



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации
информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5

Тема: Применение стека и очереди при преобразовании
арифметических выражений в постфиксную, префиксную
нотации и вычисление значений выражений

по дисциплине

«Дисциплина структуры и алгоритмы обработки данных»

Выполнил студент группы ИНБО-15-20

Ло Ван Хунг

Принял ассистент кафедры МОСИТ

Коваленко М.А.

Практическая
работа выполнена

«__»_____2021 г.

«Зачтено»

«__»_____2021 г.

Москва 2021

Содержание:

Отчёт по упражнениям задания 1	3
Отчет по программной реализации задания 2	7
Вывод	18
Информационные источники	18

Вариант: 3

1. Отчёт по упражнениям задания 1

1.1)

Условие: Провести преобразование инфиксной записи выражения в префиксную нотацию, расписывая процесс по шагам $S = a + (b - c * k) - d * e - f$

$a + (b - c * k) - d * e - f$		
Исходная строка	Стек	Выходная строка
$a + (b - c * k) - d * e -$		f
$a + (b - c * k) - d * e$	-	f
$a + (b - c * k) - d *$	-	ef
$a + (b - c * k) - d$	-*	ef
$a + (b - c * k) -$	-*	def
$a + (b - c * k)$	--	*def
$a + (b - c * k$	--)	*def
$a + (b - c *$	--)	k*def
$a + (b - c$	--)*	k*def
$a + (b -$	--)*	ck*def
$a + (b$	--)-	*ck*def
$a + ($	--)-	b*ck*def
$a +$	--	-b*ck*def
a	--+	a-b*ck*def
		--+a-b*ck*def

Ответ: --+a-b*ck*def

1.2)

Условие: Представить постфиксную нотацию выражений $a + (c - b) / (b * d)$, $(a + b) * c - (d + e * f / ((g / h + i - j) * k)) / r$

$a+(c-b)/(b*d)$		
Исходная строка	Стек	Выходная строка
$+(c-b)/(b*d)$		a
$(c-b)/(b*d)$	+	a
$c-b)/(b*d)$	+(a
$-b)/(b*d)$	+(ac
$b)/(b*d)$	+(-	ac
$)/(b*d)$	+(-	acb
$/(b*d)$	+	acb-
$(b*d)$	+/	acb-
$b*d)$	+/ (acb-
$*d)$	+/ (acb-b
$d)$	+/ (*	acb-b
$)$	+/ (*	acb-bd
		acb-bd*/+

Ответ: acb-bd*/+

$(a+b)*c-(d+e*f/((g/h+i-j)*k))/r$		
Исходная строка	Стек	Выходная строка
$a+b)*c-(d+e*f/((g/h+i-j)*k))/r$	(
$+b)*c-(d+e*f/((g/h+i-j)*k))/r$	(a
$b)*c-(d+e*f/((g/h+i-j)*k))/r$	(+	a
$)*c-(d+e*f/((g/h+i-j)*k))/r$	(+	ab
$*c-(d+e*f/((g/h+i-j)*k))/r$		ab+
$c-(d+e*f/((g/h+i-j)*k))/r$	*	ab+
$-(d+e*f/((g/h+i-j)*k))/r$	*	ab+c
$(d+e*f/((g/h+i-j)*k))/r$	-	ab+c*
$d+e*f/((g/h+i-j)*k))/r$	-(ab+c*
$+e*f/((g/h+i-j)*k))/r$	-(ab+c*d
$e*f/((g/h+i-j)*k))/r$	-(+	ab+c*d
$*f/((g/h+i-j)*k))/r$	-(+	ab+c*de
$f/((g/h+i-j)*k))/r$	-(+*	ab+c*de
$/((g/h+i-j)*k))/r$	-(+*	ab+c*def
$((g/h+i-j)*k))/r$	-(+/*	ab+c*def*
$(g/h+i-j)*k))/r$	-(+/(ab+c*def*
$g/h+i-j)*k))/r$	-(+/(ab+c*def*
$/h+i-j)*k))/r$	-(+/(ab+c*def*g
$h+i-j)*k))/r$	-(+/(ab+c*def*g
$+i-j)*k))/r$	-(+/(ab+c*def*gh
$i-j)*k))/r$	-(+/(ab+c*def*gh/
$-j)*k))/r$	-(+/(ab+c*def*gh/i
$j)*k))/r$	-(+/(ab+c*def*gh/i+
$)*k))/r$	-(+/(ab+c*def*gh/i+j
$*k))/r$	-(+/(ab+c*def*gh/i+j-
$k))/r$	-(+/(ab+c*def*gh/i+j-
$))/r$	-(+/(ab+c*def*gh/i+j-k
$)/r$	-(+/*	ab+c*def*gh/i+j-k*
$/r$	-	ab+c*def*gh/i+j-k*/+
r	-/	ab+c*def*gh/i+j-k*/+r
	-/	ab+c*def*gh/i+j-k*/+r-

