Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №8  
по дисциплине  
«Основы программирования»  
на тему:  
« Обратная польская запись»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Закрепление навыков работы с линейными динамическими структурами, навыков разработки комбинированных алгоритмов и реализации их на языке программирования С++.

**Постановка задачи**

Реализовать вычисление алгебраических выражений, используя Обратную польскую запись.

**Теоретическая часть**

Рассмотрим классическую задачу преобразования формы записи арифметического выражения в обратную польскую запись (ОПЗ) с использованием стека.

ОПЗ – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Такая форма записи не содержит скобок. В литературе можно встретить другие названия: обратная польская нотация, бесскобочная символика Лукашевича (названа именем польского математика Яна Лукашевича, предложившего такой подход в 1920 году).

Алгоритм преобразования выражения в ОПЗ с использованием стека.

Входным значением алгоритма является строка, содержащая арифметическое выражение в привычной нам форме записи. На выходе получаем строку, содержащую запись этого выражения в обратной польской нотации, при этом выражение не содержит скобок и знаки операций расположены за операндами.

Для реализации алгоритма используется стек для переменных типа char. В стек будут помещены знаки операций и открывающая скобка.

На первом шаге рассматриваем поочерёдно каждый символ входной строки: если этот символ – число (или переменная), то просто помещаем его в выходную строку; если символ – знак операции (+, –, ×, /), то проверяем приоритет данной операции.

Операции умножения и деления имеют наивысший приоритет (положим его равным трём). Операции сложения и вычитания имеют меньший приоритет – два. Наименьший приоритет – единицу – имеет открывающая скобка.

Далее необходимо проверить стек. Если стек все ещё пуст или находящиеся в нём символы (знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.

Если символ, находящийся на вершине стека, имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к предыдущему пункту.

Если текущий символ – открывающая скобка, то помещаем её в стек. Если текущий символ – закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т. е. символ с приоритетом, равным единице), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке ещё остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере выражения a + (b – c) \* d. Согласно изложенному выше алгоритму, поочерёдно перебираем все символы входной строки.

Получаем выражение в обратной польской нотации: a b c – d \* +

Алгоритм вычисления выражения, записанного в ОПЗ.

Для реализации этого алгоритма используют стек чисел (или переменных, если они встречаются в исходном выражении). Алгоритм достаточно прост. В качестве входной строки теперь рассматриваем выражение, записанное в ОПЗ.

1. Если очередной символ входной строки – число, то помещаем его в стек.

2. Если очередной символ – знак операции, то извлекаем из стека два верхних числа, используем их в качестве операндов для этой операции, затем помещаем результат обратно в стек.

Когда вся входная строка будет разобрана, в стеке останется одно число, которое и будет результатом данного выражения.

**Практическая часть**

Листинг кода приложения:

#include <iostream>

using namespace std;

struct Node

{

char data;

Node\* next;

};

//добавление

void AddElem(Node\*& elem, char value) {

Node\* temp = new Node;

temp->data = value;

temp->next = NULL;

if (elem == NULL) {

elem = temp;

}

else {

temp->next = elem;

elem = temp;

}

}

//извлечение

char Izvl(Node\*& \_elem) {

Node\* temp = \_elem;

char a = temp->data;

\_elem = temp->next;

delete(temp);

return a;

}

bool Check(char str[]) {

int count = 0, i = 0;

while (str[i] != '\0')

{

//проверка на скобки

if (str[i] == '(') count++;

if (str[i] == ')' && count == 0) return false;

if (str[i] == ')') count--;

if ((str[i] == '(') && (str[i + 1] == ')')) return false;

