МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 8

Тема:

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

Выполнил студент Цемкало А. Р.
Фамилия И. О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

Цель. Получить знания и навыки по реализации структуры стек и очередь и их применения при решении задач.

Вариант 3.

Задание 1. Выполнить упражнения и представить процесс выполнения в отчете.

1. Преобразование инфиксной записи выражения в префиксную нотацию

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

Процесс выполнения:

Выражение a+(b-c*k)-d*e-f переворачивается в f-e*d-(k*c-b)+a, затем разворачивается по алгоритму постфиксной нотации с помощью стека и очереди.

Стек: -* Очередь: fed
Стек: - Очередь: fed*
Стек: --(* Очередь: fed*kc
Стек: --(Очередь: fed*kc*
Стек: --(Очередь: fed*kc*b
Стек: - Очередь: fed*kc*b--
Стек: + Очередь: fed*kc*b---

Очередь: fed*kc*b---a+

Затем выражение переворачивается: +a---b*ck*def

2. Преобразование инфиксной записи выражения в постфиксную нотацию

Условие упражнения: представить постфиксную нотацию выражений

1. a+(c-b)/(b*d)
2. (a+b)*c-(d+e*f/((g/h+i-j)*k))/r

Процесс выполнения:

a+(c-b)/(b*d)

Стек: +(- Очередь: acb
Стек: + Очередь: acb-
Стек: +/(* Очередь: acb-bd
Стек: +/(Очередь: acb-bd*
Очередь: acb-bd*/+

(a+b)*c-(d+e*f/((g/h+i-j)*k))/r

Стек: (+ Очередь: ab
Стек: (Очередь: ab+
Стек: (* Очередь: ab+c
Стек: (Очередь: ab+c*
Стек: (-(+*

Очередь: ab+c*def
Стек: (- Очередь: ab+c*def*+
Стек: (-/(/ Очередь: ab+c*def*+gh
Стек: (-/(Очередь: ab+c*def*+gh/
Стек: (-/(- Очередь: ab+c*def*+gh/i+j
Стек: (-/ Очередь: ab+c*def*+gh/i+j-
Стек: (Очередь: ab+c*def*+gh/i+j-/-
Стек: (* Очередь: ab+c*def*+gh/i+j-/-k
Стек: (Очередь: ab+c*def*+gh/i+j-/-k*
Очередь: ab+c*def*+gh/i+j-/-k*r/

3. Представить префиксную нотацию выражений п.2

Инфиксная форма: a+(c-b)/(b*d)

Префиксная форма: acb-bd*/+

Инфиксная форма: (a+b)*c-(d+e*f/((g/h+i-j)*k))/r

Префиксная форма: ab+c*def*+gh/i+j-/-k*r/

4. Провести вычисление значения выражения, представленного в постфиксной форме, расписывая процесс по шагам 7 2 – 3 2 +*

