|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(национальный исследовательский университет)*** |

ФАКУЛЬТЕТ ФУНДАМЕНТАЛЬНЫЕ НАУКИ

КАФЕДРА вычичлительная математика И МАТЕМАТИЧЕСКАЯ ФИЗИКА (ФН-11)

**Отчет**

**по домашнему заданию часть 2.3**

Использование бинарных деревьев для вычисления выражений

**Дисциплина:** Информатика

Студент гр. ФН11 – 22Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Писарев Г. А.**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Никитин А. В.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019

**Задание**

Использовать бинарные деревья для подсчета выражений.

Выражение в соответствии с вариантом (x/2+7)\*(7\*(y-4)+3)

**Теоретические основы деревьев синтаксического разбора**

**Дерево синтаксического разбора** – результат синтаксического анализа, т.е. результат процесса сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой.

**Синтаксический анализатор**— это программа или часть программы, выполняющая синтаксический анализ.

В ходе синтаксического анализа исходный текст преобразуется в структуру данных, обычно — в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки.

Разбор арифметического выражения

Как транслятор обрабатывает и выполняет арифметические и логические выражения, которые он встречает в программе?

Один из вариантов - представить это выражение в виде двоичного дерева.

Что получается при прохождении таких двоичных деревьев? Прямое прохождение дерева (корень - левое - правое) дает **/ + a b + - c d 1**

то есть знак операции (корень) предшествует своим операндам.

Такая форма записи арифметических выражений называется **префиксной**.

Симметричное (левое –корень –правое) дает запись вида **a + b / c - d + 1**

или **инфиксную форму**, которая совпадает с обычной записью, но без учета скобок. Поскольку скобок нет, правильный порядок операций невозможно восстановить по инфиксной записи.

В трансляторах широко используется **постфиксная запись** выражений, которая получается в результате обратного обхода (левое - правое - корень).

В ней знак операции стоит после обоих операндов: **a b + c d - 1 /**

Порядок выполнения такого выражения однозначно определяется следующим алгоритмом, который использует стек:

* 1. Взять очередной элемент.
  2. Если это операнд (не знак операции), то записать его в стек.
  3. Если это знак операции, то:
     + выбрать из стека второй операнд
     + выбрать из стека первый операнд
     + выполнить операцию с этими данными и результат записать в стек

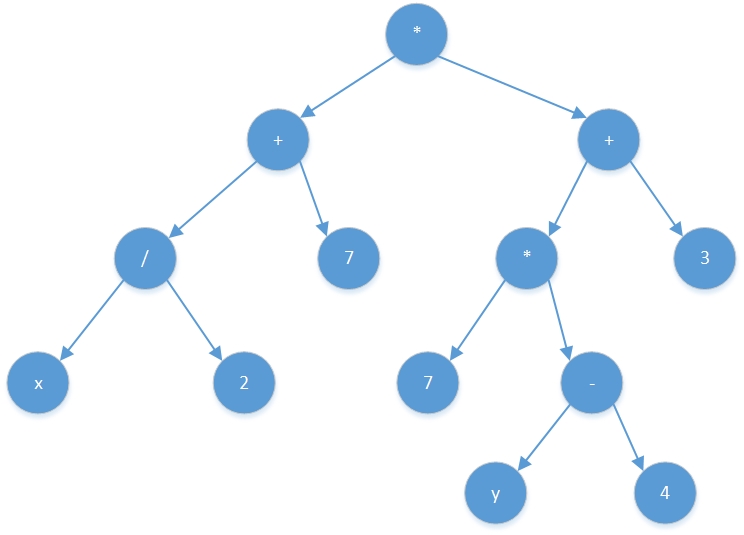
Алгоритм построения дерева

Пусть задано арифметическое выражение. Надо построить для него дерево синтаксического разбора и различные формы записи.

