Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ОТЧЕТ

по лабораторной работе №7

Тема работы:

Работа со строками

Выполнил

студент: гр. 151004 Данилов Ф.А.

Проверил: Фадеева Е.П.

Минск 2021

Содержание

[1 Постановка задачи 3](#_Toc90912532)

[2 Методика решения 4](#_Toc90912533)

[3 Структура данных 6](#_Toc90912534)

[4 Схема алгоритма решения задачи по ГОСТ 19.701-90 7](#_Toc90912535)

[5 Результаты расчетов 11](#_Toc90912536)

[Приложение А 12](#_Toc90912537)

[Приложение Б 16](#_Toc90912538)

# Постановка задачи

Преобразовать выражение (т.е. текст специального вида), составленное из ­цифр и знаков четырех арифметических операций (сложения, вычитания, умножения, деления), из инфиксной в постфиксную форму. Операнды – однозначные числа.

Правила постановки:

1. Пользователь вводит математическое выражение в инфиксной форме с клавиатуры.
2. Программа учитывает некорректный ввод пользователя (некорректным вводом считаются любые символы, не перечисленные в условии задачи).
3. Программа не проверяет семантическую структуру введенного пользователем выражения.
4. Вводимые пользователем символы должны быть цифрами или знаками простейших арифметических операций (сложение, вычитание, умножение, деление).
5. Программа выводит введенное пользователем выражение, но уже в постфиксной форме.

# Методика решения

Существует несколько алгоритмов для превращения инфиксных формул в обратную польскую запись. Рассматривается переработанный алгоритм, идея которого предложена Э. Дейкстра. Предпологается, что формула состоит из переменных, двухоперандных операторов +,-,\*,/, а также левой и правой скобок. Чтобы отметить конец формулы, можно вставлять символ после её последнего символа и перед первым символом следующей формулы.

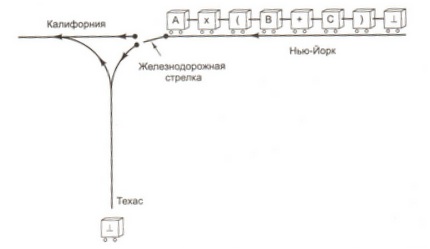


Рисунок 1 – Визуализация алгоритма обратной польской записи

На рисунке схематично показана железная дорога из Нью-Йорка в Калифорнию с ответвлением, ведущим в Техас. Каждый символ формулы представлен одним вагоном. Поезд движется на запад (налево). Перед стрелкой каждый вагон должен останавливаться и узнавать, должен ли он двигаться прямо в Калифорнию, или ему нужно будет по пути заехать в Техас. Вагоны, содержащие операнды, всегда направляются в Калифорнию и никогда не едут в Техас. Вагоны, содержащие все прочие символы, должны перед прохождением стрелки узнавать о содержимом ближайшего вагона, отправившегося в Техас.  
В таблице показана зависимость ситуации от того, какой вагон отправился в Техас последним и какой вагон находится у стрелки. Первый вагон (помеченный символом ⊥) всегда отправляется в Техас.

Таблица 1 – Зависимость действий от текущего символа

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ⊥ | + | - | \* | / | ( | ) |
| ⊥ | 4 | 1 | 1 | 1 | 1 | 1 | 5 |
| + | 2 | 2 | 2 | 1 | 1 | 1 | 2 |
| - | 2 | 2 | 2 | 1 | 1 | 1 | 2 |
| \* | 2 | 2 | 2 | 2 | 2 | 1 | 2 |
| / | 2 | 2 | 2 | 2 | 2 | 1 | 2 |
| ( | 5 | 1 | 1 | 1 | 1 | 1 | 3 |

Числа соответствуют следующим ситуациям:

1. Вагон на стрелке отправляется в Техас
2. Последний вагон, направившийся в Техас, разворачивается и направляется в Калифорнию
3. Вагон, находящийся на стрелке, и последний вагон, отправившийся в Техас, угоняются и исчезают
4. Остановка. Символы, находящиеся на Калифорнийской ветке, представляют собой формулу в обратной польской записи, если читать слева направо
5. Остановка. Произошла ошибка. Изначальная формула была некорректно сбалансирована

После каждого действия производится новое сравнение вагона, находящегося у стрелки (это может быть тот же вагон, что и в предыдущем сравнении, а может быть следующий вагон), и вагона, который на данный момент последним ушёл на Техас. Этот процесс продолжается до тех пор, пока не будет достигнут шаг 4.

