МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Дополнительное задание**

по дисциплине: Основы программирования

тема: «Обратная польская запись»

Выполнил: ст. группы ПВ-202 Буйвало Анастасия Андреевна

ВТ-202 Бездольников Владимир Романович

Проверил:

Притчин Иван Сергеевич

Белгород 2020 г.

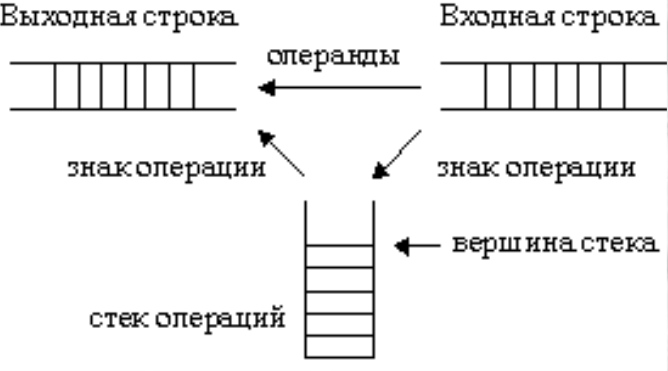
**Алгоритм построения и вычисления Обратной польской записи**

Обратная польская запись была создана для того, чтобы проще вычислять математические выражения, при решении с помощью ОПЗ не нужно задумываться над приоритетами операций, также в ней отсутствуют скобки, и она намного удобнее для машинного вычисления.

Для составления ОПЗ нам понадобится стек операций и массив выхода. Далее нужно определить приоритет операций. Это необходимо для правильного распределения порядка математических действий, чтобы, например, отдавать предпочтение умножению перед сложением.  
Высоким приоритетом обладают операции умножения и деления, а низким сложения и вычитания   
Все числа и переменные записываются в массив выхода.  
Знаки действий могут находиться как в стеке операций, так и в массиве выхода. Их расположение зависит от того, что находится последним в стеке.

Ели текущий знак является действием, а последний знак из стека операций имеет приоритет ниже или равный, то последний знак из стека уходит в массив выхода, а текущий добавляется в стек. Иначе все добавляется как обычно. Чтобы операция попала в массив выхода нужен знак с меньшим, или с таким же приоритетом на входе.

Идем по порядку, слева направо.



**Описание структур данных**

const MAX\_N = 256;

s\_signs = ['+', '-', '\*', '/'];

N = 26;

type t\_range = 1..MAX\_N;

t\_arr\_ch = array [1..N] of char;

t\_ch\_stack = record

el: array[t\_range] of char;

size : t\_range;

end;

t\_real\_stack = record

el: array[t\_range] of real;

size : t\_range;

end;

t\_complex = record

mat : real;

imagi : real;

end;

t\_arr\_comp = array [1..N] of t\_complex;

Спецификация.

1. Заголовок: function get\_word(s : string; var i : byte):string;
2. Назначение: возращает слово строки s, начиная с i-той позиции
3. Входные параметры: s, i.
4. Выходные параметры: i.

Спецификация.

1. Заголовок: procedure ch\_push(var stack : t\_ch\_stack; b : string)
2. Назначение: помещает элемент b на вершину стека stack
3. Входные параметры: stack, b.
4. Выходные параметры: stack.

Спецификация.

1. Заголовок: procedure real\_push(var stack : t\_real\_stack; b : real)
2. Назначение: помещает элемент b на вершину стека stack
3. Входные параметры: stack, b.
4. Выходные параметры: stack.

Спецификация.

1. Заголовок: function is\_empty(stack : t\_ch\_stack): boolean
2. Назначение: возвращает значение «истина», если стек stack пустой и значение «ложь», если нет
3. Входные параметры: stack.
4. Выходные параметры: нет.

Спецификация.

1. Заголовок: function ch\_pop(var stack : t\_ch\_stack) : char
2. Назначение: возвращает верхний элемент из стека stack и удаляет его из стека stack
3. Входные параметры: stack.
4. Выходные параметры: stack.

Спецификация.

1. Заголовок: function real\_pop(var stack : t\_real\_stack) : real
2. Назначение: возвращает верхний элемент из стека stack и удаляет его из стека stack
3. Входные параметры: stack.
4. Выходные параметры: stack.

Спецификация.

1. Заголовок: procedure clear\_ch\_stack(var s : string; var stack : t\_ch\_stack);
2. Назначение: записывает в строку s все элементы из стека stack
3. Входные параметры: s, stack.
4. Выходные параметры: s, stack.

