Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**

по выполнению лабораторной работы

«Реализация стека и основных алгоритмов его обработки»

Минск

2018

**Лабораторная работа № 5**

**Тема работы: «Реализация стека и основных алгоритмов его обработки»**

**1. Цель работы**

Изучение принципов работы со структурой данных стек, формирование умений применения структуры для вычисления арифметических выражений.

**2. Задание**

1. Используя стек, напечатать содержимое текстового файла, выписывая литеры каждой его строки в обратном порядке.

2. Выполнить задание в соответствии с вариантом:

**Вариант 1.** Разместить в стеке 10 символов и распечатать их в обратном порядке.

**Вариант 2.** Ввести с клавиатуры 8 чисел и сформировать из них два стека. В первый поместить числа, которые делятся на 2, а во второй - остальные. Распечатать оба стека.

3. С использованием стека вычислить арифметическое выражение, предварительно преобразовав его в постфиксную форму записи.

4\*. С использованием стека вычислить арифметическое выражение, предварительно преобразовав его в префиксную форму записи.

**3. Оснащение работы**

ПК, Pascal.

**4. Основные теоретические сведения**

**Стеком** называется упорядоченный набор элементов, в котором размещение новых и удаление существующих происходит только с одного конца, называемого **вершиной**.

**Дисциплина обслуживания** — это совокупность правил (упорядочение и алгоритм) обслуживания элементов динамической структуры данных.

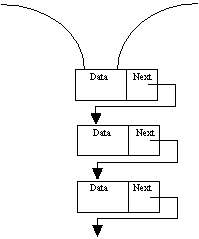
В стеке реализуется дисциплина обслуживания LIFO:

**LAST** – последний;

**INPUT** – вошел;

**FIRST** – первый;

**OUTPUT** – вышел.



Графическое представление стека

Основные операции над стеком:

*- инициализация стека;*

- включение нового элемента (английское название **push** - заталкивать);

- исключение элемента из стека (англ. **pop** - выскакивать);

*полезными могут быть также вспомогательные операции:*

- определение текущего числа элементов в стеке;

- очистка стека;

- неразрушающее чтение элемента из вершины стека.

Способы реализации стека *(существует несколько способов реализации стека)*:

- с помощью одномерного массива;

- с помощью связанного списка;

- с помощью класса объектно-ориентированного программирования.

**Реализация стека с помощью связанного списка**

Значением указателя, представляющего стек, является ссылка на вершину стека, и каждый элемент стека содержит поле ссылки на соседний, "нижний" элемент.

*Таким образом, описать стек можно следующим образом:*

**Type**

**EXST = ^ST;**

**ST = record**

**Data : integer;**

**Next : EXST;**

**end;**

**Var**

**Stack : EXST; {Текущая переменная}**

Если стек пуст, то значение указателя равно **Nil**.

Процедура формирования стека будет иметь следующий вид:

**Procedure FormStack;**

**Var**

**Stack : EXST; {Текущая переменная}**

**Digit : integer;**

**Procedure writeStack(Var u : EXST; Digit : integer);**

**Var**

**x : EXST;**

**Begin**

**new(x); {выделяем память под хранение нового элемента стека}**

**x^.Data := Digit; {заполняем поле данных элемента}**

**x^.Next := u; {новый элемент "связываем" со стеком}**

**u := x; {созданный элемент определяем как вершину стека}**

**End;**

**Begin**

**Stack := Nil; {инициализация стека}**

**writeln('Введите элементы стека. Окончание ввода – 0');**

**read(Digit);**

**while Digit <> 0 do**

**begin**

**writeStack(Stack, Digit);**

**read(Digit);**

**end;**

**End;**

***Занесение*** *элемента в стек производится аналогично вставке нового элемента в начало списка. Процедура занесения элемента в стек должна содержать два параметра: первый задает вершину стека, в который нужно занести элемент, второй – заносимое значение элемента стека.*

***Извлечение*** *элемента из стека. В результате выполнения этой операции некоторой переменной* ***i*** *должно быть присвоено значение первого элемента стека, и значение указателя на начало списка должно быть перенесено на следующий элемент стека.*

**Procedure readStack(Var u : EXST; Var i : integer);**

**Var**

**x : EXST;**

**Begin**

**i := u^.Data; {считываем значение поля данных в переменную}**

**x := u; {запоминаем адрес вершины стека}**

**u := u^.Next; {переносим вершину стека на следующий элемент}**

**dispose(x); {освобождаем память, занятую уже ненужным элементом стека}**

**End.**

*Недостатком описанной процедуры является предположение о том, что стек не пуст. Для его исправления следует разработать логическую функцию, определяющую, пуст ли обрабатываемый стек.*

**Function FreeStack(u : EXST) : boolean;**

*Примеры работы со стеком:*

**type**

**{указательный тип данных на элемент Стека}**

**Tptr = ^Telem;**

**{запись, состоящая из двух полей, описывающая элемент Стека}**

**Telem = record**

**inf : integer;  {информационное поле - хранит целые числа}**

**link : Tptr;    {указательное поле - ссылка на следующий элемент Стека}**

**end;**

**{раздел описания переменных}**

**var**

**{глобальная переменная - указатель на вершину Стека}**

**top : Tptr;**

**Удаление всех элементов из стека.**

**procedure delAllStack;**

**var**

**{вспомогательный указатель}**

**p : Tptr;**

**begin**

**{удаление элементов начинается с первого элемента Стека}**

**p := top;**

**{пока не будут удалены все элементы из Стека, то есть пока указатель р**

**не станет равным NIL начинаем цикл}**

**while(p <> NIL) do**

**begin**

**{чтобы удалить текущий элемент и не потерять связь со следующим, переставляем**

**указатель top на следующий элемент Стека}**

**top := p^.link;**

**{окончательно разрушаем связь между удаляемым и следующим элементом, так как**

**текущий элемент сейчас будет удаляться}**

**p^.link := NIL;**

**{удаляем верхний элемент из Стека}**

**dispose(p);**

**{установка указателя р снова на вершину Стека}**

**p := top;**

**end;**

**{конец тела процедуры}**

**{\*примечание: указатель на вершину Стека top должен ссылаться на NIL}**

**end;**

**Нахождение количества элементов в стеке.**

**function getCountElem : integer;**

**var**

**{вспомогательный указатель}**

**p : Tptr;**

**{хранит количество элементов в Стеке}**

**k : integer;**

**{начало тела функции}**

**begin**

**{зануляем количество, так как расчет еще не начался}**

**k := 0;**

**{указатель p стартует с самого верхнего элемента Стека}**

**p := top;**

**{до тех пор пока р не выйдет за последний элемент, то есть в NIL**

**будем вести подсчет элементов}**

**while(p <> NIL) do**

**begin**

**{счетчик, отвечающий за количество элементов увеличиваем на единицу}**

**k := k + 1;**

**{переходим на следующий элемент Стека}**

**p := p^.link;**

**end;**

**{в качестве ответа возвращаем количество элементов в Стеке}**

**getCountElem := k;**

**end;**

**Реализация стека на базе массива**

Если максимальный размер данных можно определить до начала их использования, более эффективным может оказаться однократное выделение непрерывной области памяти. Это можно сделать либо в динамической памяти, либо с помощью обычных массивов. Связь элементов при этом осуществляется не через указатели, а с помощью вспомогательных массивов и переменных, в которых хранятся номера элементов.

*Рассмотрим реализацию стека. Пусть известно, что количество его элементов не превышает* ***n****. Кроме массива элементов, соответствующих типу данных стека, достаточно иметь одну переменную целого типа для хранения индекса элемента массива, являющего вершиной стека. При помещении в стек индекс увеличивается на единицу, а при выборке – уменьшается.*

*Например, стек, описанный с помощью указателей*

**Type**

**pnode=^node;**

**node=record**

**d:word; {информационная}**

**s:string; {часть}**

**p:pnode; {указатель}**

**end;**

*можно описать следующим образом:*

**const n = 100;**

**Type node = record**

**d: word;**

**s:string;**

**end;**

**stack = array [1..n] of node;**

**var st: stack;**

**top:word;**

*В начале работы со стеком переменная* ***top*** *обнуляется. Занесение в стек выглядит примерно так:*

**inc(top):**

**if top > n then begin**

**writeln (‘переполнение стека’);**

**exit**

**end;**

**st[top].d:=d;**

**st[top].s:=s;**

*При использовании динамической памяти вместо массива описывается указатель на массив:*

**var st:= ^stack;**

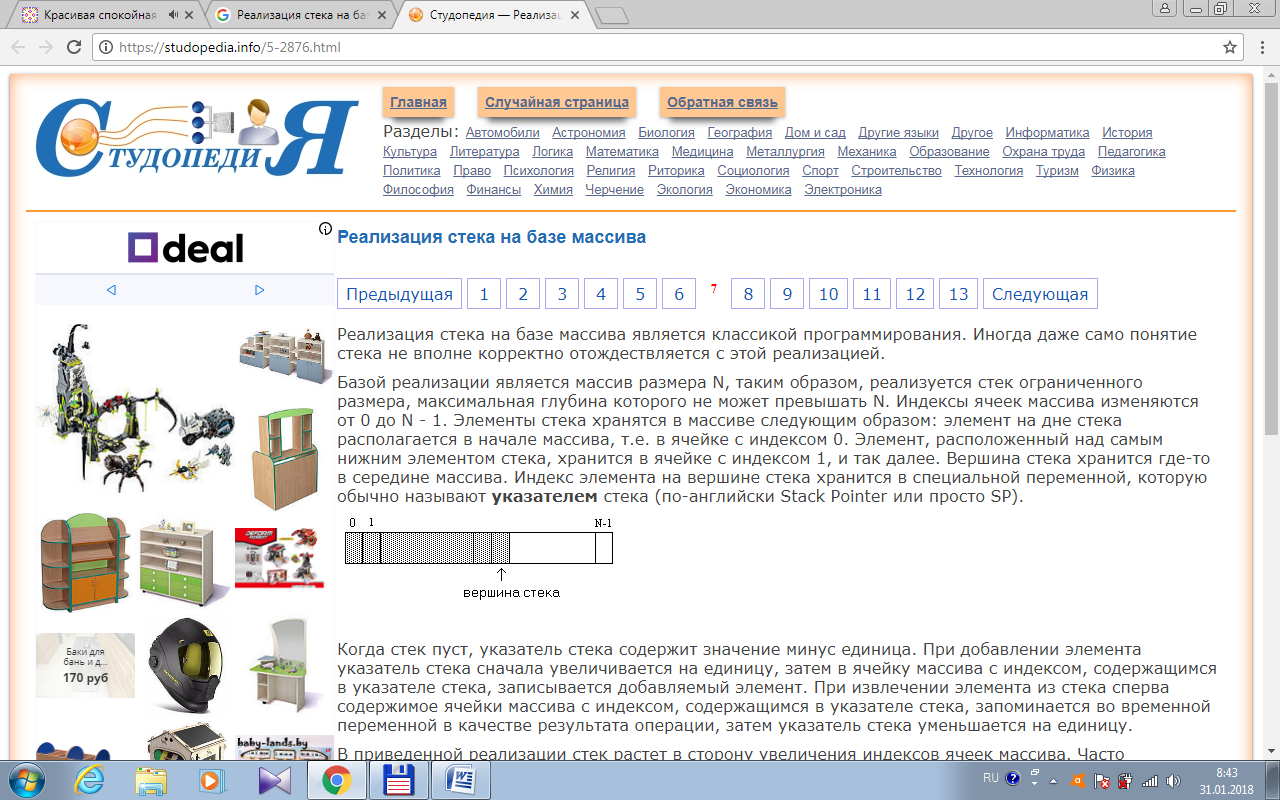
*Перед использованием такого стека необходимо выделить под него память:*

**new(st);**

*Обращение к элементу стека будет содержать операцию разадресации:*

**st^[top].d:=d;**

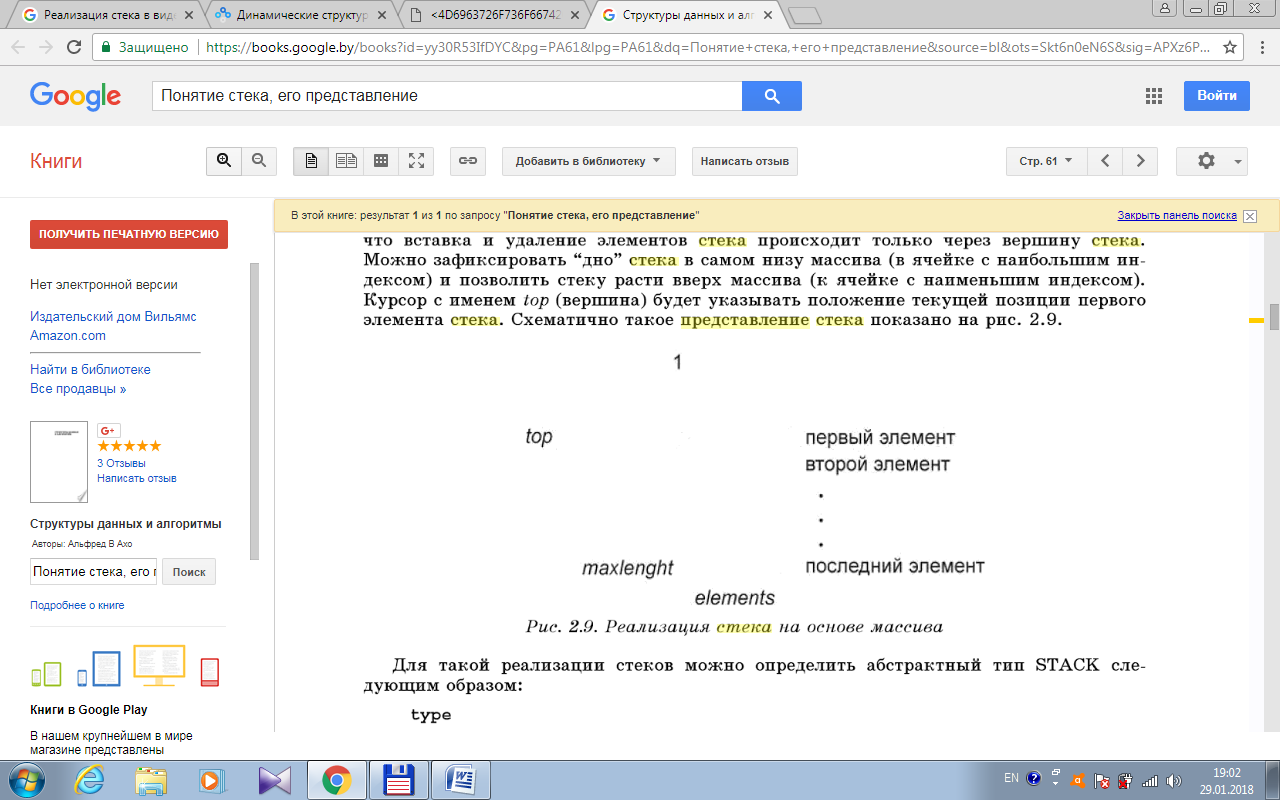
*Базой реализации является массив размера* ***N,*** *таким образом, реализуется стек ограниченного размера, максимальная глубина которого не может превышать* ***N****. Индексы ячеек массива изменяются от* ***0*** *до* ***N - 1****. Элементы стека хранятся в массиве следующим образом: элемент на дне стека располагается в начале массива, т.е. в ячейке с индексом 0. Элемент, расположенный над самым нижним элементом стека, хранится в ячейке с индексом 1, и так далее. Вершина стека хранится где-то в середине массива. Индекс элемента на вершине стека хранится в специальной переменной, которую обычно называют указателем стека (по-английски Stack Pointer или просто SP).*

**

*Когда стек пуст, указатель стека содержит значение минус единица. При добавлении элемента указатель стека сначала увеличивается на единицу, затем в ячейку массива с индексом, содержащимся в указателе стека, записывается добавляемый элемент. При извлечении элемента из стека сперва содержимое ячейки массива с индексом, содержащимся в указателе стека, запоминается во временной переменной в качестве результата операции, затем указатель стека уменьшается на единицу.*

*В приведенной реализации стек растет в сторону увеличения индексов ячеек массива. Часто используется другой вариант реализации стека на базе вектора, когда дно стека помещается в последнюю ячейку массива, т.е. в ячейку с индексом* ***N - 1****. Элементы стека занимают непрерывный отрезок массива, начиная с ячейки, индекс которой хранится в указателе стека, и заканчивая последней ячейкой массива. В этом варианте стек растет в сторону уменьшения индексов. Если стек пуст, то указатель стека содержит значение* ***N*** *(которое на единицу больше, чем индекс последней ячейки массива).*

*Таким образом, можно зафиксировать «дно» стека в самом низу массива (в ячейке с наибольшим индексом) и позволить ему расти вверх массива (к ячейке с наименьшим индексом). Курсор с именем* ***top*** *(вершина) будет указывать положение текущей позиции первого элемента стека. Схематично такое представление стека будет выглядеть следующим образом:*

****

*Для такой реализации стеков можно определить абстрактный тип Stack следующим образом:*

**type**

**stack = record**

**top: integer;**

**element: array [1..maxlength] of elementtype**

**end;**

*В этой реализации стек состоит из последовательности элементов element[top], element[top+1], …, element[maxlength]. Отметим, если top=maxlength+1, то стек пустой.*

*Процедуры для реализации пяти типовых операторов, выполняемых над стеками:*

**procedure MAKENIL (var S: STACK);**

**begin**

**S.top:=maxlength+1**

**end; { MAKENIL }**

**function EMPTY (S: STACK): Boolean;**

**begin**

**if S.top > maxlength then**

**return (true)**

**else**

**return (false)**

**end; {EMPTY}**

**function TOP (var S: STACK):elementtype;**

**begin**

**if EMPTY (S) then**

**error (‘Стек пустой’)**

**else**

**return (S.elements[S.top])**

**end; { TOP }**

**procedure POP (var S: STACK);**

**begin**

**if EMPTY (S) then**

**error (‘Стек пустой’)**

**else**

**S.top:=S.top+1**

**end; {POP}**

**procedure PUSH (x: elementtype; var S: STACK);**

**begin**

**if S.top=1 then**

**error (‘Стек полон’)**

**else begin**

**S.top:=S.top-1**

**S.elements[S.top]:=x**

**end**

**end; {PUSH}**

*значение, возвращаемое функцией top, имеет тип elementtype, который должен быть разрешенным типом результата, возвращаемого функцией. Если это не так, данную функцию нужно преобразовать в процедуру или она должна возвращать указатель на элемент типа elementtype.*

*Пример:*

Программный модуль иллюстрирует операции над стеком, расширяющимся в сторону уменьшения адресов. Указатель стека всегда указывает на первый свободный элемент.