//проверка на знаки

if ((str[i] == '+') && ((str[i + 1] == '+') || (str[i + 1] == '-') || (str[i + 1] == '\*') || (str[i + 1] == '/') || (str[i + 1] == '\\0'))) return false;

if ((str[i] == '-') && ((str[i + 1] == '+') || (str[i + 1] == '-') || (str[i + 1] == '\*') || (str[i + 1] == '/') || (str[i + 1] == '\\0'))) return false;

if ((str[i] == '\*') && ((str[i + 1] == '+') || (str[i + 1] == '-') || (str[i + 1] == '\*') || (str[i + 1] == '/') || (str[i + 1] == '\\0'))) return false;

if ((str[i] == '/') && ((str[i + 1] == '+') || (str[i + 1] == '-') || (str[i + 1] == '\*') || (str[i + 1] == '/') || (str[i + 1] == '\\0'))) return false;

i++;

}

if (count == 0) return true;

return false;

}

//получение приоритетов

int GetPriority(char ch) {

switch (ch)

{

case '\*':

case '/':

return 3;

case '+':

case '-':

return 2;

case '(':

return 1;

default:

return 0;

}

}

//выполнение операций

char ApplyOperations(char operation, int a, int b) {

switch (operation) {

case '+': {

return a + b;

}

case '-': {

return a - b;

}

case '\*': {

return a \* b;

}

case '/': {

if (b == 0) {

throw "\nОшибка. Деление на ноль!";

}

return a / b;

}

default:

cout << "Некорректная операция" << endl;

return -1;

}

}

void ToOPZ(char str[], char OPZstr[])

{

if (Check(str)) {

int i = 0, j = 0;

Node\* stack = NULL;

while (str[i] != '\0') {

if (str[i] == '(') AddElem(stack, str[i]);

else if (str[i] == ')') {

while (stack != nullptr && stack->data != '(') {

OPZstr[j] = Izvl(stack);

cout << OPZstr[j];

j++;

}

if (stack != nullptr && stack->data == '(') {

Izvl(stack);

}

}

else if (str[i] == '+' || str[i] == '-' || str[i] == '\*' || str[i] == '/') {

while (stack != nullptr && GetPriority(stack->data) >= GetPriority(str[i])) {

OPZstr[j] = Izvl(stack);

cout << OPZstr[j];

j++;

}

AddElem(stack, str[i]);

}

else {

OPZstr[j] = str[i];

cout << OPZstr[j];

j++;

}

i++;

}

while (stack != nullptr) {

OPZstr[j] = Izvl(stack);

cout << OPZstr[j];

j++;

}

OPZstr[j] = '\0';

}

else cout << "ERROR";

}

int OPZ(char expression[]) {

Node\* stack = NULL;

int i = 0;

while (expression[i] != '\0') {

if (expression[i] >= '0' && expression[i] <= '9') {

AddElem(stack, expression[i]);

}

else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '\*' || expression[i] == '/') {

int b = Izvl(stack) - '0';

int a = Izvl(stack) - '0';

int result;

try {

result = ApplyOperations(expression[i], a, b);

}

catch (const char\* e) {

cout << e << endl;

return -1;

}

AddElem(stack, result + '0');

}

i++;

}

return Izvl(stack) - '0';

}

void CheckDel(char \*str) {

int res = OPZ(str);

if (res == -1) {

cout << "Посчитать не получилось, из-за ошибки." << endl;

}

else {

cout << "\nОтвет: " << OPZ(str) << endl;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №8 (Обратная польская запись)" << endl;

char mass[100], str[100];

cout << "Введите выражение: ";

cin.getline(mass, 100);

cout << "До преобразования:\n" << mass << endl;

cout << "После преобразования:\n";

ToOPZ(mass, str);

CheckDel(str);

system("pause");

return 0;