Спецификация.

1. Заголовок: procedure unar\_minus(var s : string; i : byte)
2. Назначение: заменяет операцию унарного минуса на выражение 0 – и сдвигает строку s, начиная с i символа
3. Входные параметры: s, i.
4. Выходные параметры: s.

Спецификация.

1. Заголовок: procedure transform\_unar\_minus\_str (var s : string)
2. Назначение: поиск в строке s унарного минуса и преобразование его в бинарный
3. Входные параметры: s.
4. Выходные параметры: s.

Спецификация.

1. Заголовок: procedure opz\_sign\_write(var opz\_s : string; var stack : t\_ch\_stack; var fl1 : byte; sign : char);
2. Назначение: распределяет операцию sign либо в строку opz\_s, либо в стек stack, в зависимости от приоритета предыдущей операции fl1
3. Входные параметры: stack, fl1, sign.
4. Выходные параметры: opz\_s, stack, fl1.

Спецификация.

1. Заголовок: procedure bracket(var s, opz\_s : string; var i : byte)
2. Назначение: переводит обычную запись в скобках из строки s начиная с i-той позиции в ОПЗ и записывает в строку opz\_s
3. Входные параметры: s, i.
4. Выходные параметры: s, i, opz\_s.

Спецификация.

1. Заголовок: procedure opz (var opz\_s : string; s : string);
2. Назначение: преобразует выражение в строке s в обратную польскую запись и записывает в строку opz\_s
3. Входные параметры: s.
4. Выходные параметры: opz\_s.

Спецификация.

1. Заголовок: function check\_el(s : char; a : t\_arr\_ch): boolean
2. Назначение: возвращает значение «истина», если найдено s в массиве а, иначе «ложь»
3. Входные параметры: s, a.
4. Выходные параметры: нет.

Спецификация.

1. Заголовок: procedure record\_let(s: string; var a : t\_arr\_ch; var k : byte)
2. Назначение: запись в массив a не повторяющихся k букв из строки s
3. Входные параметры: s, a.
4. Выходные параметры: a, k.

Спецификация.

1. Заголовок: procedure sort\_arr\_let(var a : t\_arr\_ch; k : byte);
2. Назначение: сортировка массива переменных a длины k лексикографически
3. Входные параметры: a, k.
4. Выходные параметры: a.

Спецификация.

1. Заголовок: procedure read\_value(var arr\_comp\_val : t\_arr\_comp; arr\_var : t\_arr\_ch; k : byte)
2. Назначение: заполняет массив значений val, переменных, содержащихся в массиве arr\_var длины k
3. Входные параметры: arr\_var, k.
4. Выходные параметры: arr\_comp\_val.

Спецификация.

1. Заголовок: procedure complex\_plus(var c : t\_complex; a, b : t\_complex)
2. Назначение: сложение комплексных чисел a и b и запись результата в переменную c
3. Входные параметры: a, b.
4. Выходные параметры: c.

Спецификация.

1. Заголовок: procedure complex\_plus(var c : t\_complex; a, b : t\_complex)
2. Назначение: вычитание комплексных чисел a и b и запись результата в переменную c
3. Входные параметры: a, b.
4. Выходные параметры: c.

Спецификация.

1. Заголовок: procedure complex\_plus(var c : t\_complex; a, b : t\_complex)
2. Назначение: умножение комплексных чисел a и b и запись результата в переменную c
3. Входные параметры: a, b.
4. Выходные параметры: c.

Спецификация.

1. Заголовок: procedure complex\_plus(var c : t\_complex; a, b : t\_complex)
2. Назначение: деление комплексных чисел a и b и запись результата в переменную c
3. Входные параметры: a, b.
4. Выходные параметры: c.

Спецификация.

1. Заголовок: procedure calc\_comp\_opz (var rez : t\_complex ; arr\_comp\_val : t\_arr\_comp; opz\_s : string);
2. Назначение: нахождение значения выражения опз opz\_s для значений переменных массива arr\_comp\_val и запись ответа в переменную rez
3. Входные параметры: arr\_comp\_val, opz\_s.
4. Выходные параметры: rez.