*Предполагается, что в модуле будут уточнены определения предельного размера структуры и типа данных для элемента структуры:*

**const SIZE = ...;**

**type data = ...;**

**{ Стек }**

**unit Stack;**

**Interface**

**const SIZE=...; { предельный размер стека }**

**type data = ...; { эл-ты могут иметь любой тип }**

**Procedure StackInit;**

**Procedure StackClr;**

**Function StackPush(a : data) : boolean;**

**Function StackPop(Var a : data) : boolean;**

**Function StackSize : integer;**

**Implementation**

**Var StA : array[1..SIZE] of data; { Стек - данные }**

**{ Указатель на вершину стека, работает на префиксное вычитание }**

**top : integer;**

**Procedure StackInit; {\*\* инициализация - на начало }**

**begin top:=SIZE; end; {\*\* очистка = инициализация }**

**Procedure StackClr;**

**begin top:=SIZE; end;**

**{ \*\* занесение элемента в стек }**

**Function StackPush(a: data) : boolean;**

**begin**

**if top=0 then StackPush:=false**

**else begin { занесение, затем - увеличение указателя }**

**StA[top]:=a; top:=top-1; StackPush:=true;**

**end; end; { StackPush }**

**{ \*\* выборка элемента из стека }**

**Function StackPop(var a: data) : boolean;**

**begin**

**if top=SIZE then StackPop:=false**

**else begin { указатель увеличивается, затем - выборка }**

**top:=top+1; a:=StA[top]; StackPop:=true;**

**end; end; { StackPop }**

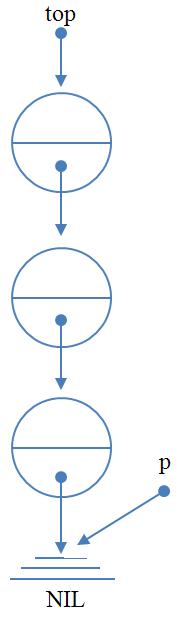
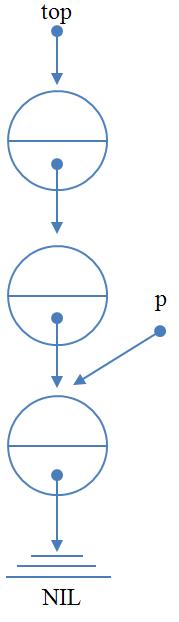
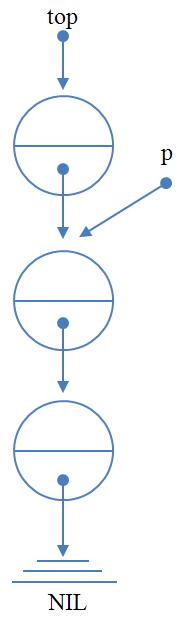
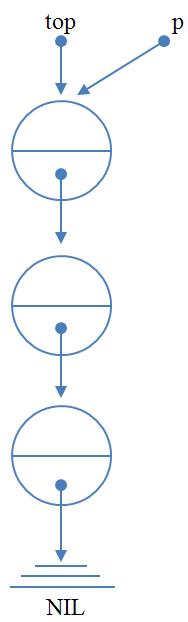
**Function StackSize : integer; {\*\* определение размера }**

**begin StackSize:=SIZE-top; end;**

**END.**

**Печать** элементов структуры данных стек нельзя относить к фундаментальным операциям, а лишь к вспомогательным операциям.

*Предположим, что начальный стек содержит три элемента. Для визуализации значения информационных полей элементов стека необходим дополнительный указатель. Введем идентификатор p.*



*Выше представлен последовательность всех итераций по печати элементов стека на экран пользователя.*

Важно заметить следующее, что распечатка всех элементов на экран - итерационный процесс, следовательно, при программировании используют циклы, как правило, цикл с предусловием, то есть цикл **while-do**.

Как видно из представленной схемы, как только вспомогательный указатель **p** достигнет **NIL**, операция вывода будет официально завершена. После окончания печати всех элементов стека, сам стек остается в согласованном состоянии.

**type**

**{указательный тип данных на элемент Стека}**

**Tptr = ^Telem;**

**{запись, состоящая из двух полей, описывающая элемент Стека}**

**Telem = record**

**inf : integer;  {информационное поле - хранит целые числа}**

**link : Tptr;    {указательное поле - ссылка на следующий элемент Стека}**

**end;**

**{раздел описания переменных}**

**var**

**{глобальная переменная - указатель на вершину Стека}**

**top : Tptr;**

**procedure printFromTop;**

**var**

**{вспомогательный указатель, ссылающийся на текущий элемент Стека}**

**p : Tptr;**

**{начало тела процедуры}**

**begin**

**{устанавливаем указатель р на первый элемент Стека}**

**p := top;**

**{выпечатываем на экран диалог}**

**write('Элементы стека имеют вид: ');**

**{до тех пор, пока указатель р не выйдет за последний элемент Стека}**

**while(p <> NIL) do**

**begin**

**{печатаем на экран пользователя информационное поле текущего элемента Стека,**

**причем целые числа печатаются через пробел, чтобы вывод был понятным}**

**write(p^.inf, ' ');**

**{переход на следующий элемент Стека}**

**p := p^.link;**

**end;**

**{конец тела процедуры}**

**end;**

Нестандартные операции для работы со стеком:

1. Расщепление структуры исходного стека на два различных подстека по какому-либо признаку.

2. Слияние элементов из двух или более однородных стековых структур в консолидирующий стек по какому-либо признаку.

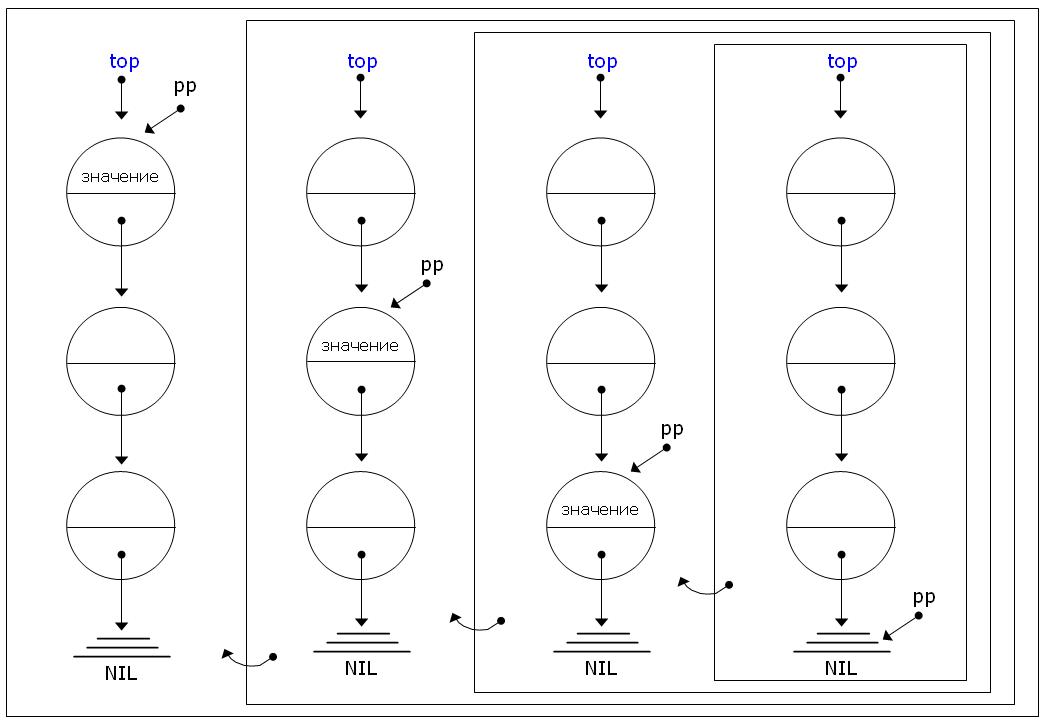
3. Сортировка стека по заданному ключу.

4. Поиск элемента стека по заданному критерию (в качестве критерия, как правило, выступает значение информационного поля элемента).

5. Переворот элементов стека. Элемент, являющийся вершиной стека становится его последним (замыкающим) элементом, а последний (замыкающий) элемент становится первым, то есть превращается в ведущий элемент.

6. Печать элементов стека от замыкающего к начальному элементу, используя рекурсивный обход.

**Печать** элементов стека от замыкающего к начальному элементу, **используя рекурсивный вызов.**



*Как видно из представленной схемы рекурсивной печати элементов, для обхода стека требуется вспомогательный указатель (им является идентификатор* ***pp****).*

*На схеме показаны вложенные копии вызова рекурсивной процедуры, причем в каждой версии процедуры существует "свой" указатель* ***pp****, указывающий на соответствующий элемент стека.*

Для обхода стека, состоящего из трех элементов, потребовалось 4-ре раза вызвать рекурсивную процедуру, что, несомненно, является затратным с точки зрения использования динамической памяти программы.

Как видно из изображения, рекурсия закончилась, когда указатель **pp** достиг критического значения, то есть вышел в **NIL**. Как только это случилось, рекурсивные копии процедур начали закрываться, и именно в этот момент происходит визуализация значений информационных полей на дисплей пользователя.

Подобный вариант рекурсии имеет название рекурсия на возврате. В каждой копии процедуры имеется элемент, у которого подписано слово "значение", вот именно данное значение и будет распечатано на экране.

**type**

**{указательный тип данных на элемент Стека}**

**Tptr = ^Telem;**

**{запись, состоящая из двух полей, описывающая элемент Стека}**

**Telem = record**

**inf : integer;  {информационное поле - хранит целые числа}**

**link : Tptr;    {указательное поле - ссылка на следующий элемент Стека}**

**end;**

**{раздел описания переменных}**

**var**

**{глобальная переменная - указатель на вершину Стека}**

**top : Tptr;**

**procedure printRecFromEnd(pp : Tptr);**

**begin**

**{любая рекурсия обязана иметь элементарный случай, чтобы завершится. В**

**данной процедуре, элементарным случаем является ситуация, когда**

**указатель вышел за последний элемент стека, то есть ссылается на NIL}**

**if(pp <> NIL) then**

**begin**

**{рекурсивный вызов, в качестве параметра передается указатель на**

**следующий элемент Стека}**

**printRecFromEnd(pp^.link);**

**{печать значения информационного поля текущего элемента**

**причем целые числа печатаются через пробел, чтобы вывод был понятным}**

**write(pp^.inf, ' ');**

**end;**

**{конец тела процедуры}**

**end;**

*Пример:*

**Program Stek;**

**uses**

**crt; {Для использования readkey и clrscr}**

**type**

**Tinf=integer; {тип данных, который будет храниться в элементе стека}**

**List=^TList;  {Указатель на элемент типа TList}**

**TList=record {А это наименование нашего типа "запись" обычно динамические структуры описываются через запись}**

**data:TInf;  {данные, хранимые в элементе}**

**next:List;   {указатель на следующий элемент}**

**end;**

**{Процедура добавляющая элемент в стек}**

**procedure AddElem(var stek1:List;znach1:TInf);**

**var**

**tmp:List;**

**begin**

**GetMem(tmp,sizeof(TList)); {выделяем в памяти место для нового элемента}**

**tmp^.next:=stek1;  {указатель на следующий элемент "направляем" на вершину стека}**

**tmp^.data:=znach1; {добавляем к элементу данные}**

**stek1:=tmp; {вершина стека изменилась, надо перенести и указатели на неё}**

**end;**

**{Процедура вывода стека}**

**procedure Print(stek1:List);**

**begin**

**if stek1=nil then {проверка на пустоту стека}**

**begin**

**writeln('Стек пуст.');**

**exit;**

**end;**

**while stek1<>nil do {пока указатель stek1 не станет указывать в пустоту}**

**begin   {а это произойдёт как только он перейдёт по ссылке последнего элемента}**

**Write(stek1^.data, ' '); {выводить данне}**

**stek1:=stek1^.next  {и переносить указатель вглубь по стеку}**

**end;**

**end;**

**{Процедура освобождения памяти занятой стеком}**

**Procedure FreeStek(stek1:List);**

**var**

**tmp:List;**

**begin**

**while stek1<>nil do {пока stek1 не станет указывать в "пустоту" делать}**

**begin**

**tmp:=stek1; {указатель tmp направим на вершину стека}**

**stek1:=stek1^.next; {вершину стека перенесём на следующий за данной вершиной элемент}**

**FreeMem(tmp,SizeOf(Tlist)); {освободим память занятую под старую вершину}**

**end;**

**end;**

**{Поиск элемента в стеке по значению}**

**Function SearchElemZnach(stek1:List;znach1:TInf):List;**

**begin**

**if stek1<>nil then {если стек не пуст, то}**

**while (Stek1<>nil) and (znach1<>stek1^.data) do {пока stek1 не укажет в "пустоту" или пока мы не нашли нужный нам элемент}**

