



МИНОБРНАУКИ РОССИИ

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 выполнению индивидуального задания № 5

по дисциплине «Структуры и алгоритмы обработки данных»

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

Выполнил:

Студентка группы ИКБО-29-20

Хан Анастасия Александровна

Проверил:

Копылова А.В.

Москва 2021

СОДЕРЖАНИЕ

Отчет по заданию 1

Провести преобразование инфиксной записи выражения в префиксную нотацию, расписывая процесс по шагам	4
Представить постфиксную нотацию выражений	5
Представить префиксную нотацию выражений п.2	8
Провести вычисление значения выражения представленного в постфиксной форме, расписывая процесс по шагам.....	13

Отчет по заданию 2

Задание 1.....	14
Условие задачи.....	14
Постановка задачи.....	14
Описание используемых функций.. ..	14
Код реализации структуры данных на однонаправленном списке... ..	14
Код реализации структуры данных на динамическом массиве.....	15
Код всей программы (реализация на однонаправленном списке).....	15
Код всей программы (реализация на динамическом массиве).....	20
Результаты тестирования	23
Задание 2.....	24
Условие задачи.....	24
Постановка задачи.....	24
Описание используемых функций.. ..	24
Код реализации функций и основной программы.	25
Результаты тестирования	29

Задание 3.....	31
Условие задачи.....	31
Постановка задачи.....	31
Описание используемых функций.. ..	31
Код реализации функций и основной программы.	32
Результаты тестирования.....	35
Выводы	36
Информационные источники	36

Отчет по заданию 1

Выполнить упражнения и представить процесс их выполнения

Персональный вариант: $29\%3 + 1 = 3$

1. Провести преобразование инфиксной записи выражения в префиксную нотацию, расписывая процесс по шагам.

Условие: $S = a + (b - c * k) - d * e - f$

Таблица 1

Стек операндов	a	a	a	b a	b a	c b a	k c b a
Стек операций		+	(+	(+	- (+	- (+	* - (+
Комментарий							<p>Далее следует $)$, значит, из стека операций извлекаются и записываются два верхних элемента: ck</p> <p>Из стека операндов извлекается $*$: $*ck$. Данный операнд заносится в стек операндов, далее опять извлекается два верхних операнда и операция: $-b*kc$</p> <p>Далее следует $)$, извлечение прекращается</p>

Продолжение таблицы 1

Стек операндов	$-b*kc$ a	d $+a-b*kc$	d $+a-b*kc$	e d $+a-b*kc$
Стек операций	+	-	* -	* -
Комментарий	<p>Далее идет $-$, в вершине стека $+$, две равных по приоритету операции, Извлекаем из стека $+$, помещаем $-$ в стек операций</p>			<p>Далее следует $-$, по приоритету меньше, чем $*$ в вершине стека Извлекаем $*$, а $-$ помещаем в стек</p>

Продолжение таблицы 1

Стек операндов	f *ed +a-b*kc	-*edf +a-b*kc	-+a-b*kc-*edf
Стек операций	- -	-	
Комментарий	Операции и операнды в исходном выражении закончились. Извлекаем из стека -, получаем -*edf	Извлекаем последнюю операцию и записываем результат	

Результат: $S_{\text{префиксное}} = -+a-b*kc-*edf$

2. Представить постфиксную нотацию выражений.

Условие: $S = a + (c-b)/(b*d)$

Таблица 2

Постфиксная запись	a	acb	acb-	acb-	acb-	acb-bd	acb-bd*	acb-bd*/+
Стек операций	(+	- (+	+	/ +	(/ +	* (/ +	/ +	
Комментарий		Далее идет), извлекаем все операции до (Далее идет /, + ниже по приоритету, чем деление, помещаем / в стек			Далее идет), извлекаем из стека все операции, стоящие до (Извлекаем из стека все оставшиеся операции и записываем в ответ	

Результат: $S_{\text{постфиксное}} = acb-bd*/+$

$$\text{Условие: } S = (a+b)*c-(d+e*f/((g/h+i-j)*k))/r$$

Таблица 3

Постфиксная запись	a	ab	ab+c	ab+c*
Стек операций	(+ (*	(-
Комментарий		Далее идет), извлекаем все операции до (Далее идет -, минус ниже по приоритету, чем умножение, помещаем - в стек, * извлекаем	

Продолжение таблицы 3

Постфиксная запись	ab+c*d	ab+c*de	ab+c*de	ab+c*de*
Стек операций	+ (-	* + (-	* + (-	/ + (-
Комментарий			Далее идет операция деления, она одинакова по приоритету с умножением, которое находится в вершине стека. Извлекаем * и вносим / в стек	

