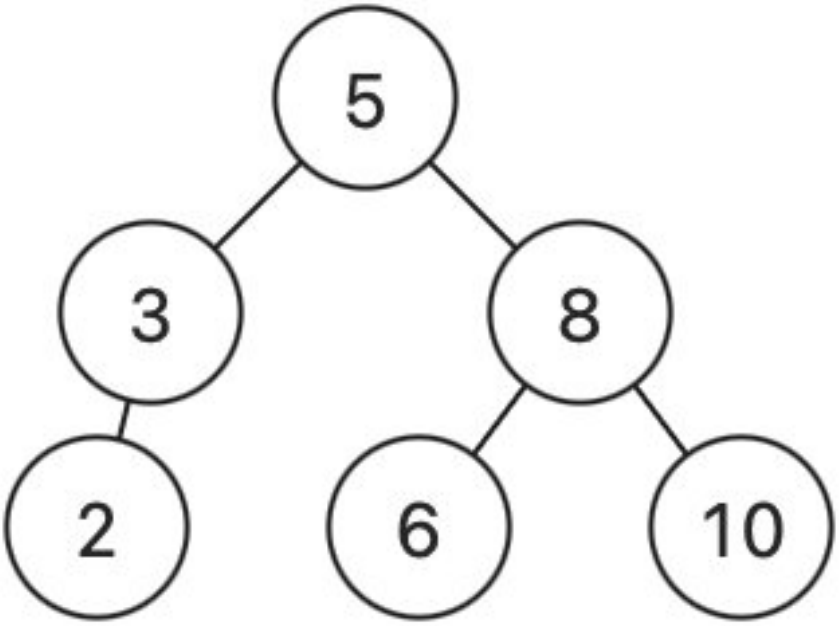
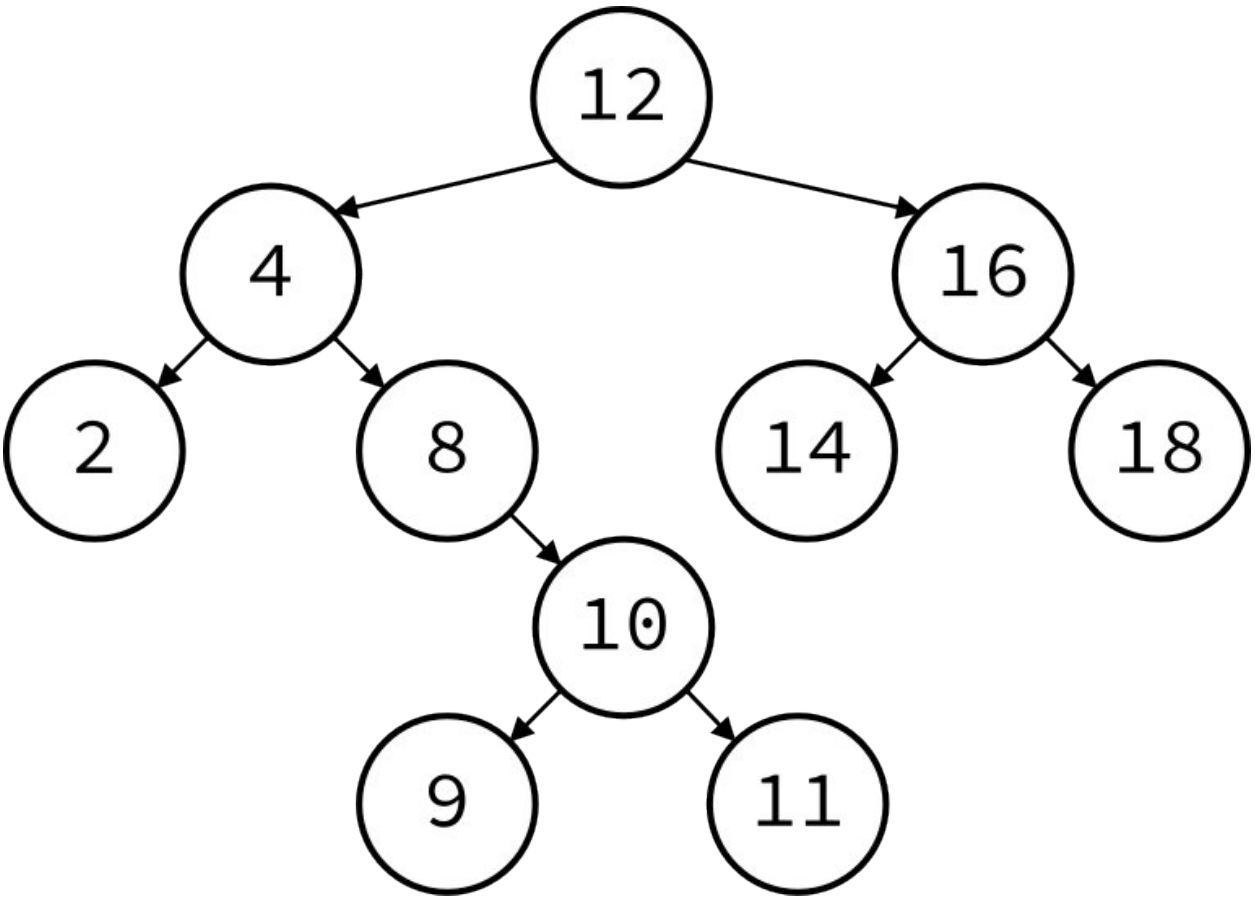
Студент: Пантюхин А.Е., группа: ДТ-460а