Линия на Техас используется как стек, где отправка вагона в Техас – это помещение элемента в стек, а разворот отправленного в Техас вагона в сторону Калифорнии – это выталкивание элемента из стека.

Порядок следования операндов в инфиксной и постфиксной записи одинаков. Однако порядок следования операторов не всегда один и тот же. В обратной польской записи операторы появляются в том порядке, в котором они будут выполняться.

# Структура данных

Таблица 2 – Данные

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| InfNot | String | Строка, записанная в инфиксной форме |
| PostNot | String | Строка, записанная в постфиксной форме |
| Stack | TStack<String> | Стек, в который записываются операторы |
| Digits | Set of Integer | Множество символов цифр для проверки текущего символа строки InfNot. Принимает значения = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] |
| i | Integer | Счетчик для цикла |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

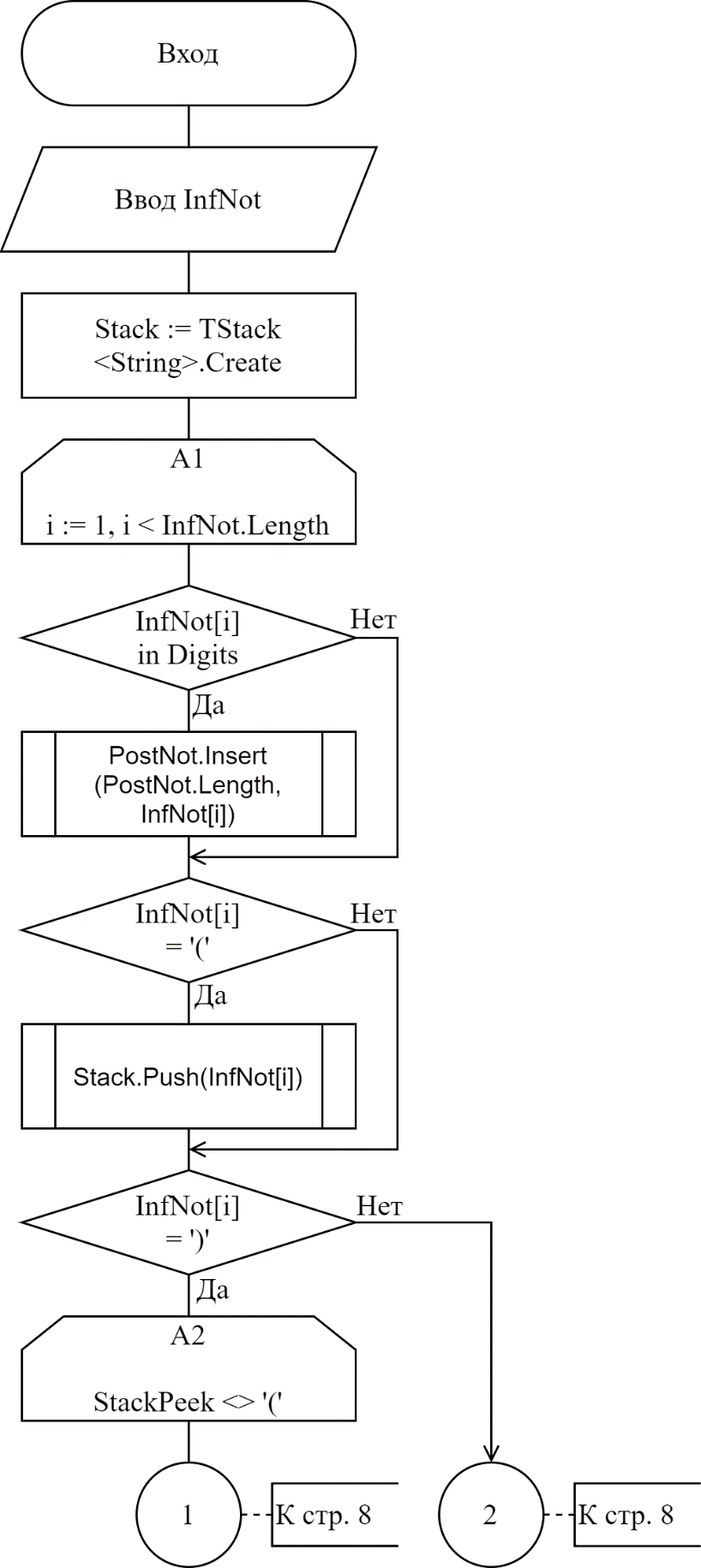


Рисунок 2 - Схема алгоритма решения задачи по ГОСТ 19.701-90 (часть 1)