**stek1:=stek1^.next; {переносить указатель}**

**SearchElemZnach:=stek1;{функция возвращает указатель на найденный элемент}**

**end;         {в случае если элемент не найден, она вернёт nil}**

**{Процедура удаления элемента по указателю}**

**Procedure DelElem(var stek1:List;tmp:List);**

**var**

**tmpi:List;**

**begin**

**if (stek1=nil) or (tmp=nil) then {если стек пуст или указатель никуда не указывает, то выходим}**

**exit;**

**if tmp=stek1 then {если мы удаляем элемент который является вершиной стека, то}**

**begin**

**stek1:=tmp^.next;{следует перенести вершину и}**

**FreeMem(tmp,SizeOf(TList)); {высвободить память из под элемента}**

**end**

**else {в случае, если удаляемый элемент не вершина стека, то}**

**begin**

**tmpi:=stek1; {ставим указатель на вершину стека}**

**while tmpi^.next<>tmp do {доходим до элемента стоящего "перед" тем, который нам следует удалить}**

**tmpi:=tmpi^.next;**

**tmpi^.next:=tmp^.next; {указатель элемента переносим на следующий элемент за удаляемым}**

**FreeMem(tmp,sizeof(TList)); {удаляем элемент}**

**end;**

**end;**

**{Процедура удаления элемента по значению}**

**procedure DelElemZnach(var Stek1:List;znach1:TInf);**

**var**

**tmp:List;**

**begin**

**if Stek1=nil then {Если стек пуст, то выводим сообщение и выходим}**

**begin**

**Writeln('Стек пуст');**

**exit;**

**end;**

**tmp:=SearchElemZnach(stek1,znach1); {tmp указывает на удаляемый элемент}**

**if tmp=nil then {если элемент не был найден, то выводим сообщение и выходим}**

**begin**

**writeln('Элемент с искомым значением ' ,znach1, ' отсутствует в стеке.');**

**exit;**

**end;**

**DelElem(stek1,tmp); {удаляем элемент из стека }**

**Writeln('Элемент удалён.'); {сообщаем о выполнении действия}**

**end;**

**{Удаление элемента по порядковому номеру (вершина имеет номер 1)}**

**Procedure DelElemPos(var stek1:List;posi:integer);**

**var**

**i:integer;**

**tmp:List;**

**begin**

**if posi<1 then {проверка на ввод информации}**

**exit;**

**if stek1=nil then {если стек пуст}**

**begin**

**Write('Стек пуст');**

**exit**

**end;**

**i:=1; {будет считать позиции}**

**tmp:=stek1;**

**while (tmp<>nil) and (i<>posi) do {пока tmp не укажет в "пустоту" или мы не найдём искомый элемент}**

**begin**

**tmp:=tmp^.next; {переходим на следующий элемент}**

**inc(i)   {увеличиваем значение счётчика}**

**end;**

**if tmp=nil then {если элемента нет выводим соответствующие сообщения и выходим}**

**begin**

**Writeln('Элемента с порядковым номером ' ,posi, ' нет в стеке.');**

**writeln('В стеке ' ,i-1, ' элемента(ов).');**

**exit**

**end;**

**DelElem(stek1,tmp); {если мы не вышли, то элемент есть и его следует удалить}**

**Writeln('Элемент удалён.'); {сообщаем о выполнении действия}**

**end;**

**{Процедура сортировки "пузырьком" с изменением только данных}**

**procedure SortBublInf(nach:list);**

**var**

**tmp,rab:List;**

**tmps:Tinf;**

**begin**

**GetMem(tmp,SizeOf(Tlist)); {выделяем память для рабочего "буфера" обмена}**

**rab:=nach; {рабочая ссылка, становимся на вершину стека}**

**while rab<>nil do {пока мы не дойдём до конца стека делать}**

**begin**

**tmp:=rab^.next; {перейдём на следующий элемент}**

**while tmp<>nil do {пока не конец стека делать}**

**begin**

**if tmp^.data<rab^.data then {проверяем следует ли менять элементы}**

**begin**

**tmps:=tmp^.data; {стандартная замена в 3 операции}**

**tmp^.data:=rab^.data;**

**rab^.data:=tmps**

**end;**

**tmp:=tmp^.next {переход к следующему элементу}**

**end;**

**rab:=rab^.next {переход к следующему элементу}**

**end**

**end;**

**{Процедура сортировки "пузырьком" с изменением только адресов}**

**procedure SortBublLink(nach:List);**

**var**

**tmp,pered,pered1,pocle,rab:List; {все рабочие ссылки}**

**begin**

**rab:=nach; {становимся на вершину стека}**

**while rab<>nil do{пока не конец стека делать}**

**begin**

**tmp:=rab^.next; {переходим к следующему за сортируемым элементу}**

**while tmp<>nil do {пока не конец стека делать}**

**begin**

**if tmp^.data<rab^.data then {если следует произвести замену, то}**

**begin**

**pered:=nach; {становимся в вершину стека}**

**pered1:=nach; {становимся в вершину стека}**

**if rab<>nach then {если мы не стоим на изменяемом элементе, то}**

**while pered^.next<>rab do pered:=pered^.next; {станем на элементе перед изменяемым}**

**while pered1^.next<>tmp do pered1:=pered1^.next; {станем на элементе перед изменяемым, который находится за**

**первым изменяемым}**

**pocle:=tmp^.next; {запоминаем адрес элемента после второго изменяемого}**

**if rab^.next=tmp then {если элементы "соседи", то}**

**begin**

**tmp^.next:=rab; {меняем ссылки, тут если не понятно рисуйте на листочке}**

**rab^.next:=pocle**

**end**

**else {в случае если элементы не соседи, то}**

**begin**

**tmp^.next:=rab^.next;{меняем ссылки, тут если не понятно рисуйте на листочке}**

**rab^.next:=pocle;**

**end;**

**if pered1<>rab then{советую просмотреть на листочке}**

**pered1^.next:=rab;**

**if rab<>nach then{советую просмотреть на листочке}**

**pered^.next:=tmp**

**else{всё советую просмотреть на листочке}**

**nach:=tmp;**

**pered1:=tmp;{советую просмотреть на листочке}**

**tmp:=rab;{советую просмотреть на листочке}**

**rab:=pered1;{советую просмотреть на листочке}**

**end;**

**tmp:=tmp^.next; {переходим на следующий элемент}**

**end;**

**rab:=rab^.next;{переходим на следующий элемент}**

**end;**

**end;**

**var**

**Stk, {переменная, которая всегда будет указывать на "вершину" стека}**

**tmpl:List; {рабочая переменная}**

**znach:Tinf; {данные вводимые пользователем}**

**ch:char; {для работы менюшки}**

**begin**

**Stk:=nil;**

**repeat {цикл для нашего меню}**

**clrscr; {очистка экрана, далее идёт вывод самого меню}**

**Write('Программа для работы со ');**

**Textcolor(4);**

**Writeln('стеком.');**

**Textcolor(7);**

**Writeln('Выберите желаемое действие:');**

**Writeln('1) Добавить элемент.');**

**Writeln('2) Вывод стека.');**

**Writeln('3) Удаление элемента по значению.');**

**Writeln('4) Удаление элемента по порядковому номеру.');**

**Writeln('5) Поиск элемента по значению');**

**Writeln('6) Сортировка стека методом "Пузырька", меняя только данные.');**

**Writeln('7) Сортировка стека с изменением адресов.');**

**Writeln('8) Выход.');**

**writeln;**

**ch:=readkey; {ожидаем нажатия клавиши}**

**case ch of {выбираем клавишу}**

**'1':begin**

**write('Введите значение добавляемого элемента: ');**

**readln(znach); {считываем значение добавляемого нового элемент}**

**AddElem(Stk,znach);**

**end;**

**'2':begin**

**clrscr; {очистка экрана}**

**Print(Stk); {вызов процедуры вывода}**

**readkey; {ожидаем нажатия клавиши}**

**end;**

**'3':begin**

**Write('Введите значение удаляемого элемента: ');**

**readln(znach); {ввод значения удаляемого элемента}**

**DelElemZnach(Stk,znach); {вызов процедуры удаления элемента по значению}**

**readkey;{ожидаем нажатия клавиши}**

**end;**

**'4':begin**

**Write('Введите порядковый номер удаляемого элемента: ');**

**readln(znach); {ввод позиции удаляемого файла}**

**DelElemPos(Stk,znach);{вызов процедуры удаления элемента по значению}**

**readkey;{ожидаем нажатия клавиши}**

**end;**

**'5':begin**

**write('Введите значение искомого элемента: ');**

**readln(znach); {ввод искомого значения}**

**tmpl:=SearchElemZnach(Stk,znach); {вызываем процедуру поиска элемента по значению}**

**if tmpl=nil then {проверяем найден ли элемент и выводим соответствующие сообщения}**

**write('Искомый элемент отсутствует в стеке')**

**else**

**write('Элемент ',tmpl^.data,' найден');**

**readkey;{ожидаем нажатия клавиши}**

**end;**

**'6':begin**

**if Stk=nil then {проверяем не пустой ли стек}**

**begin**

**Write('Стек пуст.');**

**readkey{ожидаем нажатия клавиши}**

**end**

**else**

**begin**

**SortBublInf(Stk);{вызов процедуры сортировки стека с изменением данных}**

**Write('Стек отсортирован.');**

**readkey;{ожидаем нажатия клавиши}**

**end**

**end;**

**'7':begin**

**if Stk=nil then{проверяем не пустой ли стек}**

**begin**

**Write('Стек пуст.');**

**readkey{ожидаем нажатия клавиши}**

**end**

**else**

**begin**

**SortBublLink(Stk);{вызов процедуры сортировки стека с изменением адресов}**

**Write('Стек отсортирован.');**

**readkey;{ожидаем нажатия клавиши}**

**end**

**end;**

**end;**

**until ch='8';**

**FreeStek(Stk); {освобождаем память занятую стеком}**

**end.**

**Преобразование инфиксного выражения в постфиксное**

Любое арифметическое выражение можно переписать в виде бесскобочного постфиксного выражения, в котором знак операции стоит после своих операндов, а обозначение функции — после всех ее аргументов.

*В постфиксном выражении можно определить все операнды, связанные с любой операцией, поэтому необходимость в применении скобок отпадает.*

А с помощью стека можно выполнить эти операции и вычислить значение любого постфиксного выражения. Перемещаясь слева направо, мы интерпретируем каждый операнд как команду "занести операнд в стек", а каждый знак операции - как команды "извлечь из стека два операнда, выполнить операцию и занести результат в стек".

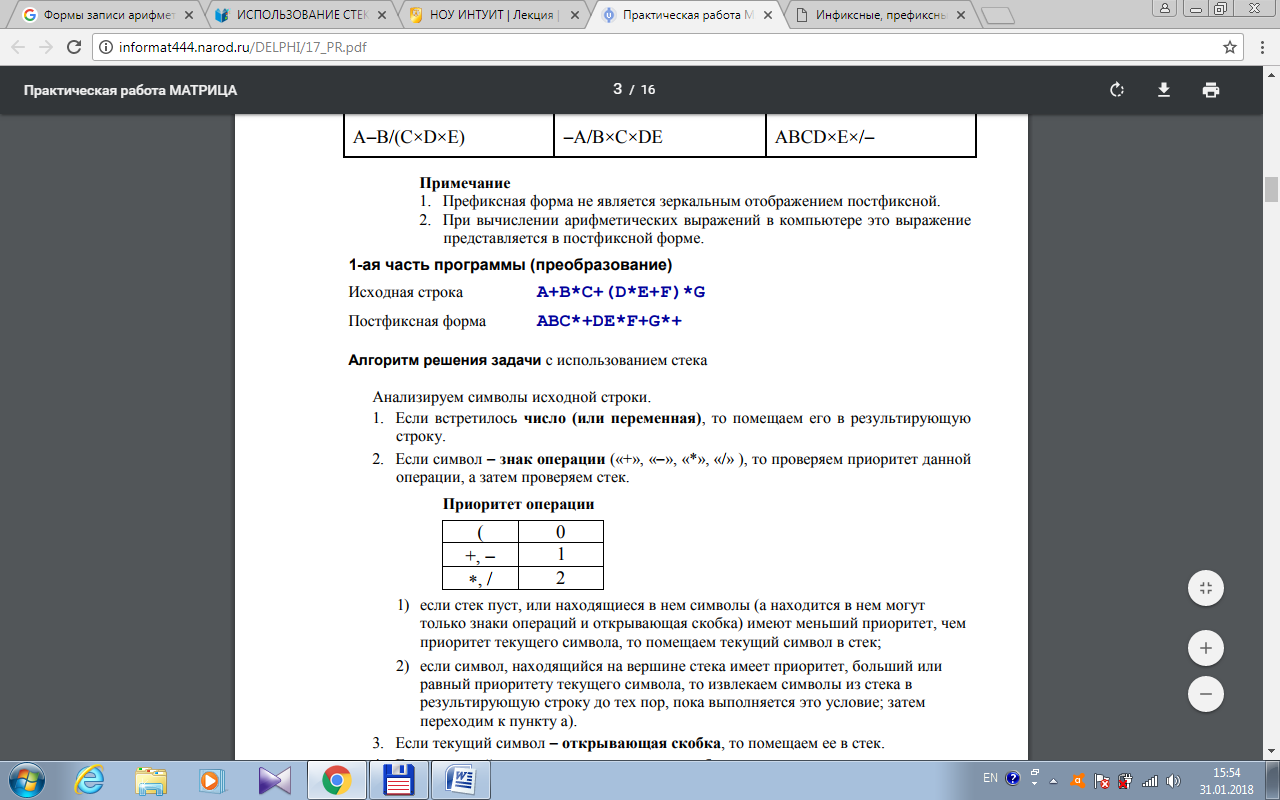
Исходная строка A+B\*C+(D\*E+F)\*G

Алгоритм решения задачи с использованием стека. Анализируем символы исходной строки:

1. Если встретилось число (или переменная), то помещаем его в результирующую строку.

2. Если символ − знак операции («+», «−», «\*», «/» ), то проверяем приоритет данной операции, а затем проверяем стек:

приоритет операции:

****

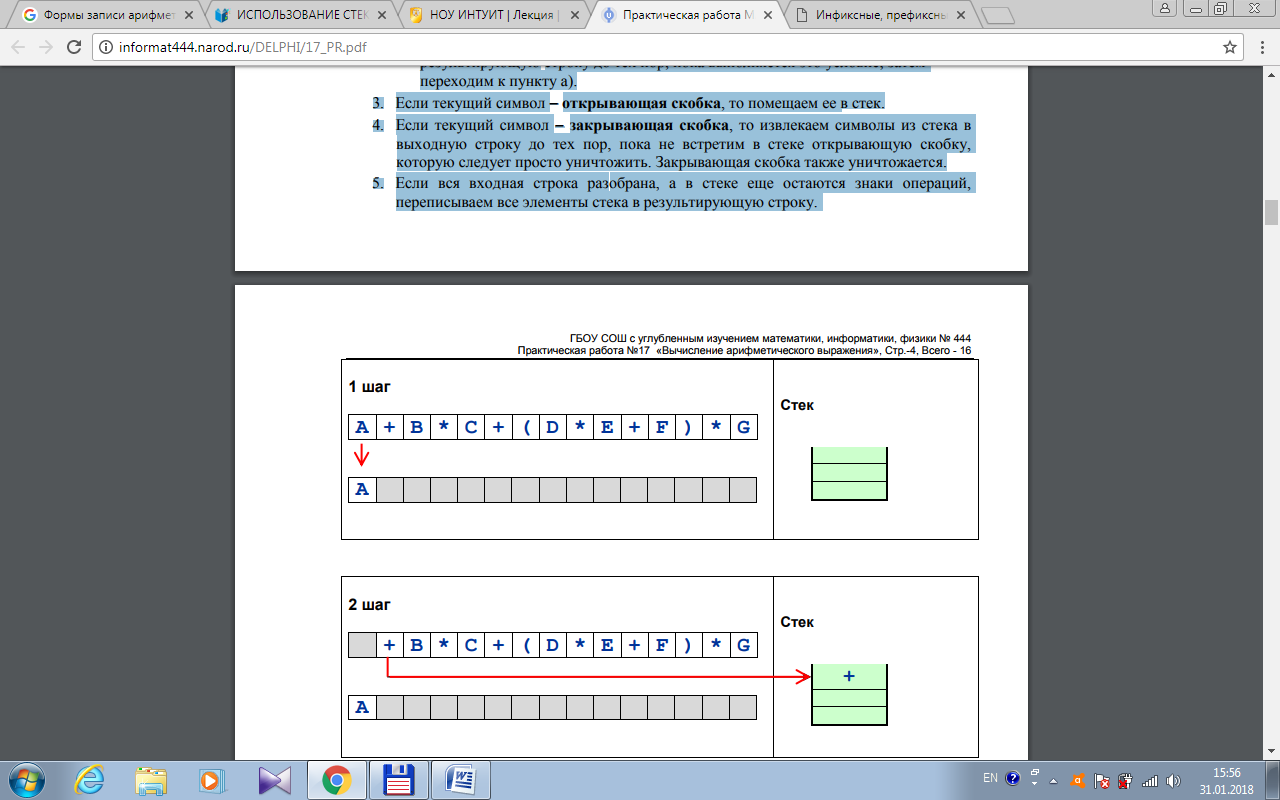
а) если стек пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек;

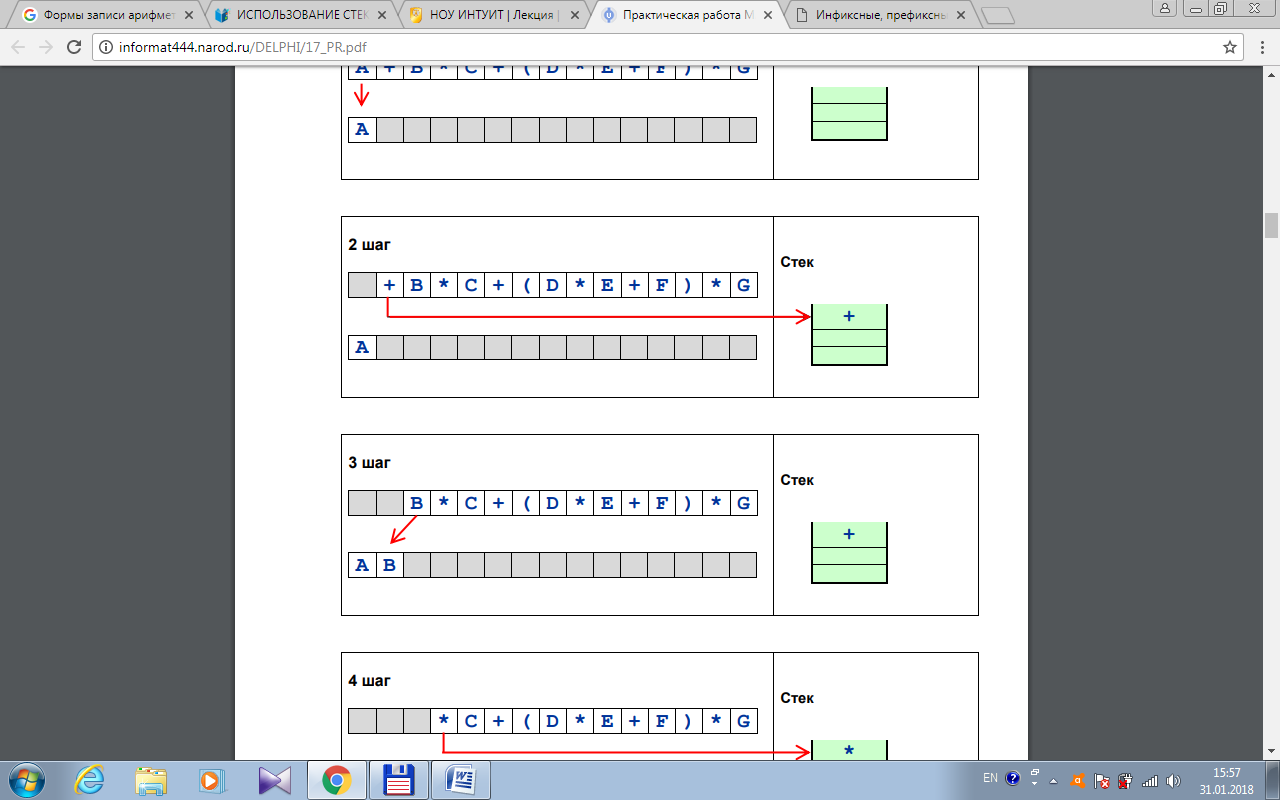
б) если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в результирующую строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

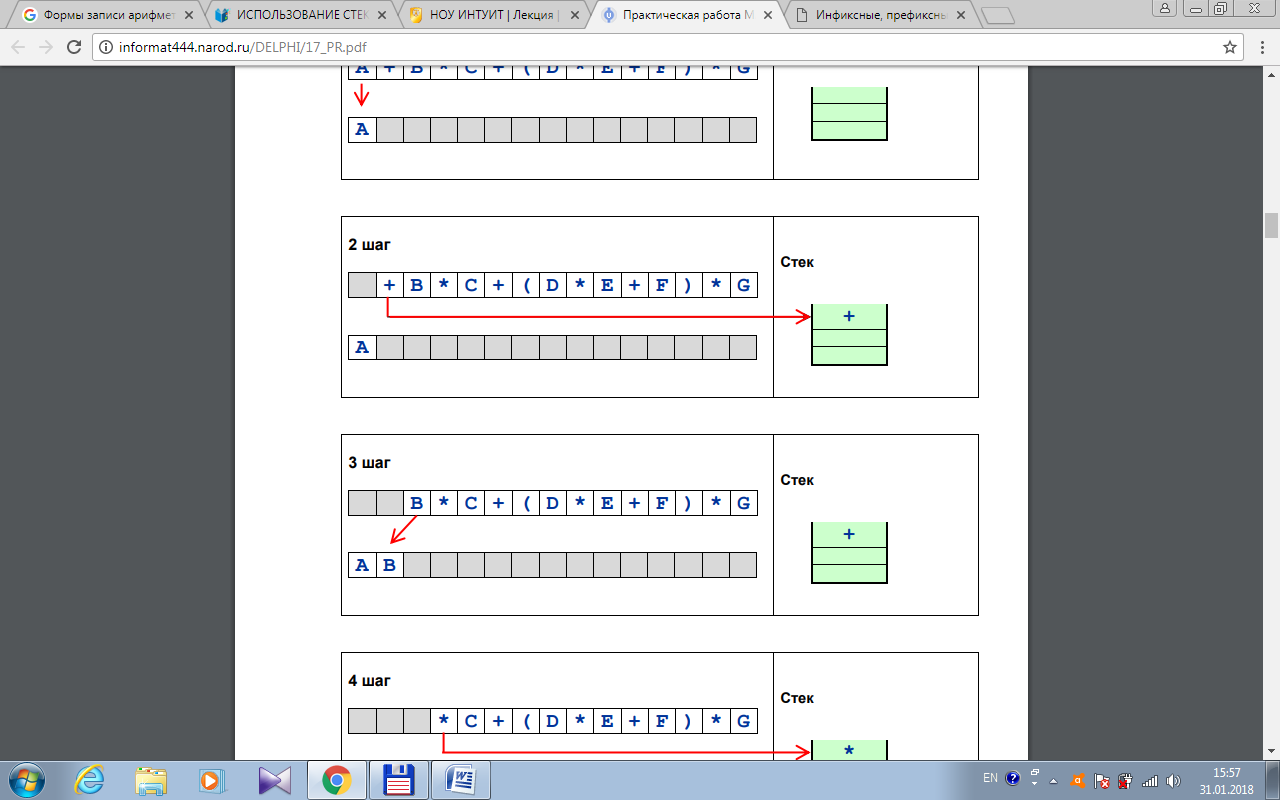
3. Если текущий символ − открывающая скобка, то помещаем ее в стек.