**Тестовые данные:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Выражение | ОПЗ | Значения переменных | Результат |
| 1 | 3 + a \* b | 3 a b \* + | a = 5  b = 2 | 13 |
| 2 | 249 + 3.5 | 249 3.5 + | - | 252,5 |
| 3 | с \* ( f + x ) | cfx+\* | c = 3  f = 1  x = 2 | 9 |
| 4 | a \* c | a c \* | a = 2i  c = 3 +2 i | -4 + 6i |
| 5 | a + ( a \* b \* ( c + d ) + ( a + b ) ) | a a b \* c d + \* a b + + + | a = 1  b = 2  c = 3  d = 4 | 18 |
| 6 | a + ( a + b \* ( c + d ) + ( a + b ) ) | a a b c d + \* + a b + + + | a = 1  b = 2  c = 3  d = 4 | 19 |
| 7 | 5 + a | 5 a + | a = 1 + 3i | 6 + 3i |
| 8 | -a + z | 0 a – z + | a = 7  z = 10 | 3 |
| 9 | a + ( - b – c ) | a 0 b – c - + | a = 1  b = 2  c = 3 | -4 |

**Текст программы:**

const MAX\_N = 256;

s\_signs = ['+', '-', '\*', '/'];

N = 26;

type t\_range = 1..MAX\_N;

t\_arr\_ch = array [1..N] of char;

t\_ch\_stack = record

el: array[t\_range] of char; //знаки

size : t\_range;

end;

t\_real\_stack = record

el: array[t\_range] of real;

size : t\_range;

end;

t\_complex = record

mat : real;

imagi : real;

end;

t\_arr\_comp = array [1..N] of t\_complex;

{Запись в переменную w, слово строки s, начиная с i-той позиции}

function get\_word(s : string; var i : byte):string;

var w : string;

len : integer;

begin

w := '';

len := length (s);

while (i <= len) and (s[i] <= ' ') do

i := i + 1;

while (i <= len) and (s[i] > ' ') do

begin

w := w + s[i];

i := i + 1;

end;

get\_word := w;

end;

{помещает элемент на вершину стека}

procedure ch\_push(var stack : t\_ch\_stack; b : string);

begin

stack.size := stack.size + 1;

stack.el[stack.size] := b[1];

end;

{помещает элемент на вершину стека}

procedure real\_push(var stack : t\_real\_stack; b : real);

begin

stack.size := stack.size + 1;

stack.el[stack.size] := b;

end;

{возвращает значение «истина», если стек пустой и значение «ложь», если нет}

function is\_empty(stack : t\_ch\_stack): boolean;

begin

is\_empty := stack.size = 0

end;

{возвращает верхний элемент и удаляет его из стека}

function ch\_pop(var stack : t\_ch\_stack) : char;

var ch :char;

begin

ch := stack.el[stack.size];

stack.size := stack.size - 1;

ch\_pop := ch;

end;

{возвращает верхний элемент и удаляет его из стека}

function real\_pop(var stack : t\_real\_stack) : real;

var ch :real;

begin

ch := stack.el[stack.size];

stack.size := stack.size - 1;

real\_pop := ch;

end;

{записывает в строку s все элементы из стека stack}

procedure clear\_ch\_stack(var s : string; var stack : t\_ch\_stack);

begin

while is\_empty (stack) = false do

s := s + ' ' + ch\_pop(stack);

end;

{заменяет операцию унарного минуса на выражение 0 – и сдвигает строку s, начиная с i символа}

procedure unar\_minus(var s : string; i : byte);

var j, len : byte;

begin

len := length (s);

s := s + ' ' + ' ';

for j := len downto i do

begin

s[j+2] := s[j];

end;

s[i] := '0';

end;

{поиск в строке s унарного минуса и преобразование его в бинарный}

procedure transform\_unar\_minus\_str (var s : string);

var w : string;

i : byte;

begin

i := 1;

if s[1] = '-' then

unar\_minus(s, i);

while i <= length (s) do

begin

w := get\_word (s, i);

if w[1] = '(' then

begin

w := get\_word (s, i);

if w[1] = '-' then

unar\_minus(s, i-1);

end;

end;

end;

{распределяет операцию sign либо в строку opz\_s, либо в стек stack, в зависимости от приоритета предыдущей операции fl1}

procedure opz\_sign\_write(var opz\_s : string; var stack : t\_ch\_stack; var fl1 : byte; sign : char);

var fl2 : byte;

begin // если стэк не пустой

if is\_empty (stack) = true then //если стек пустой

begin

ch\_push (stack, sign);

if (sign = '-') or (sign = '+') then

fl1 := 0

else

fl1 := 1;

end

else

begin

if (sign = '-') or (sign = '+') then

fl2 := 0

else

fl2 := 1;

if fl1 >= fl2 then //приоритет такой же или меньше, то предыдущий в новую

begin //строку и убрать из стэка

opz\_s := opz\_s + ' ' + ch\_pop (stack);

ch\_push (stack, sign);

end // добавляем в стэк новый

else

ch\_push (stack, sign);

fl1 := fl2;

end;

end;