Продолжение таблицы 3

Постфиксная запись	$ab+c*de*g$	$ab+c*de*g$	$ab+c*de*gh$	$ab+c*de*gh/i$
Стек операций	((+ (-	/ ((+ (-	/ ((+ (-	+ ((+ (-
Комментарий			Далее идет +, у плюса меньший приоритет, чем у /, поэтому извлекаем /, а + заносим в стек	Далее идет -, у минуса одинаковый приоритет с +, поэтому + извлекаем из стека, а – помещаем в стек

Продолжение таблицы 3

Постфиксная запись	$ab+c*de*gh/i+j$	$ab+c*de*gh/i+j-k$	$ab+c*de*gh/i+j-k*$
Стек операций	- ((+ (-	* (+ (-	(+ (-
Комментарий	Далее идет). Извлекаем все операции до первой встретившейся ((Далее идет). Извлекаем все операции до первой встретившейся ((Далее идет). Извлекаем все операции до первой встретившейся (. Потом опять стоит), извлекаем все операции до первой встретившейся))

Продолжение таблицы 3

Постфиксная запись	$ab+c*de*gh/i+j-k*/+r$	$ab+c*de*gh/i+j-k*/+r/-$
Стек операций	/	-
Комментарий	Конец строки, извлекаем все операции из стека	

Результат: $S_{\text{постфиксное}} = ab+c*de*gh/i+j-k*/+r/-$

Условие: $S = (a+b)*c-(d+e*f(((g/h+i-j)*k)))/r$

3. Представить префиксную нотацию выражений п.2

Условие: $S = a + (c-b)/(b*d)$

Таблица 4

Стек операндов	a	a	a	c	c	b	-cb	-cb
Стек операций		+	((-	c	a	a
Комментарий			+	+	(a	+	+
					+			
						Далее идет), извлекаем b, a, - из стека операций, (извлекаем из стека операций тоже, записываем: -cb Результат помещаем в стек операндов		

Продолжение таблицы 4

Стек операндов	-cb a	b -cb a	b -cb a	d b -cb a	*bd -cb a
Стек операций	(/ +	(/ +	* (/ +	* (/ +	/ +
Комментарий				Далее идет), извлекаем d, b, операцию *, (извлекаем из стека тоже, записываем: *bd Результат помещаем в стек операндов	Строка инфиксная закончилась. Извлекаем операцию / из стека операций, операнды *bd, -cb, записываем: /-cb*bd Результат помещаем в стек операндов

Продолжение таблицы 4

Стек операндов	/-cb*bd a		+a/-cb*bd
Стек операций	+		
Комментарий	Извлекаем операцию + из стека операций, операнды /- cb*bd , a, записываем: +a/-cb*bd Результат помещаем в стек операндов		Стека операций пуст, инфиксная строка тоже, значит, в стеке операндов записана искомая префиксная строка

Результат: $S_{\text{префиксное}} = +a/-cb*bd$

$$\text{Условие: } S = (a+b)*c-(d+e*f/((g/h+i-j)*k))/r$$

Таблица 5

Стек операндов		a	a	b a	+ab	c +ab	*+abc
Стек операций	((+	+	*	*	(-
Комментарий				Далее), извлекаем b, a, операцию +, (извлекаем из стека тоже, записываем: +ab Результат помещаем в стек операндов		Далее идет операция -, она ниже по приоритету, чем *, извлекаем из стека c, +ab, из стека операций извлекаем * и помещаем туда – Записываем: *+abc Результат помещаем в стек операндов	

Продолжение таблицы 5

Стек операндов	*+abc	d *+abc	d *+abc	e d *+abc	e d *+abc	f e d *+abc
Стек операций	(-	(-	+	+	*	*
			(-	(-	(-	+
						(-
Комментарий						Далее идет /, имеет тот же приоритет, что и * в вершине стека. Извлекаем из стека операндов f, e, из стека операций * и помещаем туда /. Записываем: *ef Результат помещаем в стек операндов

Продолжение таблицы 5

Стек операндов	*ef d *+abc	*ef d *+abc	g *ef d *+abc	h g *ef d *+abc	i /gh *ef d *+abc
Стек операций	(/ + (-	((/ + (-	/ ((/ + (-	/ ((/ + (-	+ ((/ + (-
Комментарий				Далее идет +, ниже по приоритету, чем / в вершине стека. Извлекаем из стека операндов h, g, из стека операций / и помещаем туда +. Записываем: /gh Результат помещаем в стек операндов	Далее идет -, имеет тот же приоритет, что и + в вершине стека. Извлекаем из стека операндов i, /gh, из стека операций + и помещаем туда -. Записываем: -/ghi Результат помещаем в стек операндов