№ шага	Стек	Знак
1	7	
2	7 2	
3	5	-
4	5 3	
5	5 3 2	
6	5 3 2	
7	5 5	+
8	25	*

Код программы:

```
#include <iostream>
#include <queue>
#include <stack>
#include <string>
using namespace std;

void pop(stack <char>* St, queue <char>* Q) {
	for (int i = 0; i < (*St).size(); i++) {
		if ((*St).top() == '(') {
			(*St).pop();
			break;
		}
		(*Q).push((*St).top());
		(*St).pop();
	}
}

string reverse_expression(string expression) {
	string new_expression = "";
	for (int i = expression.size() - 1; i >= 0; i--) {
		if (expression[i] == '(') {
			new_expression += ')';
		}
		else if (expression[i] == ')') {
			new_expression += '(';
		}
		else {
			new_expression += expression[i];
		}
	}
	return new_expression;
}
```

```
string infix_to_postfix(string expression) {
	stack <char> St;
	queue <char> Q;
	for (int i = 0; i < expression.size(); i++) {
		if (expression[i] == '(') {
			St.push(expression[i]);
		}
		else if (expression[i] == ')') {
			pop(&St, &Q);
		}
		else if (expression[i] == '+' || expression[i] == '-') {
			if (St.empty() || St.top() == '(') {
				St.push(expression[i]);
			}
			else if (St.top() == '*' || St.top() == '/') {
				pop(&St, &Q);
				St.push(expression[i]);
			}
			else {
				Q.push(St.top());
				St.pop();
				St.push(expression[i]);
			}
		}
		else if (expression[i] == '*' || expression[i] == '/') {
			if (!St.empty() && (St.top() == '*' || St.top() == '/')) {
				pop(&St, &Q);
			}
			St.push(expression[i]);
		}
		else {
			Q.push(expression[i]);
		}
	}
	while (!St.empty()) {
		if (St.top() != '(') {
			Q.push(St.top());
		}
		St.pop();
	}
	string new_expression = "";
	while (!Q.empty()) {
		new_expression += Q.front();
		Q.pop();
	}
	return new_expression;
}
string infix_to_prefix(string expression) {
	expression = reverse_expression(expression);
	expression = infix_to_postfix(expression);
	expression = reverse_expression(expression);
	return expression;
}
int calculate_postfix(string expression) {
	stack <int> St;
	for (int i = 0; i < expression.size(); i++) {
		if (expression[i] == '+') {
			int a = St.top();
			St.pop();
			a += St.top();
			St.pop();
			St.push(a);
		}
		else if (expression[i] == '*') {
			int a = St.top();
```

```
                St.pop();
                a *= St.top();
                St.pop();
                St.push(a);
            }
            else if (expression[i] == '/') {
                int a = St.top();
                St.pop();
                a = St.top() / a;
                St.pop();
                St.push(a);
            }
            else if (expression[i] == '-') {
                int a = St.top();
                St.pop();
                a = St.top() - a;
                St.pop();
                St.push(a);
            }
            else {
                int a = static_cast<int>(expression[i]) - '0';
                St.push(a);
            }
        }
        return St.top();
}

int calculate_prefix(string expression) {
        stack <int> St;
        for (int i = expression.size() - 1; i >= 0; i--) {
            if (expression[i] == '+') {
                int a = St.top();
                St.pop();
                a += St.top();
                St.pop();
                St.push(a);
            }
            else if (expression[i] == '*') {
                int a = St.top();
                St.pop();
                a *= St.top();
                St.pop();
                St.push(a);
            }
            else if (expression[i] == '/') {
                int a = St.top();
                St.pop();
                a /= St.top();
                St.pop();
                St.push(a);
            }
            else if (expression[i] == '-') {
                int a = St.top();
                St.pop();
                a -= St.top();
                St.pop();
                St.push(a);
            }
            else {
                St.push(static_cast<int>(expression[i]) - '0');
            }
        }
        return St.top();
}
```

```
int main() {
	string expression_infix = "(a+b)*c-(d+e*f/((g/h+i-j)*k))/r";
	string expression_postfix;
	string expression_prefix;
	expression_postfix = infix_to_postfix(expression_infix);
	expression_prefix = infix_to_prefix(expression_infix);
	cout << expression_postfix << endl;
	cout << expression_prefix << endl;
	cout << calculate_postfix(expression_postfix) << endl;
	cout << calculate_prefix(expression_prefix);
	return 0;
}
```

Задание 2.

Задание 2. п.1

1.1. Условие задачи.

Создать класс или просто заголовочный файл с функциями. Применить операции для вычисления значения выражения п.1 данного варианта.

1.2. 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.

1.3. Выбор подхода к реализации структуры данных

Прототипы функций:

empty() – проверяет, является ли stack пустым.

pop() – удаляет элемент из верхней части stack.

push() – добавляет элемент в верхнюю часть stack.

size() – возвращает количество элементов в контейнере stack.

front() – возвращает элемент в верхней части stack.

clear() – очищает stack.