{Переводт обычную запись в скобках из строки s начиная с i-той позиции

в ОПЗ и записывает в строку opz с использованием стэка stack}

procedure bracket(var s, opz\_s : string; var i : byte);

var k\_open, k\_close, fl1 : byte;

w : string;

stack, zero\_stack, temp\_stack : t\_ch\_stack;

begin

k\_open := 1;

k\_close := 0;

while k\_open <> k\_close do

begin

w := get\_word (s, i);

if w[1] = '(' then

begin

inc(k\_open);

temp\_stack := stack;

stack := zero\_stack;

end;

if w[1] = ')' then

begin

clear\_ch\_stack (opz\_s, stack);

inc(k\_close);

stack := temp\_stack;

temp\_stack := zero\_stack;

clear\_ch\_stack (opz\_s, stack);

end;

if (w[1] in s\_signs) then

opz\_sign\_write (opz\_s, stack, fl1, w[1])

else

if (w[1] <> ')') and (w[1] <> '(') then

opz\_s := opz\_s + ' ' + w;

end;

end;

{преобразует выражение в строке s в обратную польскую запись и записывает в строку opz\_s}

procedure opz (var opz\_s : string; s : string);

var w : string;

i, len, fl1 : byte;

stack : t\_ch\_stack;

begin

i := 1;

len := length (s);

while i <= len do

begin

w := get\_word (s, i);

if w[1] in s\_signs then // если слово является знаком

opz\_sign\_write (opz\_s, stack, fl1, w[1])

else

begin

if w[1] = '(' then /// если не знак

bracket(s, opz\_s, i)

else

opz\_s := opz\_s + ' ' + w;

end;

if i = (len + 1) then

clear\_ch\_stack (opz\_s, stack);

end;

end;

{возвращает значение «истина», если найдено s в массиве а, иначе «ложь»}

function check\_el(s : char; a : t\_arr\_ch): boolean;

var i : byte;

flag : boolean;

begin

i:= 1;

flag := false;

while (flag = false) and (i <= N )do

begin

if s = a[i] then

flag := true;

i := i + 1;

end;

check\_el := flag;

end;

{записывает в массив а буквы из строки s}

procedure record\_let(s:string; var a : t\_arr\_ch; var k : byte);

var i : byte;

begin

k := 1;

for i := 1 to length (s) do

if (s[i] in ['A'..'z']) and (check\_el(s[i],a) = false) then

begin

a[k] := s[i];

k := k + 1 ;

end;

k := k - 1;

end;

{сортировка массива переменных a длины k лексикографически}

procedure sort\_arr\_let(var a : t\_arr\_ch; k : byte);

var i : byte;

f : boolean;

t : char;

begin

f := true;

while f = true do

begin

f := false;

for i:= 1 to k - 1 do

if a[i] > a[i+1] then

begin

f := true;

t:= a[i];

a[i] := a [i+1];

a[i+1] := t;

end;

end;

end;

{ввод значений в массив val для массива переменных a длины k}

procedure read\_value(var arr\_comp\_val : t\_arr\_comp;

arr\_var : t\_arr\_ch; k : byte);

var i, j : byte;

mat : real;

s : string;

begin

s := ' ';

for j := 1 to ord(arr\_var[1])- ord('a') do

arr\_comp\_val[j].mat := 0;

for i := 1 to k do

begin

if ord(arr\_var[i]) = ord(arr\_var[1]) + i - 1 then

begin

write(arr\_var[i],' = ');

read(mat);

read(s);

if length(s) > 0 then

begin

s[length(s)] := ' ';

arr\_comp\_val[ord(arr\_var[i]) - ord('a') + 1].mat := mat;

arr\_comp\_val[ord(arr\_var[i]) - ord('a') + 1].imagi := strtofloat(s);

end

else

begin

arr\_comp\_val[ord(arr\_var[i]) - ord('a') + 1].mat := mat;

arr\_comp\_val[ord(arr\_var[i]) - ord('a') + 1].imagi := 0;

end;

end

else

begin

for j := (ord(arr\_var[i-1]) - ord('a') + 2) to (ord(arr\_var[i]) - ord('a') ) do

arr\_comp\_val[j].mat := 0;

write(arr\_var[i],' = ');

read(mat);

read(s);

if length(s) > 0 then

begin

s[length(s)] := ' ';

arr\_comp\_val[j + 1].mat := mat;

arr\_comp\_val[j + 1].imagi := strtofloat(s);

end

else

begin

arr\_comp\_val[j + 1].mat := mat;

arr\_comp\_val[j + 1].imagi := 0;

end;

end;

end;

end;