Ответ: $ab+c*def*gh/i+j-k*/+r/-$

1.3) Представить префиксную нотацию выражений п.2

a+(c-b)/(b*d)		
Исходная строка	Стек	Выходная строка
a+(c-b)/(b*d)	
a+(c-b)/(b*)	d
a+(c-b)/(b)*	d
a+(c-b)/()*	bd
a+(c-b)/		*bd
a+(c-b)	/	*bd
a+(c-b)	*bd
a+(c-)	b*b
a+(c)-	b*b
a+()-	cb*b
a+		-cb*b
a	+	-cb*b
	+	a-cb*b
		+a-cb*b

Ответ: +a-cb*b

(a+b)*c-(d+e*f/((g/h+i-j)*k))/r		
Исходная строка	Стек	Выходная строка
(a+b)*c-(d+e*f/((g/h+i-j)*k))/		r
(a+b)*c-(d+e*f/((g/h+i-j)*k))	/	r
(a+b)*c-(d+e*f/((g/h+i-j)*k)	/)	r
(a+b)*c-(d+e*f/((g/h+i-j)*k	/))	r
(a+b)*c-(d+e*f/((g/h+i-j)*	/))	kr
(a+b)*c-(d+e*f/((g/h+i-j)	/))*	kr
(a+b)*c-(d+e*f/((g/h+i-j	/))*)	kr
(a+b)*c-(d+e*f/((g/h+i-	/))*)	jkr
(a+b)*c-(d+e*f/((g/h+i	/))*)-	jkr
(a+b)*c-(d+e*f/((g/h+	/))*)-	ijkr
(a+b)*c-(d+e*f/((g/h	/))*)-+	ijkr
(a+b)*c-(d+e*f/((g/	/))*)-+	hijkr
(a+b)*c-(d+e*f/((g	/))*)-+/	hijkr
(a+b)*c-(d+e*f/((/))*)-+/	ghijkr
(a+b)*c-(d+e*f/(/))*	-+/ghijkr
(a+b)*c-(d+e*f/	/)	*-+/ghijkr
(a+b)*c-(d+e*f	/)/	*-+/ghijkr

$(a+b)*c-(d+e*$	$/)/$	$f*-/ghijkr$
$(a+b)*c-(d+e$	$/)/*$	$f*-/ghijkr$
$(a+b)*c-(d+$	$/)/*$	$ef*-/ghijkr$
$(a+b)*c-(d$	$/)+$	$/ef*-/ghijkr$
$(a+b)*c-($	$/)+$	$d/ef*-/ghijkr$
$(a+b)*c-$	$/$	$+d/ef*-/ghijkr$
$(a+b)*c$	$-$	$/+d/ef*-/ghijkr$
$(a+b)*$	$-$	$c/+d/ef*-/ghijkr$
$(a+b)$	$-*$	$c/+d/ef*-/ghijkr$
$(a+b$	$-*)$	$c/+d/ef*-/ghijkr$
$(a+$	$-*)$	$bc/+d/ef*-/ghijkr$
$(a$	$-*)+$	$bc/+d/ef*-/ghijkr$
$($	$-*)+$	$abc/+d/ef*-/ghijkr$
		$-*+abc/+d/ef*-/ghijkr$

Ответ: $-*+abc/+d/ef*-/ghijkr$

1.4)

Условие: Провести вычисление значения выражения представленного в постфиксной форме, расписывая процесс по шагам $7\ 2\ -\ 3\ 2\ 3\ +\ *\ +$

72-323+*+	
Ввод	Стек
7	7
2	7 2
-	5
3	5 3
2	5 3 2
3	5 3 2 3
+	5 3 5
*	5 15
+	20

Ответ: 20

2. Отчет по программной реализации задания 2

2.1) Постановка задачи: Реализовать операции стек: втолкнуть элемент в стек, вытолкнуть элемент из стека, вернуть значение элемента в вершине стека, сделать стек пустым, определить пуст ли стек. Рассмотреть два варианта реализации: на массиве (или строке); на однонаправленном списке. Создать класс или просто заголовочный файл с функциями. Применить операции для вычисления значения выражения п.4 данного варианта.