Продолжение таблицы 5

Стек операндов	j +/ghi *ef d *+abc	-+/ghij *ef d *+abc	k /*ef-+/ghij d *+abc
Стек операций	- ((/ + (-	/ (+ (-	* (+ (-
Комментарий	Далее идет), извлекаем j, +/ghi, операцию -, (извлекаем из стека тоже, записываем: - +/ghij Результат помещаем в стек операндов	Далее идет *, имеет тот же приоритет, что и / в вершине стека. Извлекаем из стека операндов - +/ghij, *ef, из стека операций / и помещаем туда *. Записываем: /*ef-+/ghij Результат помещаем в стек операндов	Далее идет), извлекаем k, /*ef-+/ghij, операцию *, (извлекаем из стека тоже, записываем: /*ef-+/ghijk Результат помещаем в стек операндов

Продолжение таблицы 5

Стек операндов	/*ef-+/ghijk d *+abc	+d/*ef-+/ghijk *+abc	r +d/*ef-+/ghijk *+abc
Стек операций	+ (-	/ -	/ -
Комментарий	Далее идет), извлекаем /*ef- +/ghijk, d, операцию +, (извлекаем из стека тоже, записываем: +d/*ef-+/ghijk Результат помещаем в стек операндов		Инфиксная строка кончилась. Извлекаем r, +d/*ef- +/ghijk, операцию /, записываем: /+d/*ef-+/ghijk Результат помещаем в стек операндов

Продолжение таблицы 5

Стек операндов	$/+d*/*ef-/ghijkr$ $*+abc$	$-*+abc /+d*/*ef-/ghijkr$
Стек операций	-	
Комментарий	Извлекаем $/+d*/*ef-/ghijkr$, $*+abc$, операцию -, записываем: $-*+abc /+d*/*ef-/ghijkr$ Результат помещаем в стек операндов	Стека операций пуст, инфиксная строка тоже, значит, в стеке операндов записана искомая префиксная строка

Результат: $S_{\text{префиксное}} = -*+abc /+d*/*ef-/ghijkr$

4. Провести вычисление значения выражения представленного в постфиксной форме, расписывая процесс по шагам

Условие: $S_{\text{постфиксное}} = 7\ 2 - 3\ 23 + *$

Начинаем:

Заносим в стек операндов 7: (7)

Заносим в стек операндов 2: (2 7)

Встретили -: Извлекаем 2, 7, выполняем: $7 - 2 = 5$. Заносим в стек операндов: (5)

Заносим в стек операндов 3: (3 5)

Заносим в стек операндов 23: (23 3 5)

Встретили +: Извлекаем 23, 3, выполняем: $3 + 23 = 26$. Заносим в стек операндов: (26 5)

Встретили *: Извлекаем 26, 5, выполняем: $5 * 26 = 130$. Заносим в стек операндов: (130)

Строка пуста

Ответ: $S = 130$

Отчет по заданию 2

Выполнить программную реализацию задач в соответствии с вариантом.

1. Задание 1.

Условие задачи

Реализовать операции стек: на массиве и однонаправленном списке.

Постановка задачи.

Разработать структуру данных стек(на однонаправленном списке и динамическом массиве), реализовать основные операции с ним, функцию вычисления значения выражения, представленного в постфиксной форме.

Описание используемых функций.

void pop(Stack* &s) – удаляет элемент из стека.

void push(Stack* &s, string symb) – заносит элемент в стек.

bool isEmpty(Stack *s) – проверяет стек на наличие элементов. Если стек пуст, то возвращает true, в противном случае false.

string top(Stack* s) – возвращает верхний элемент в стеке

checkSymb(string a) проверяет, является символ a операцией, если является, то возвращает true, иначе false.

void inputExpression(Stack* &s) считывает постфиксное выражение, заносит в стек s, ввод заканчивается, подается ключевое слово end.

int operation(int a, int b, char p) по символу p выполняет операцию между a и b и возвращает результат.

string to_string(int res) – возвращает строковое представление числа.

void printStack(Stack* s) – выводит стек.

int computePost(Stack* &s) – вычисляет значение постфиксного выражения. Пока стек s не пуст выполняются действия с его элементами. Если текущий символ не является операцией, то он заносится в стек операндов. Если текущий символ – операция, то из стека операндов извлекаются два верхних операнда, между ними происходит вычисление выражения, результат приводится к строковому виду и заносится в стек операндов.