Структура информационной части элемента (узла) линейного списка:

```
struct Node {
	char val;
	Node* previous;
	Node(char value) : previous(nullptr), val(value) {}
};
```

1.4. Код реализации структуры данных

Реализация на массиве:

```
#ifndef __Task8_massiv_H
#define __Task8_massiv_H
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
using namespace std;

struct massivStack {
	int size = 0;
	char* x = new char[size];

	bool empty() {
		return size == 0;
	}

	void push_back(char value) {
		x[size++] = value;
	}

	void pop() {
		if (empty()) return;
		size--;
	}

	char front() {
		return x[size - 1];
	}

	void clear() {
		delete[]x;
		size = 0;
		x = new char[size];
	}
};
#endif
```

Реализация на однонаправленном списке:

```
#ifndef __Task8_list_H
#define __Task8_list_H
#include <iostream>
using namespace std;

struct Node {
	char val;
	Node* previous;
	Node(char value) : previous(nullptr), val(value) {}
};

struct listStack {
	Node* last;
	listStack() : last(nullptr) {}

	bool empty() {
		return last == nullptr;
	}

	void push_back(char value) {
		Node* p = new Node(value);
		if (empty()) {
			last = p;
```

```
                return;
            }
            p->previous = last;
            last = p;
        }

        void pop() {
            if (empty()) return;
            Node* p = last->previous;
            delete last;
            last = p;
        }

        char front() {
            return last->val;
        }

        void clear() {
            while (!empty()) {
                pop();
            }
            last = nullptr;
        }
};
#endif
```

1.5. Код функции вычисления значения выражения

```
#include <iostream>
#include <queue>
#include "task_8_2_list.h"
using namespace std;

void pop(listStack <char>* St, queue <char>* Q) {
    while (!(*St).empty()) {
        if ((*St).top() == '(') {
            (*St).pop();
            break;
        }
        (*Q).push((*St).top());
        (*St).pop();
    }
}

string reverse_expression(string expression) {
    string new_expression = "";
    for (int i = expression.size() - 1; i >= 0; i--) {
        if (expression[i] == '(') {
            new_expression += ')';
        }
        else if (expression[i] == ')') {
            new_expression += '(';
        }
        else {
            new_expression += expression[i];
        }
    }
    return new_expression;
}

string infix_to_postfix(string expression) {
    listStack <char> St;
    queue <char> Q;
    for (int i = 0; i < expression.size(); i++) {
```

```
            if (expression[i] == '(') {
                St.push(expression[i]);
            }
            else if (expression[i] == ')') {
                pop(&St, &Q);
            }
            else if (expression[i] == '+' || expression[i] == '-') {
                if (St.empty() || St.top() == '(') {
                    St.push(expression[i]);
                }
                else if (St.top() == '*' || St.top() == '/') {
                    pop(&St, &Q);
                    St.push(expression[i]);
                }
                else {
                    Q.push(St.top());
                    St.pop();
                    St.push(expression[i]);
                }
            }
            else if (expression[i] == '*' || expression[i] == '/') {
                if (!St.empty() && (St.top() == '*' || St.top() == '/')) {
                    pop(&St, &Q);
                }
                St.push(expression[i]);
            }
            else {
                Q.push(expression[i]);
            }
        }
        while (!St.empty()) {
            if (St.top() != '(') {
                Q.push(St.top());
            }
            St.pop();
        }
        string new_expression = "";
        while (!Q.empty()) {
            new_expression += Q.front();
            Q.pop();
        }
        return new_expression;
}

string infix_to_prefix(string expression) {
        expression = reverse_expression(expression);
        expression = infix_to_postfix(expression);
        expression = reverse_expression(expression);
        return expression;
}
```

1.6. Код основной программы

```
int main() {
        //string expression_infix = "(a+b)*c-(d+e*f/((g/h+i-j)*k))/r";
        string expression_infix = "a+b*c-d/e*h";
        string expression_prefix;
        expression_prefix = infix_to_prefix(expression_infix);
        cout << expression_prefix << endl;
        return 0;
}
```

1.7. Результаты тестирования

Выражение из задания 1 п.1: S=a+(b-c*k)-d*e-f

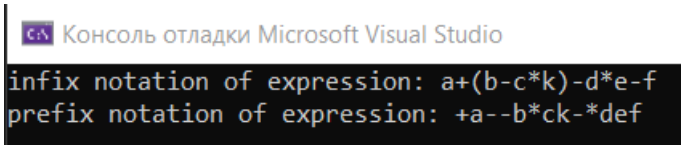

Рисунок 1 – Скриншот тестирования программы

Задание 2. п.2

2.1. Условие задачи

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

2.2. Постановка задачи

Доработать программу из задания 1, добавив функцию, вычисляющую значение выражения.

2.3. Описание подхода решения задачи

Для вычисления значения выражения в префиксной форме поочерёдно рассматриваем символы, записанные в выражении.

Если символ является числом, записываем его в стек, в котором хранятся числа, над которыми ещё предстоит совершить какие-то действия.

Если символ является знаком, берём два верхних числа из стека и производим над ними действие в зависимости от знака.

2.4. Алгоритм на псевдокоде и описание всех используемых переменных

Function calculate_prefix

Pass In: expression – выражение (строка)

INIT стек, в который записываются целочисленные значения

INIT целочисленная вспомогательная переменная *a*

FOR каждый символ в строке expression, начиная с конца

IF символ = '+' THEN

a := значение верхнего элемента стека

удалить из стека верхний элемент

a := *a* + значение верхнего элемента стека

удалить из стека верхний элемент

записать в стек *a*

ELSE IF символ = '*' THEN

a := значение верхнего элемента стека

удалить из стека верхний элемент

a := *a* * значение верхнего элемента стека

удалить из стека верхний элемент

записать в стек *a*

ELSE IF символ = '/' THEN

a := значение верхнего элемента стека

удалить из стека верхний элемент

a := *a* / значение верхнего элемента стека

удалить из стека верхний элемент

записать в стек *a*

ELSE IF символ = '-' THEN

a := значение верхнего элемента стека

удалить из стека верхний элемент

a := *a* - значение верхнего элемента стека

удалить из стека верхний элемент

записать в стек *a*

ELSE

записать в стек число (текущий символ)

ENDIF

ENDFOR

Pass Out: верхний элемент стека

Endfunction

2.5. Код реализации функции и основной программы, доказывающей соответствие алгоритма требованиям постановки задачи.

```
int calculate_prefix(string expression) {
	stack <int> St;
	int a;
	for (int i = expression.size() - 1; i >= 0; i--) {
		if (expression[i] == '+') {
			a = St.top();
			St.pop();
			a += St.top();
			St.pop();
			St.push(a);
		}
		else if (expression[i] == '*') {
			a = St.top();
			St.pop();
			a *= St.top();
			St.pop();
			St.push(a);
		}
		else if (expression[i] == '/') {
			a = St.top();
			St.pop();
			a /= St.top();
			St.pop();
			St.push(a);
		}
		else if (expression[i] == '-') {
			a = St.top();
			St.pop();
			a -= St.top();
			St.pop();
			St.push(a);
		}
		else {
			St.push(static_cast<int>(expression[i]) - '0');
		}
	}
	return St.top();
}
```

```
int main() {
	string expression_infix = "(8-5)*2+6+4*(3+5)";
	cout << "infix notation of expression: " << expression_infix << endl;
	string expression_prefix;
	expression_prefix = infix_to_prefix(expression_infix);
	cout << "prefix notation of expression: " << expression_prefix << endl;
	cout << "result: " << calculate_prefix(expression_prefix);
	return 0;
}
```

2.6. Результаты тестирования функции

№	Входные данные	Ожидаемый результат	Результат выполнения программы
1	*+52-78	-7	
2	+*-852+6*4+35	44	44

Задание 2. п.3

3.1. Условие задачи

Сложить два длинных числа, которые не могут быть размещены в переменной стандартного типа. Числа поступают либо как последовательность цифр, либо как строка.

3.2. Постановка задачи

Записать числа формат отличный от int, long и т.д. Затем сложить.

3.3. Описание подхода решения задачи с указанием структуры данных, которая обеспечит эффективный алгоритм

Поскольку числа не могут быть размещены в переменной стандартного типа, будем записывать поочерёдно цифры чисел в очереди (по одной очереди на число). Будет удобнее записывать число с конца, так как сложение двух чисел начинается с последних цифр.

3.4. Алгоритм на псевдокоде и описание всех используемых переменных

START

INIT long_int_1 – строка, в которую записывается первое число

INIT long_int_2 – строка, в которую записывается второе число

INIT result – строка, в которую записывается результат

INPUT ввод первого и второго числа

INIT Q1 и Q2 – очереди для записи цифр длинных чисел

FOR каждый символ в строке long_int_1, начиная с конца

    Запись символа, преобразованного в цифру, в стек Q1

ENDFOR

FOR каждый символ в строке long_int_2, начиная с конца

    Запись символа, преобразованного в цифру, в стек Q2

ENDFOR

INIT q – число, созданное для вычисления суммы цифр

INIT r – число, созданное для хранения остатка, который нужно перенести в следующий разряд

WHILE q ≠ 0 or r ≠ 0 or Q1 – непустой or Q2 – непустой

IF Q1 и Q2 – непустые

q := первая цифра очереди Q1 + первая цифра очереди Q2 + r

удалить первый элемент очереди Q1

удалить первый элемент очереди Q2

r := q / 10

result := result + число q mod 10, преобразованное в строку

ELSE IF только Q1– непустой

q := первая цифра очереди Q1 + r

удалить первый элемент очереди Q1

r := q / 10

result := result + число q mod 10, преобразованное в строку

ELSE IF только Q2– непустой

q := первая цифра очереди Q2 + r

удалить первый элемент очереди Q2

r := q / 10

result := result + число q mod 10, преобразованное в строку

ELSE

q := r

r := q / 10

IF q ≠ 0 or r ≠ 0

result := result + число q mod 10, преобразованное в строку

ENDIF

ENDIF

ENDWHILE

Перевернуть строку result

PRINT result

END

3.5. Код реализации функции и основной программы, доказывающей соответствие алгоритма требованиям постановки задачи

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main() {
	string long_int_1, long_int_2, result = "";
	cin >> long_int_1 >> long_int_2;
	queue <int> Q1;
	queue <int> Q2;
	for (int i = long_int_1.size() - 1; i >= 0; i--) {
		Q1.push(static_cast<int>(long_int_1[i]) - '0');
	}
	for (int i = long_int_2.size() - 1; i >= 0; i--) {
		Q2.push(static_cast<int>(long_int_2[i]) - '0');
	}
	int q = 1, r = 0;
	while(q != 0 || r != 0 || !Q1.empty() || !Q2.empty()) {
		if (!Q1.empty() && !Q2.empty()) {
			q = Q1.front() + Q2.front() + r;
			Q1.pop();
			Q2.pop();
			r = q / 10;
			result += to_string(q % 10);
		}
		else if (!Q1.empty()) {
			q = Q1.front() + r;
			Q1.pop();
			r = q / 10;
			result += to_string(q % 10);
		}
		else if (!Q2.empty()) {
			q = Q2.front() + r;
			Q2.pop();
			r = q / 10;
			result += to_string(q % 10);
		}
		else {
			q = r;
			r = q / 10;
			if (!(q == 0 && r == 0)) {
				result += to_string(q % 10);
			}
		}
	}
	reverse(result.begin(), result.end());
	std::cout << result;
	return 0;
}
```

3.6. Результаты тестирования функции

№	Входные данные	Ожидаемый результат	Результат выполнения программы
1	985123345 + 168955	985292300	985292300
2	992233720368547758 07+ 999922337203685477 5807	100984467440737095516 14	100984467440737095516 14

ВЫВОДЫ

В ходе выполнения задания получены знания и практические навыки по реализации структуры стек и очередь, и их применения при решении задач. Разработаны:

1. Функция преобразования инфиксной записи выражения в префиксную нотацию;
2. Функция преобразования инфиксной записи выражения в постфиксную нотацию;
3. Функция вычисления значения выражения, представленного в постфиксной форме;
4. Функция вычисления значения выражения, представленного в префиксной форме;
5. Операции стек: втолкнуть элемент в стек, вытолкнуть элемент из стека, вернуть значение элемента в вершине стека, сделать стек пустым, определить пуст ли стек;
6. Программа сложения двух больших целых чисел.

Тестирования всех операций пройдены успешно.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования – Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. Документация по Microsoft C/C++ | Microsoft Docs – [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160
3. Все публикации подряд / Хабр – [Электронный ресурс] URL: https://habr.com/ru/