}

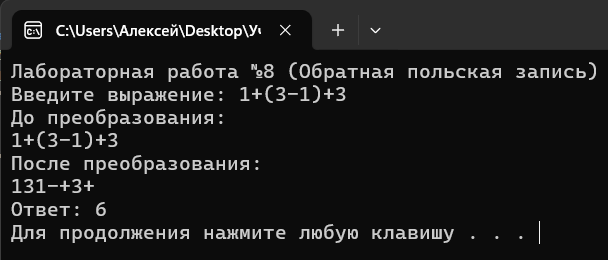
****

Рисунок 1 – Результат успешного выполнения кода

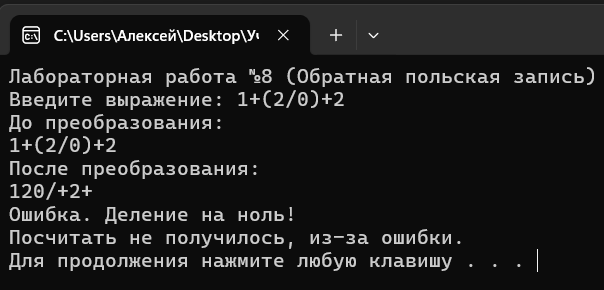
****

Рисунок 2 – Результат выполнения кода с ошибкой деления на 0