Код реализации структуры данных на однонаправленном списке.

```
struct Stack
{
    string a;
    Stack* next;
```

```
};
```

Код реализации структуры данных на динамическом массиве.

```
struct Stack
{
    vector<string> a = {};
};
```

Код реализации функции вычисления значения постфиксного выражения.

```
int computePost(Stack* &s)
{
    int res = 0;
    string a, b, oper;
    Stack* w1 = 0;
    while (!isEmpty(s))
    {
        if (checkSymb(top(s)))
        {
            b = top(w1);
            pop(w1);
            a = top(w1);
            pop(w1);
            oper = top(s);
            res = operation(atoi(a.c_str()), atoi(b.c_str()), oper[0]);
            push(w1, to_string(res));
        }
        else
        {
            push(w1, top(s));
        }
        pop(s);
    }
    return res;
}
```

Код всей программы (реализация на однонаправленном списке).

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
```

```

using namespace std;
string masOper[] = {"+", "-", "/", "*"};
struct Stack
{
    string a;
    Stack* next;
};
string to_string(int res)
{
    bool neg = false;
    string ans;
    char d;
    res == 0? ans = "0": ans = "";
    if (res < 0)
    {
        neg = true;
        res*=-1;
    }
    while (res != 0)
    {
        d = 48 + res % 10;
        res /= 10;
        ans = d + ans;
    }
    if (neg == true)
    {
        ans = '-' + ans;
    }
    return ans;
}
bool checkSymb(string a)
{
    bool ans = false;
    for (int i = 0 ; i < 4; i++)
    {
        if (masOper[i] == a)
        {
            ans = true;
            break;
        }
    }
    return ans;
}
int operation(int a, int b, char p)

```



```

{
    int res = 0;
    switch (p)
    {
        case '+':
        {
            res = a + b;
            break;
        }
        case '-':
        {
            res = a - b;
            break;
        }
        case '*':
        {
            res = a * b;
            break;
        }
        case '/':
        {
            res = a / b;
            break;
        }
    }
    return res;
}

void makeEmpty(Stack* &s)
{
    Stack* el;
    while (s != 0)
    {
        el = s;
        s = s->next;
        delete el;
    }
}

bool isEmpty(Stack *s)
{
    bool cond;
    s == 0 ? cond = true: cond = false;
    return cond;
}

string top(Stack* s)

```

```

{
    return s->a;
}
void pop(Stack* &s)
{
    Stack* a = s;
    s = s->next;
    delete a;
}
void push(Stack* &s, string symb)
{
    if (s != 0)
    {
        Stack* newEl = new Stack;
        newEl->a = s->a;
        newEl->next = s->next;
        s->a = symb;
        s->next = newEl;
    }
    else
    {
        s = new Stack;
        s->a = symb;
        s->next = 0;
    }
}
void printStack(Stack* s)
{
    while (s != 0)
    {
        cout << s->a << ' ';
        s = s->next;
    }
}
int computePost(Stack* &s)
{
    int res = 0;
    string a, b, oper;
    Stack* w1 = 0;
    while (!isEmpty(s))
    {
        if (checkSymb(top(s)))
        {
            b = top(w1);

```

```

        pop(w1);
        a = top(w1);
        pop(w1);
        oper = top(s);
        res = operation(atoi(a.c_str()), atoi(b.c_str()), oper[0]);
        push(w1, to_string(res));
    }
    else
    {
        push(w1, top(s));
    }
    pop(s);
}
return res;
}
void inputExpression(Stack* &s)
{
    vector<string> expr;
    string symb;
    cout << "Ввод выражения прекращается, когда вводится end." <<
endl;
    cin >> symb;
    while (symb != "end")
    {
        expr.push_back(symb);
        cin >> symb;
    }
    for (int i = expr.size() - 1; i >= 0; i--)
    {
        push(s, expr[i]);
    }
}
int main()
{
    setlocale(LC_ALL, "Russian");
    Stack* stack = 0;
    inputExpression(stack);
    cout << computePost(stack);
}

```

Код всей программы (реализация на динамическом массиве).

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
using namespace std;
string masOper[] = {"+", "-", "/", "*"};
struct Stack
{
    vector<string> a = { };
};
string to_string(int res)
{
    bool neg = false;
    string ans;
    char d;
    res == 0? ans = "0": ans = "";
    if (res < 0)
    {
        neg = true;
        res*=-1;
    }
    while (res != 0)
    {
        d = 48 + res % 10;
        res /= 10;
        ans = d + ans;
    }
    if (neg == true)
    {
        ans = '-' + ans;
    }
    return ans;
}
bool checkSymb(string a)
{
    bool ans = false;
    for (int i = 0 ; i < 4; i++)
    {
        if (masOper[i] == a)
        {
            ans = true;
            break;
        }
    }
}
```

```

        }
    }
    return ans;
}
int operation(int a, int b, char p)
{
    int res = 0;
    switch (p)
    {
        case '+':
        {
            res = a + b;
            break;
        }
        case '-':
        {
            res = a - b;
            break;
        }
        case '*':
        {
            res = a * b;
            break;
        }
        case '/':
        {
            res = a / b;
            break;
        }
    }
    return res;
}
void makeEmpty(Stack* &s)
{
    s->a.clear();
}
bool isEmpty(Stack *s)
{
    bool cond;
    s->a.size() == 0 ? cond = true: cond = false;
    return cond;
}
string top(Stack* s)
{

