Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине «Основы алгоритмизации и программирования»

За 2 семестр

Тема: «Динамические структуры. Деревья»

Выполнили:

Студенты 1 курса

Группы ИИ-21(1)

Кабак Д. Н.

Проверил:

Гирель Т. Н.

Брест 2022

**Цель работы:** приобретение навыков работы с динамической памятью и указателями на C++. Изучение принципов работы с динамическими структурами данных: деревьями.

**Ход работы:**

**Вариант 3**

**Задание 1**

Создайте программой числовое двоичное дерево. Опишите функцию, которая находит наибольший элемент непустого дерева. В программе используйте подпрограммы.

**Код:**

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

bool is\_left;

int tabs = 0;

int MAX;

struct Node {

    int val;

    Node\* left;

    Node\* right;

    Node(int \_val) : val(\_val), left(nullptr), right(nullptr) {}

};

struct tree {

    Node\* root;

    tree() : root(nullptr) {}

    bool is\_empty() {

        return root == nullptr;

    }

    void push\_back(int \_val, Node\* minor\_root) {

        if (is\_empty()) {

            Node\* p = new Node(\_val);

            root = p;

            return;

        }

        Node\* q = minor\_root;

        if (\_val < q->val) {

            if (q->left == nullptr) {

                Node\* p = new Node(\_val);

                q->left = p;

                return;

            }

            q = q->left;

        }

        else {

            if (q->right == nullptr) {

                Node\* p = new Node(\_val);

                q->right = p;

                return;

            }

            q = q->right;

        }

        push\_back(\_val, q);

    }

    void print(Node\* minor\_root) {

        if (!minor\_root) return;

        tabs++;

        print(minor\_root->left);

        for (int i = 0; i < tabs; i++) cout << " ";

        cout << minor\_root->val << endl;

        print(minor\_root->right);

        tabs--;

        return;

    }

    void task1(Node\* minor\_root) {

        if (!minor\_root) {

            if (is\_empty()) return;

            cout << "Max element: "<< MAX;

            return;

        }

        MAX = minor\_root->val;

        task1(minor\_root->right);

        return;

    }

    void free\_tree(Node\* minor\_root) {

        if (!minor\_root) return;

        free\_tree(minor\_root->left);

        free\_tree(minor\_root->right);

        delete minor\_root;

        return;

    }

};

int main() {

    tree tr;

    string data;

    cout << "Enter the values you need to write to the list. To stop recording write 'q':" << endl;

    while (1) {

        cin >> data;

        if (data == "q") {

            break;

        }

        try {

           tr.push\_back(atoi(data.c\_str()), tr.root);

        }

        catch(const std::exception& e) {

            std::cerr << e.what() << '\n';

        }

    }

    cout << "\n\n";

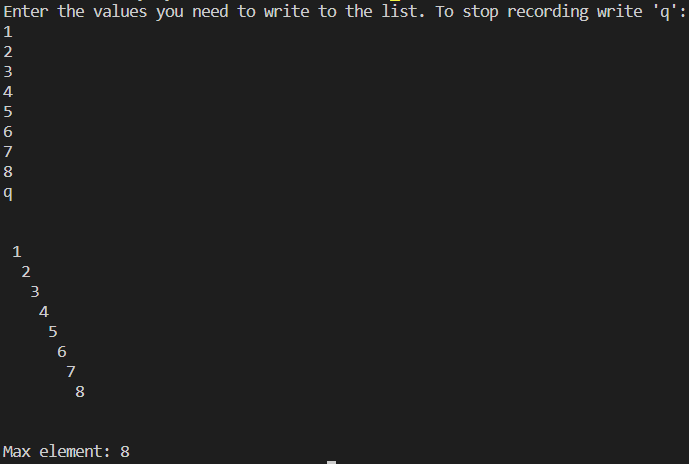
    tr.print(tr.root);

    cout << "\n\n";

    tr.task1(tr.root);

    tr.free\_tree(tr.root);

}



**Задание 2**

По бесскобочной постфиксной записи арифметического выражения с операндами, записанными в виде строк символов, построить дерево выражения и получить полноскобочное инфиксное выражение.

**Код:**

#include <iostream>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

enum optype {power = 3, devide = 2, multiply = 2, minus = 1, plus = 1, null = 0}; // приоритеты операций

struct stack {

    char val[100];

    optype type;

    stack \* next;

} \*head;

void push(char[], optype);

void push(stack[]);

stack \* pop();

void post\_to\_in(char[], char[]);

int main() {

    char post[100], in[100]; // входная и выходная строка

    gets(post);

    post\_to\_in(post, in);

    std::cout << in << std::endl;

}

void push(stack \*t) {

    t->next = head;

    head = t;

}

void push(char str[], optype type) {

    stack \*t = new stack;

    strcpy(t->val, str);

    t->type = type;

    t->next = head;

    head = t;

}

stack \* pop() {

    stack \*t;

    if(head == NULL) return NULL;

    t = head;

    head = t->next;

    return t;

}

void post\_to\_in(char \* input, char \* output) {

    char c, temp[100];

    int p\_temp = 0;

    stack \*h1, \*h2; // переменные для хранения первых двух элементов стека

    optype type;

    head = NULL;

    while(\*input) { // пока есть символы строке

        c = (\*input);

        if(c >= '0' && c <= '9' || c == '.') { // если текущий символ часть числа

            temp[p\_temp++] = c; // то добавляем его во временную строку

            temp[p\_temp] = '\0';

        } else if(c == ' ') {

            if(p\_temp != 0) {

                push(temp, null); // добавляем число в стек

                p\_temp = 0;

            }

            temp[0] = '\0'; // опустошаем временную строку

        } else if(c == '+' || c == '-'|| c == '\*' || c == '/' || c == '^') { // если читаем знак операции

            h1 = pop(); // выталкиваем первый элемент

            h2 = pop(); // выталкиваем второй элемент

            // находим приоритет операции

            if(c == '+') type = plus;

            else if(c == '-') type = minus;

            else if(c == '\*') type = multiply;

            else if(c == '/') type = devide;

            else if(c == '^') type = power;

            if(h2->type != null && h2->type < type) { // если приоритет для 1-го элемента меньше

                temp[0] = '('; temp[1] = '\0'; // берем выражение в скобки

                h2->val[strlen(h2->val) + 2] = '\0';

                h2->val[strlen(h2->val) + 1] = c; // приписываем знак операции

                h2->val[strlen(h2->val)] = ')';

            } else {

                h2->val[strlen(h2->val) + 1] = '\0';

                h2->val[strlen(h2->val)] = c;

            }

            strcat(temp, h2->val);

            if(h1->type != null && h1->type < type) {  // если приоритет для 2-го элемента меньше

                strcat(temp, "(");

                h1->val[strlen(h1->val) + 1] = '\0';

                h1->val[strlen(h1->val)] = ')'; // берем выражение в скобки

            }

            strcat(temp, h1->val);

            strcpy(h2->val, temp);

            delete h1;

            h2->type = type; // устанавливаем новый приоритет операции

            push(h2); // добавляем новый элемент в стек

        }

        input++;

    }

    strcpy(output, (pop())->val); // копируем выражение из вершины стека в строку результата

}



**Вывод:** приобрел навыки работы с динамической памятью и указателями на C. Изучил принципы работы с динамическими структурами данных: деревьями.