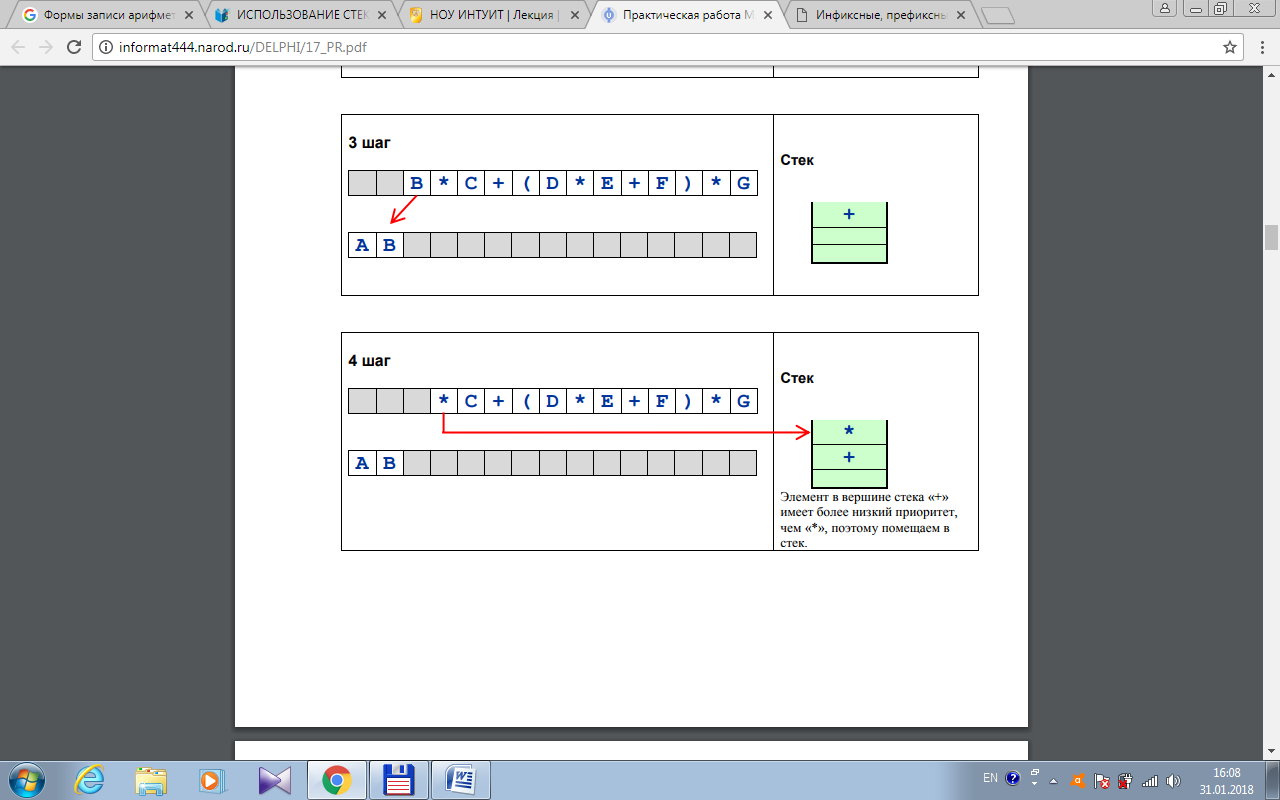
4. Если текущий символ − закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку, которую следует просто уничтожить. Закрывающая скобка также уничтожается.

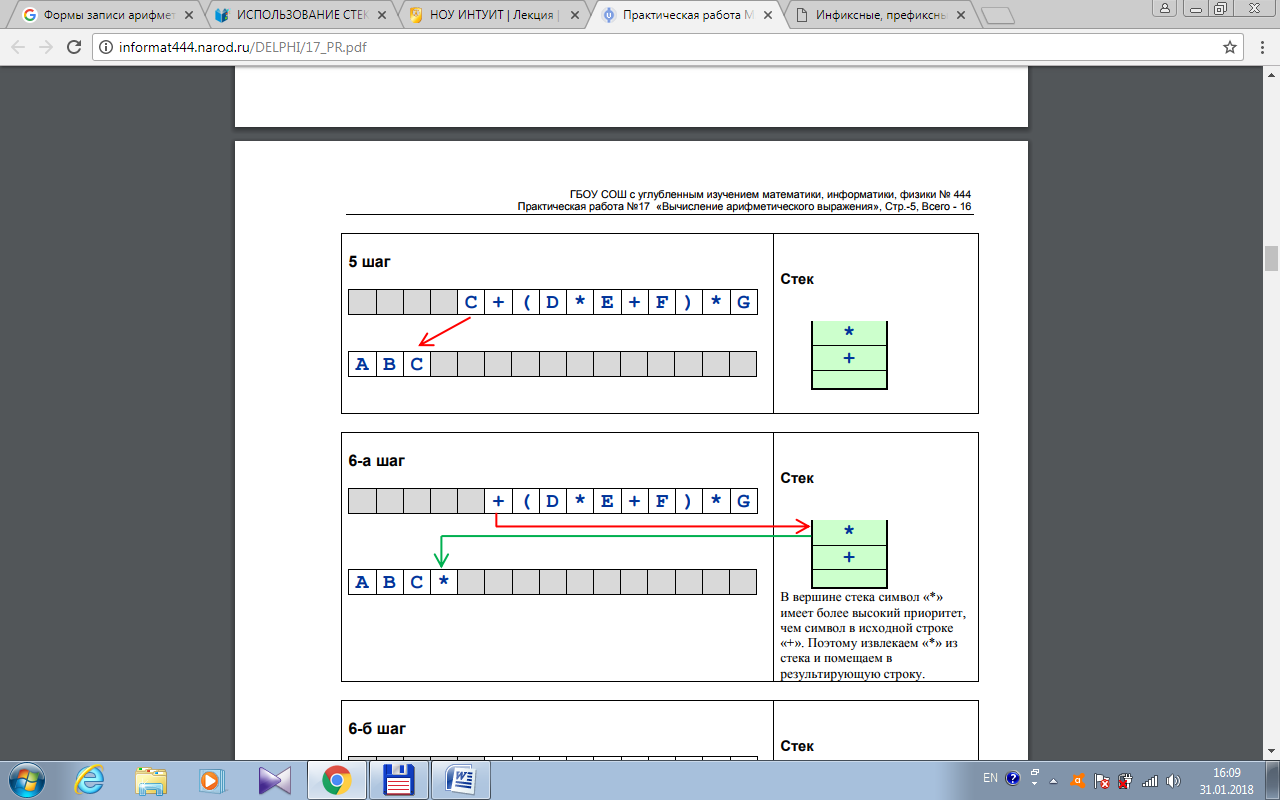
5. Если вся входная строка разобрана, а в стеке еще остаются знаки операций, переписываем все элементы стека в результирующую строку.