```

```

        return s->a.back();
    }
void pop(Stack* &s)
{
    s->a.pop_back();
}
void push(Stack* &s, string symb)
{
    if (s == 0)
    {
        s = new Stack;
    }
    s->a.push_back(symb);
}
void printStack(Stack* s)
{
    for (int i = 0; i < s->a.size(); i++)
    {
        cout << s->a[i] << ' ';
    }
}
int computePost(Stack* &s)
{
    int res = 0;
    string a, b, oper;
    Stack* w1 = 0;
    while (!isEmpty(s))
    {
        if (checkSymb(top(s)))
        {
            b = top(w1);
            pop(w1);
            a = top(w1);
            pop(w1);
            oper = top(s);
            res = operation(atoi(a.c_str()), atoi(b.c_str()), oper[0]);
            push(w1, to_string(res));
        }
        else
        {
            push(w1, top(s));
        }
        pop(s);
    }
}

```

```

    }
    return res;
}
void inputExpression(Stack* &s)
{
    vector<string> expr;
    string symb;
    cout << "Ввод выражения прекращается, когда вводится end." <<
endl;
    cin >> symb;
    while (symb != "end")
    {
        expr.push_back(symb);
        cin >> symb;
    }
    for (int i = expr.size() - 1; i >= 0; i--)
    {
        push(s, expr[i]);
    }
}
int main()
{
    setlocale(LC_ALL, "Russian");
    Stack* stack = 0;
    inputExpression(stack);
    cout << computePost(stack);
}

```

Результаты тестирования.

Скриншот слева – однонаправленный список, справа – динамический массив.

Пример из своего варианта:

Входные данные: 7 2 – 3 23 + * end

Ожидалось: 130

Результат работы программы:

```

Ввод выражения прекращается, когда вводится end.
7 2 - 3 23 + * end
130
-----
Process exited with return value 0
Press any key to continue . . .

```

```

Ввод выражения прекращается, когда вводится end.
7 2 - 3 23 + * end
130
-----
Process exited with return value 0
Press any key to continue . . .

```

Входные данные: 80 3 * 40 – 40 / 5 - end

Ожидалось: 0

Результат работы программы: 0

```
Ввод выражения прекращается, когда вводится end.
80 3 * 40 - 40 / 5 - end
0
-----
Process exited with return value 0
Press any key to continue . . .
```

```
Ввод выражения прекращается, когда вводится end.
80 3 * 40 - 40 / 5 - end
0
-----
Process exited with return value 0
Press any key to continue . . .
```

Входные данные: 1 1 – 35 7 / * 2 5 + + end

Ожидалось: 7

Результат работы программы: 7

```
Ввод выражения прекращается, когда вводится end.
1 1 - 35 7 / * 2 5 + + end
7
-----
Process exited with return value 0
Press any key to continue . . .
```

```
Ввод выражения прекращается, когда вводится end.
1 1 - 35 7 / * 2 5 + + end
7
-----
Process exited with return value 0
Press any key to continue . . .
```

2. Задание 2

Условие задачи

Вычислить значение выражения, представленного в префиксной форме.

Постановка задачи.

Разработать функцию вычисления значения выражения, представленного в префиксной форме.

Описание используемых функций.

Структура данных, которая будет использоваться – очередь. К стандартным функциям – операциям с очередью добавится несколько функций: Функция, дающая ответ, является символ операцией или нет, функция, выбирающая по операции, какое действие сделать с числами, функция, которая считывает префиксное выражение и функция, в которой будет происходить вычисление значения выражения.