Определение прототипов функций, реализующих операции с пред- и постусловием.

```
// Receive a new element to be pushed onto the stack
// The new element is pushed onto the stack
template <typename T>
void push(T);

// Receive nothing
// Pops the last element
out template <typename T>
void pop();

// Receives nothing
// Returns the data field of the last
element template <typename T>
T top();

// Receives nothing
// Makes the stack empty
template <typename T>
void makeEmpty();

// Receives nothing
// Returns true if the stack is empty, otherwise returns false
template <typename T>
bool isEmpty();

// Receives nothing
// Returns the size of the stack
template <typename T>
size_t getSize();
```

Структура однонаправленного списка

```
struct Node
{
    T data;
    Node* next = nullptr;
};
```

Код реализации стека на динамическом массиве

arrayStack.hpp

```
#pragma once

#include <cstdlib>

template <typename T>
class ArrayStack
{
public:
    ArrayStack() {}
    ~ArrayStack()
    {
        makeEmpty();
    }

private:
    size_t size = 0;
    T* stack = nullptr;

public:
    // Receive a new element to be pushed onto the stack
    // The new element is pushed onto the stack
```



```

void push(T newElement)
{
    stack = static_cast<T*>(std::realloc(stack, sizeof(T) * (size +
1))); stack[size] = newElement;
    ++size;
}

// Receive nothing
// Pops the last element
out void pop()
{
    stack = static_cast<T*>(std::realloc(stack, sizeof(T) * (--size));
}

// Receives nothing
// Returns the data field of the last element
T top()
{
    return stack[size - 1];
}

// Receives nothing
// Makes the stack
empty void makeEmpty()
{
    std::free(stack);
    size = 0;
}

// Receives nothing
// Returns true if the stack is empty, otherwise returns false
bool isEmpty()
{
    return size == 0 ? true : false;
}

// Receives nothing
// Returns the size of the stack
size_t getSize()
{
    return size;
}
};

```

Код реализации стека на однонаправленном списке

linkedListStack.hpp

```

#pragma once

template <typename T>
class LinkedListStack
{
public:
    LinkedListStack() {}
    ~LinkedListStack() { makeEmpty(); }

private:
    struct Node
    {
        T data;
        Node* next = nullptr;
    };
};

```

```

size_t size;
Node* stack;

public:
    // Receive a new element to be pushed onto the stack
    // The new element is pushed onto the stack
    void push(T newElement)
    {
        if (stack)
        {
            stack->next = new Node;
            stack = stack->next;
        }
        else
            stack = new Node;

        stack->data = newElement;
        stack->next = nullptr;

        size++;
    }

    // Receive nothing
    // Pops the last element
    out void pop()
    {
        Node* current = stack, *last = nullptr;

        if (current)
        {
            for (; current->next; last = current, current = current->next);

            delete current;

            if (last)
                last->next = nullptr;

            stack = last;

            --size;
        }
    }

    // Receives nothing
    // Returns the data field of the last
    element T top()
    {
        Node* current = stack;
        for (; current && current->next; current = current->next);
        return current->data;
    }

    // Receives nothing
    // Makes the stack empty
    void makeEmpty()
    {
        Node* current = stack, *last = nullptr;

        for (; current; last = current, current = current->next)
        {
            if (last)
                delete last;
        }
    }

```

```

        if (!last)
            delete last;

        size = 0;
    }

    // Receives nothing
    // Returns true if the stack is empty, otherwise returns false
    bool isEmpty()
    {
        return size == 0 ? true : false;
    }

    // Receives nothing
    // Returns the size of the stack
    size_t getSize()
    {
        return size;
    }
};

```

Код функции вычисления данного постфиксного выражения

main.cpp

```

#include <iostream>
#include <cstdlib>
#include <vector>
#include <string>

#include "LinkedListStack.hpp"
#include "arrayStack.hpp"

std::vector<std::string> tokenize(const std::string str)
{
    std::string token;
    std::vector<std::string> tokenList;
    bool completeToken = false;

    for (auto i = str.begin(); i != str.end(); i++)
    {
        if (token.empty() && (*i == '+' || *i == '-' || *i == '*' || *i == '/' || *i
== '(' || *i == ')'))
        {
            token.push_back(*i);
            completeToken = true;
        }
        else if (isdigit(*i))
        {
            token.push_back(*i);
        }
        else if (!token.empty())
        {
            completeToken = true;
            --i;
        }

        if (completeToken)
        {
            tokenList.push_back(token);
            completeToken = false;
            token.clear();
        }
    }
}