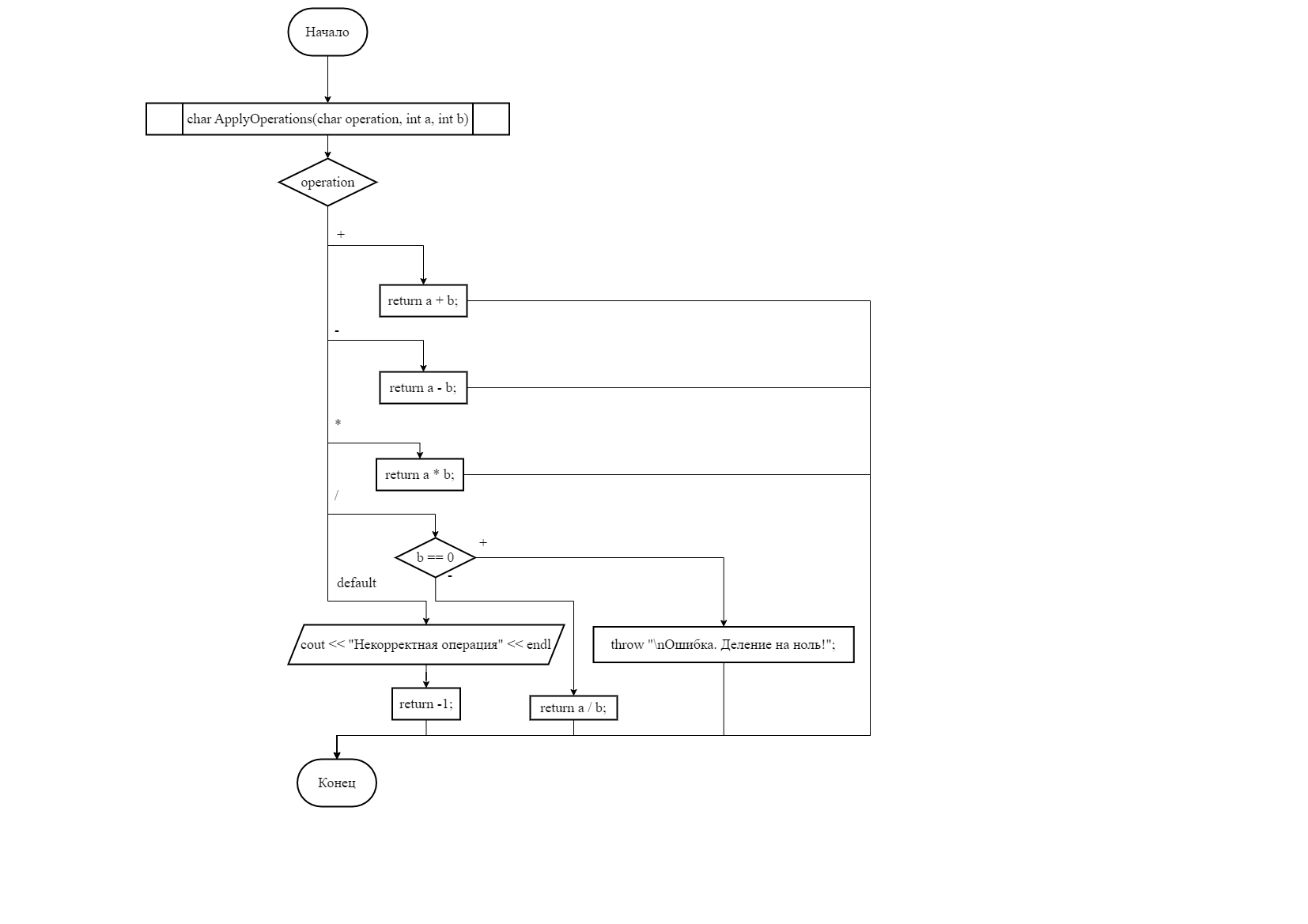


Рисунок 3 – Блок-схема функции ApplyOperations

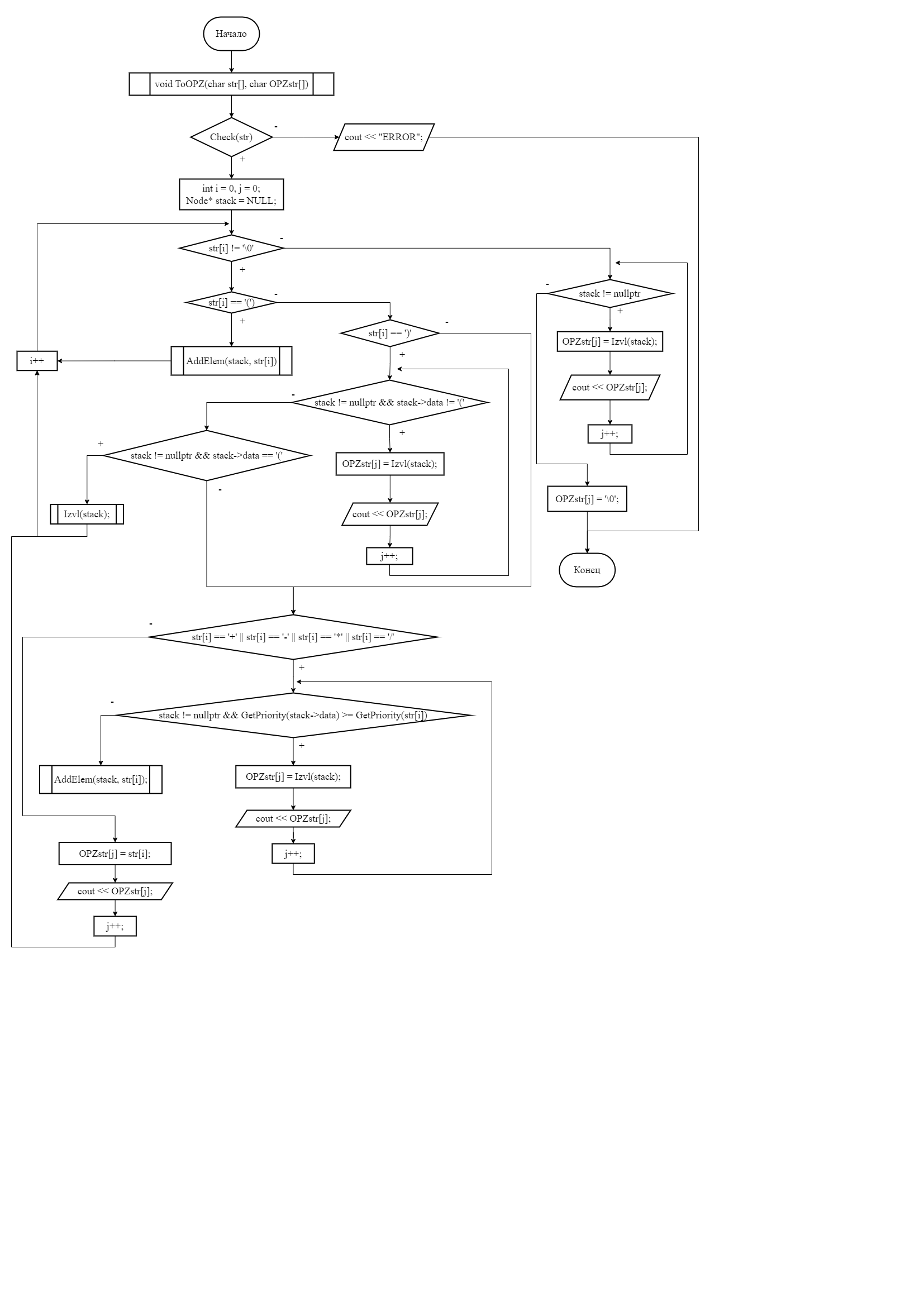


Рисунок 4 – Блок-схема функции ToOPZ