Рассмотрим самый простой вариант, введя следующие упрощения:

1. В выражении могут присутствовать целые числа, имена переменных и знаки операций + - \* /. Разбор арифметического выражения
2. Запрещается использование вызовов функций, скобок, унарных знаков плюс и минус (например, запрещено выражение -a+5, вместо него надо писать 0-a+5).
3. Предполагается, что выражение записано верно, то есть не делается проверки на правильность.
4. Выражение уже разбито на отдельные элементы трех типов (числа, переменные, знаки операций), записанные в массив символьных строк Term размером N.

**Графическое изображение дерева**

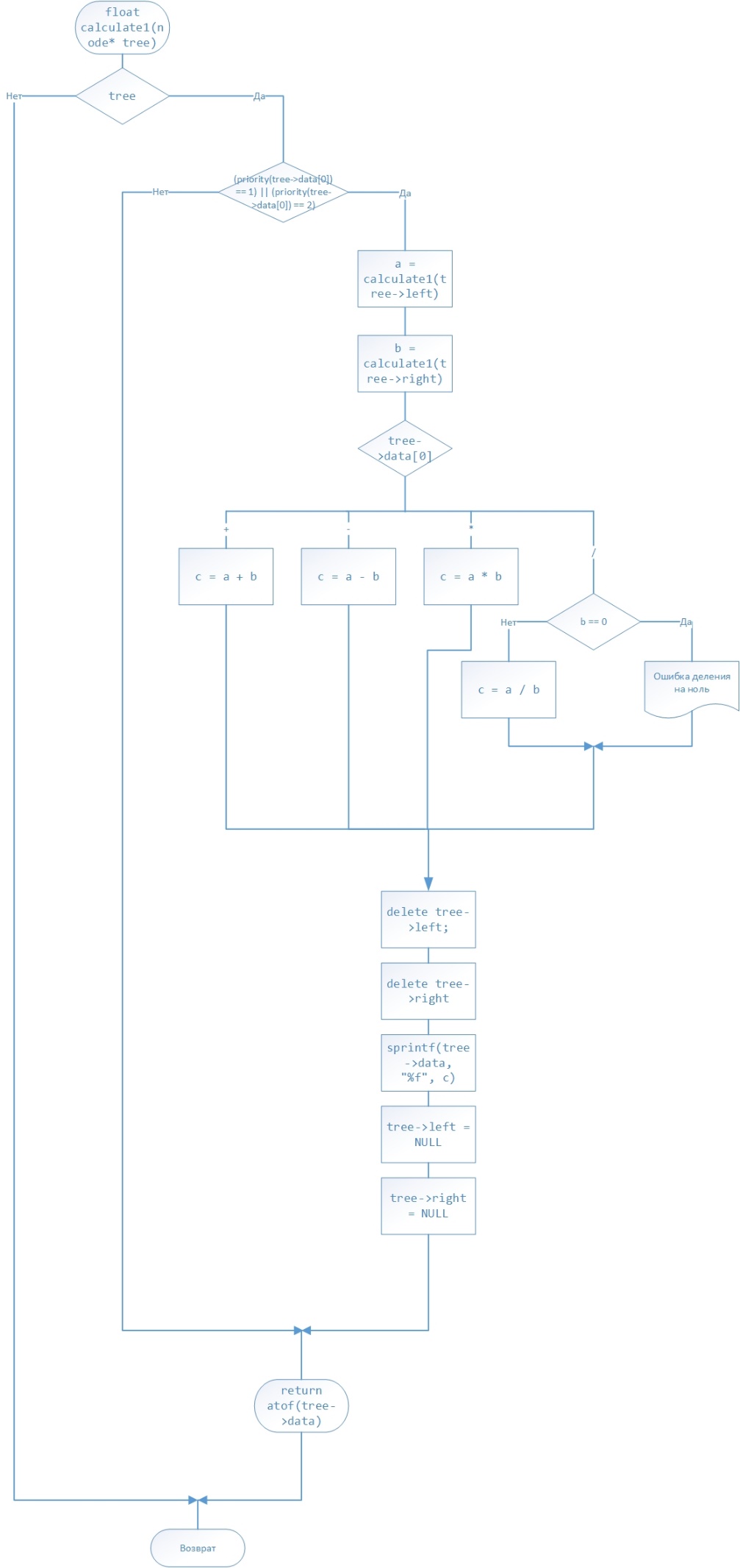
****

**Количество узлов – 13**

**Количество листьев – 7**

**Глубина дерева – 4**

**Блок-схема алгоритма подпрограммы colculate1**

****

**Код программы**

#include <iostream>

#include <ctime>

using namespace std;

typedef struct node {

char data[10];

node \*left;

node \*right;

};

int Priority(char c)

{

switch (c)

{

case '+':

return 1;

break;

case '-':

return 1;

break;

case '/':

return 2;

break;

case '\*':

return 2;

break;

default:

return 100;

break;

}

}

int priority(char c)

{

switch (c)

{

case '+':

return 1;

break;

case '-':

return 1;

break;

case '/':

return 2;

break;

case '\*':

return 2;

break;

default:

return 0;

break;

}

}

void addnull(char\* C, int i)

{

C[strlen(C) + 1] = '\0';

for (int j = strlen(C); j >= i; j--)

{

if (j == i)

{

C[j] = '0';

}

else

{

C[j] = C[j - 1];

}

}

}

void changeconst(char\* C, int first, int k)

{

bool b = true;

int last = first;

char c[100];

int i = 0;

char x[100];

x[0] = rand() % 10 + 48;

x[1] = rand() % 10 + 48;

x[2] = '\0';