Функция `bool checkSymb(string a)` проверяет, является символ `a` операцией, если является, то возвращает истину, иначе ложь.

Функция `void inputExpression(Queue* &q)` считывает префиксное выражение в очередь `q`, когда ввод заканчивается, подается ключевое слово `end`.

Функция `int operation(int a, int b, char p)` по символу `p` выполняет операцию между `a` и `b` и возвращает результат.

Функция `int computePre(Queue* &q)` – функция, в которой происходит вычисление значения префиксного выражения. Пока очередь не пуста, выполняются вычисления значений операций между операндами. Если перед текущей операцией стояла тоже операция, то заносим предыдущую операцию в конец очереди; если перед операцией стоит один операнд, то помещаем его тоже в конец очереди. Если же перед операндом был операнд, то вычисляем

значение выражения и помещаем в конец очереди. Когда в очереди не будет элементов, то функция возвращает значение, находящееся в переменной ответа res.

Код реализации функций и основной программы.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
string masOper[] = {"+", "-", "/", "*"};
```

```
struct Queue
{
    string a;
    Queue* last;
};
```

```
void pop(Queue* &q)
{
    Queue* a = q;
    q = q->last;
    delete a;
}
```

```
bool isEmpty(Queue* q)
{
    bool t;
    q == 0? t = true: t = false;
    return t;
}
```

```
string top(Queue* q)
{
    return q->a;
}
```

```
Queue* getLast(Queue* q)
{
    Queue* elem = q;
    while (elem->last != 0)
    {
        elem = elem->last;
    }
}
```

```

    }
    return elem;
}

void push(Queue* &q, string symb)
{
    if (q == 0)
    {
        q = new Queue;
        q->a = symb;
        q->last = 0;
    }
    else
    {
        Queue* lastEl = getLast(q);
        Queue* newEl = new Queue;
        newEl->a = symb;
        newEl->last = 0;
        lastEl->last = newEl;
    }
}

void printQueue(Queue* &queue)
{
    Queue* q = queue;
    while (q != 0)
    {
        cout << q->a << ' ';
        q = q->last;
    }
    cout << endl;
}

void inputExpression(Queue* &q)
{
    string symb;
    cout << "Ввод выражения прекращается, когда вводится end." <<
endl;
    cin >> symb;
    while (symb != "end")
    {
        push(q, symb);
        cin >> symb;
    }
}

```

```

}

string to_string(int res)
{
    bool neg = false;
    string ans;
    char d;
    res == 0? ans = "0": ans = "";
    if (res < 0)
    {
        neg = true;
        res*=-1;
    }
    while (res != 0)
    {
        d = 48 + res % 10;
        res /= 10;
        ans = d + ans;
    }
    if (neg == true)
    {
        ans = '-' + ans;
    }
    return ans;
}

bool checkSymb(string a)
{
    bool ans = false;
    for (int i = 0 ; i < 4; i++)
    {
        if (masOper[i] == a)
        {
            ans = true;
            break;
        }
    }
    return ans;
}

int operation(int a, int b, char p)
{
    int res = 0;
    switch (p)
    {
        case '+':

```

```

        {
            res = a + b;
            break;
        }
    case '-':
    {
        res = a - b;
        break;
    }
    case '*':
    {
        res = a * b;
        break;
    }
    case '/':
    {
        res = a / b;
        break;
    }
}
return res;
}

int computePre(Queue* &q)
{
    int res;
    string lastEl = top(q);
    pop(q);
    while (!isEmpty(q))
    {
        if (!checkSymb(lastEl) && checkSymb(top(q)) )
        {
            push(q, lastEl);
            lastEl = top(q);
            pop(q);
        }
        else if (!checkSymb(lastEl) && !checkSymb(top(q)) )
        {
            push(q, lastEl);
            lastEl = top(q);
            pop(q);
        }
        else if (checkSymb(lastEl) && checkSymb(top(q)))
        {

```

```

        push(q, lastEl);
        lastEl = top(q);
        pop(q);
    }
    else if (checkSymb(lastEl) && !checkSymb(top(q)) &&
checkSymb(top(q->last)))
    {
        push(q, lastEl);
        lastEl = top(q);
        pop(q);
    }
    else
    {
        res = operation(atoi(top(q).c_str()),      atoi(top(q-
>last).c_str()), lastEl[0]);
        pop(q);
        lastEl = to_string(res);
        pop(q);
    }
}
return res;
}
int main()
{
    setlocale(LC_ALL, "Russian");
    Queue* queue = 0;
    inputExpression(queue);
    cout << endl << computePre(queue);
}