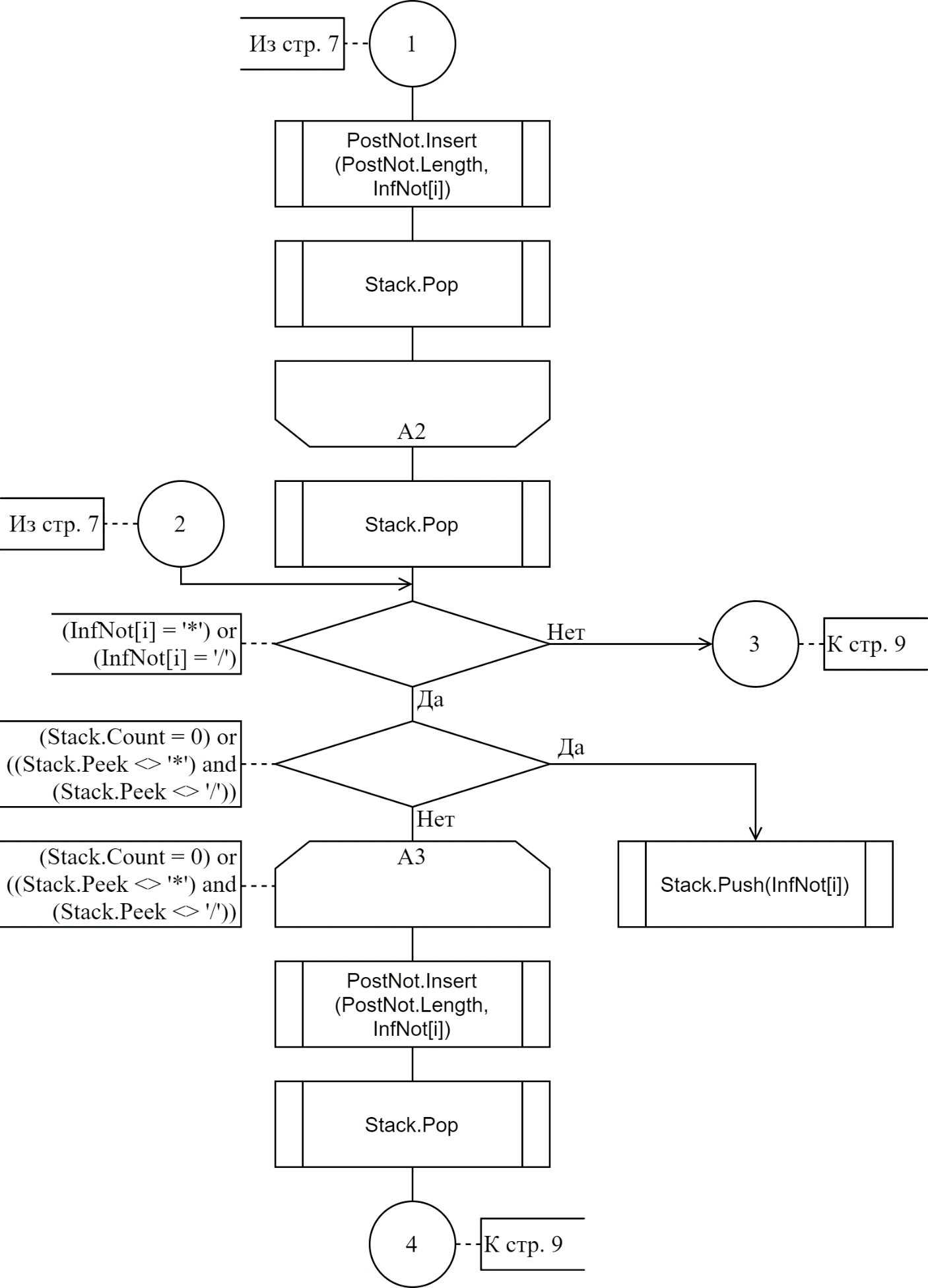


Рисунок 3 - Схема алгоритма решения задачи по ГОСТ 19.701-90 (часть 2)