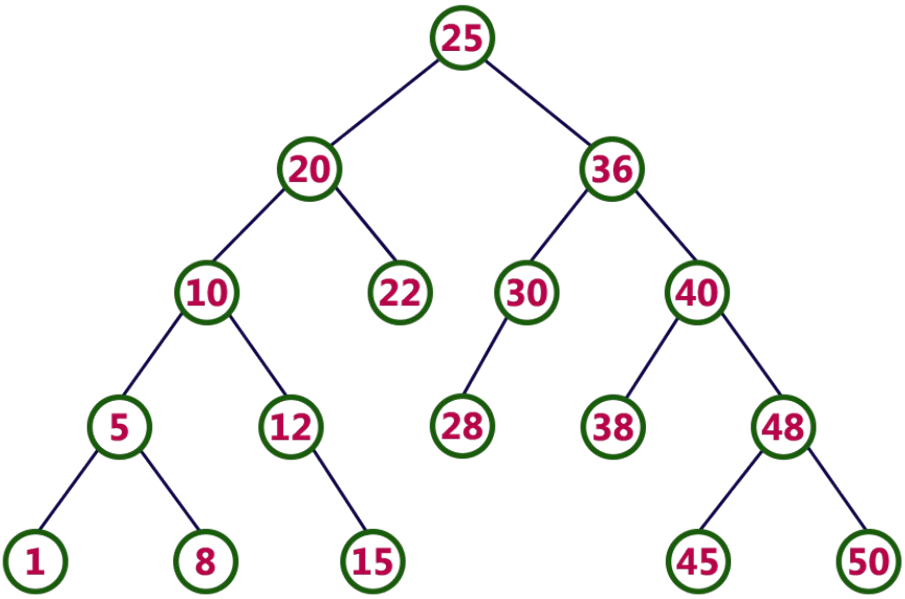
Лабораторная работа №3

Задание: построить дерево Т2 как копию дерева Т1 (вариант 9)

Тестовые данные:







Этапы решения задачи:

Приведённые в презентации к лабораторной работе функции реализованы на C++. Вследствие требований к программе, реализованы функции на C для аналогичных задач.

Добавление элемента:

1) Валидация входных данный (указатель на адрес дерева не нулевой)
2) Если корня дерева нет – выделение памяти под структуру элемента дерева, инициализация значения корня входным значением, ветвей – указателями на NULL.
3) Если корень есть, рекурсивный вызов функции для:
   Если значение в корне больше входного: левой ветви от корня.
   Если значение в корне меньше входного: правой ветви от корня.

Вывод дерева на экран:

1) Валидация входных значений (указатель на дерево не нулевой)
2) Рекурсивный вызов функции для левой ветви дерева.
3) Вывод значения текущего корня.
4) Рекурсивный вызов функции для правой ветви дерева.