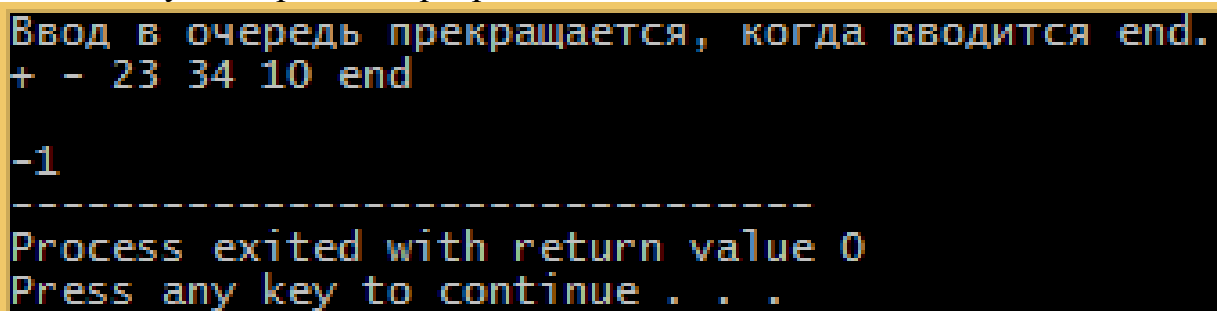
```

Результаты тестирования функции

Входные данные: + - 23 34 10 end

Ожидалось: -1

Результат работы программы: -1



```

Ввод в очередь прекращается, когда вводится end.
+ - 23 34 10 end

-1
-----
Process exited with return value 0
Press any key to continue . . .

```

Входные данные: + - + - 9 8 7 6 5 end

Ожидалось: 7

Результат работы программы: 7

```
Ввод в очередь прекращается, когда вводится end.  
+ - + - 9 8 7 6 5 end  
  
7  
-----  
Process exited with return value 0  
Press any key to continue . . .
```

Входные данные: * / 35 7 - + * 1 2 3 4 end

Ожидалось: 5

Результат работы программы: 5

```
Ввод в очередь прекращается, когда вводится end.  
* / 35 7 - + * 1 2 3 4 end  
  
5  
-----  
Process exited with return value 0  
Press any key to continue . . .
```

Пример из практического занятия №3:

Ожидалось: -31 (в лекции 31, но там опечатка, $(-46) + 15 = -31$)

Результат работы программы: -31

```
Ввод в очередь прекращается, когда вводится end.  
+ - 2 48 * / 12 4 5 end  
  
-31  
-----  
Process exited with return value 0  
Press any key to continue . . .
```

3. Задание 3

Условие задачи.

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

Постановка задачи.

Необходимо написать функцию, которая принимает на вход две строки – два числа – и кладет их в стеки. Далее посимвольно вычисляет их сумму и возвращает результат.

Описание используемых функций.

Для вычисления будем использовать структуру данных стек, с информационной частью типа `string`, реализованной на однонаправленном списке.

Функция (`string computeSumBigNumbers(string n1, string n2)`) получает на вход два числа как две строки. Далее каждая из них посимвольно заносится в стек слева направо, вследствие чего в стеках, в которые они были записаны, находятся искомые числа как последовательность символов, расположенные в обратном порядке.

Далее, пока два стека не являются пустыми, выполняется сложение соответствующих цифр числа и переменной, хранящей единицу переноса (изначально имеет значение 0) (осуществляется преобразование строк в числа и производится сложение). В переменную единицы переноса заносится целая часть от целочисленного деления суммы цифр на 10. Остаток суммы от деления на 10 преобразуется в строку и заносится слева в результат сложения чисел.

Далее, у нас два варианта: либо оба стека пусты, либо один не пуст. Проверяем, не остался ли какой-нибудь из двух стеков непустым. Если какой из стеков непустой, то вызывается функция `finishPerenos` (сделана, чтобы уменьшить количество кода). Если же оба стека пусты и есть единица переноса, то перед строкой результата добавляется цифра 1.

Функция возвращает окончательный результат сложения двух чисел в строковом формате.

Функция `string finishPerenos(Stack* &n, string res, int perenos)` получает на вход непустой стек – число, которое не до конца было сложено, результат сложения двух чисел (неполный), единицу переноса. Пока стек не пуст, выполняется сложение числа из стека и единицы переноса. В переменную единицы переноса заносится целая часть от целочисленного деления суммы цифр на 10. Остаток суммы от деления на 10 преобразуется в строку и заносится слева в результат сложения чисел.

По окончании сложения, если единица переноса осталась, то перед строкой результата добавляется цифра 1. Функция возвращает окончательный результат сложения.

Код реализации функций и основной программы