```

```

    }
}

if (!token.empty())
    tokenList.push_back(token);

return tokenList;
}

int doMath(int operand1, int operand2, char operation)
{
    if (operation == '+')
        return operand1 + operand2;
    else if (operation == '-')
        return operand1 - operand2;
    else if (operation == '*')
        return operand1 * operand2;

    return operand1 / operand2;
}

int solvePostfixExpression(const std::string postfixExpression)
{
    int result = 0;
    ArrayStack<int> operandStack;
    std::vector<std::string> tokenList;

    tokenList = tokenize(postfixExpression);

    for (auto i = tokenList.begin(); i != tokenList.end(); i++)
    {
        if (isdigit((*i)[0]))
        {
            operandStack.push(std::stoi(*i));
        }
        else
        {
            int operand2 = operandStack.top();
            operandStack.pop();
            int operand1 = operandStack.top();
            operandStack.pop();

            result = doMath(operand1, operand2,
                (*i)[0]); operandStack.push(result);
        }
    }

    return result;
}

int main()
{
    std::cout << solvePostfixExpression("7 2 - 3 2 3 + * +");

    return 0;
}

```

```

Консоль отладки Microsoft Visual Studio
20
D:\C++ Projects\ForStuding\ForStuding\Debug\ForStuding.exe (процесс 10444) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

int main()
{
    std::cout << solvePostfixExpression("7 2 - 3 2 3 + * +");

    return 0;
}

```

Рис.1 Результат тестирования алгоритма

2.2)

Постановка задачи: Разработать функцию(ии) вычисления значения выражения, представленного в префиксной форме.

Описание подхода решения: задача решается аналогично предыдущей, (если токен операнда, кладём его в стек, если оператора, то вынимаем из стека (в случае этой задачи) два операнда и производим с ними операцию) за исключением того, что обход строки идёт справа налево.

Алгоритм на псевдокоде
<pre> def solvePostfixExpression(prefixExpr): operandStack = [] result = 0 tokenList = tokenize(prefixExpr) for i in reversed(tokenList): if i in "0123456789": operandStack.push(i) else: operand2 = operandStack.pop() operand1 = operandStack.pop() result = doMath(operand1, operand2, i) operandStack.push(result) return result def doMath(operand1, operand2, operation): if operation == "+": return operand1 + operand2 elif operation == "-": return operand1 - operand2 elif operation == "*": return operand1 * operand2 return operand1 / operand2 def tokenize(str): </pre>

```

token = ""
tokenList = []
completeToken = False

for i in str:
    if empty(i) && (i == '+' || i == '-' || i == '*' || i == '/' || i == '(' || i == ')'):
        token = i
        completeToken = True
    elif i in "0123456789":
        token = token + i
    elif not empty(token):
        completeToken = True
        prev(i)

    if completeToken == True:
        tokenList.push(token)
        completeToken = False
        clear(token)

print(prefixEval('+ 7 * 1 * 3 + 4 5'))

```

Описание всех используемых переменных в функции solvePostfixExpression:

- prefixExpr – строка с префиксным выражением
- operandStack – стек операндов
- result – целочисленная переменная для хранения результата
- tokenList – массив токенов
- i – итератор массива токенов

Описание всех используемых переменных в функции doMath:

- operand1 – операнд выражения
- operand2 – операнд выражения
- operation – операция выражения

Описание всех используемых переменных в функции tokenize:

- str – строковая переменная с выражением
- token – строковая переменная, хранящая текущий токен
- tokenList – массив токенов
- completeToken – булева переменная, хранящая информацию о полноте токена
- i – итератор массива токенов

Код алгоритма

```

#include <iostream>
#include <stdlib.h>
#include <vector>
#include <string>

#include "LinkedListStack.hpp"
#include "arrayStack.hpp"

std::vector<std::string> tokenize(const std::string str)
{
    std::string token;
    std::vector<std::string> tokenList;
    bool completeToken = false;

    for (auto i = str.begin(); i != str.end(); i++)
    {
        if (token.empty() && (*i == '+' || *i == '-' || *i == '*' || *i == '/' || *i
== '(' || *i == ')'))
        {