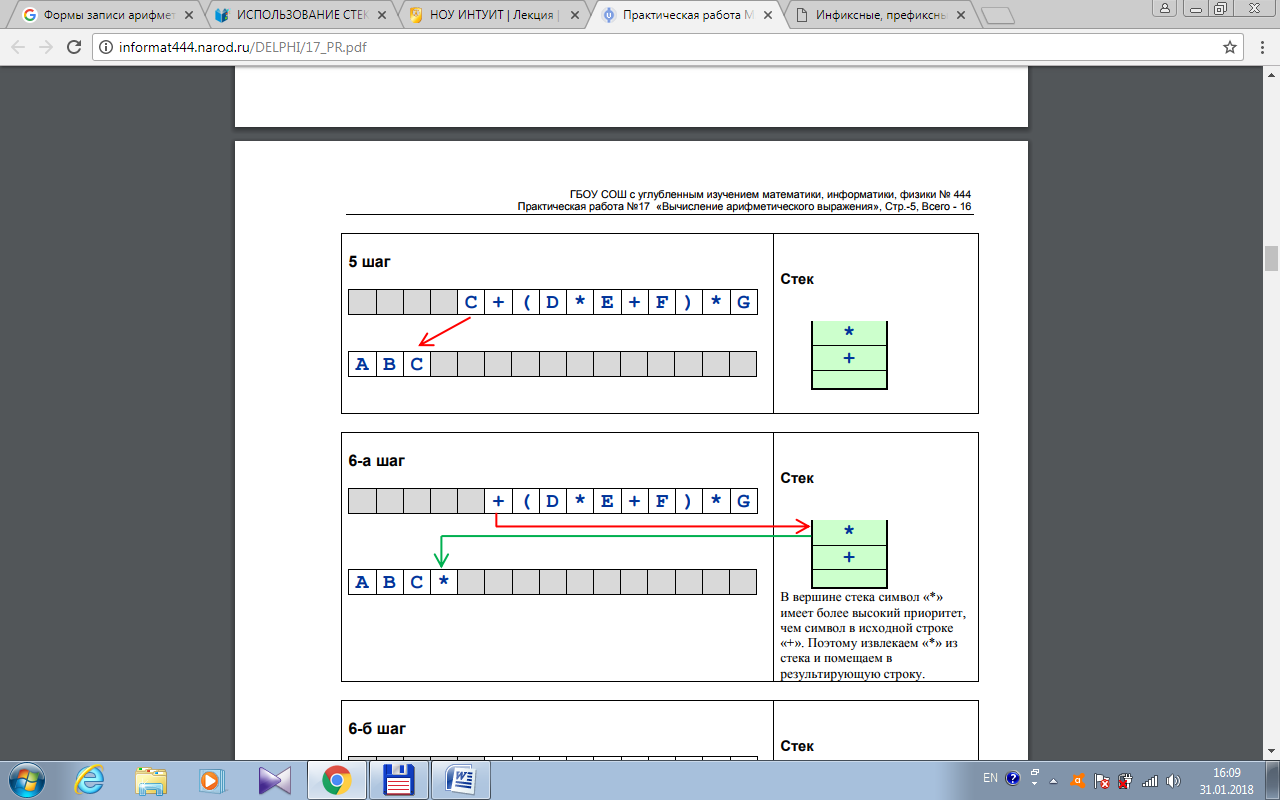
****

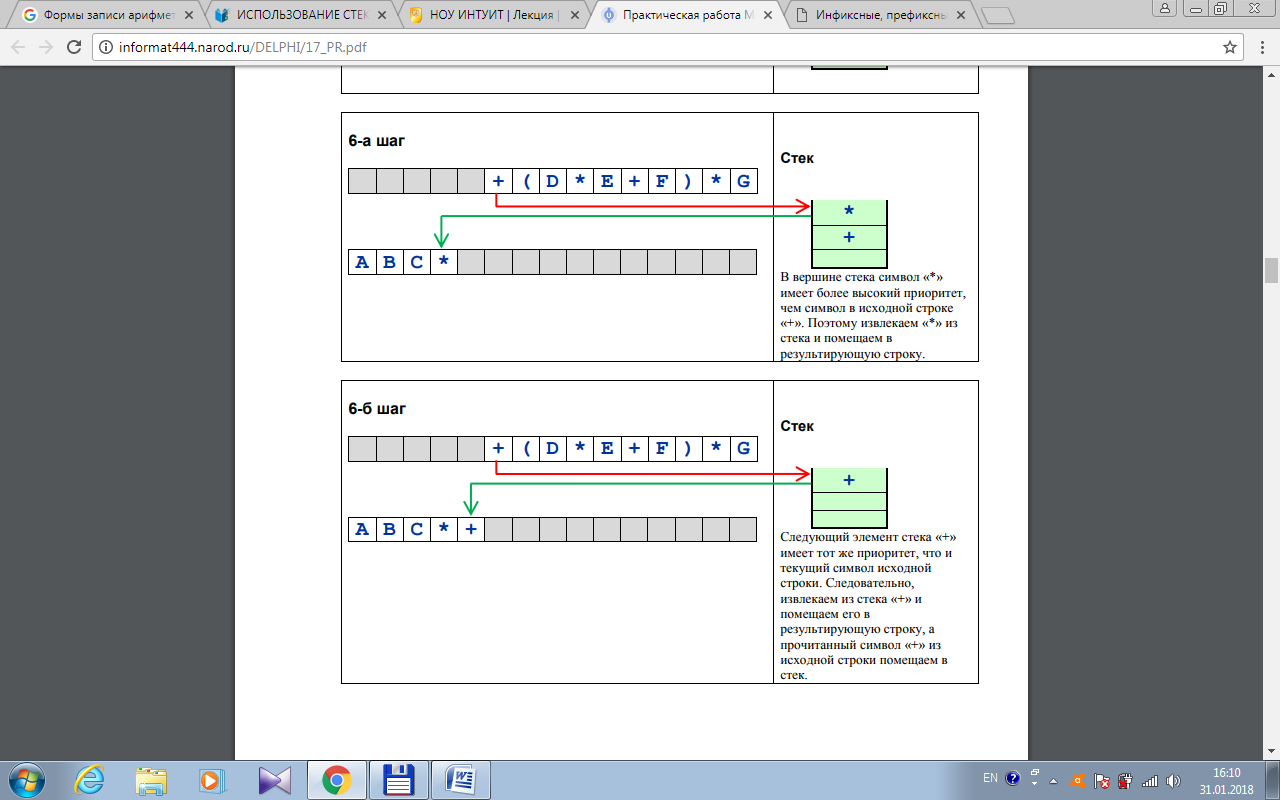
****

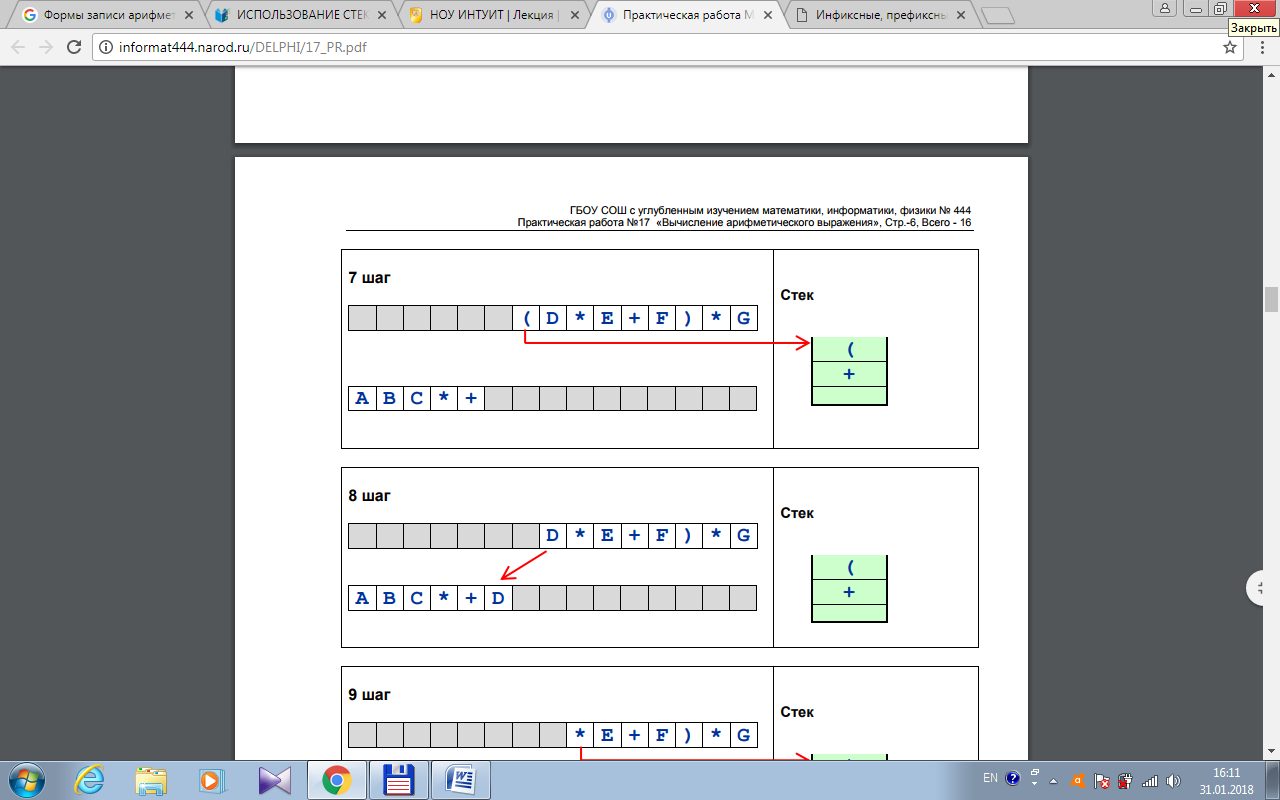
****

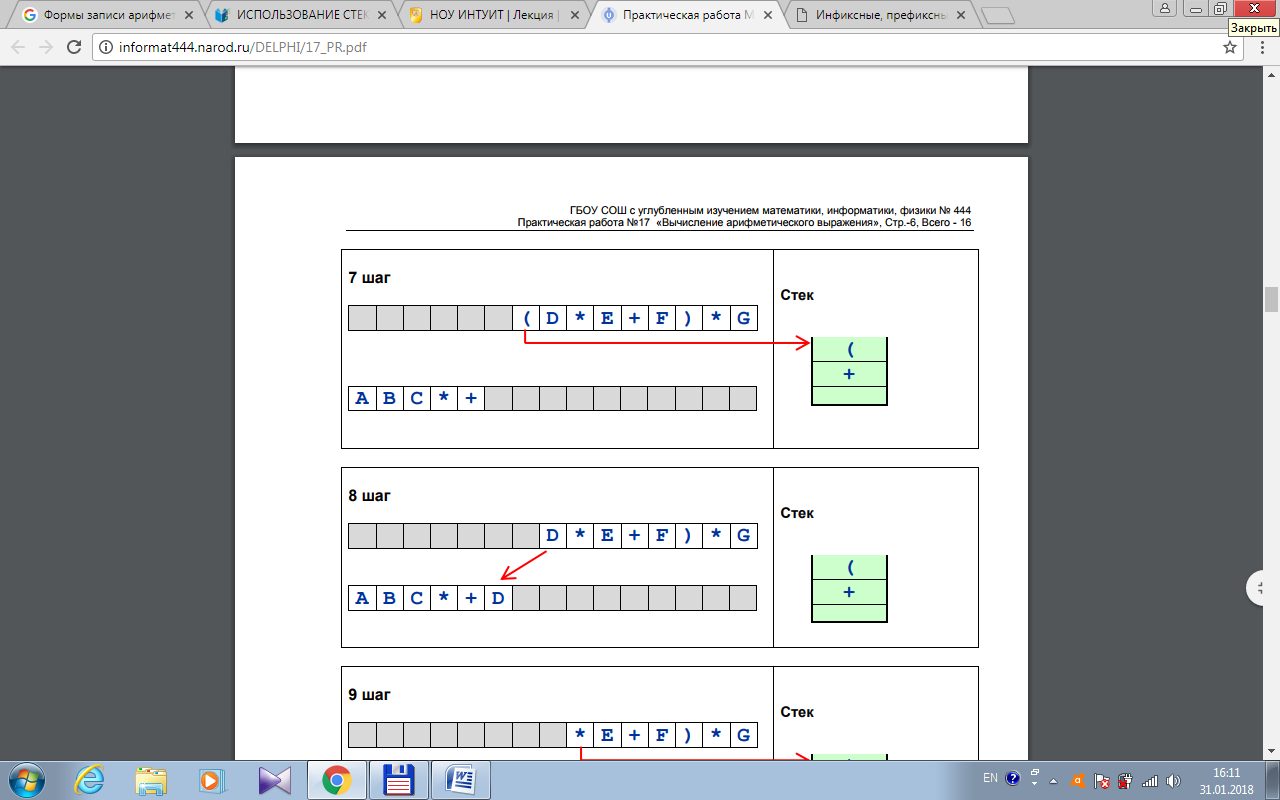
****

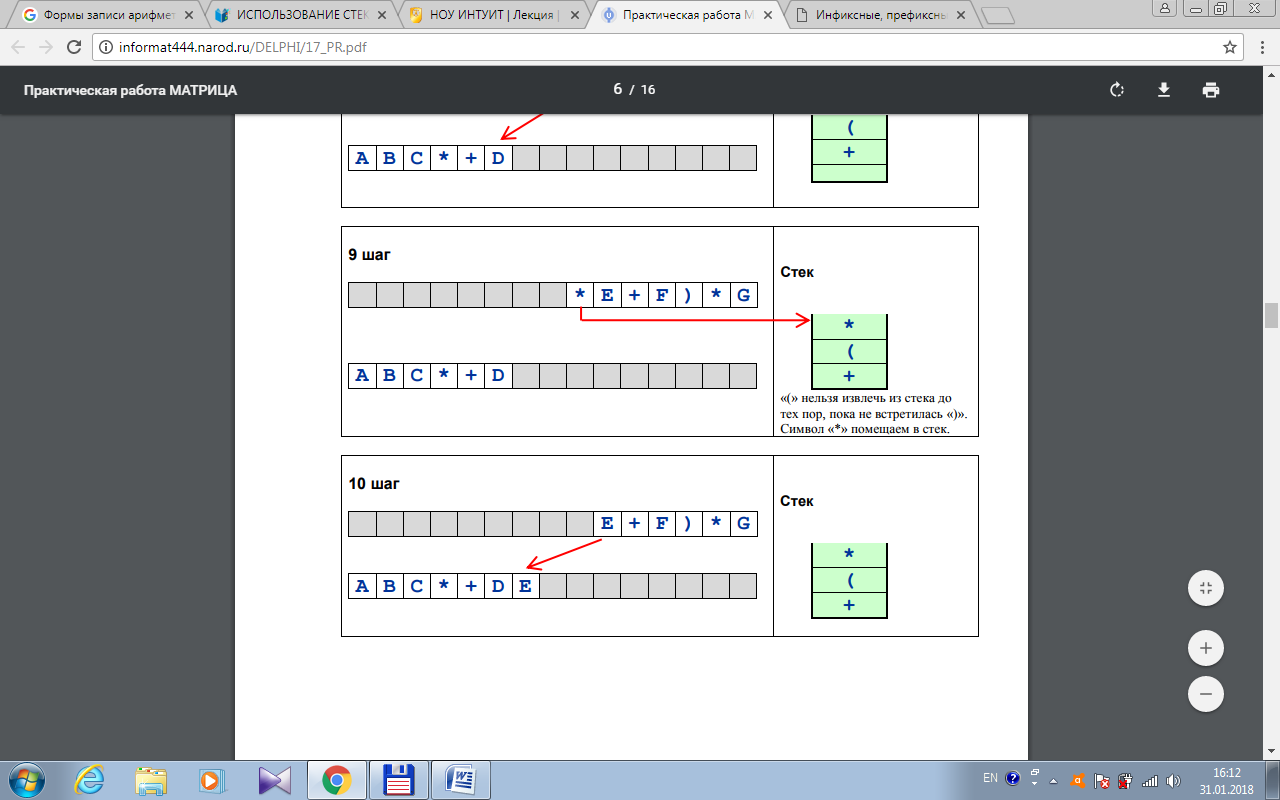
****

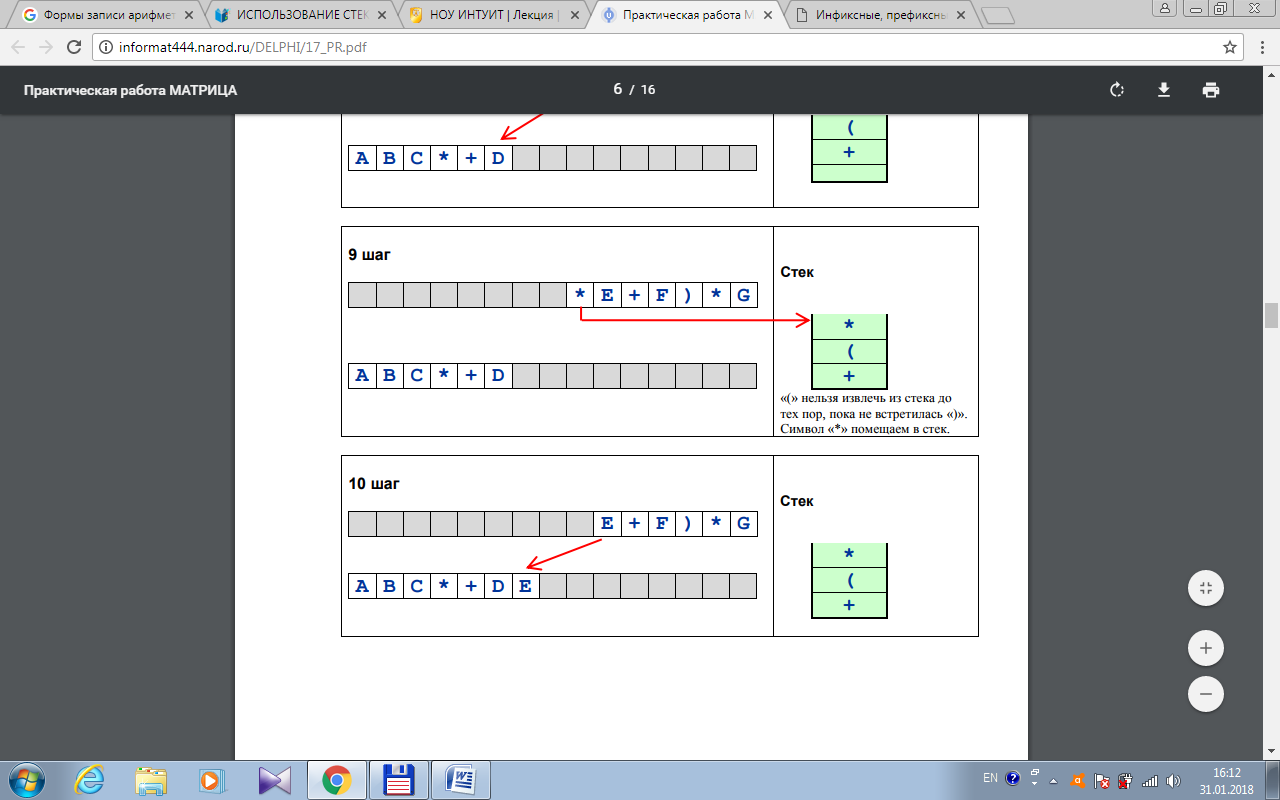
****

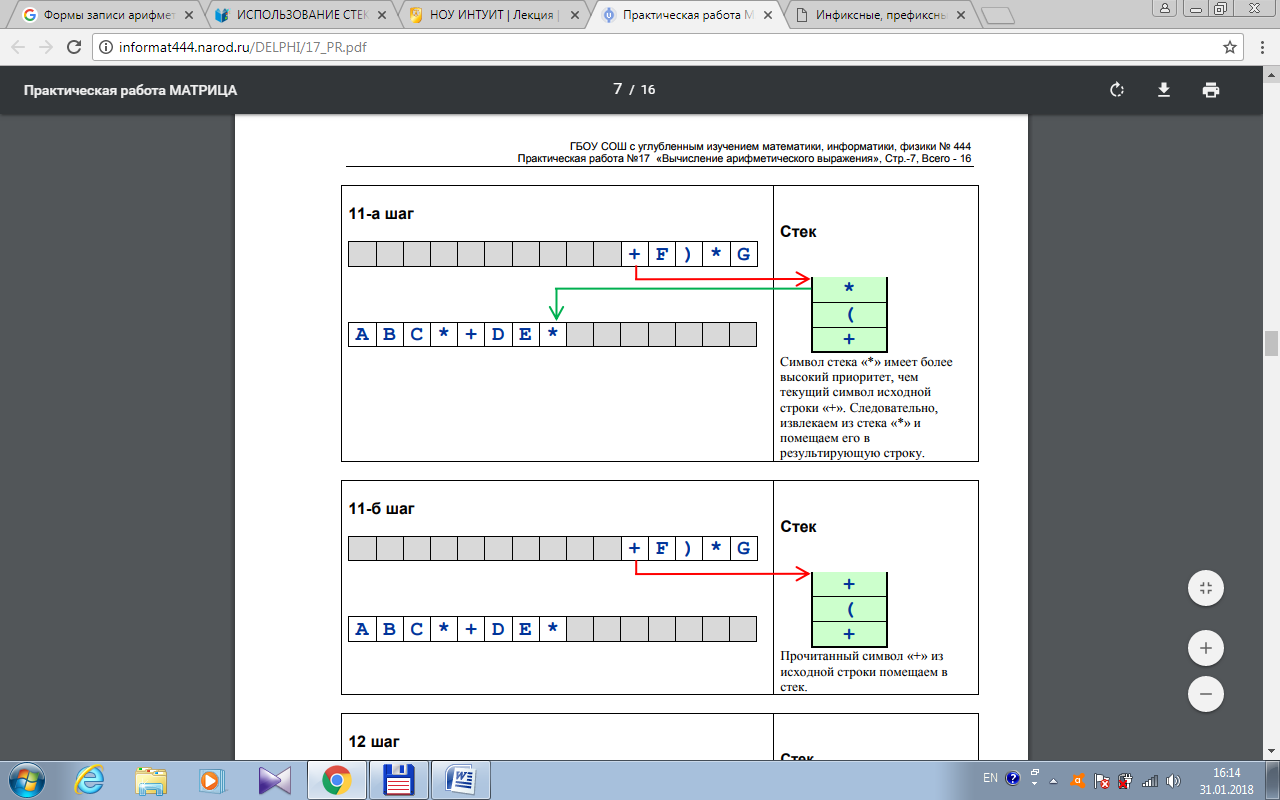
****

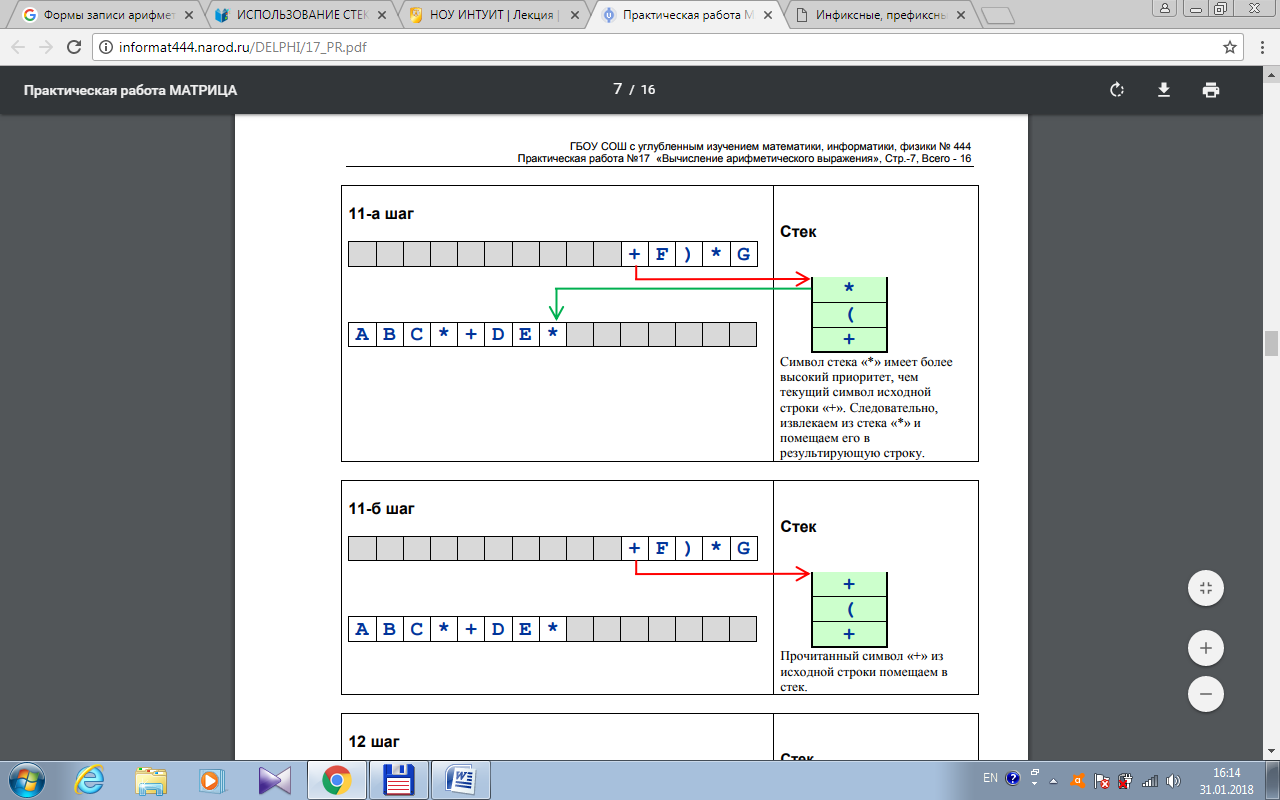