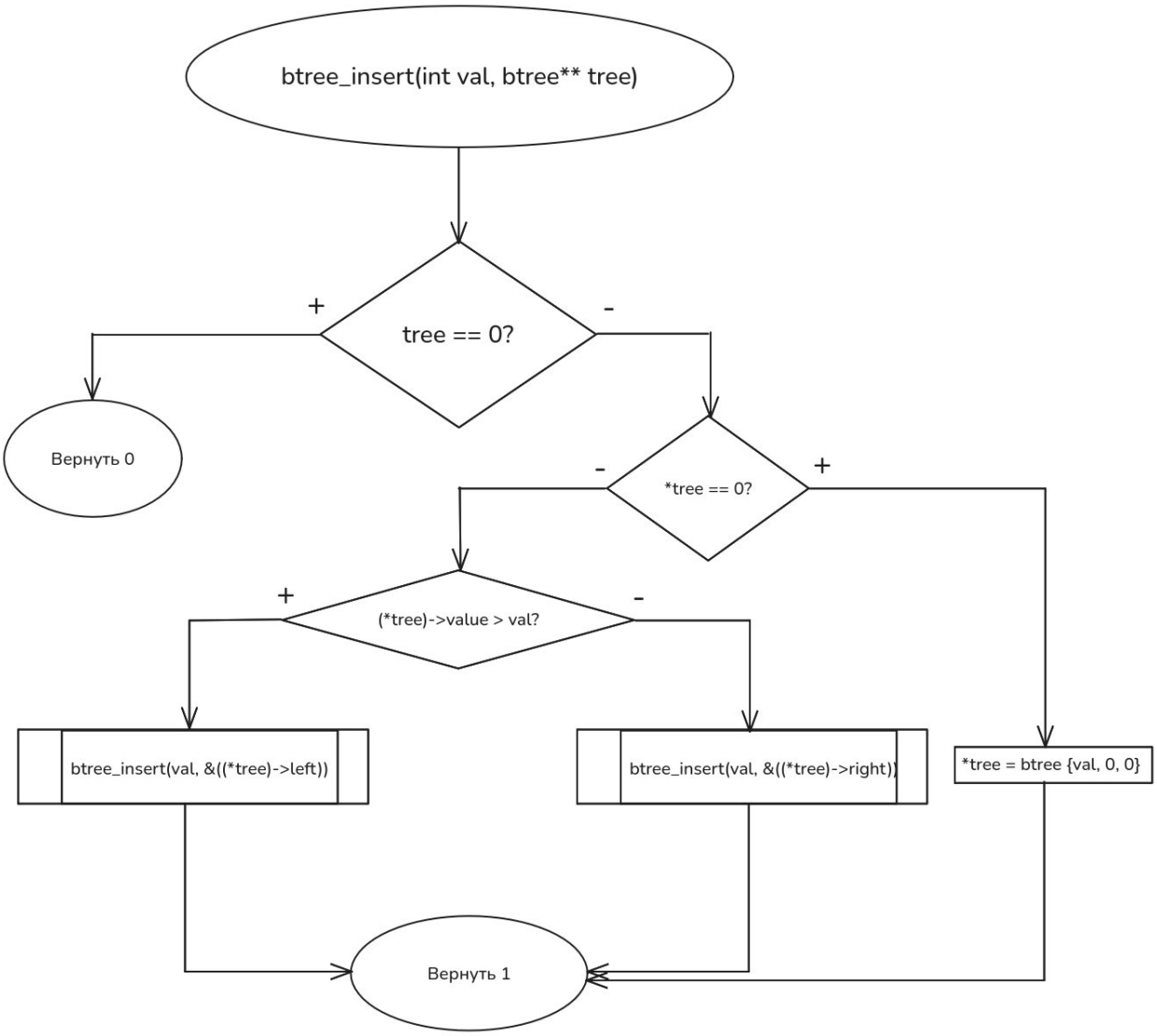
Деаллокация дерева:

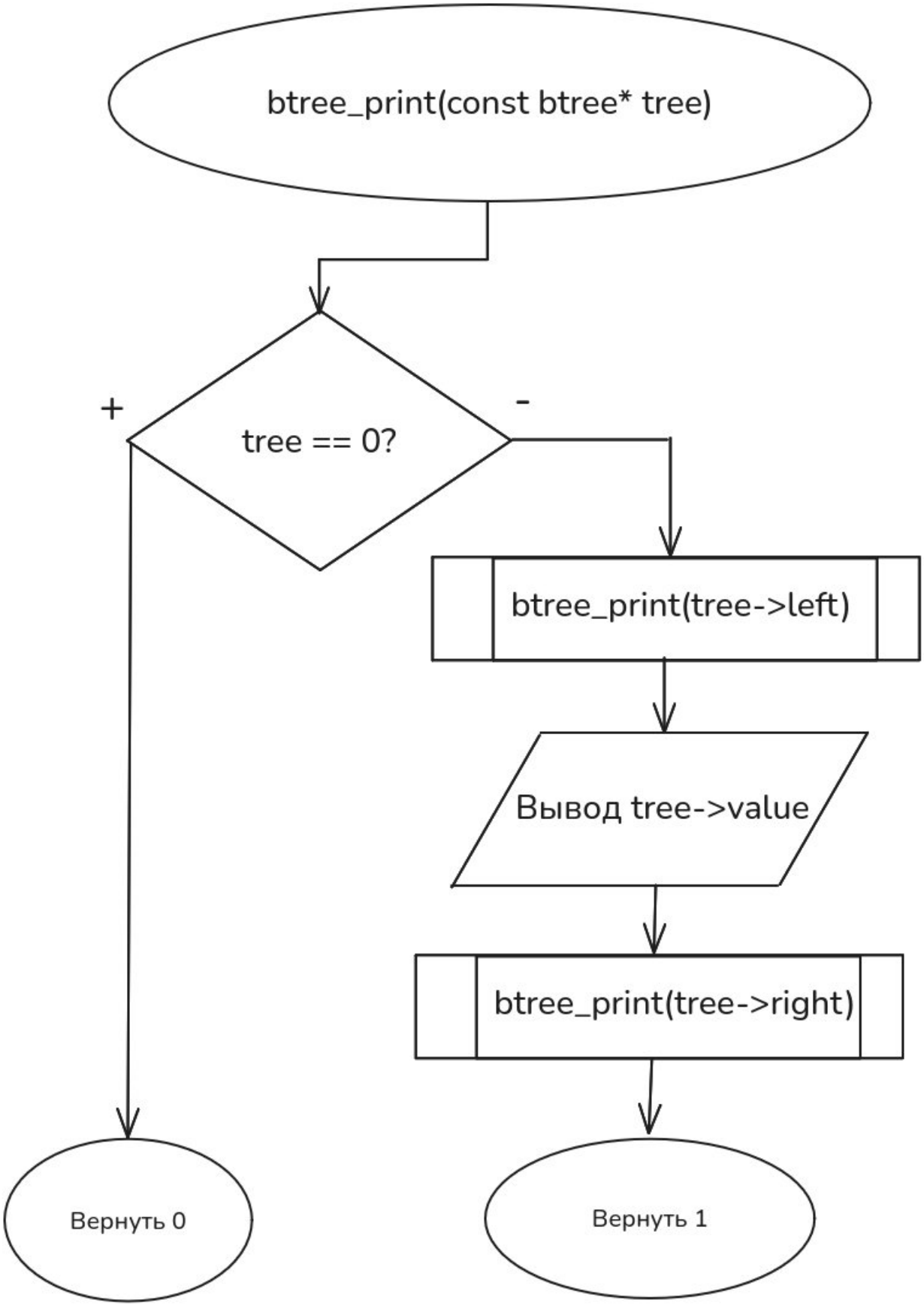
1) Валидация входных значений (указатель на адрес дерева в памяти не нулевой, адрес дерева в памяти не нулевой)
2) Рекурсивный вызов функции для левой и правой ветвей.
3) Деаллокация корня.
4) Установка значения по указателю на адрес дерева в памяти равным нулю.