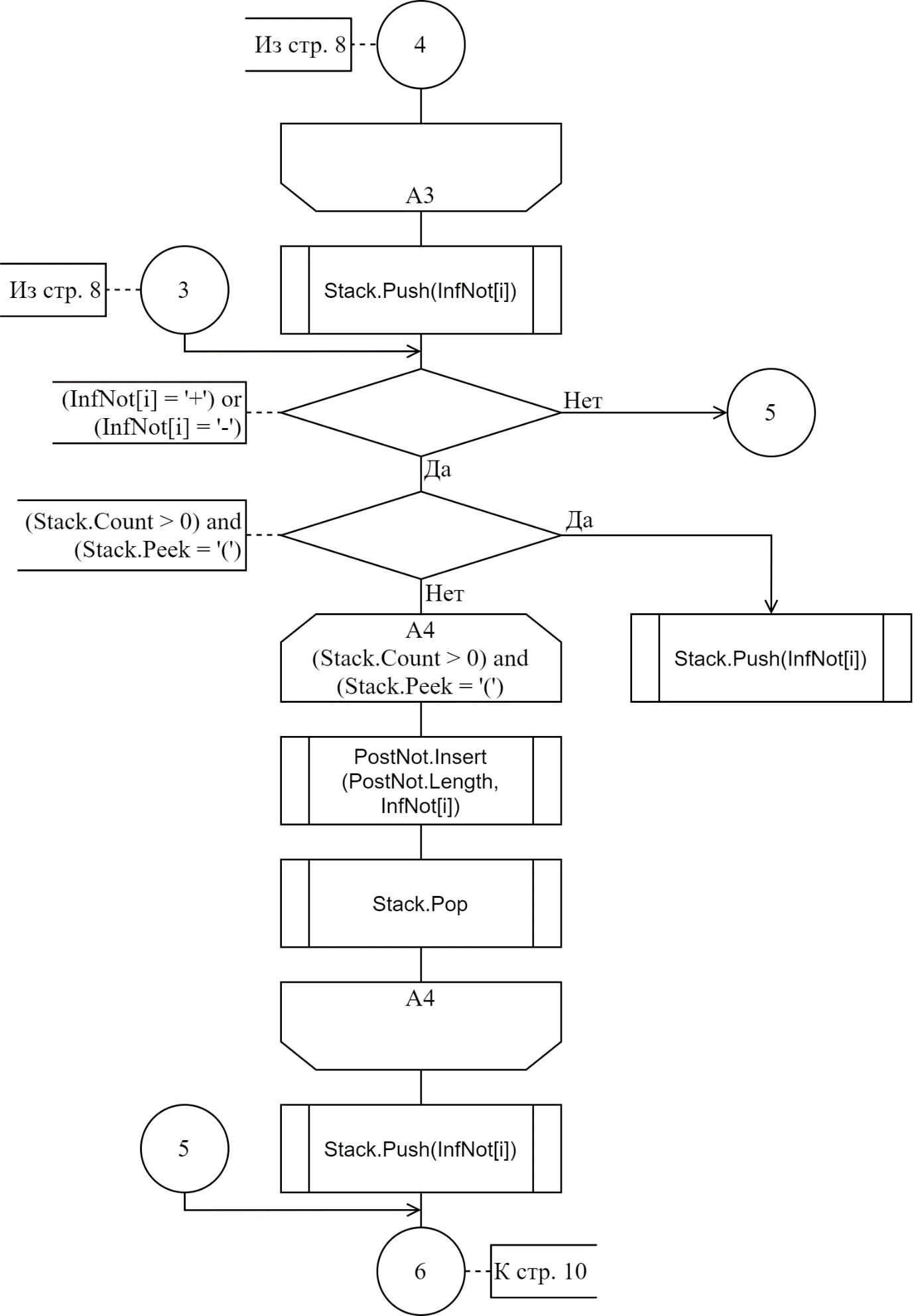


Рисунок 4 - Схема алгоритма решения задачи по ГОСТ 19.701-90 (часть 3)

