_z;**

**parent\_z = z->parent;**

**}**

**leftRotate(root, grand\_parent\_z);**

**parent\_z->color = BLACK;**

**grand\_parent\_z->color = RED;**

**z = parent\_z;**

**}**

**}**

**}**

**(\*root)->color = BLACK;**

**}**

**void insert(struct Node \*\*root, 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;**

**fixViolation(root, z);**

**}**

**void inorder(struct Node \*root)**

**{**

**if (root == NULL)**

**return;**

**inorder(root->left);**

**printf("%d ", root->data);**

**inorder(root->right);**

**}**

**int main()**

**{**

**struct Node \*root = NULL;**

**insert(&root, 10);**

**insert(&root, 20);**

**insert(&root, 30);**

**insert(&root, 15);**

**insert(&root, 25);**

**printf("In-order traversal of the created Red-Black Tree:\n");**

**inorder(root);**

**return 0;**

**}**