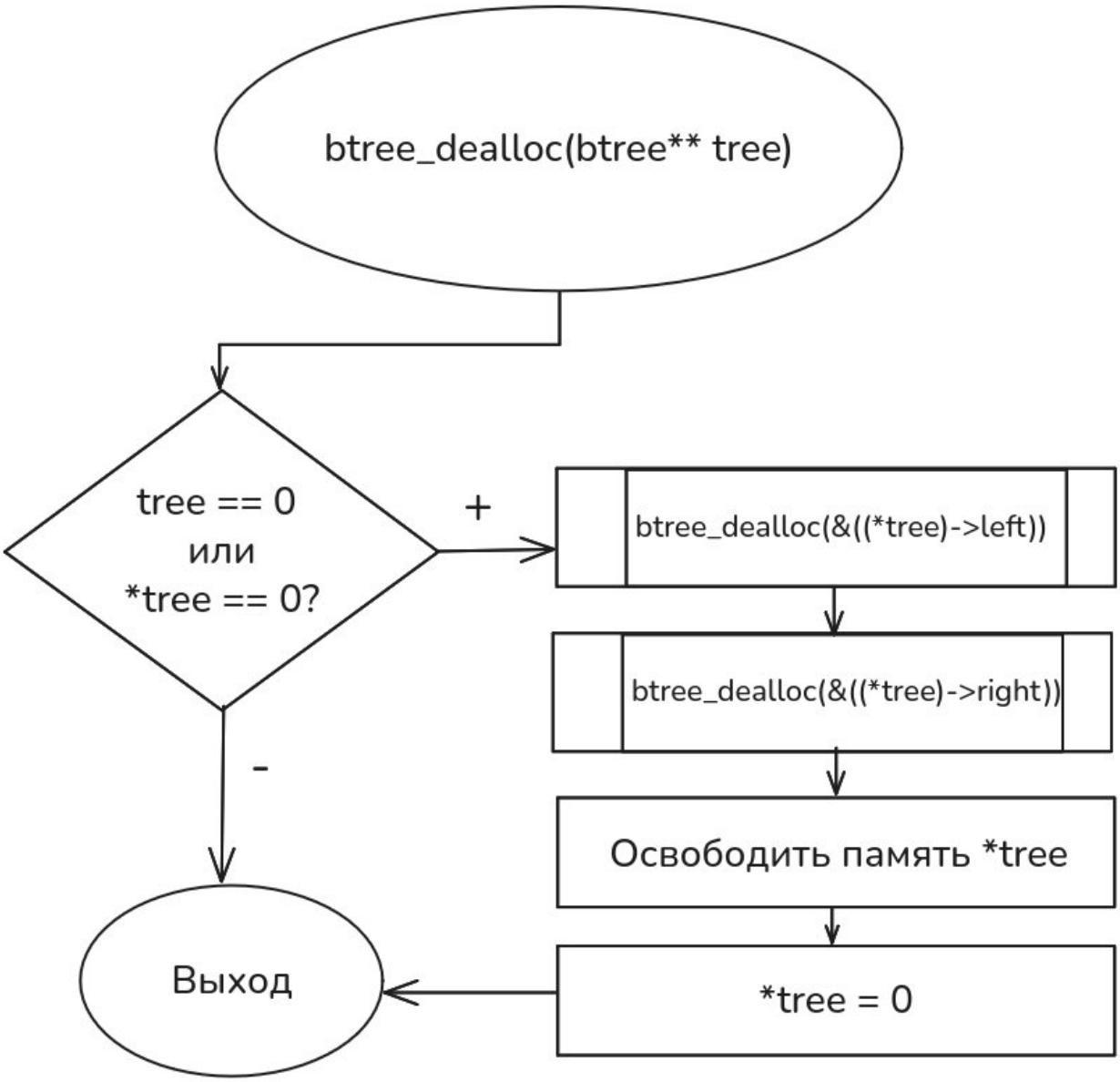
Копирование дерева:

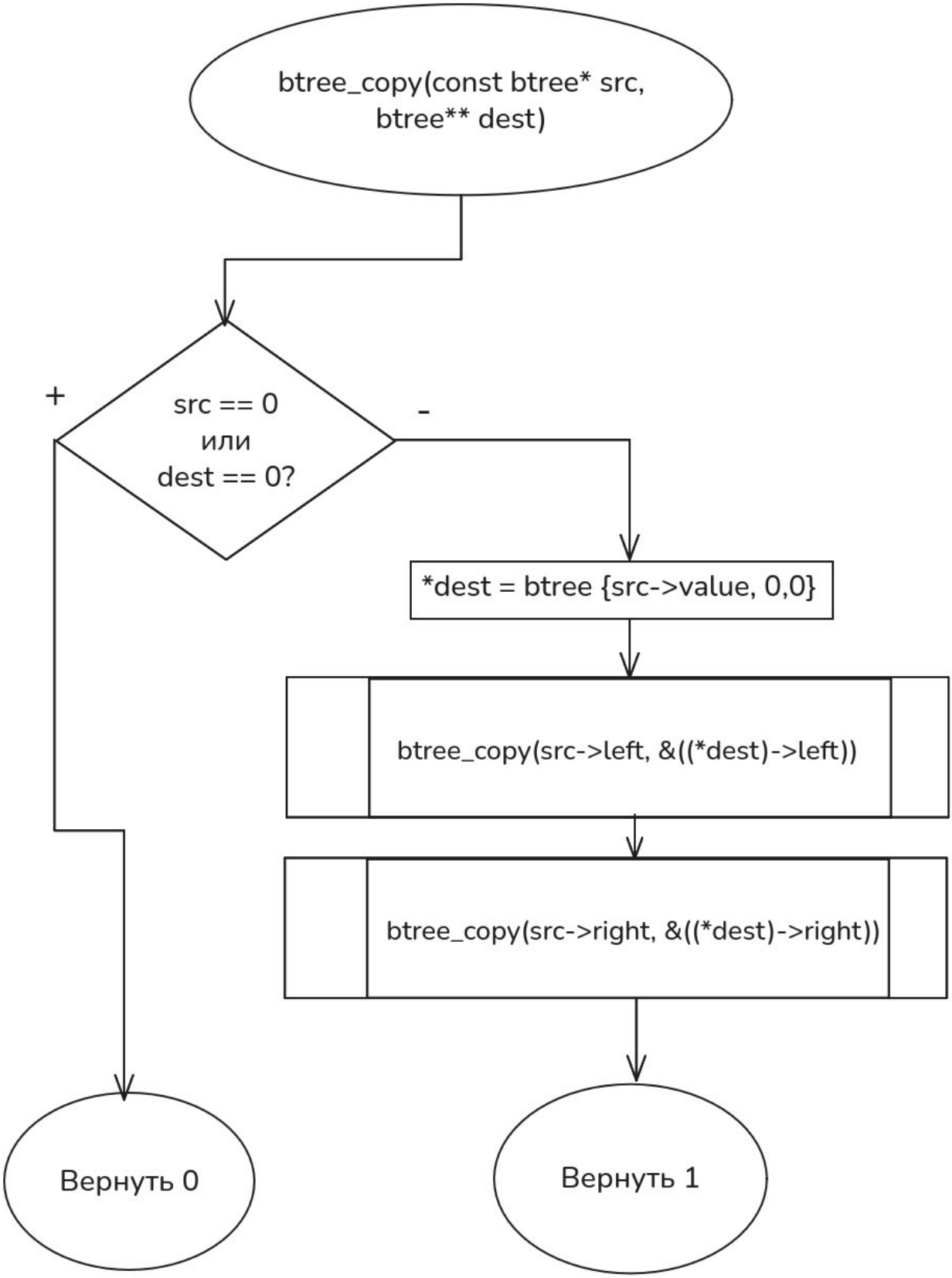
1) Валидация входных данных (указатель на дерево-источник не нулевой, указатель на адрес в памяти дерева-копии не нулевой)
2) Если по второму указателю находится не пустое дерево – деаллокация дерева-копии.
3) Выделение памяти под структуру элемента дерева-копии
4) Инициализация элемента значением корня дерева-источника
5) Рекурсивный вызов функции для левой и правой ветвей дерева-источника, левой и правой ветвей дерева-копии соответственно.

