Drzewa binarne

Przyjmując następującą definicję wierzchołka drzewa binarnego rozwiąż podane zadania.

```
struct node {
    int data;
    node* left;
    node* right;
};
```

Zad. 1. Napisz funkcję `bool search(node* tree, int x)` sprawdzającą, czy dany element, x, znajduje się w drzewie binarnym.

Zad. 2. Napisz funkcje wyznaczające: liczbę wierzchołków drzewa binarnego, liczbę liści, liczbę prawych potomków, wysokość drzewa.

```
int node_count(node* tree)
int leaves_count(node* tree)
int right_children_count(node* tree)
int tree_height(node* tree)
```

Zad. 3. Napisz funkcję sprawdzającą, czy drzewo binarne jest zbalansowane (różnica wysokości lewego i prawego poddrzewa każdego wierzchołka wynosi zero lub jeden).

```
bool is_balanced(node* tree)
```

Zad. 4. Napisz funkcję sprawdzającą, czy drzewo binarne jest drzewem BST (ang. binary search tree), czyli czy zachodzi następująca własność. Niech x będzie wierzchołkiem drzewa. Jeśli y jest wierzchołkiem znajdującym się w lewym poddrzewie wierzchołka x, to $\text{data}(x) > \text{data}(y)$. Jeśli y jest wierzchołkiem znajdującym się w prawym poddrzewie wierzchołka x, to $\text{data}(x) < \text{data}(y)$.

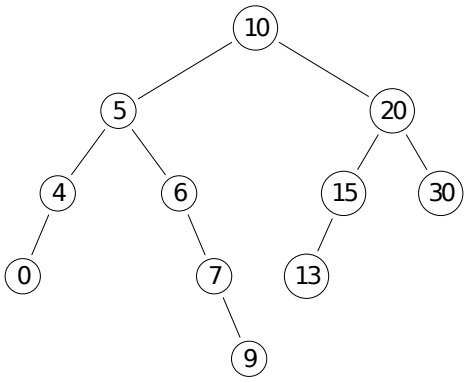
```
bool is_BST(node* tree)
```

Zad. 5. Napisz procedurę usuwającą wszystkie liście podanego drzewa binarnego.

```
void delete_leaves(node* tree)
```

Zad. 6. Zastosuj procedury `preorder()`, `inorder()`, `postorder()` do poniższego drzewa, zakładając, że odwiedzenie wierzchołka p wiąże się z następującym działaniem:

a)
```
if (p->left != 0 && p->data - p->left->data < 2)
    p->left->data += 2;
```



Rysunek 1.1: Drzewo do **Zad. 6.**

b)
```
if (p->left == 0)
    p->right = 0;
```

Zad. 7. Pokaż drzewo dla którego metody *preorder* i *inorder* generują ten sam ciąg.

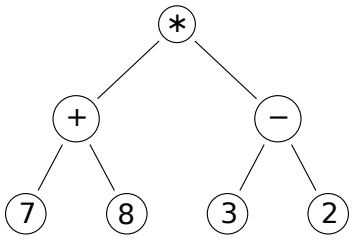
Zad. 8. Napisz funkcję tworzącą „odbicie lustrzane" podanego drzewa binarnego.
```
node* mirror_tree(node* tree)
```

Zad. 9. Napisz procedurę, która drukuje drzewo jak pokazano poniżej dla drzewa z rys. 1.1
```
10
.5
..4
...0
..6
...7
....9
.20
..15
...13
..30
```

```
void print_tree(node* tree, int depth = 0, char prefix = '')
```

Zad. 10. Strukturę drzewiastą zdefiniowano nastepująco:
```
struct node_calc {
    char op;
    int number;
    node_calc* left;
    node_calc* right;
};
```

Drzewa zdefiniowane w ten sposób mogą reprezentować wyrażenia arytmetyczne (wierzchołki *wewnętrzne* to operatory, a *zewnątrzne* to liczby). Na przykład wyrażenie $(7+8)*(3-2)$ można przedstawić jako drzewo z rys. 1.2. Napisz funkcję, która oblicza wartość takich drzew (dla drzewa z rysunku funkcja powinna zwrócić wartość 15).
```
int eval(node_calc* tree)
```



Rysunek 1.2: Drzewo do **Zad. 10.**

Zad. 11. Dana jest tablica $a = [a_1, a_2, \ldots, a_n]$ o długości $n = 2^k - 1$, dla całkowitego $k > 0$. Napisz funkcję tworzącą drzewo jak pokazano w poniższych przykładach:

- $a_1 \rightarrow \{a_1\}$,
- $a_1, a_2, a_3 \rightarrow \{a_2, \{a_1\}, \{a_3\}\}$,
- $a_1, a_2, a_3, a_4, a_5, a_6, a_7 \rightarrow \{a_4, \{a_2, \{a_1\}, \{a_3\}\}, \{a_6, \{a_5\}, \{a_7\}\}\}$.

```
node* create_perfect_tree(int* a, const int k)
```