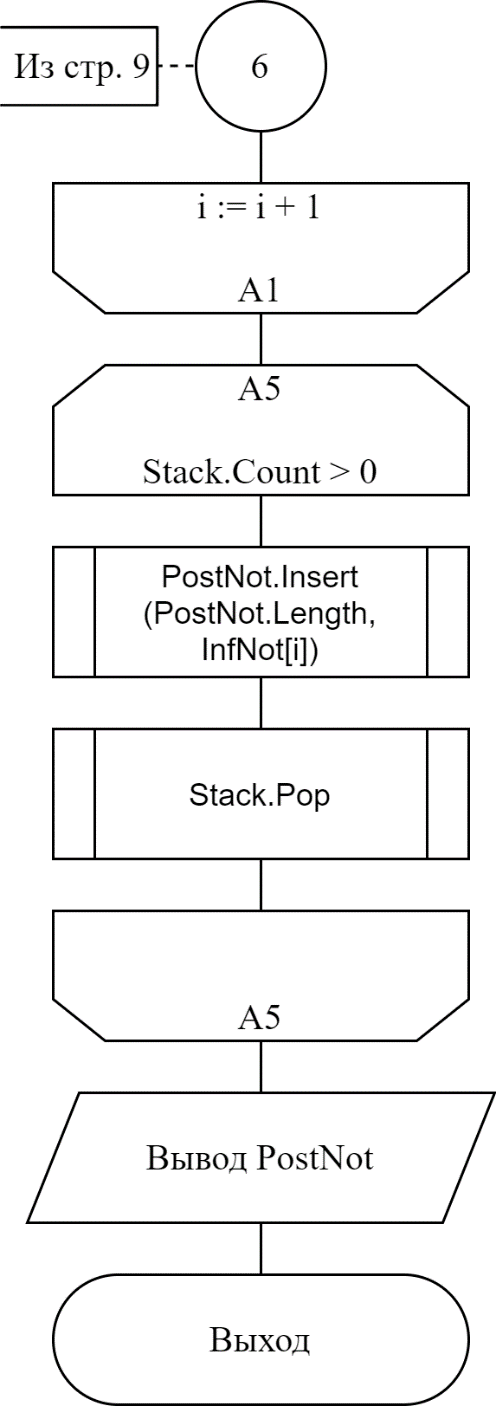


Рисунок 5 - Схема алгоритма решения задачи по ГОСТ 19.701-90 (часть 4)

# Результаты расчетов

Вследствие выполнения программы на экран выводятся следующие результаты:

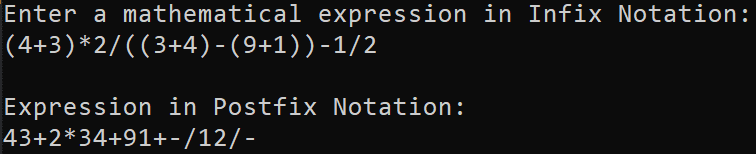


Рисунок 6 - Результаты расчётов

Приложение А

(обязательное)

program Lab7;

{

Given mathematical expression in Infix Notation

(Allowed Symbols [digits, '(', ')', '\*',

'/', '+', '-']) convert it to Postfix Notation

Input: Expression in Infix Notation - String

Output: Expression in Postfix Notation - String

}

{$APPTYPE CONSOLE}

{$R \*.res}

// Import library with TStack

uses

System.SysUtils, System.Generics.Collections;

// Error - error value to check for allowed symbols

// i - index for 'for' loop to iterate through string

// Stack - stack of elements to calculte algorythm

// AllowedSyms - set of elements that are allowed in

// given expression

// Digits - set of elements to check if current element

// is digit

// InputStr - input string

// InfNot - expression in Infix Notation

// PostNot - expression in Postfix Notation (Reverse Polish Notation)

var

Error, i: Integer;

Stack: TStack<String>;

AllowedSyms: set of Char = ['0','1','2','3','4','5','6','7',

'8','9','+','-','\*','/','(',')'];

Digits: set of Char = ['0','1','2','3','4','5','6','7','8','9'];

InputStr, InfNot, PostNot: String;

begin

// Enter a mathematical expression

// in Infix Notation

repeat

// Input line

writeln('Enter a mathematical expression in Infix

Notation:');

readln(InputStr);

// Check if all chars in InputStr are allowed

// If not, Error := index of failure

Error:=0;

i:=0;

repeat

i := i + 1;

if not(InputStr[i] in AllowedSyms) then Error:=i;

until(Error <> 0) or (i = InputStr.Length);

// If Error was found output index of failed

// element

if (Error <> 0) then

begin

writeln;

writeln('Invalid symbol on position ', Error);

writeln('Expression must consist of digits and

operators: +, -, \*, /');

writeln;

end;

until Error = 0;

// After completing all checks equate inputted string

// to InfNot

InfNot := InputStr;

// Initialize stack

Stack := TStack<String>.Create;