```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

```

using namespace std;
struct Stack
{
    string a;
    Stack* next;
};
void makeEmpty(Stack* &s)
{
    Stack* el;
    while (s != 0)
    {
        el = s;
        s = s->next;
        delete el;
    }
}
bool isEmpty(Stack *s)
{
    bool cond;
    s == 0 ? cond = true: cond = false;
    return cond;
}
string top(Stack* s)
{
    return s->a;
}
void pop(Stack* &s)
{
    Stack* a = s;
    s = s->next;
    delete a;
}
void push(Stack* &s, char symb)
{
    if (s != 0)
    {
        Stack* newEl = new Stack;
        newEl->a = s->a;
        newEl->next = s->next;
        s->a = symb;
        s->next = newEl;
    }
    else
    {

```



```

        s = new Stack;
        s->a = symb;
        s->next = 0;
    }
}
string to_string(int res)
{
    bool neg = false;
    string ans;
    char d;
    res == 0? ans = "0": ans = "";
    if (res < 0)
    {
        neg = true;
        res*=-1;
    }
    while (res != 0)
    {
        d = 48 + res % 10;
        res /= 10;
        ans = d + ans;
    }
    if (neg == true)
    {
        ans = '-' + ans;
    }
    return ans;
}
string finishPerenos(Stack* &n, string res, int perenos)
{
    int sum;
    while (!isEmpty(n))
    {
        sum = atoi(top(n).c_str()) + perenos;
        pop(n);
        perenos = sum / 10;
        res = to_string(sum % 10) + res;
    }
    if (perenos == 1)
    {
        res = "1" + res;
    }
    return res;
}

```

```

string computeSumBigNumbers(string n1, string n2)
{
    string res = "";
    Stack *num1 = 0, *num2 = 0;
    for (int i = 0; i < n1.size(); i++)
    {
        push(num1, n1[i]);
    }
    for (int i = 0; i < n2.size(); i++)
    {
        push(num2, n2[i]);
    }
    int perenos = 0;
    int sum, d1, d2;
    string d;
    while (!isEmpty(num1) && !isEmpty(num2))
    {
        d1 = atoi(top(num1).c_str());
        d2 = atoi(top(num2).c_str());
        pop(num1);
        pop(num2);
        sum = d1 + d2 + perenos;
        perenos = sum / 10;
        d = to_string(sum % 10);
        res = d + res;
    }
    if (!isEmpty(num1))
    {
        res = finishPerenos(num1, res, perenos);
    }
    else if (!isEmpty(num2))
    {
        res = finishPerenos(num2, res, perenos);
    }
    else if (perenos == 1)
    {
        res = "1" + res;
    }
    return res;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    Stack* stack = 0;

```

```

        string    bigNumber1[]    =    {"27",    "200",    "9999",    "10000",
"99999999999999999999",
        "11111111111111111111111111111111",
"12345678978675645324243"};
        string    bigNumber2[]    =    {"3",    "200",    "1",    "9000000000",
"99999999999999999999",
        "11111111111111111111111111111111",
"9954678976543214565463535"};
        for (int i = 0; i < 7; i++)
        {
                cout << bigNumber1[i] << " + " << bigNumber2[i] << " = " <<
computeSumBigNumbers( bigNumber1[i], bigNumber2[i]) << endl;
        }
}

```

Результаты тестирования функции

```

27 + 3 = 30
200 + 200 = 400
9999 + 1 = 10000
10000 + 9000000000 = 9000100000
99999999999999999999 + 99999999999999999999 = 199999999999999999998
11111111111111111111111111111111 + 11111111111111111111111111111111 = 22222222222222222222222222222222
12345678978675645324243 + 9954678976543214565463535 = 9967024655521890210787778

-----
Process exited with return value 0
Press any key to continue . . .

```

Выводы

В ходе практической работы я изучила структуру данных стек, очередь, освоила основные операции со стеком и очередью, получила навыки ручного преобразования выражений в префиксные и постфиксные записи, программно реализовала вычисление значений выражений в постфиксной и префиксной формах, с помощью стека реализовала функцию сложения больших чисел, не входящих в рамки стандартного диапазона.

Информационные источники

- 1) Лекции по Структурам и алгоритмам обработки данных Скворцова Л. А. – МИРЭА – Российский технологический университет 2021 год
- 2) Искусство программирования Т. 1 / Д. Кнут – М.: Вильямс, 2014. – 712 с.