if (k == 2) // если пользователь выбрал ручной ввод

{

while ((isdigit(C[last]) == 0) && (priority(C[last]) == 0) && (C[last] != ')') && (last < strlen(C))) // поиск константы

{

c[i] = C[last];

i++;

last++;

}

c[i] = '\0';

cout << "Значение переменной " << c << " > ";

cin >> x;

}

while (b)

{

b = false;

last = first;

i = 0;

while ((isdigit(C[last]) == 0) && (priority(C[last]) == 0) && (C[last] != ')') && (last < strlen(C))) // поиск конца константы

{

c[i] = C[last];

i++;

last++;

}

c[i] = '\0';

i = 0;

int I = 0;

int l = last, f = first;

if (l == strlen(C)) // если константа в конце

{

for (f; f < l + strlen(x) - 1; f++)

{

C[f] = x[i];

i++;

C[l + strlen(x) - 1] = '\0';

}

}

else

{

if (strlen(c) < strlen(x)) // если символов в константе меньше чем цифр в числе

{

for (int j = strlen(C); j >= l; j--)

{

C[j + strlen(x) - 1] = C[j];

}

for (int j = f; j < f + strlen(x); j++)

{

C[j] = x[I];

I++;

}

}

else

{

for (l, f; l <= strlen(C); l++, f++) //если константа в середине и символов в константе больше чем цифр в числе

{

if (i < strlen(x))

{

C[f] = x[i];

i++;

l--;

}

else

{

C[f] = C[l];

}

}

}

}

for (int j = 0; j < strlen(C); j++) // поиск повтарения константы

{

i = 0;

while (C[j] == c[i])

{

i++;

j++;

if ((i == strlen(c)) )

{

b = true;

first = j - i;

break;

}

}

if (b)

break;

}

}

}

node \*MakeTree(char term[][10], int first, int last)

{

char c;

int MinPrt, i, k = 0, ptr;

int nest = 0;

int f;

node\* TreeElement = new node;

if (first == last)

{

TreeElement->left = NULL;

TreeElement->right = NULL;

strcpy(TreeElement->data, term[first]);

return TreeElement;

}

MinPrt = 100;

for (i = first; i <= last; i++)