{сложение комплексных чисел a и b и запись результата в переменную c}

procedure complex\_plus(var c : t\_complex; a, b : t\_complex);

begin

c.mat := a.mat + b.mat;

c.imagi := a.imagi + b.imagi;

end;

{вычитание комплексных чисел a и b и запись результата в переменную c}

procedure complex\_minus(var c : t\_complex; a, b : t\_complex);

begin

c.mat := a.mat - b.mat;

c.imagi := a.imagi - b.imagi;

end;

{умножение комплексных чисел a и b и запись результата в переменную c}

procedure complex\_multiplic(var c : t\_complex; a, b : t\_complex);

begin

c.mat := a.mat \* b.mat - a.imagi \* b.imagi;

c.imagi := a.mat \* b.imagi + b.mat \* a.imagi;

end;

{деление комплексных чисел a и b и запись результата в переменную c}

procedure complex\_division(var c : t\_complex; a, b : t\_complex);

begin

c.mat := (a.mat \* b.mat + a.imagi \* b.imagi) / (sqr(b.mat) + sqr(b.imagi));

c.imagi := (-a.mat \* b.imagi + b.mat \* a.imagi) / (sqr(b.mat) + sqr(b.imagi));

end;

{нахождение значения выражения опз opz\_s для значений переменных массива arr\_comp\_val и запись ответа в переменную rez}

procedure calc\_comp\_opz (var rez : t\_complex ; arr\_comp\_val : t\_arr\_comp; opz\_s : string);

var i, len : byte;

w : string;

m, mat, imagi : real;

num1, num2 : t\_complex;

stack\_mat, stack\_imagi : t\_real\_stack;

begin

i := 1;

len := length (opz\_s);

while i <= len do

begin

w := get\_word (opz\_s, i);

if w[1] in s\_signs then

begin

num1.mat := real\_pop(stack\_mat);

num1.imagi := real\_pop(stack\_imagi);

num2.mat := real\_pop(stack\_mat);

num2.imagi := real\_pop(stack\_imagi);

case w[1] of

'+' : complex\_plus(rez, num1, num2);

'-' : complex\_minus(rez, num2, num1);

'\*' : complex\_multiplic(rez, num1, num2);

'/' : complex\_division(rez, num2, num1);

end;

real\_push(stack\_mat, rez.mat);

real\_push(stack\_imagi, rez.imagi);

end

else

begin

if (length(w) >= 1) and (w[1] in '0'..'9') then

begin

m := strtofloat(w);

real\_push (stack\_mat, m);

real\_push (stack\_imagi, 0);

end

else

begin

mat := arr\_comp\_val[ord(w[1]) - ord('a') + 1].mat;

imagi := arr\_comp\_val[ord(w[1]) - ord('a') + 1].imagi;

real\_push(stack\_mat, mat);

real\_push(stack\_imagi, imagi);

end;

end;

end;

rez.mat := real\_pop(stack\_mat);

rez.imagi := real\_pop(stack\_imagi);

end;

var arr\_var : t\_arr\_ch;

arr\_comp\_val : t\_arr\_comp;

comp\_rez : t\_complex;

s, opz\_s : string;

k : byte;

begin

k := 0;

writeln('Вас приветствует программа получения обратной польской записи.');

write('Введите выражение, для которого вы хотите получить ОПЗ: ');

read(s);

transform\_unar\_minus\_str(s);

opz(opz\_s, s);

writeln('Введенное выражение записаное в ОПЗ: ', opz\_s);

record\_let(s, arr\_var, k);

if k > 0 then

begin

sort\_arr\_let(arr\_var, k);

read\_value(arr\_comp\_val, arr\_var, k);

end;

calc\_comp\_opz(comp\_rez, arr\_comp\_val, opz\_s);

if comp\_rez.imagi = 0 then

writeln('Значение выражения: ', comp\_rez.mat:0:5)

else

writeln('Значение выражения: ', comp\_rez.mat:0:5,' + ',comp\_rez.imagi:0:5,'i');

end.

**Результаты работы программы:**