****

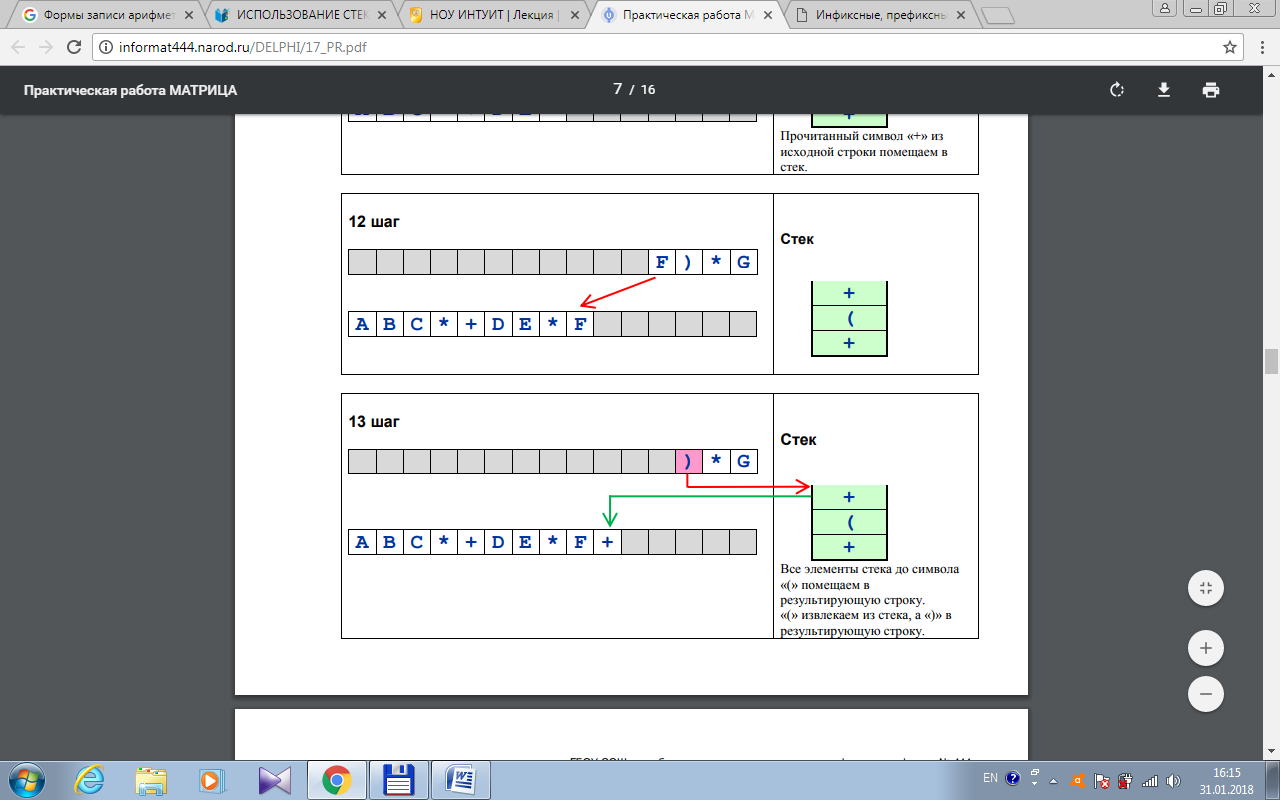
****

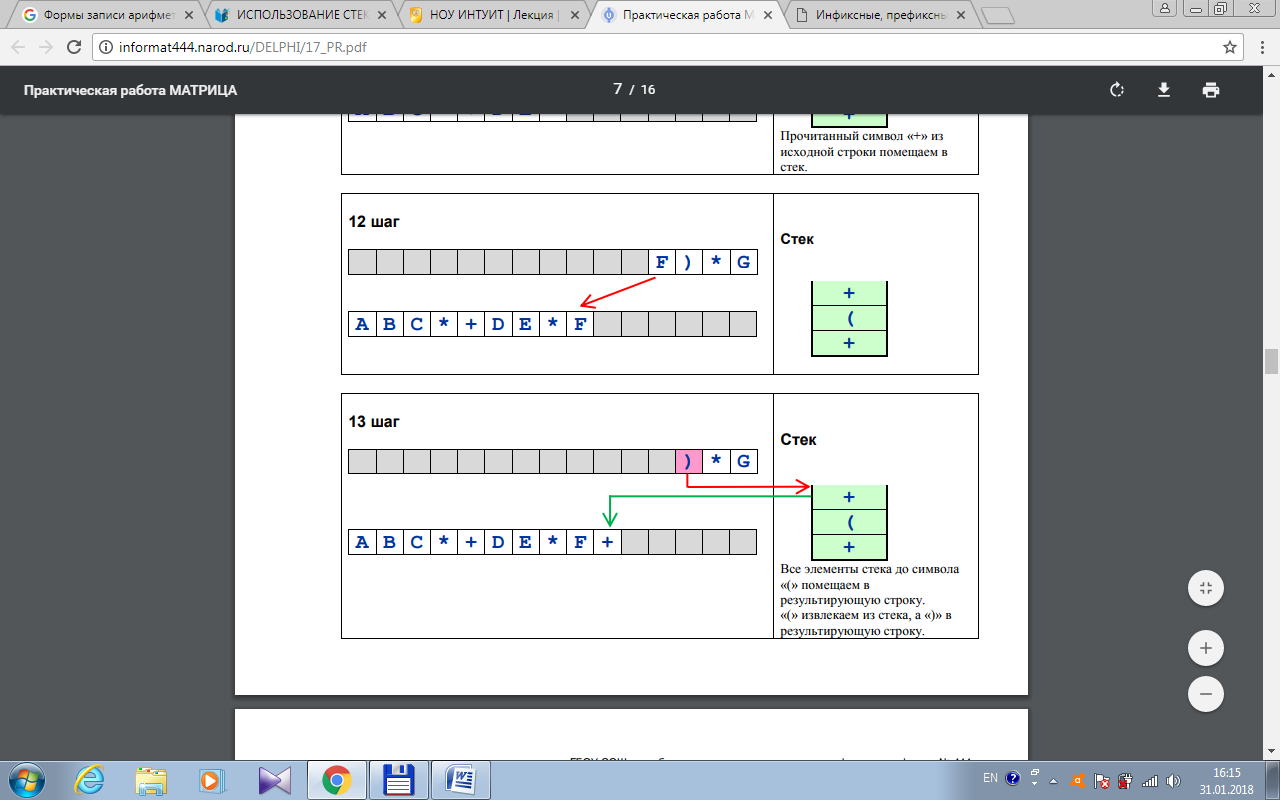
****

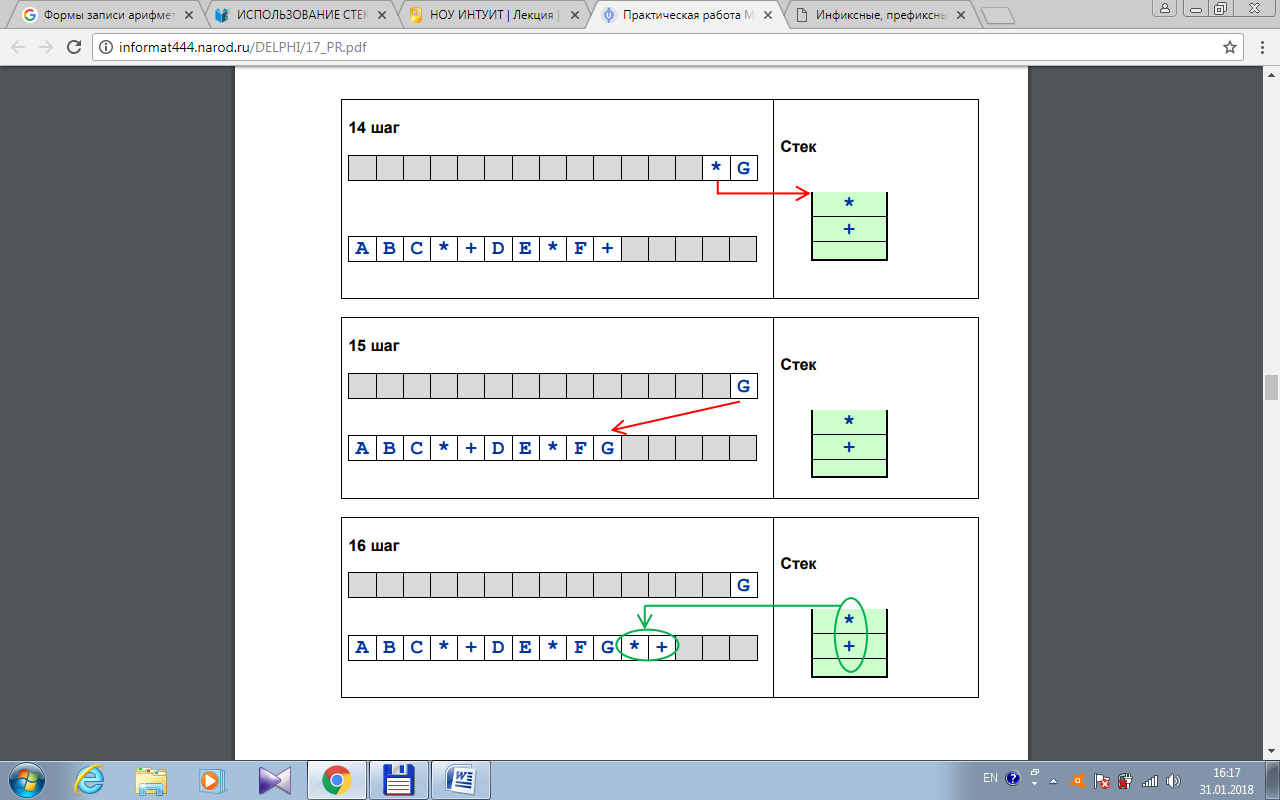
****

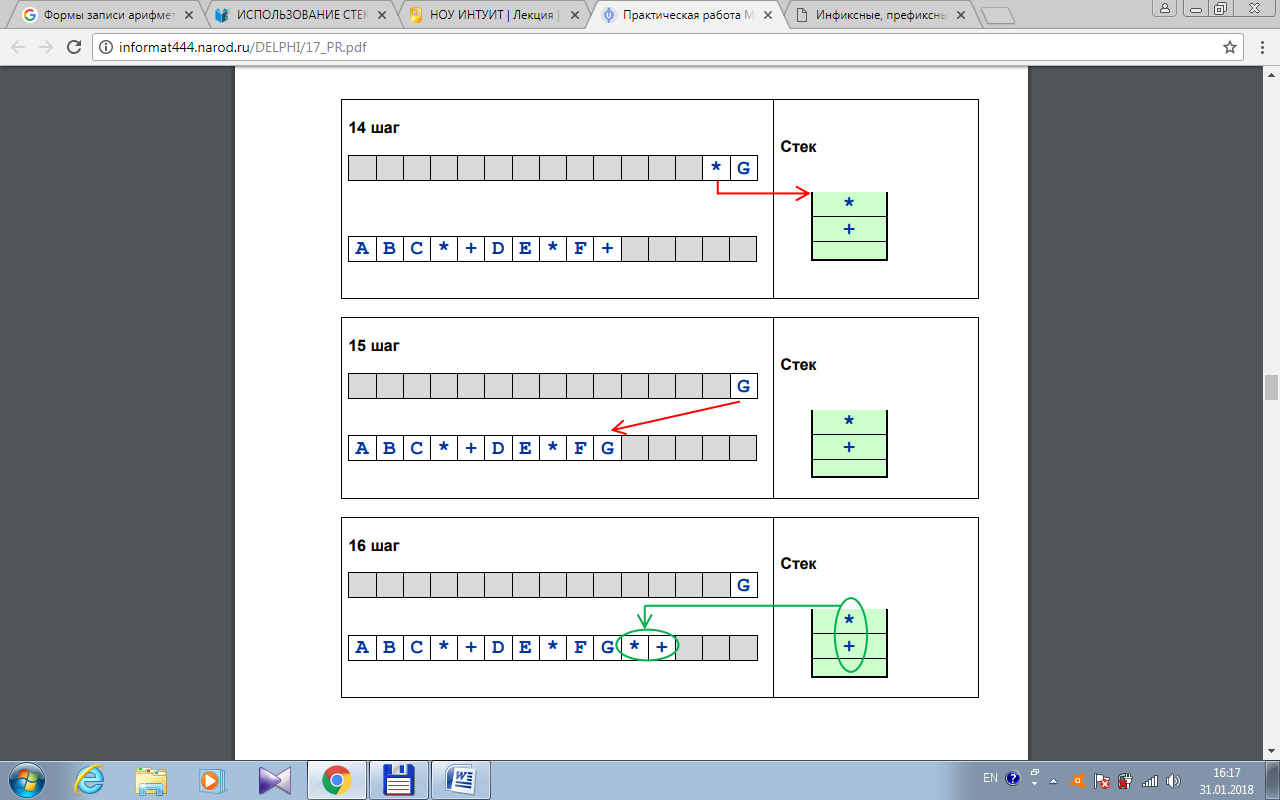
****

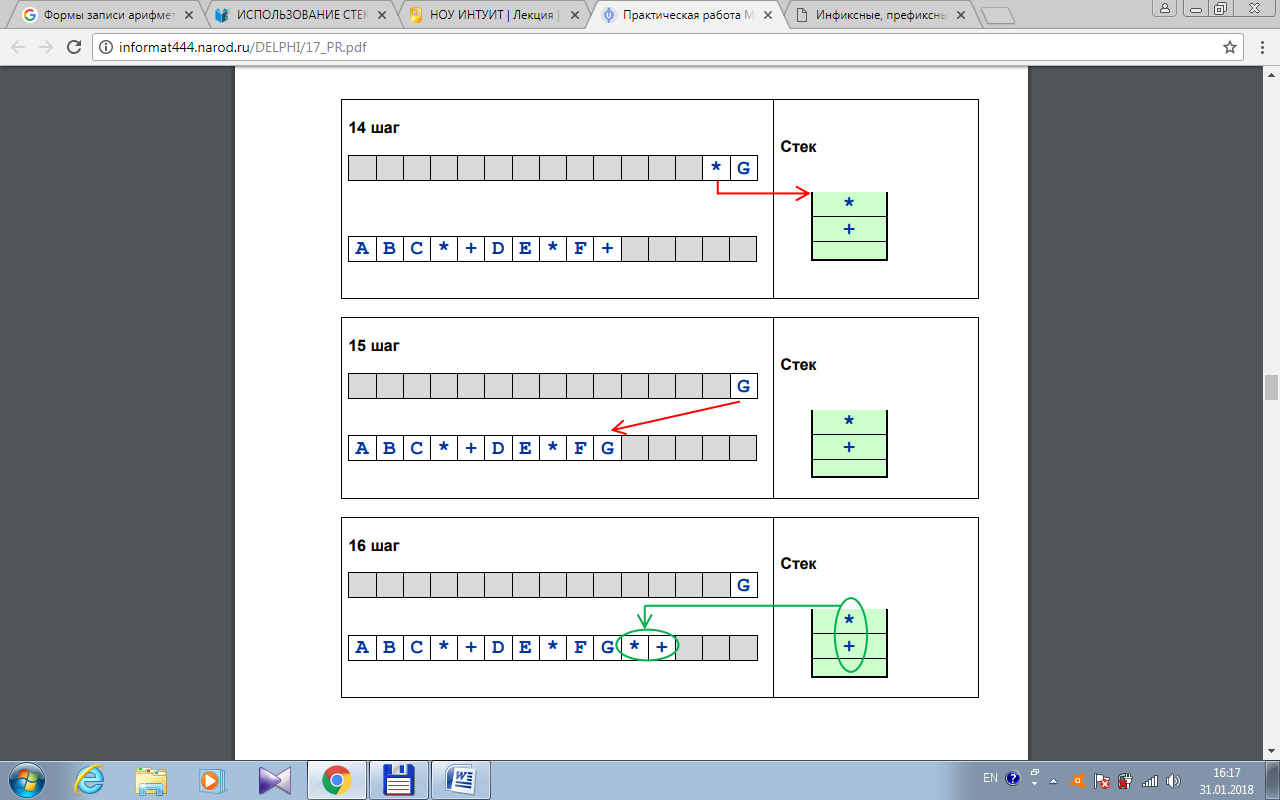
****

****

****

****

****

****

Постфиксная форма ABC\*+DE\*F+G\*+

При решении задачи будем использовать динамическую память – стек.