Блок-схемы рекурсивных функций




btree_print(const btree* tree)
tree == 0?
+
-
btree_print(tree->left)
Вывод tree->value
btree_print(tree->right)
Вернуть 0
Вернуть 1


btree_dealloc(btree** tree)
tree == 0
или
*tree == 0?
+
-
btree_dealloc(&((*tree)->left))
btree_dealloc(&((*tree)->right))
Освободить память *tree
*tree = 0
Выход



Текст программы с комментариями:

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

```
typedef struct btree
{
    int value; // информационная часть
    struct btree *left, *right; // служебная часть
}
btree;



int btree_insert(int val, btree** tree)
{
    int rval = 1; //флаг ошибки (1 == успех)
    if (!tree) return !rval; //если на входе просто 0, ничего не делаем
    btree* ptr = *tree; //разименуем указатель для удобства
    if (!ptr)
    { //выделим память под дерево, если указатель на неё 0
        ptr = malloc(sizeof(btree)); //выделение памяти под структуру
        if (!ptr) return !rval; //вернём 0 при ошибке выделения
        *ptr = (btree){val, 0,0}; //инициализация структуры
        *tree = ptr; //вернём обратно
    }
    else
    { //рекурсивный вызов
        if (ptr->value > val) rval = btree_insert(val, &ptr->left);
        else rval = btree_insert(val, &ptr->right);
    }
    return rval;
}


int btree_print(const btree* tree)
{
    if (!tree) return 0;
    //рекурсивный вызов для левой ветви
    btree_print(tree->left);
    printf("%d ", tree->value);
    //рекурсивный вызов для правой ветви
    btree_print(tree->right);
    return 1;
}


void btree_dealloc(btree** tree)
{
    if(!tree) return;
    if(!(*tree)) return;


    btree_dealloc(&(*tree)->left);
```

```
    btree_dealloc(&(*tree)->right);
    free(*tree);
    *tree = 0;
    return;
}


int btree_copy(const btree* src, btree** dest)
{
    int rval = 1;  //флаг ошибки (1 == успех)
    if (!dest || !src) return !rval;
    if (*dest) btree_dealloc(dest);
    *dest = malloc(sizeof(btree)); //выделение памяти под структуру
    (**dest) = (btree){src->value,0,0}; //инициализация структуры


    rval &= btree_copy(src->left, &((*dest)->left));
    rval &= btree_copy(src->right, &((*dest)->right));
    // &= значит, что rval останется 1 только если не было ошибок ниже.
    return rval;
}


int btree_input(btree** tree)
{
    int count, value; //кол-во значений и буфер под значение.
    printf("Укажите количество элементов дерева: ");
    scanf("%d", &count);
    printf("\n");
    if (count < 0)
    {
        printf("Ошибка! Количество элементов не может быть меньше нуля!\n");
        return 0;
    }
    printf("Количество элементов: %d\n", count);


    for(int i = 0; i < count; i++)
    {
        printf("Введите %d-й элемент дерева: ", i+1);
        scanf("%d", &value);
        if (!btree_insert(value, tree))
        {
            printf("Ошибка выделения памяти!\n");
            return 0;
        }
        printf("\n");
    }
    printf("Введённое дерево: \n");
    btree_print(*tree);
    printf("\n");
```

```
    return 1;
}


int main(void) {
    setlocale(LC_ALL, "Russian"); //Если консоль поддерживает, включим русскую
кодировку.
    //setlocale кроме кодировки устанавливает ещё форматы дат и многое другое...
    btree* tree = 0, *tree2 = 0; //указатели на память под деревья.
    printf("Данная программа выделит место в памяти\nи построит дерево, аналогичное
введённому пользователем.\n");
    if (btree_input(&tree))
    {
        btree_copy(tree, &tree2);//копируем структуру и значения

        printf("Исходное дерево: \n");
        btree_print(tree); //вывод деревьев
        printf("\n");
        printf("Копия дерева: \n");
        btree_print(tree2);
        printf("\n");
    }


    btree_dealloc(&tree); //очистка памяти
    btree_dealloc(&tree2);
    return 0;
}
```