Министерство образования Республики Беларусь

Учреждения образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №7

По дисциплине ОАИП за 2 семестр

«динамические структуры. деревья»

Выполнил:

Студент группы ИИ-21

1-го курса

Корпач Д.Р.

Проверил :

Гирель Т.Н.

Брест 2022

**Задание 1:** Создайте программой символьное двоичное дерево. Опишите логическую функцию, проверяющую, есть ли в непустом дереве хотя бы два одинаковых символа. В программе используйте подпрограммы.

#include<iostream>

using namespace std;

int tabs=0,cnt, val;

struct Branch{

  char Data;

  Branch \*LeftBranch;

  Branch \*RightBranch;

};

void Add(int aData, Branch \*&aBranch){//создание бинарного дерева

    if (!aBranch){                   //и запонение его веток

        aBranch = new Branch;

        aBranch->Data = aData;

        aBranch->LeftBranch = 0;

        aBranch->RightBranch = 0;

        return;

    }

    if (aBranch->Data>aData){

        Add(aData, aBranch->RightBranch);

    }

    else {

        Add(aData, aBranch->LeftBranch);

    }

}

void print(Branch \*aBranch){//вывод бинарного дерева

    if (!aBranch) return;

    tabs++;

    print(aBranch->LeftBranch);

    for (int i = 0; i < tabs; i++){

        cout<<" | ";

    }

    cout << aBranch->Data <<endl;

    print(aBranch->RightBranch);

    tabs--;

    return;

}

bool findDouble(Branch \*aBranch){

    if(aBranch){

        findDouble(aBranch->LeftBranch);

        if(cnt && (aBranch->Data == val)){

            ++cnt;

            if((cnt > 1) && (0 == aBranch->RightBranch)){

                for(int i = 0; i < cnt; ++i){

                    return true;

                }

                cnt = 1;

            }

        }

        else{

            if(cnt > 1){

                for(int i = 0; i < cnt; ++i){

                    return true;

                }

            }

            val = aBranch->Data;

            cnt = 1;

        }

        findDouble(aBranch->RightBranch);

    }

}

void FreeTree(Branch \*aBranch){//высвобождение памяти от бин дерева

    if (!aBranch) return;

    FreeTree(aBranch->LeftBranch);

    FreeTree(aBranch->RightBranch);

    delete aBranch;

    return;

}

int main(){

    Branch \*Root = 0;

    int n, \*even\_number;

    char cifr;

    bool double\_element=false;

    cout<<"input count cifr: ";cin >>n;

    for (int i = 0;i< n; i++){

        cin>>cifr;

        Add(cifr, Root);

    }

    print(Root);

    double\_element=findDouble(Root);

    if(double\_element==true){

        cout<<"bin tree have double element"<<endl;

    }

    else{cout<<"bin tree not to have double element"<<endl;}

    FreeTree(Root);

    return 0;

}

**Задание 2**

По постфиксной записи арифметического выражения с операндами-строками получить инфиксную запись выражения , содержащую только необходимые скобки.

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

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;

// <--Функции работы со стеком-->

void push(char[], optype);

void push(stack \*);

stack \* pop();

// <--Функция выполняющая наш алгоритм-->

void fromRPN(char \*, char \*); // (RPN) Reverse polish notation

int main() {

    char infix[100], postfix[100]; // входная и выходная строка

    gets(infix);

    fromRPN(infix, postfix);

    printf("%s\n", postfix);

    return 0;

}

void push(stack \*t) {

    t->next = head;

    head = t;

}

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

    stack \*t;

    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 fromRPN(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** 