{

c = term[i][0];

if (c == '(')

{

nest++;

continue;

}

if (c == ')')

{

nest--;

continue;

}

if (nest > 0)

continue;

ptr = Priority(c);

if (ptr <= MinPrt)

{

MinPrt = ptr;

k = i;

}

}

if ((MinPrt == 100) && (term[first][0] == '(') && (term[last][0] == ')'))

return MakeTree(term, first + 1, last - 1);

TreeElement = new node;

strcpy(TreeElement->data, term[k]);

TreeElement->left = MakeTree(term, first, k - 1);

TreeElement->right = MakeTree(term, k + 1, last);

return TreeElement;

}

void print(node\* MyTree)

{

if (MyTree != NULL)

{

print(MyTree->left);

cout << MyTree->data << " ";

print(MyTree->right);

}

}

bool IsNumber(node\* tree)

{

int i = 0;

if (!tree)

return false;

while (tree->data[i])

if (!strchr("0123456789", tree->data[i++]))

return 0;

return true;

}

float calculate1(node\* tree)

{

float a, b, c = 0;

if (tree)

{

if ((priority(tree->data[0]) == 1) || (priority(tree->data[0]) == 2))

{

a = calculate1(tree->left);

b = calculate1(tree->right);

switch (tree->data[0])

{

case '+':

c = a + b;

break;

case '-':

c = a - b;

break;

case '\*':

c = a \* b;

break;

case '/':

if (b == 0)

{

cout << "Ошибка деления на ноль" << endl;

exit(0);

}

else

c = a / b;

break;

}

delete tree->left;

delete tree->right;

sprintf(tree->data, "%f", c);

tree->left = NULL;

tree->right = NULL;

}

return atof(tree->data);

}

}

void unop(char \*C)

{

int j;

char ch[] = "1234567890qwertyuiopasdfghjklzxcvbnm";

for (int i = 0; i < strlen(C); i++)

{

if ((C[i] == '(') && (C[i + 1] == '-'))

{

j = i + 2;

C[i] = '0';

while (strchr(ch, C[j]) != 0)

j++;

i = j;

while (j < strlen(C))

{

C[j] = C[j + 1];

j++;

}

}

}

}

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(NULL));

char C[100];

int k = 0;

cout << "Ввдеите выражение" << endl;

char s[] = "+-/\*";

bool b = true;

int L, R;

char ch[] = "qwertyuiopasdfghjklzxcvbnm+-/\*1234567890()";

gets\_s(C, 100);

int fl = 0;

while (b) //доступность символа

{

if (fl > 0)

gets\_s(C, 100);

b = false;

for (int i = 0; i < strlen(C); i++)

{

if (strchr(ch, C[i]) == 0)

{

system("cls");

cout << "Недопустимая операция! Повторите ввод" << endl;

b = true;

fl++;

break;

}

}

}

b = true;

fl = 0;

while (b) // контроль скобок

{

if (fl > 0)

gets\_s(C, 100);

b = false;

L = 0;

R = 0;

for (int i = 0; i < strlen(C); i++)

{

if (C[i] == '(') // контроль скобок (

L++;

if (C[i] == ')') // контроль скобок )

R++;

}

if (L != R)

{

cout << "Не все скобки закрыты!\nПовторный ввод" << endl;

b = true;

fl++;

}

}

b = true;

fl = 0;

unop(C);

while (b)

{

if (fl > 0)

gets\_s(C, 100);

for (int i = 0; i < strlen(C); i++)

{

b = false;

if (priority(C[i]) != 0)

{

if (i == 0) //унрные операции -1 -> 0-1

{

if (priority(C[i]) == 2)

{

system("cls");

cout << "Ошибка ввода! Недопустимая операция (одинокая операция).\nПовторный ввод" << endl;

b = true;

fl++;

break;

}

else

{

addnull(C, i);

}

}

if (priority(C[i - 1]) != 0) // "--", "++"

{

system("cls");

cout << "Ошибка ввода! Недопустимая операция(--).\nПовторный ввод" << endl;

b = true;

fl++;

break;

}

if (i == (strlen(C) - 1)) // операция в конце строки

{

system("cls");

cout << "Ошибка ввода! Недопустимая операция(операция в коце строки).\nПовторный ввод" << endl;

b = true;

fl++;

break;

}

}

if ((isdigit(C[i]) == 0) && (priority(C[i]) == 0) && (C[i] != ' ') && (C[i] != '(') && (C[i] != ')'))

{

if (k == 0)

{

cout << "Для рандомного заполнения констант введите 1\nДля ручного заполнения констант введите 2\n";

cin >> k;

}

changeconst(C, i, k);

}

}

}

int j = 0;

char term[60][10];

int p = 0, q = 0;

for (int i = 0; i < strlen(C); i++)

{

j = i;

q = 0;

while (isdigit(C[j]) != 0)

{

term[p][q] = C[j];

q++;

j++;

}

if (q == 0)

{

term[p][q] = C[j];

term[p][1] = '\0';

}

else

{

term[p][q] = '\0';

if ((priority(C[j]) == 1) || (priority(C[j]) == 2) || (C[j] == ')') || (C[j] == '('))

{

p++;

term[p][0] = C[j];

term[p][1] = '\0';

}

}

p++;

i = j;

}

node\* MyTree = MakeTree(term, 0, p - 1);

cout << "Дерево\t";

print(MyTree);

cout << endl;

cout << C << " = ";

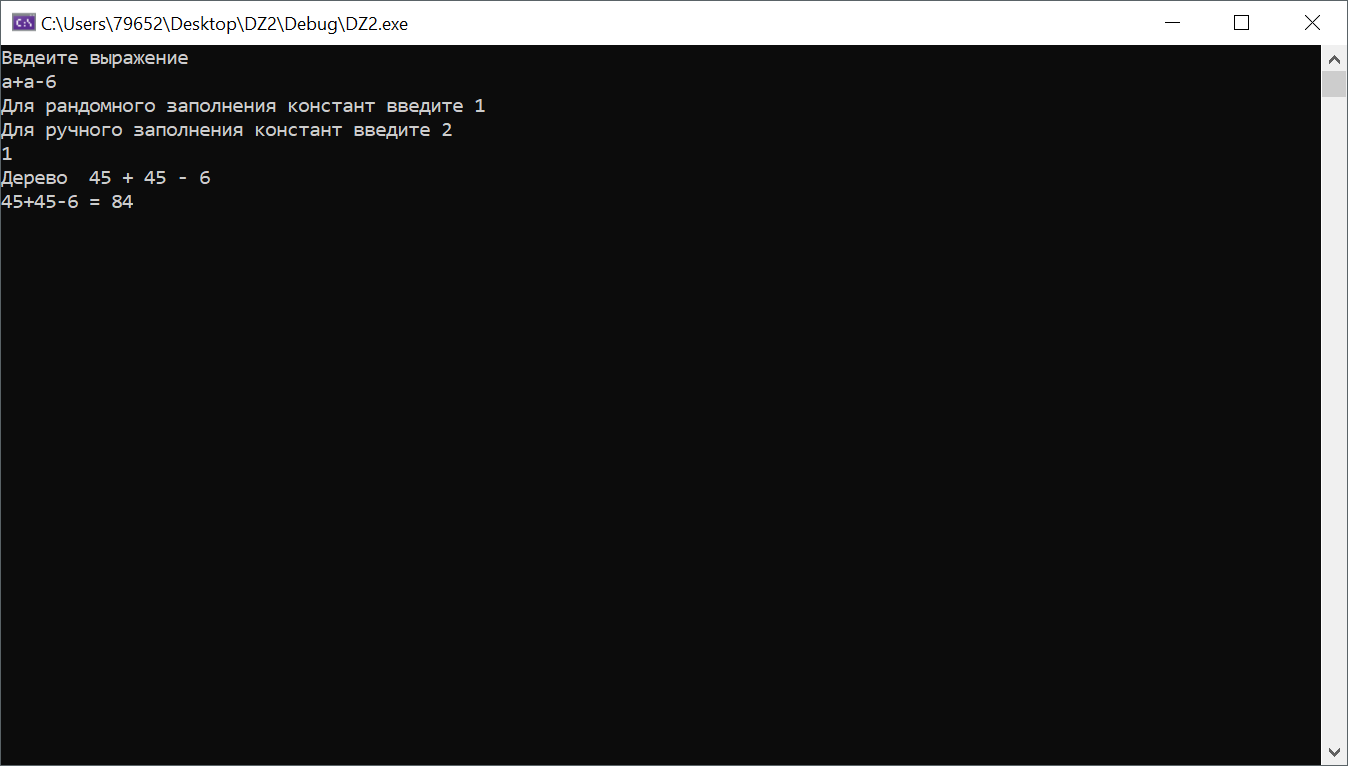
cout << calculate1(MyTree);

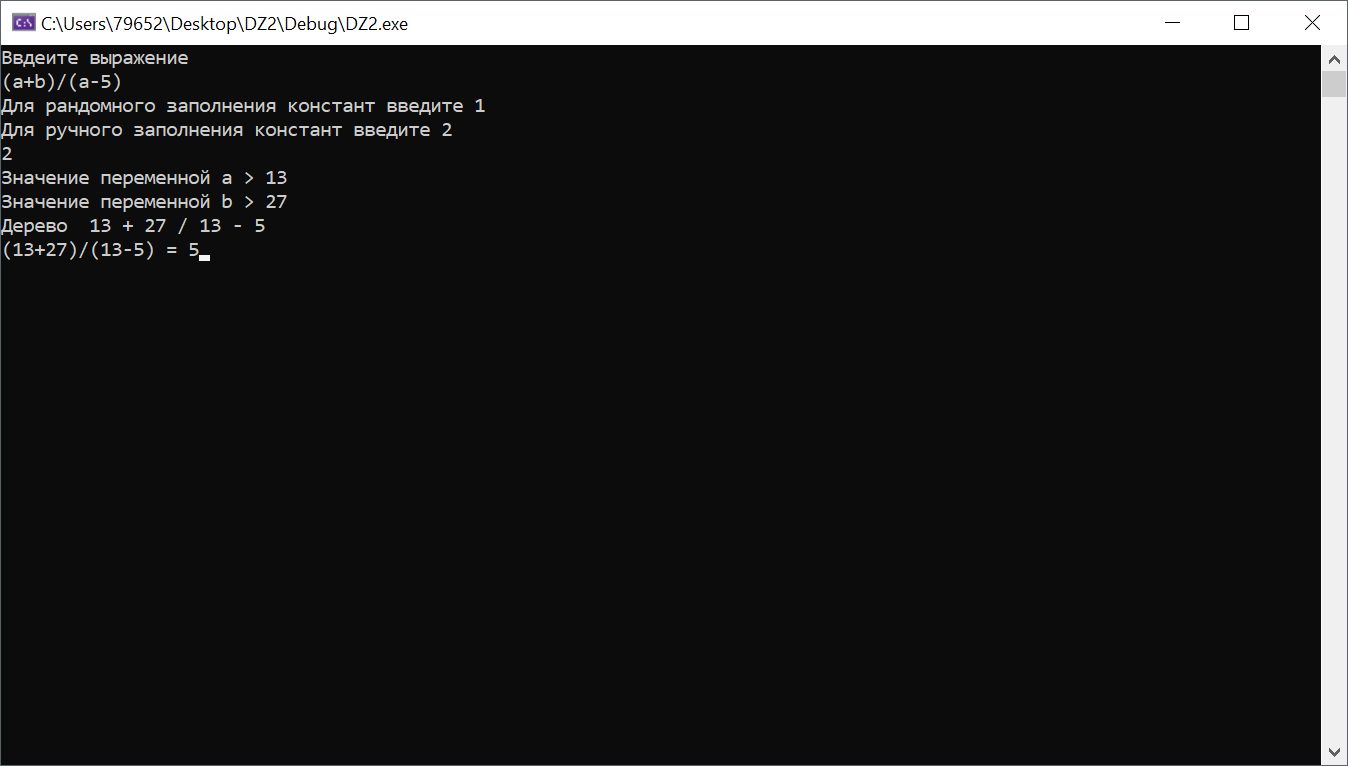
system("pause>>void");

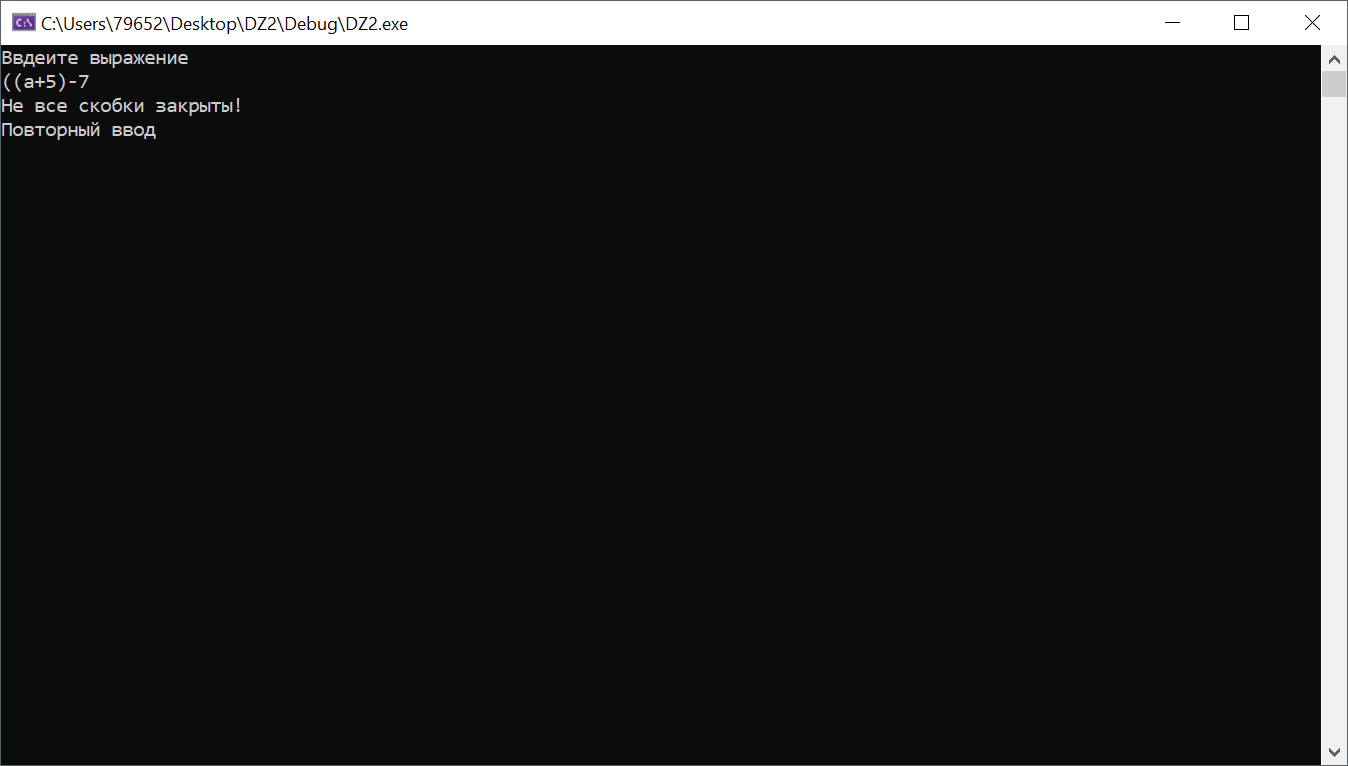
}

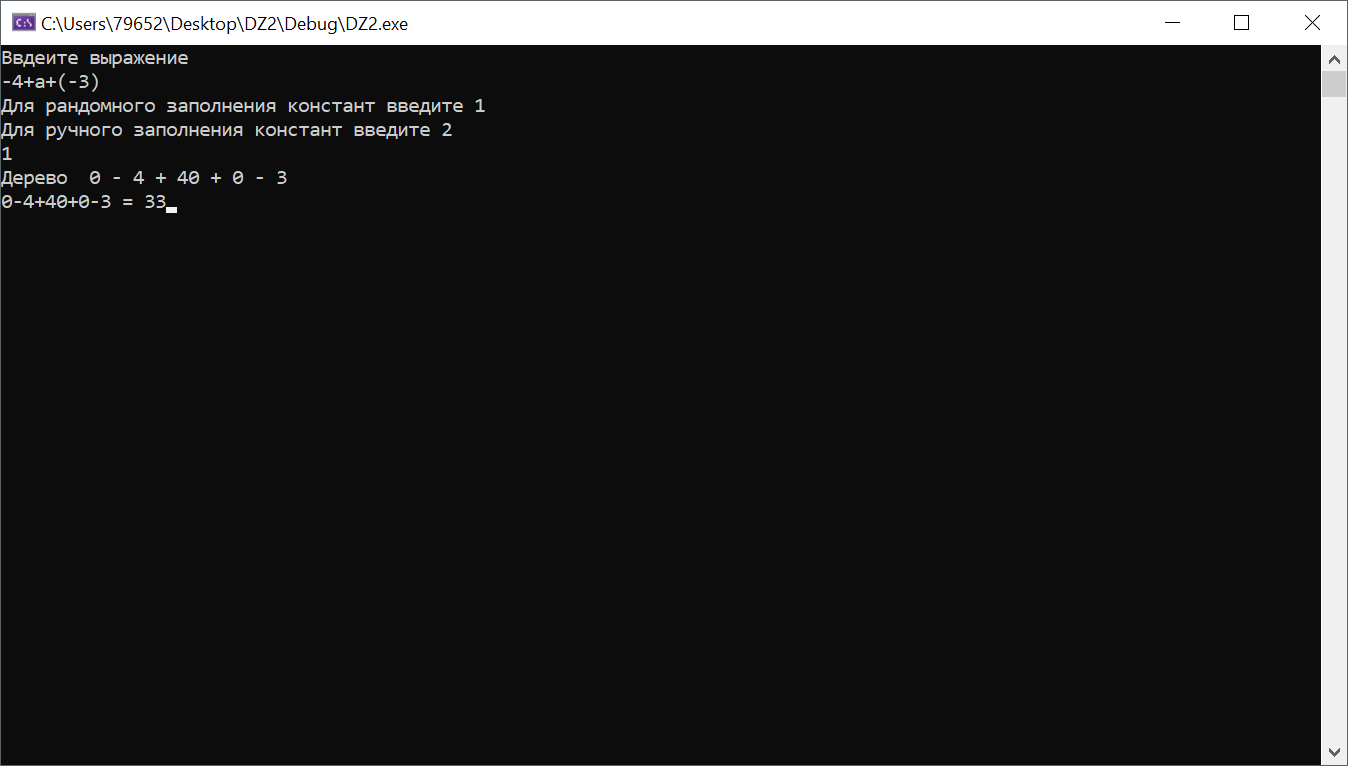
**Тесты программы**











**Программы работает правильно**