// Iterate through chars of inputted line InfNot

for i := 1 to InfNot.Length do

begin

// Check if current symbol is number

// If so, add it to final line

if InfNot[i] in Digits then

PostNot.Insert(PostNot.Length, InfNot[i]);

// Check if current symbol is '('

// If so, push it to stack

if InfNot[i] = '(' then

Stack.Push(InfNot[i]);

// Check if current symbol is ')'

// If so, move all elements from stack to final

// line until '(' is present

if InfNot[i] = ')' then

begin

while Stack.Peek <> '(' do

begin

PostNot.Insert(PostNot.Length, Stack.Peek);

Stack.Pop;

end;

Stack.Pop;

end;

// Check if current symbol is either '\*' or '/'

if (InfNot[i] = '\*') or (InfNot[i] = '/') then

begin

// Check if top of stack contains the same symbol

// If not, push current symbol to stack

// Else, move all elements from stack to final

// line while top of stack is '\*' or '/'

if (Stack.Count = 0) or ((Stack.Peek <> '\*')

and (Stack.Peek <> '/')) then

Stack.Push(InfNot[i])

else

begin

while (Stack.Count > 0) and ((Stack.Peek = '\*')

or (Stack.Peek = '/')) do

begin

PostNot.Insert(PostNot.Length, Stack.Peek);

Stack.Pop;

end;

Stack.Push(InfNot[i]);

end;

end;

// If current symbol is eiter '+' of '-'

if (InfNot[i] = '+') or (InfNot[i] = '-') then

begin

// Check if top of stack is '('

// If so, push current symbol to stack

// Else, move all elements from stack to

// final line until '(' is present

if (Stack.Count > 0) and (Stack.Peek = '(') then

Stack.Push(InfNot[i])

else

begin

while(Stack.Count > 0)and(Stack.Peek <> '(')do

begin

PostNot.Insert(PostNot.Length, Stack.Peek);

Stack.Pop;

end;

Stack.Push(InfNot[i]);

end;

end;

end;

// After cycle move all elements from stack to final

// line

while Stack.Count > 0 do

begin

PostNot.Insert(PostNot.Length, Stack.Peek);

Stack.Pop;

end;

// Output final line in reverse polish notation

writeln;

writeln('Expression in Postfix Notation:');

writeln(PostNot);

// Stop console from closing

readln;

end.

Приложение Б

(обязательное)

Тестовые наборы

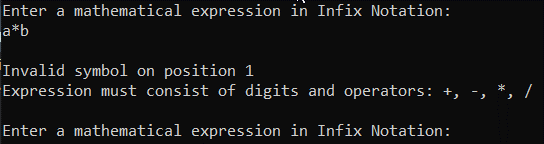
Тестовые наборы для проверки ввода данных

Тест 1

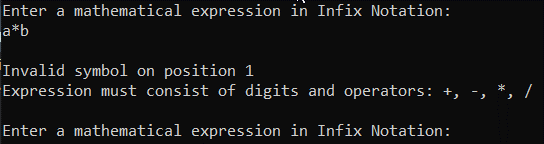
Тестовая ситуация: операнд, не входящий в AllowedSyms

Исходные данные:

Ожидаемый резуьтат:



Полученный результат:

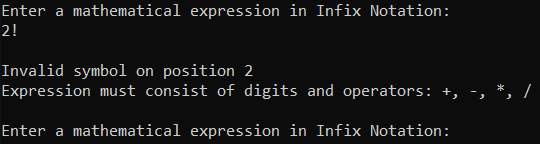


Тест 2

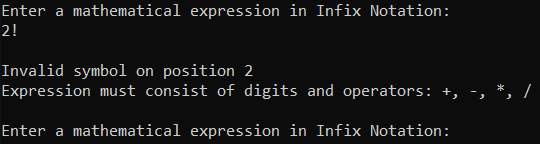
Тестовая ситуация: оператор, не входящий в AllowedSyms

Исходные данные:

Ожидаемый резуьтат:



Полученный результат:



Тестовые наборы для проверки вычислений

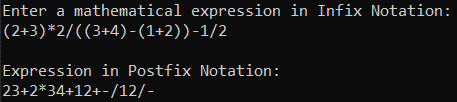
Тест 1

Тестовая ситуация: проверка выполнения алгоритма

Исходные данные:

Ожидаемый резуьтат:

Полученный результат:



Тест 2

Тестовая ситуация: проверка выполнения алгоритма

Исходные данные:

Ожидаемый резуьтат:

Полученный результат:

