NAME: **FRANK GODFREY NGOWI**

REG NO: **BCS/18/23/014/TZ**

**QUESTION 1:**

**1.i)**

int search(Node\* root, int key) {

while (root != NULL) {

if (key == root->key) return 1;

root = (key < root->key) ? root->left : root->right;

}

return 0;

}

**1.ii.**

int Height(Node\* root) {

if (root == NULL) return -1;

int leftHeight = Height(root->left);

int rightHeight = Height(root->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

**1.iii.**

void preOrder(Node\* root) {

if (root == NULL) return;

Node\* stack[100];

int top = -1;

stack[++top] = root;

while (top >= 0) {

Node\* current = stack[top--];

printf("%d ", current->key);

if (current->right) stack[++top] = current->right;

if (current->left) stack[++top] = current->left;

}

}

**1.iv.**

void inOrder(Node\* root) {

Node\* stack[100];

int top = -1;

Node\* current = root;

while (current != NULL || top >= 0) {

while (current != NULL) {

stack[++top] = current;

current = current->left;

}

current = stack[top--];

printf("%d ", current->key);

current = current->right;

}

}

**1.v.**

Node\* findMin(Node\* root) {

while (root->left != NULL) root = root->left;

return root;

}

Node\* delete(Node\* root, int key) {

if (root == NULL) return root;

if (key < root->key) root->left = delete(root->left, key);

else if (key > root->key) root->right = delete(root->right, key);

else {

if (root->left == NULL) {

Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

free(root);

return temp;

}

Node\* temp = findMin(root->right);

root->key = temp->key;

root->right = delete(root->right, temp->key);

}

return root;

}

FULL PROGRAM

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int key;

struct Node \*left, \*right;

} Node;

Node\* createNode(int key) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->key = key;

newNode->left = newNode->right = NULL;

return newNode;

}

Node\* insert(Node\* root, int key) {

if (root == NULL) return createNode(key);

if (key < root->key) root->left = insert(root->left, key);

else if (key > root->key) root->right = insert(root->right, key);

return root;

}

int search(Node\* root, int key) {

while (root != NULL) {

if (key == root->key) return 1;

root = (key < root->key) ? root->left : root->right;

}

return 0;

}

int Height(Node\* root) {

if (root == NULL) return -1;

int leftHeight = Height(root->left);

int rightHeight = Height(root->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

void preOrder(Node\* root) {

if (root == NULL) return;

Node\* stack[100];

int top = -1;

stack[++top] = root;

while (top >= 0) {

Node\* current = stack[top--];

printf("%d ", current->key);

if (current->right) stack[++top] = current->right;

if (current->left) stack[++top] = current->left;

}

}

void inOrder(Node\* root) {

Node\* stack1[100], \*stack2[100];

int top1 = -1, top2 = -1;

Node\* current = root;

while (current != NULL || top1 >= 0) {

while (current != NULL) {

stack1[++top1] = current;

current = current->left;

}

current = stack1[top1--];

printf("%d ", current->key);

current = current->right;

}

}

Node\* findMin(Node\* root) {

while (root->left != NULL) root = root->left;

return root;

}

Node\* delete(Node\* root, int key) {

if (root == NULL) return root;

if (key < root->key) root->left = delete(root->left, key);

else if (key > root->key) root->right = delete(root->right, key);

else {

if (root->left == NULL) {

Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

free(root);

return temp;

}

Node\* temp = findMin(root->right);

root->key = temp->key;

root->right = delete(root->right, temp->key);

}

return root;

}

int main() {

Node\* root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 70);

insert(root, 20);

insert(root, 40);

insert(root, 60);

insert(root, 80);

printf("Search 40: %d\n", search(root, 40));

printf("Height: %d\n", Height(root));

printf("PreOrder: ");

preOrder(root);

printf("\n");

printf("InOrder: ");

inOrder(root);

printf("\n");

root = delete(root, 40);

printf("InOrder after deletion: ");

inOrder(root);

printf("\n");

return 0;

}

**QUESTION 2**

**2.**

#include <stdio.h>

#define MAX 100000

#define MAXLENGTH 300

int main() {

int runCounts[MAXENGTH] = {0};

int maxValues[MAXLENGTH] = {0};

for (int n = 1; n <= MAX; n++) {

int length = 0;

int current = n;

while (current != 1) {

if (current % 2 == 0) {

current /= 2;

} else {

current = (3 \* current + 1) / 2;

}

length++;

}

length++;

runCounts[length]++;

if (n > maxValues[length]) {

maxValues[length] = n;

