Практична робота №8
Завірюха Еліна, МП-21

Побудова АВЛ-дерева:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
    int height;
} Node;

int max(int a, int b) {
    return (a > b) ? a : b;
}

int height(Node* node) {
    if (node == NULL)
        return 0;
    return node->height;
}

int getBalance(Node* node) {
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}

Node* newNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;  // New node is initially at height 1
    return node;
}

Node* rightRotate(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;
```

```
    // Update heights
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    return x;
}

Node* leftRotate(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}

Node* insert(Node* node, int data) {
    // Standard BST insertion
    if (node == NULL)
        return newNode(data);

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    else  // Duplicate data not allowed
        return node;

    // Update height of current node
    node->height = 1 + max(height(node->left), height(node->right));

    // Get the balance factor to check if this node became unbalanced
    int balance = getBalance(node);

    // Left Left Case
    if (balance > 1 && data < node->left->data)
        return rightRotate(node);

    // Right Right Case
    if (balance < -1 && data > node->right->data)
```

```
        return leftRotate(node);

    // Left Right Case
    if (balance > 1 && data > node->left->data) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    // Right Left Case
    if (balance < -1 && data < node->right->data) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    // No rotation needed, return the unchanged node
    return node;
}

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

int main() {
    int keys[] = {12, 8, 7, 9, 14, 11, 10, 50, 70, 90, 25, 20, 30, 80, 25, 48, 77,
16, 10, 20, 62};
    int n = sizeof(keys) / sizeof(keys[0]);

    Node* root = NULL;

    for (int i = 0; i < n; i++) {
        root = insert(root, keys[i]);
    }

    printf("Preorder traversal of the AVL tree: ");
    preOrderTraversal(root);

    return 0;
}
```

2. Програма для створення дерева арифметичного виразу за його префіксним записом:

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <ctype.h>

typedef struct ExpressionNode {
    char data;
    struct ExpressionNode* left;
    struct ExpressionNode* right;
} ExpressionNode;

ExpressionNode* createExpressionTree(char expression[], int* index) {
    char token = expression[*index];
    (*index)++;

    ExpressionNode* newNode =
(ExpressionNode*)malloc(sizeof(ExpressionNode));
    newNode->data = token;
    newNode->left = NULL;
    newNode->right = NULL;

    if (isdigit(token)) {
        return newNode;
    } else {
        newNode->left = createExpressionTree(expression, index);
        newNode->right = createExpressionTree(expression, index);
        return newNode;
    }
}

void inOrderTraversal(ExpressionNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%c ", root->data);
        inOrderTraversal(root->right);
    }
}

int main() {
    char prefixExpression[] = "*+AB-CD";
    int index = 0;

    ExpressionNode* root = createExpressionTree(prefixExpression, &index);

    printf("Inorder traversal of the expression tree: ");
    inOrderTraversal(root);

    return 0;
}
```