Для работы со стеком будем использовать следующие процедуры и функцию:

1) разместите в блоке реализации описание типов и переменных:

**Type pt\_1=^el\_1;**

**el\_1=Record //описание стека для первой части программы**

**data:Char;**

**next:pt\_1;**

**End;**

**pt\_2=^el\_2;**

**el\_2=Record //описание стека для второй части программы**

**data:Real;**

**next:pt\_2;**

**End;**

**Var s:String;**

**f:Boolean;**

**rez\_1:String;**

**rez\_2:Real;**

2) запись символа в стек:

**Procedure WriteStack\_1(Var u:pt\_1; dig:Char);**

**//запись в стек**

**//u – указатель на начало стека**

**//dig – значение символа, которое записывается в стек**

**Var x: pt\_1;**

**Begin**

**New(x);**

**x^.data:=dig;**

**x^.next:=u;**

**u:=x;**

**End;**

3) извлечение элемента из стека. *В результате выполнения процедуры параметру* ***dig*** *присваивается значение первого элемента стека и изменяется значение указателя на начало списка:*

**Procedure ReadStack\_1(Var u:pt\_1; Var dig:Char);**

**//извлечение из стека**

**//u – указатель на начало стека**

**//dig – значение символа, которое извлекается из стека**

**Var x: pt\_1;**

**Begin**

**dig:=u^.Data;**

**x:=u;**

**u:=u^.Next;**

**Dispose(x);**

**End;**

4) определение приоритета операции. *Результатом выполнения функции является значение приоритета операции. Эту функцию будем использовать при записи операции в стек:*

**Function Priority(c: Char): Byte;**

**Var R:Byte;**

**Begin**

**R := 0;**

**Case c of**

**'+', '-': R:= 1;**

**'\*', '/': R:= 2;**

**end;**

**Priority:=R;**

**End;**

5) определение есть ли записи в стеке. *Результатом выполнения функции является значение* ***True****, если стек не пуст и* ***False****, если стек пуст. Эту функцию будем использовать перед процедурой* ***ReadStack****:*

**Function Free\_1(u:pt\_1):Boolean;**

**//u – указатель на начало стека**

**Begin**

**If u=Nil Then Free\_1:=False**

**Else**

**Free\_1:= True;**

**End;**

6) преобразование исходного арифметического выражения в строку, представленную в постфиксной форме:

**Procedure Convert\_expression (a:String;Var z:String);**

**Var head:pt\_1;**

**i:Integer;**

**w:Char;**

**Begin**

**head:=Nil;**

**z:='';**

**i:=1;**

**While i<=Length(a) Do**

**Begin**

**Case a[i] of**

**'0'..'9': z:=z+a[i];**

**'(': WriteStack\_1(head,a[i]);**

**')': Begin //Считываем из стека до символа «(»**

**ReadStack\_1(head,w);**

**While w<>'(' Do**

**Begin z:=z+w;**

**ReadStack\_1(head,w);**

**End;**

**End;**

**'+','-','\*','/':**

**Begin**

**If not Free\_1(head) Then WriteStack\_1(head,a[i])**

**else**

**begin**

**w:=head^.data;**

**While Free\_1(head)and(Priority(head^.data)>= Priority(a[i])) do**

**Begin**

**ReadStack\_1(head,w);**

**z:=z+w;**

**End;**

**WriteStack\_1(head,a[i]) ;**

**end;**

**end;**

**End;**

**Inc(i);**

**End;**

**//Дополняем строку символами операций, запомненных в стеке**

**While Free\_1(head) Do**

**Begin**

**ReadStack\_1(head,w);**

**z:=z+w;**

**End;**

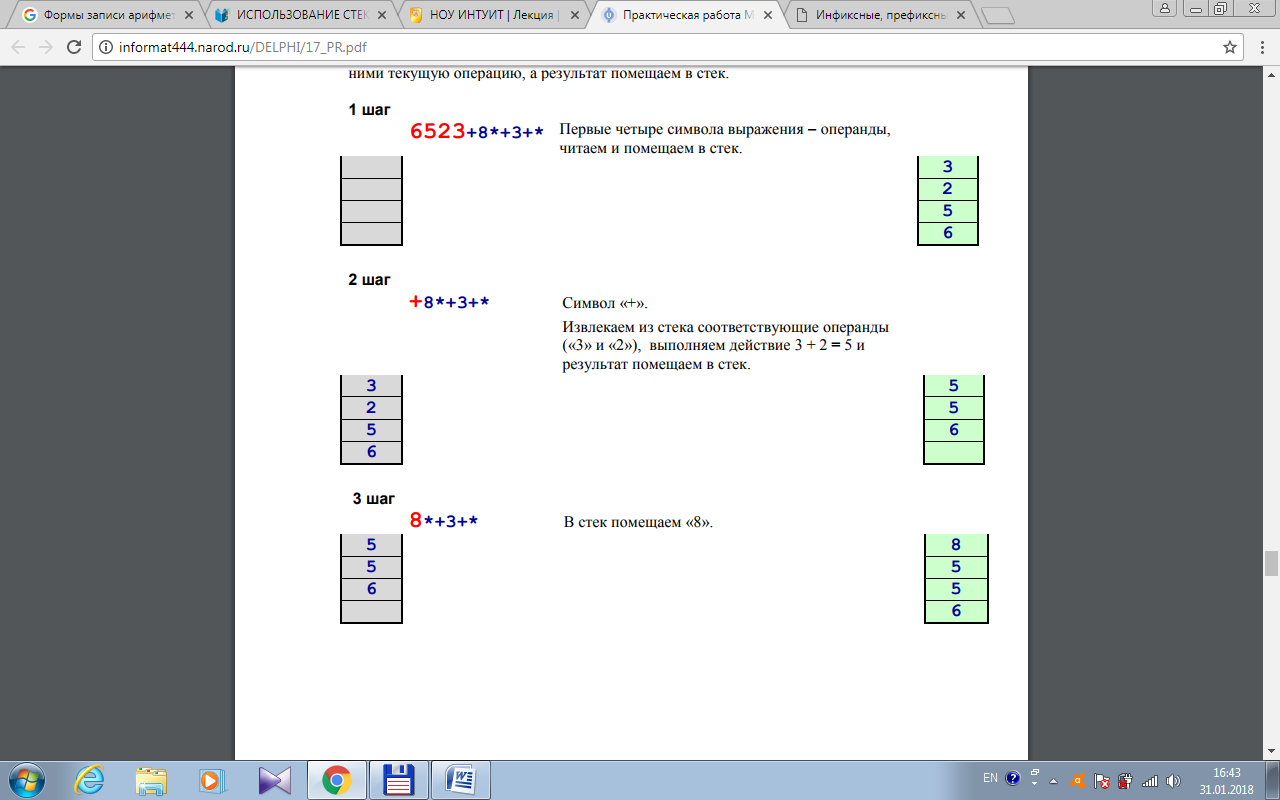
**End;**

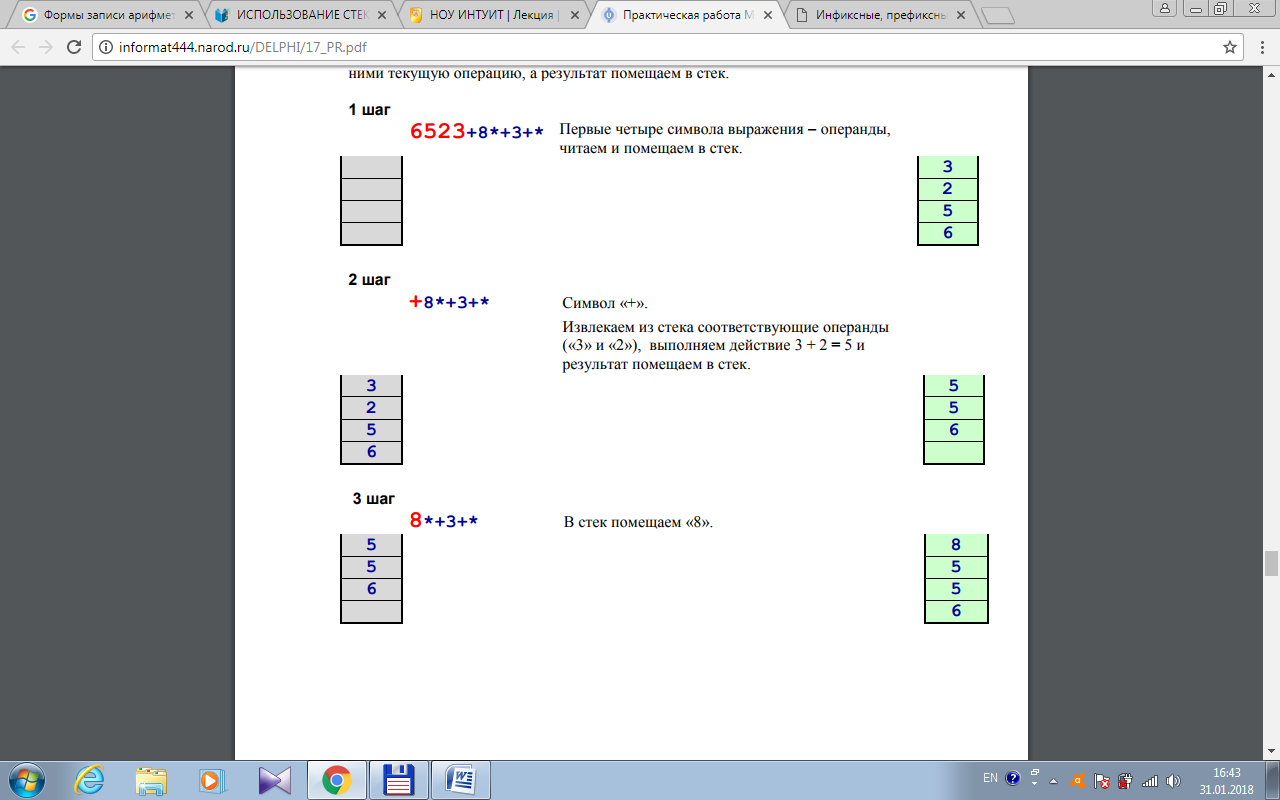
**Вычисление выражения, записанного в постфиксной форме**

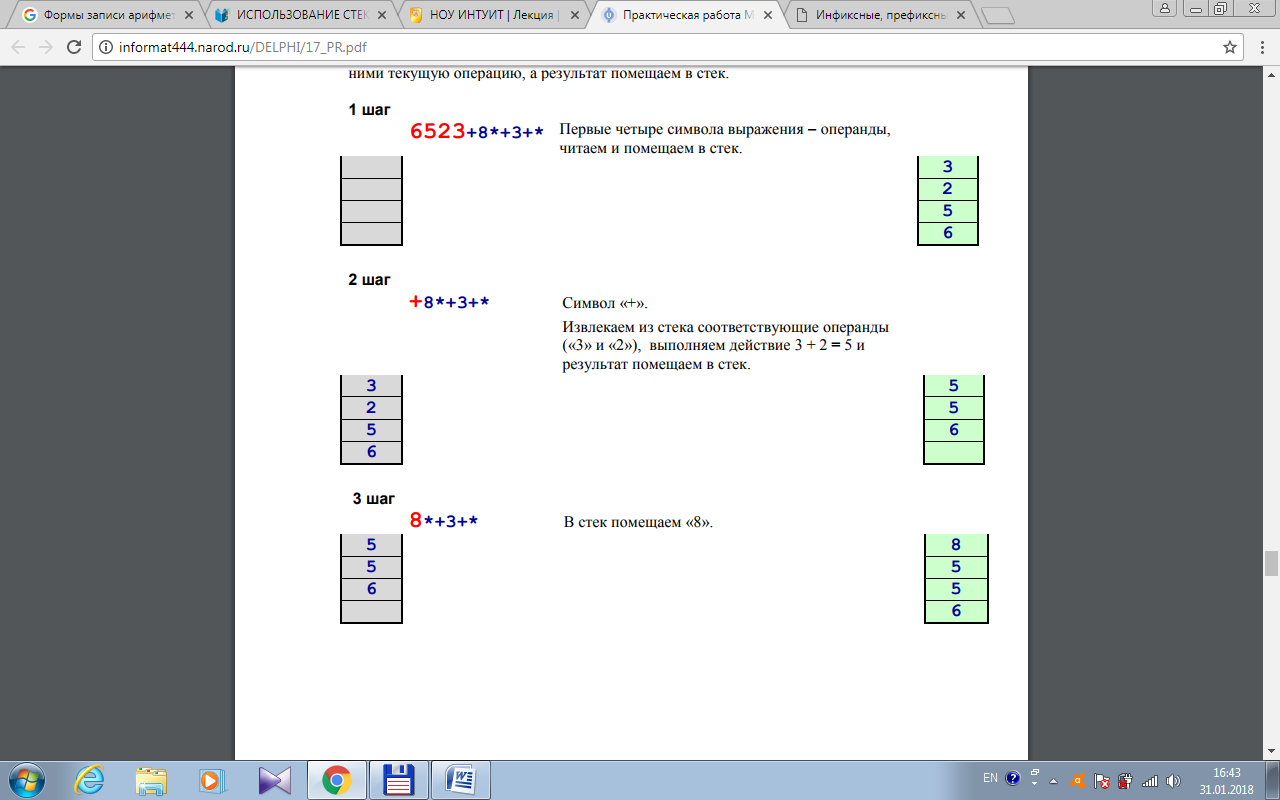
Алгоритм решения задачи с использованием стека:

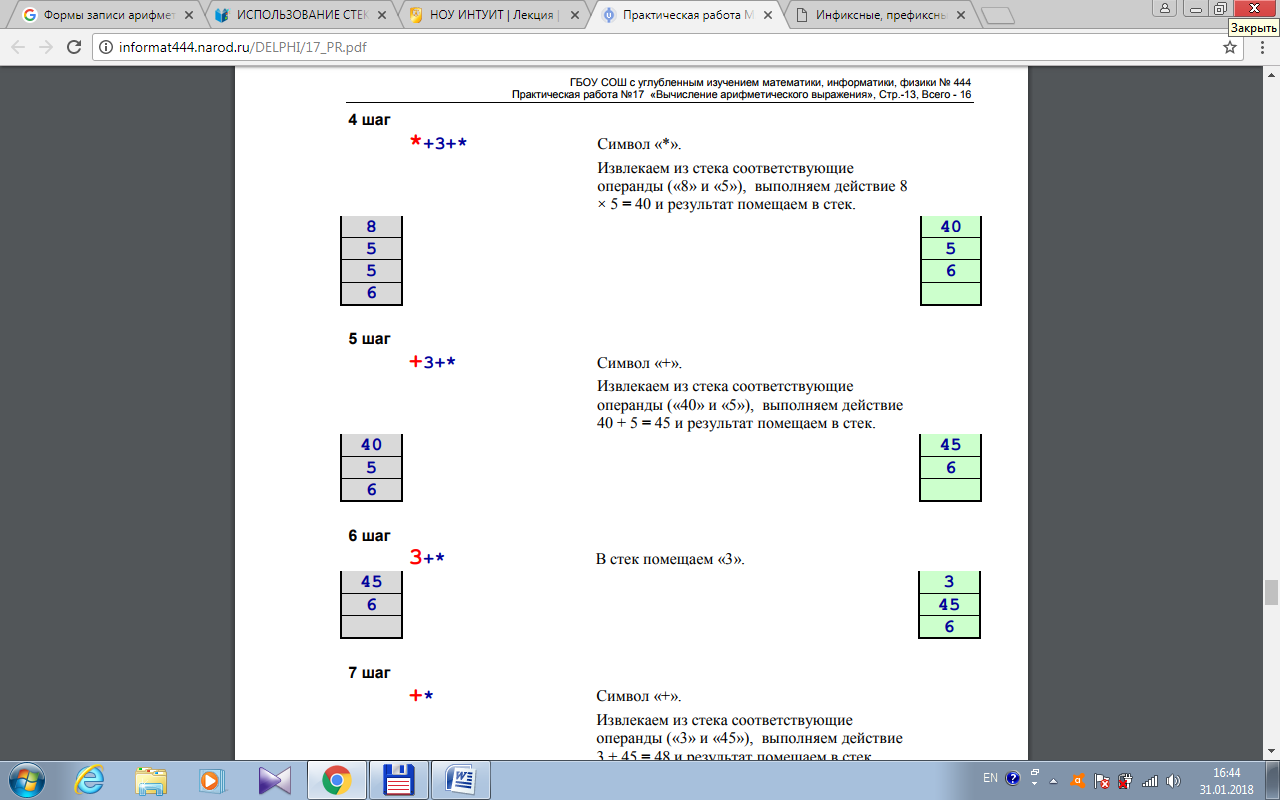
*В постфиксной записи каждая операция относится к двум предшествующим операндам. Один из этих операндов может быть результатом операции, выполненной ранее.*

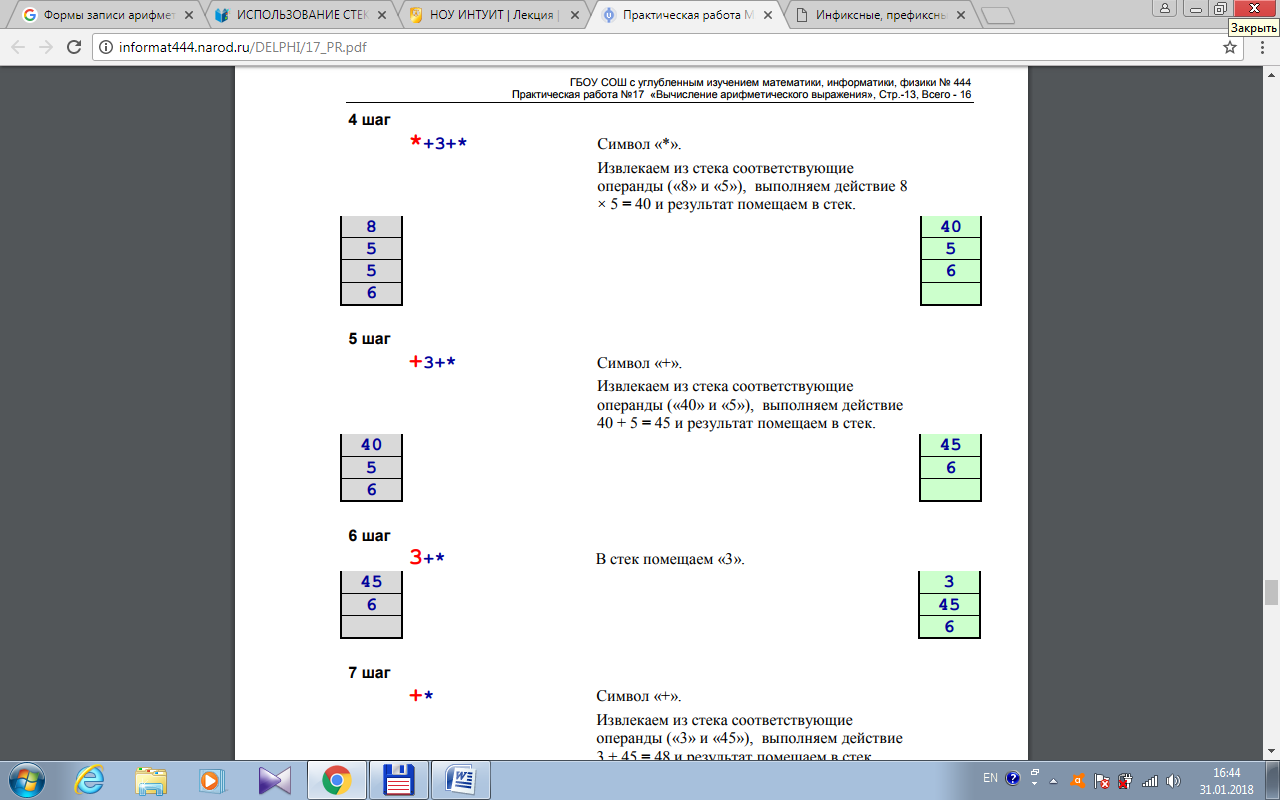
*Будем записывать в стек каждый встретившийся операнд. Если встречается операция, то ее операндами будут два верхних элемента стека. Извлекаем эти элементы, выполняем над ними текущую операцию, а результат помещаем в стек.*