}

}

for (int i = 1; i < MAXLENGTH; i++) {

if (runCounts[i] > 0) {

printf("%d %d %d\n", i, runCounts[i], maxValues[i]);

}

}

return 0;

}

**QUESTION 3**

**3.a.**

#include <stdio.h>

#include <stdlib.h>

typedef struct Request {

int priority;

char device[10];

struct Request\* next;

} Request;

Request\* createRequest(char device[], int priority) {

Request\* newRequest = (Request\*)malloc(sizeof(Request));

newRequest->priority = priority;

strcpy(newRequest->device, device);

newRequest->next = NULL;

return newRequest;

}

void insertRequest(Request\*\* head, char device[], int priority) {

Request\* newRequest = createRequest(device, priority);

if (\*head == NULL || (\*head)->priority > priority) {

newRequest->next = \*head;

\*head = newRequest;

return;

}

Request\* temp = \*head;

while (temp->next != NULL && temp->next->priority <= priority) {

temp = temp->next;

}

newRequest->next = temp->next;

temp->next = newRequest;

}

void processRequests(Request\*\* head) {

while (\*head != NULL) {

printf("Processing %s with priority %d\n", (\*head)->device, (\*head)->priority);

Request\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

}

int main() {

Request\* requestList = NULL;

insertRequest(&requestList, "Keyboard", 1);

insertRequest(&requestList, "Mouse", 2);

insertRequest(&requestList, "Scanner", 3);

insertRequest(&requestList, "Printer", 4);

processRequests(&requestList);

return 0;

}

**3.b)**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* push(Node\* top, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = top;

return newNode;

}

void displayStack(Node\* top) {

while (top != NULL) {

printf("%d ", top->data);

top = top->next;

}

printf("\n");

}

int main() {

Node\* top = NULL;

top = push(top, 10);

top = push(top, 20);

top = push(top, 30);

displayStack(top);

return 0;

}

**3.c)**

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX 100

typedef struct Stack {

char data[MAX];

int top;

} Stack;

void push(Stack\* stack, char value) {

stack->data[++stack->top] = value;

}

char pop(Stack\* stack) {

return stack->data[stack->top--];

}

int precedence(char op) {

if (op == '^') return 3;

if (op == '\*' || op == '/') return 2;

if (op == '+' || op == '-') return 1;

return 0;

}

void infixToPostfix(char\* infix, char\* postfix) {

Stack stack;

stack.top = -1;

int i = 0, j = 0;

while (infix[i] != '\0') {

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

} else if (infix[i] == '(') {

push(&stack, infix[i]);

} else if (infix[i] == ')') {

while (stack.top != -1 && stack.data[stack.top] != '(') {

postfix[j++] = pop(&stack);

}

pop(&stack);

} else {

while (stack.top != -1 && precedence(stack.data[stack.top]) >= precedence(infix[i])) {

postfix[j++] = pop(&stack);

}

push(&stack, infix[i]);

}

i++;

}

while (stack.top != -1) {

postfix[j++] = pop(&stack);

}

postfix[j] = '\0';

}

int main() {

char infix[] = "((a+b)^((c/d)\*e))";

char postfix[MAX];

infixToPostfix(infix, postfix);

printf("Postfix: %s\n", postfix); // Output: "ab+cd/e\*^"

return 0;

}

**3.d)**

#include <stdio.h>

void deposit(double\* balance, double amount) {

\*balance += amount;

printf("Deposited: %.2f, New Balance: %.2f\n", amount, \*balance);

}

void withdraw(double\* balance, double amount) {

if (amount > \*balance) {

printf("Insufficient Balance\n");

} else {

\*balance -= amount;

printf("Withdrawn: %.2f, New Balance: %.2f\n", amount, \*balance);

}

}

int main() {

double balance = 1000.0;

deposit(&balance, 500);

withdraw(&balance, 200);

withdraw(&balance, 2000);

return 0;

}

**3.e)**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* insertAtBeginning(Node\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = head;

return newNode;

}

Node\* insertAtEnd(Node\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

if (head == NULL) return newNode;

Node\* temp = head;

while (temp->next != NULL) temp = temp->next;

temp->next = newNode;

return head;

}

void displayList(Node\* head) {

while (head != NULL) {

printf("%d ", head->data);

head = head->next;

}

printf("\n");

}

int main() {

Node\* head = NULL;

head = insertAtBeginning(head, 10);

head = insertAtBeginning(head, 20);

head = insertAtEnd(head, 30);

head = insertAtEnd(head, 40);

displayList(head);

return 0;

}