```

```

        token.push_back(*i);
        completeToken = true;
    }
    else if (isdigit(*i))
    {
        token.push_back(*i);
    }
    else if (!token.empty())
    {
        completeToken = true;
        --i;
    }

    if (completeToken)
    {
        tokenList.push_back(token);
        completeToken = false;
        token.clear();
    }
}

if (!token.empty())
    tokenList.push_back(token);

return tokenList;
}

int doMath(int operand1, int operand2, char operation)
{
    if (operation == '+')
        return operand1 + operand2;
    else if (operation == '-')
        return operand1 - operand2;
    else if (operation == '*')
        return operand1 * operand2;

    return operand1 / operand2;
}

int solvePrefixExpression(const std::string prefixExpression)
{
    int result = 0;
    ArrayStack<int> operandStack;
    std::vector<std::string> tokenList;

    tokenList = tokenize(prefixExpression);

    for (auto i = tokenList.rbegin(); i != tokenList.rend(); i++)
    {
        if (isdigit((*i)[0]))
        {
            operandStack.push(std::stoi(*i));
        }
        else
        {
            int operand2 = operandStack.top();
            operandStack.pop();
            int operand1 = operandStack.top();
            operandStack.pop();

            result = doMath(operand1, operand2,
                (*i)[0]); operandStack.push(result);
        }
    }
}

```

```

    }

    return result;
}

int main()
{
    std::cout << solvePrefixExpression("-*+2 3 5 * 2 3 2");

    return 0;
}

```

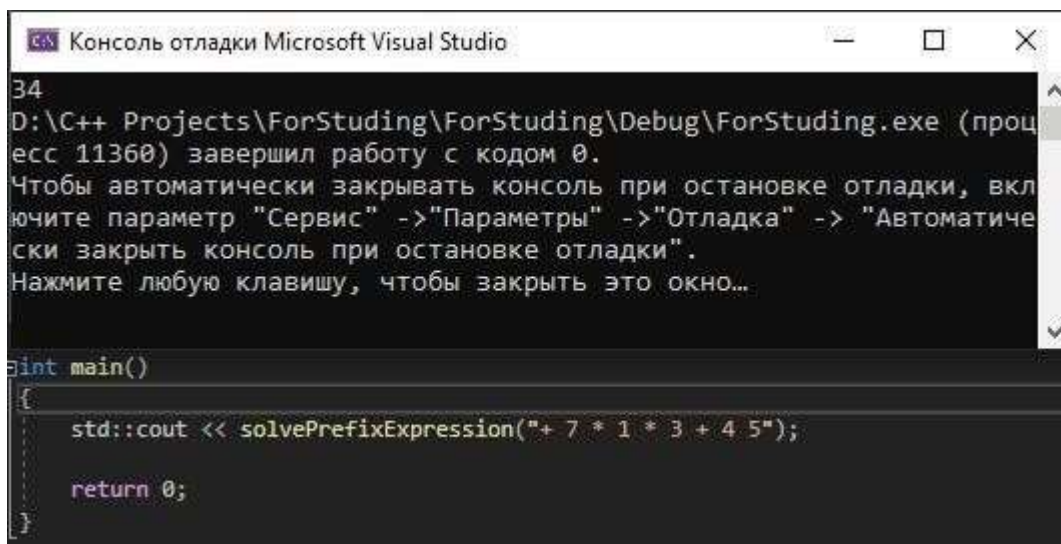


Рис. 2 Результат тестирования алгоритма

2.3)

Постановка задачи: Разработать программу сложения двух больших целых чисел (не попадающих в диапазон стандартных типов), вводимых с клавиатуры, как последовательность символов.

Описание подхода решения задачи: алгоритм принимает на вход два числа в строковом формате, поочерёдно складывает цифры двух строк, записывая результат в третью строку; если одна строка оказывается длиннее второй, то к результирующей строке добавляются все оставшиеся цифры; все операции сложения идут с учётом переполнения.

Алгоритм на псевдокоде

```

def alternativeSumming(a, b):
    c = ""
    ai = a.rbegin(), bi = b.rbegin()
    carry = 0
    result = 0

    for ai in a and bi in b:
        result = (ai - 48) + (bi - 48) + carry
        carry = result / 10
        result = result % 10
        c.push_back(result + 48)

```



```

for ai in a:
    result = (ai - 48) + carry
    carry = result / 10
    result = result % 10
    c.push_back(result + 48)

for bi in b:
    result = (bi - 48) + carry
    carry = result / 10
    result = result % 10
    c.push_back(result + 48)

if (carry)
    c.push_back(carry + 48)

reverse(c)

return c

```

Описание всех используемых переменных в функции alternativeSumming:

- a – строковая переменная с числом
- b – строковая переменная с числом
- c – строковая переменная с результирующим числом
- carry – целочисленная переменная, хранящая переполнение
- result – целочисленная переменная, хранящая результат операции
- ai – итератор по a
- bi – итератор по b

Код алгоритма

main.cpp

```

#include <iostream>
#include <stdlib.h>
#include <vector>
#include <string>

std::string alternativeSumming(const std::string& a, const std::string& b)
{
    std::string c;
    auto ai = a.rbegin(), bi = b.rbegin();
    char carry = 0;
    char result = 0;

    for (; ai != a.rend() && bi != b.rend(); ++ai, ++bi)
    {
        result = (*ai - 48) + (*bi - 48) +
            (int)carry; carry = result / 10;
        result %= 10;
        c.push_back(result + 48);
    }

    for (; ai != a.rend(); ++ai)
    {
        result = (*ai - 48) + carry;
        carry = result / 10;
        result %= 10;
        c.push_back(result + 48);
    }
}

```

```

for (; bi != b.rend(); ++bi)
{
    result = (*bi - 48) + carry;
    carry = result / 10;
    result %= 10;
    c.push_back(result + 48);
}

if (carry)
    c.push_back(carry + 48);

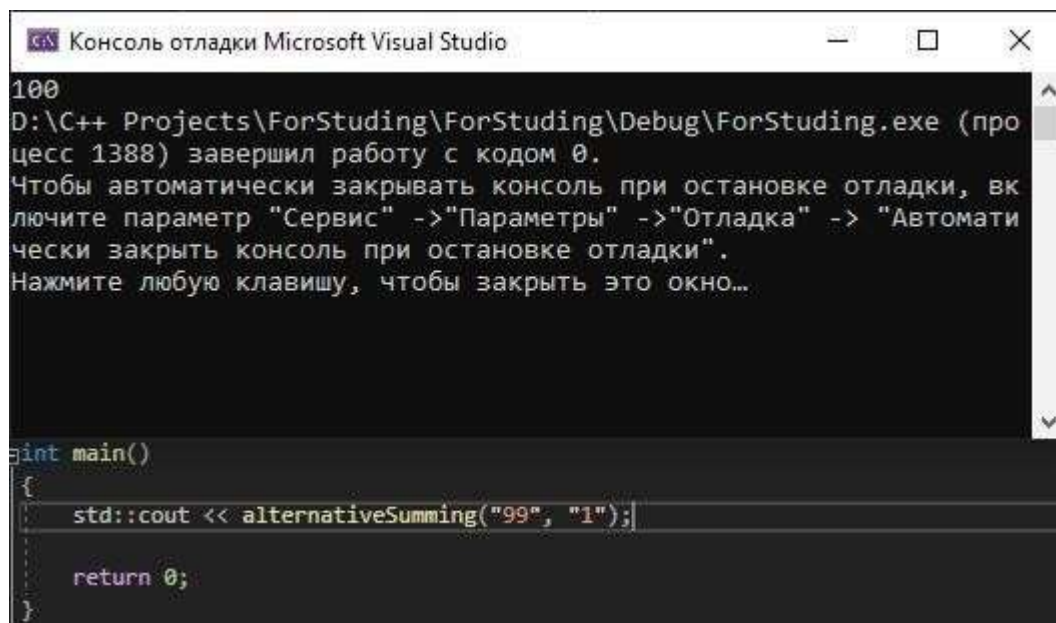
std::reverse(c.begin(), c.end());

return c;
}

int main()
{
    std::cout << alternativeSumming("99", "1");

    return 0;
}

```



Консоль отладки Microsoft Visual Studio

```

100
D:\C++ Projects\ForStuding\ForStuding\Debug\ForStuding.exe (процесс 1388) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

int main()
{
    std::cout << alternativeSumming("99", "1");

    return 0;
}

```

Рис. 3 Результат работы алгоритма

Вывод: я закрепил навыки вычисления префиксных и постфиксных выражений Информационные источники:

1. https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C
2. https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C