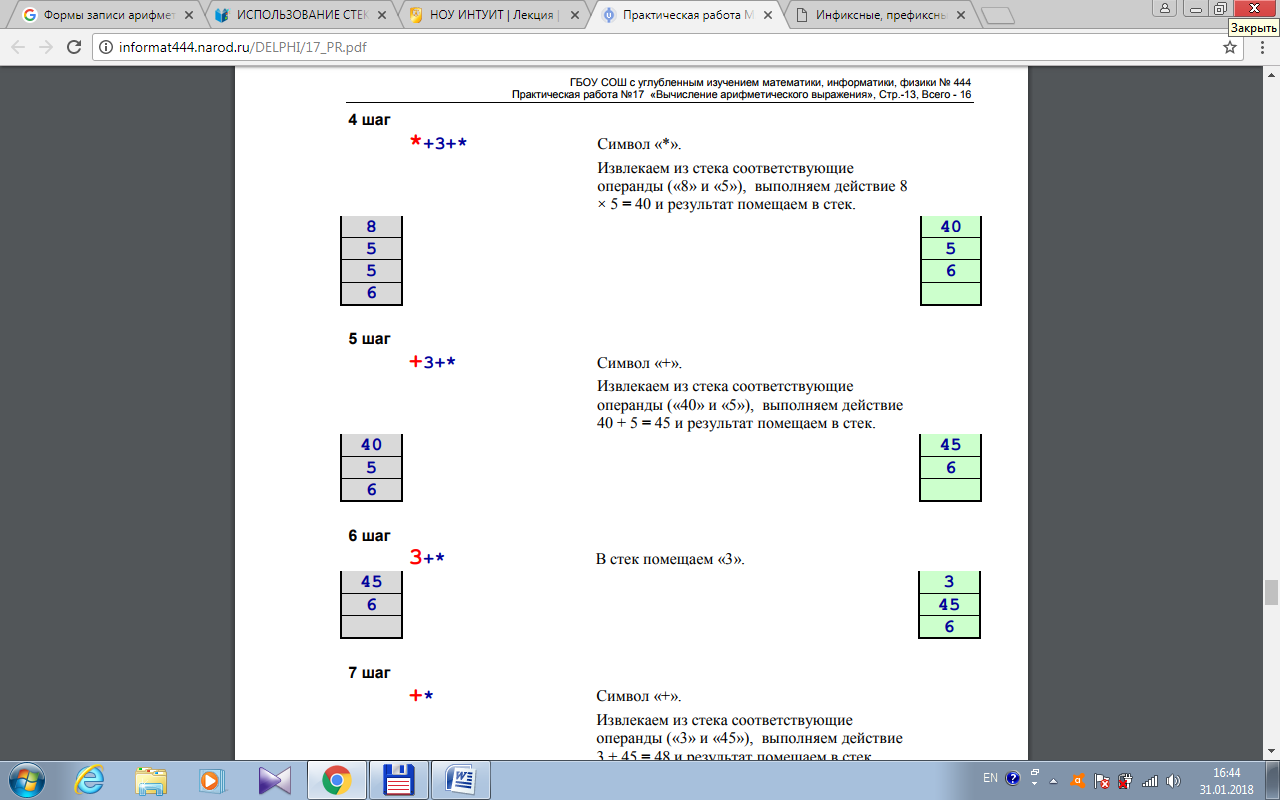
****

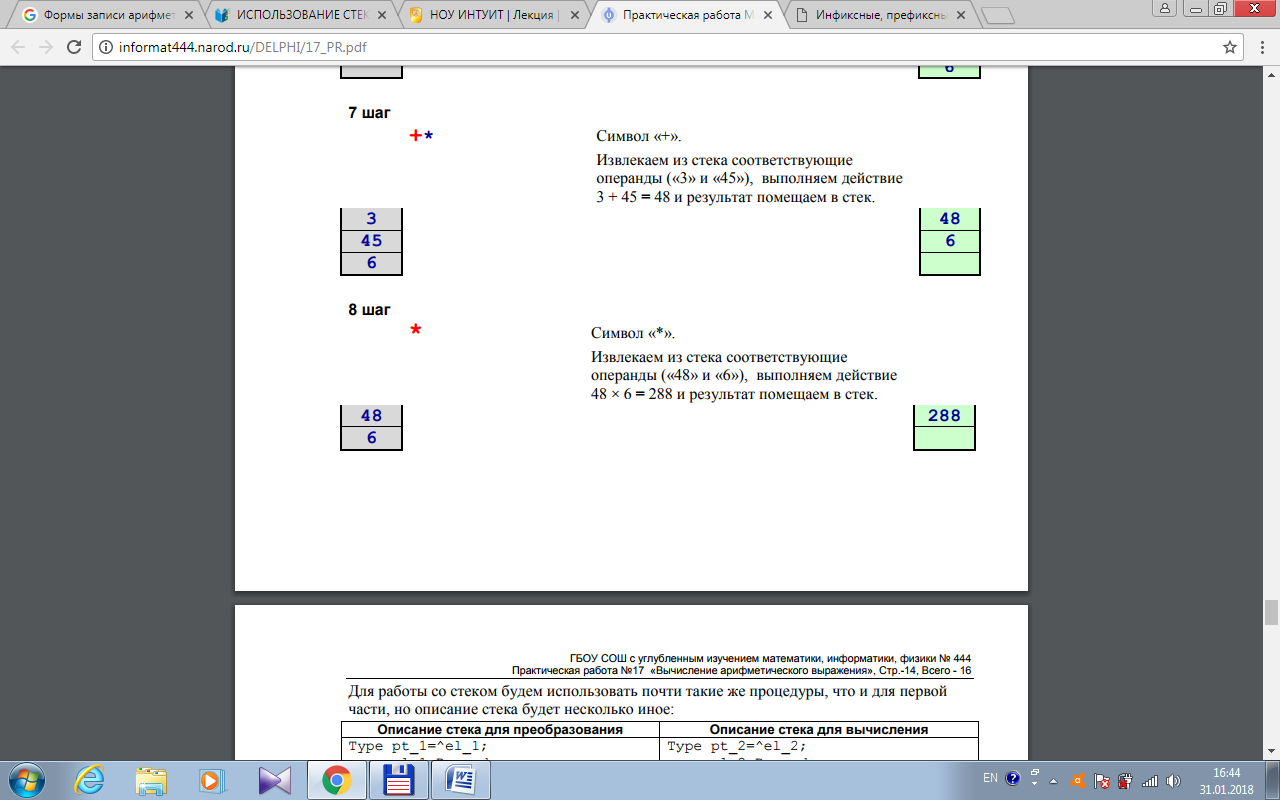
****

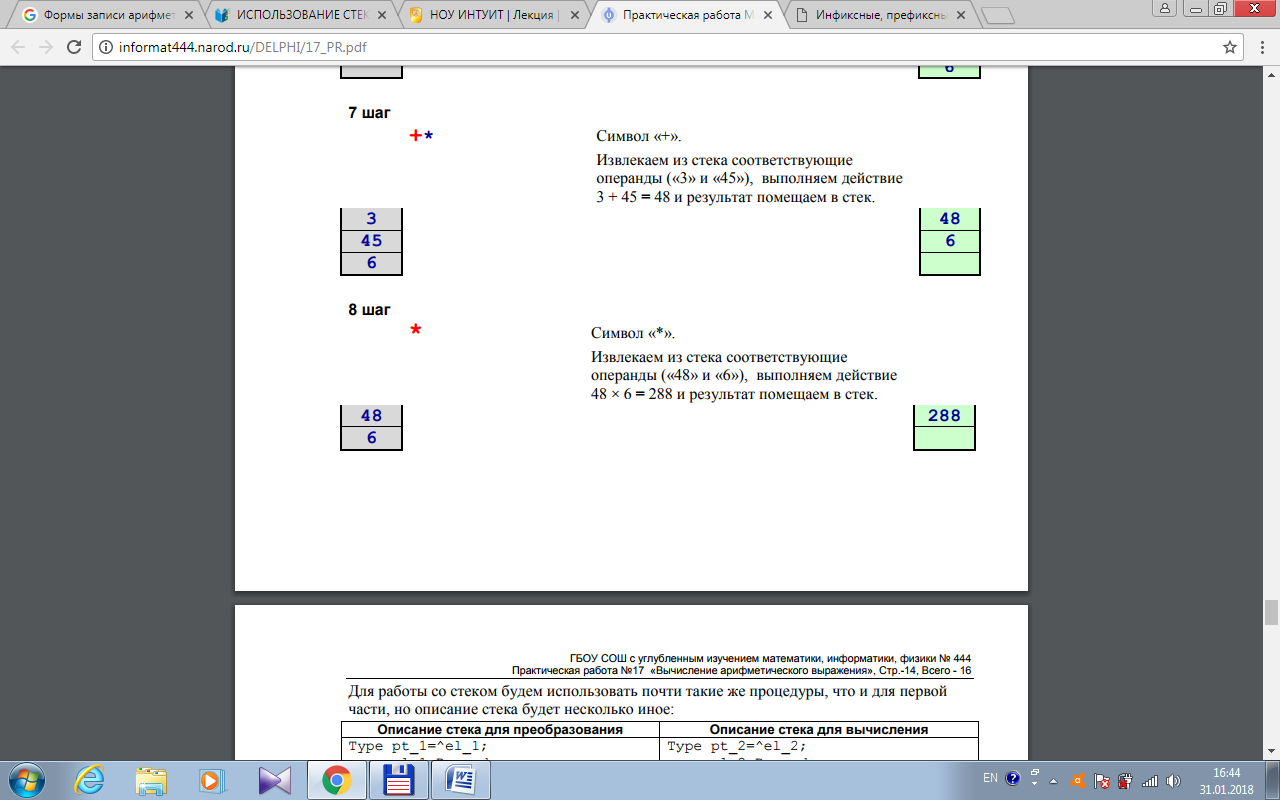
****

****

****

****

****

****

*Результат выполнения последней операции является результатом вычисления всего выражения.*

*Для работы со стеком будем использовать почти такие же процедуры, что и для первой части, но описание стека будет несколько иное:*

|  |  |
| --- | --- |
| Описание стека для преобразования | Описание стека для вычисления |
| **Type pt\_1=^el\_1;**  **el\_1=Record**  **data:Char;**  **next:pt\_1;**  **End;** | **Туре pt\_2=^el\_2;**  **el\_2=Record**  **data:Real;**  **next:pt\_2;**  **End;** |

1) запись символа в стек:

**Procedure WriteStack\_2(Var u:pt\_2; dig:Real);**

**//запись в стек**

**//u – указатель на начало стека**

**//dig – значение символа, которое записывается в стек**

**Var x: pt\_2;**

**Begin**

**New(x); x^.data:=dig;**

**x^.next:=u;**

**u:=x;**

**End;**

2) извлечение элемента из стека. *В результате выполнения процедуры параметру dig присваивается значение первого элемента стека и изменяется значение указателя на начало списка:*

**Procedure ReadStack\_2(Var u:pt\_2; Var dig:Real);**

**//извлечение из стека**

**//u – указатель на начало стека**

**//dig – значение символа, которое извлекается из стека**

**Var x: pt\_2;**

**Begin**

**dig:=u^.Data;**

**x:=u;**

**u:=u^.Next;**

**Dispose(x);**

**End;**

3) определение, есть ли записи в стеке. *Результатом выполнения функции является значение* ***True****, если стек не пуст и* ***False****, если стек пуст. Эту функцию будем использовать перед процедурой* ***ReadStack****:*

**Function Free\_2(u:pt\_2):Boolean;**

**//u – указатель на начало стека**

**Begin**

**If u=Nil Then Free\_2:=False**

**Else**

**Free\_2:= True;**

**End;**

4) преобразование исходного арифметического выражения в строку, представленную в постфиксной форме:

**Procedure Convert\_calculate(a:String;Var pp:Boolean;Var z:Real);**

**Var head: pt\_2;**

**i,k: Integer;**

**r,w: Real;**

**Begin**

**head:=Nil;**

**pp:=True;**

**i:=1;**

**While (i<=Length(a)) And pp Do**

**Begin**

**If a[i]<>' ' Then Begin{\*Пропускаем пробелы.\*}**

**If Not(a[i] In ['+','-','\*','/'])**

**Then Begin**

**Val(a[i],r,k);**

**If k=0 Then WriteStack\_2(head,r) Else pp:=False;**

**End**

**Else Begin**

**If Free\_2(head) Then ReadStack\_2(head,r) Else pp:=False;**

**If Free\_2(head) Then ReadStack\_2(head,w) Else pp:=False;**

**If pp Then Begin**

**Operation(a[i],r,w,r);**

**WriteStack\_2(head,r);**

**End;**

**End;**

**End;**

**Inc(i);**

**End;**

**If Free\_2(head) Then ReadStack\_2(head,z) Else pp:=False;**

**End;**

**Перевод из инфиксной формы записи выражения в префиксную форму**

Инфиксное выражение сканируется справа налево, и префиксная строка строится также справа налево. Алгоритм преобразования такой же, как и при преобразовании в постфиксную форму, только открывающие скобки меняются на закрывающие и, наоборот. При определении приоритета операции отношение «<» изменяется на «<», чтобы равноприортетные операции выполнялись слева направо.

Вычисление значения выражения по его префиксной записи. Алгоритм вычисления полностью совпадает с алгоритмом вычисления по постфиксной записи, но префиксная строка сканируется справа налево.

3. С использованием стека вычислить арифметическое выражение, предварительно преобразовав его в постфиксную форму записи.

4\*. С использованием стека вычислить арифметическое выражение, предварительно преобразовав его в префиксную форму записи.

**5. Порядок выполнения работы**

1. Создать текстовый файл, содержащий 3 строки символов. Используя стек, вывести содержимое текстового файла следующим образом: литеры каждой строки в обратном порядке.

2. Выполнить задание по варианту. При этом доступ к элементам стека должен осуществляться только с помощью стандартных для данной структуры операций.

3. Преобразовать инфиксную форму записи арифметического выражения (задание взять у преподавателя) в постфиксную любым способом. С использованием стека вычислить результат выражения. Ответ представить следующим образом:

*Исходная строка 5+4\*3+(4\*2+1)\*6*

*Результирующая строка 543\*+42\*1+6\*+*

*Результат 71*

4. Преобразовать инфиксную форму записи арифметического выражения (задание взять у преподавателя) в префиксную любым способом. С использованием стека вычислить результат выражения. Ответ представить следующим образом:

*Исходная строка 5+4\*3+(4\*2+1)\*6*

*Результирующая строка 543\*+42\*1+6\*+*

*Результат 71*

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы и задания**

1. Что такое стековый принцип сохранения элементов?

2. Какие основные операции реализуются для стеков?

3. Какие шаги выполняются при добавлении элемента в стек-массив?

4. Какие шаги выполняются при удалении элемента из стека-массива?

5. Какую структуру должен иметь элемент стека при динамической реализации?

6. Как между собой связываются соседние элементы стека?

7. Какие типы данных необходимы для динамической реализации стека?

8. Какие переменные необходимы для реализации операций с динамическим стеком?

9. Приведите достоинства и недостатки динамической и статической реализации стека.

**8. Рекомендуемая литература**

1.Ахо, А.В.Структуры данных и алгоритмы / А.В. Ахо, Дж.Хопкрофт, Дж. Д.Ульман. – пер. с англ. – М.: Издательский дом "Вильямс", 2007.-400 с.

2. Вирт,Н. Алгоритмы и структуры данных / Н. Вирт. –СПб.:Невский диалект, 2008. – 352с.

3. Вирт, Н. Алгоритмы и структуры данных с примерами на Паскале / Н. Вирт[перевод с английского Д. Б. Подшивалова]; – 2-е изд., испр., – СПб.: Невский Диалект, 2005. – 352с.

4. Гагарина, Л.Г. Алгоритмы и структуры данных / Л.Г. Гагарина, В.Д. Колдаев. – учеб.пособие – М: Финансы и статистика, 2009. – 304с.

5. Котов, В.М. Алгоритмы и структуры данных: учеб.пособие / В.М. Котов, Е.П. Соболевская, А.А. Толстиков – Минск: БГУ, 2011. – 267с.

6. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл – 2-е дополненное издание – М.:Техносфера, 2006. – 368с.

7. Окулов, С.М. Программирование в алгоритмах / С.М. Окулов. – 3-е изд. – М.: БИНОМ. Лаборатория знаний, 2007. – 383с.