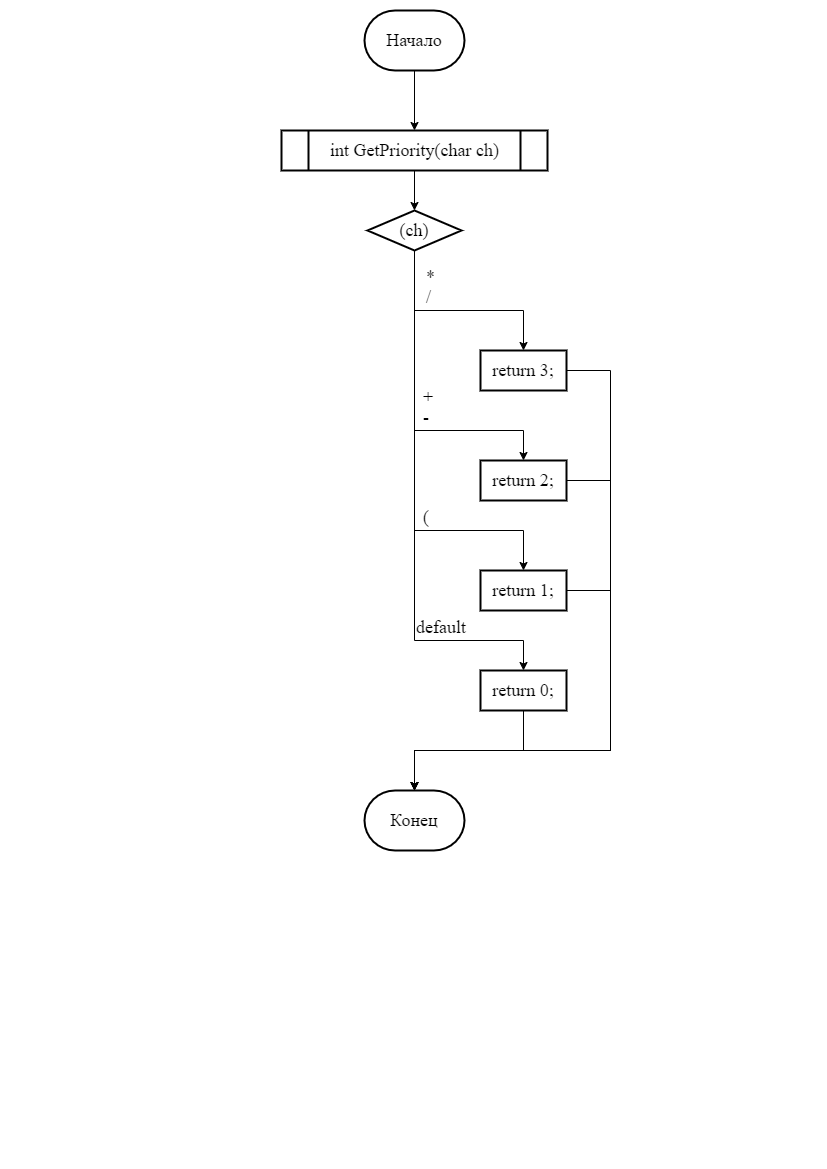


Рисунок 5 – Блок-схема функции GetPriority

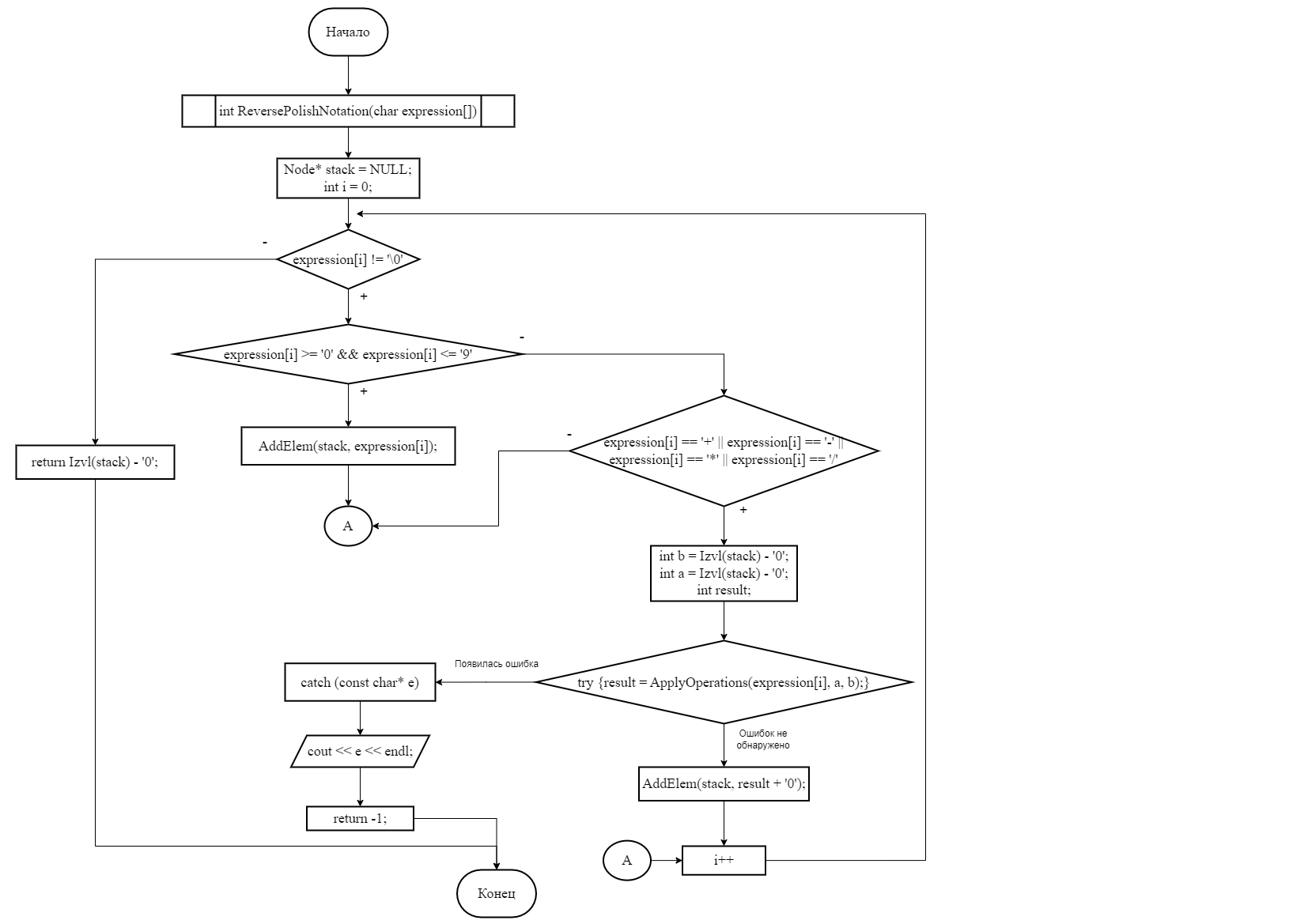


Рисунок 6 – Блок-схема функции ReversePolishNotation

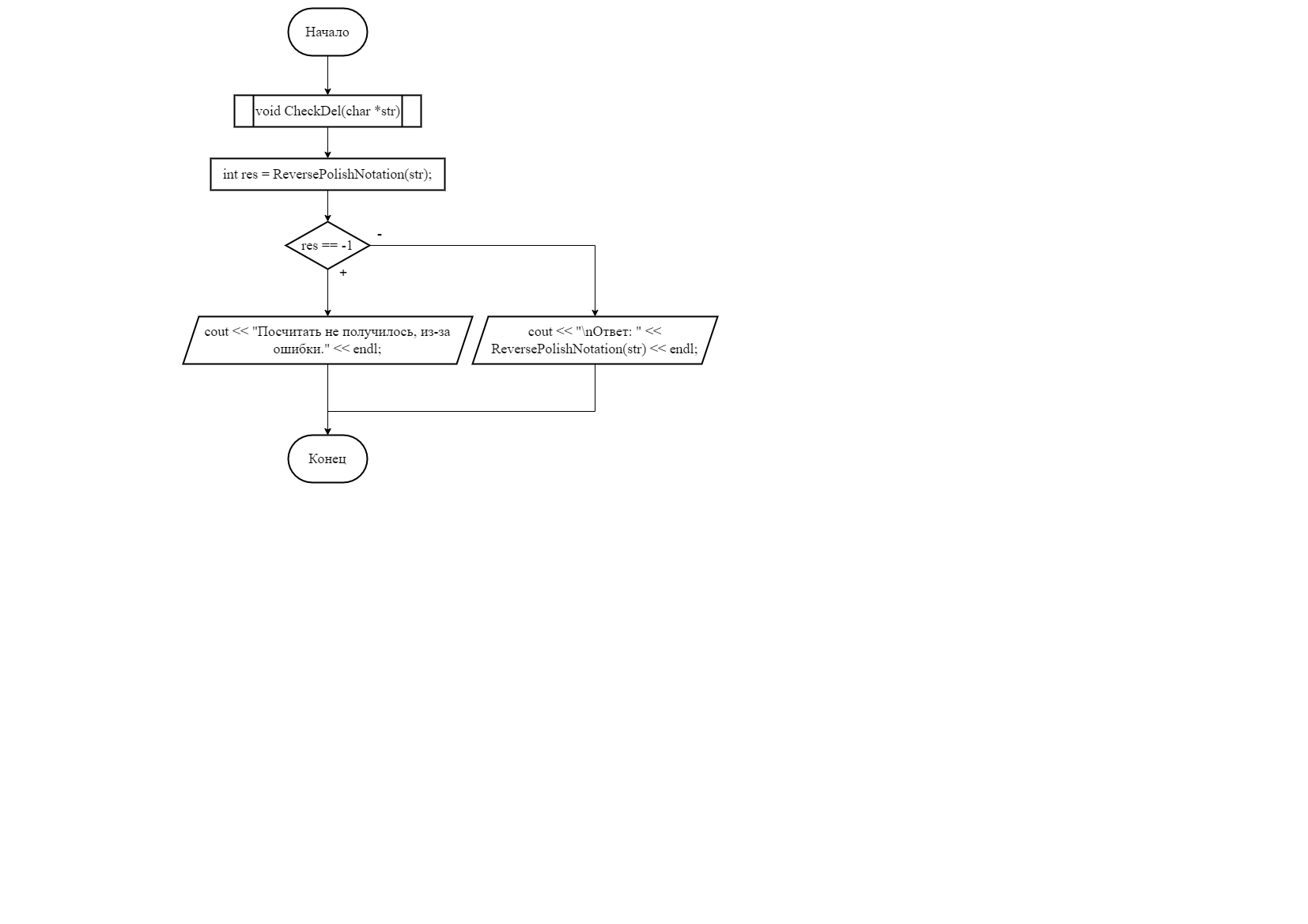


Рисунок 7 – Блок-схема функции CheckDel

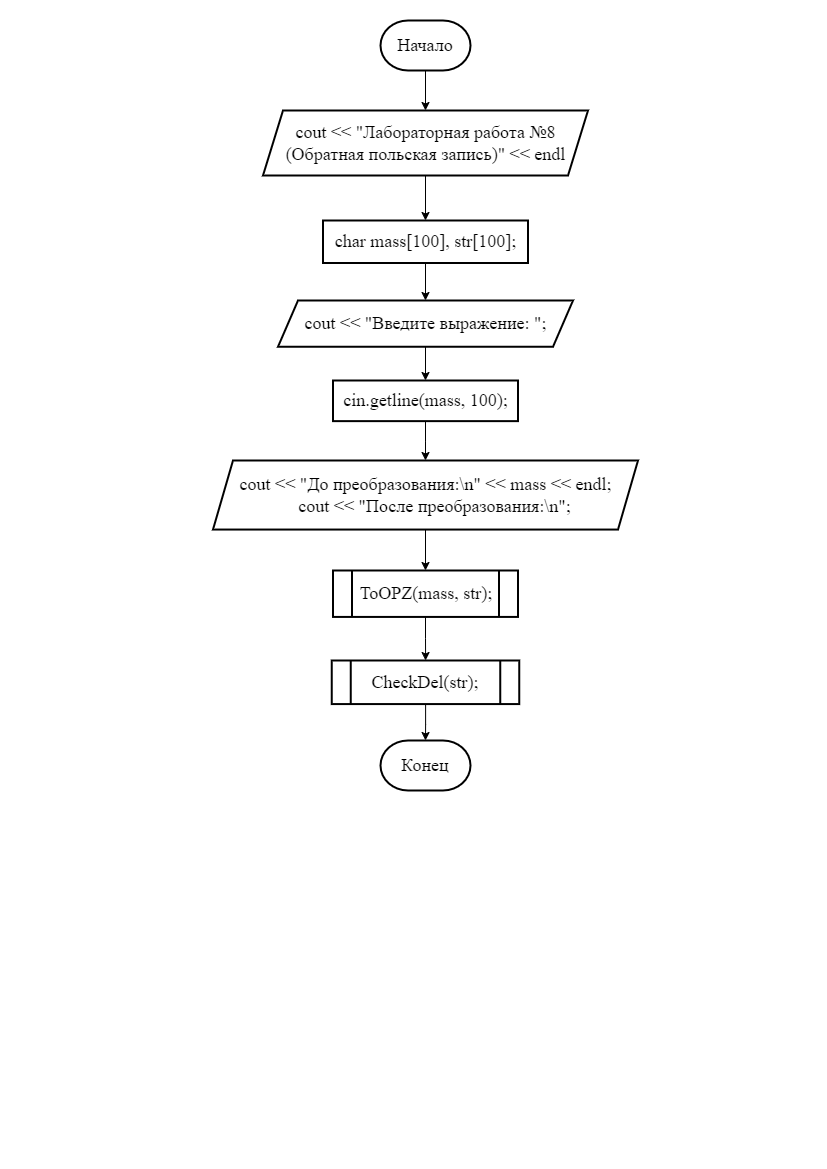


Рисунок 8 – Блок-схема функции Main

**Вывод:** в ходе лабораторной работы были получены навыки в вычислении алгебраических выражений, используя алгоритм обратной польской записи.

Преимущества обратной польской записи:

Устранение необходимости в порядке операций и скобках, то есть она может быть вычислена линейно; удобство в формулах, использующих стеки.

Недостатки:

Более сложная читаемость в сравнении со стандартной формой записи; необходимость использования дополнительной памяти.