Практическое занятие по СПО №7

Густов Владимир Владимирович
gutstuf@gmail.com

Цитата дня: Каждое поколение считает себя более умным, чем предыдущее, и более мудрым, чем последующее. (с) Джордж Оруэлл

Стек

1 (esp == 0xFFFF0A0C)

Адрес	Значение
0xFFFF0A0C	XXX
0xFFFF0A08	XXX
0xFFFF0A04	XXX
0xFFFF0A00	XXX

```
1) nothing
2) pushl $5
3) pushl %ebp
4) popl  %ebp
```

2 (esp == 0xFFFF0A08)

Адрес	Значение
0xFFFF0A0C	XXX
0xFFFF0A08	5
0xFFFF0A04	XXX
0xFFFF0A00	XXX

3 (esp == 0xFFFF0A04)

Адрес	Значение
0xFFFF0A0C	XXX
0xFFFF0A08	5
0xFFFF0A04	0
0xFFFF0A00	XXX

esp — указатель на «вершину» стека;
ebp — указатель на фрейм стека;

фрейм — отдельный фрагмент памяти стека.

`x/4d $esp` – вывод последних 4х значений в стеке

4 (esp == 0xFFFF0A08)

Адрес	Значение
0xFFFF0A0C	XXX
0xFFFF0A08	5
0xFFFF0A04	XXX
0xFFFF0A00	XXX

```
main:
  xorl  %eax, %eax
  pushl %ebp        # сохраним текущий указатель стека
  movl  %esp, %ebp # сохраним новый (текущий) адрес вершины стека
op1:
  movl $5, a
op2:
  pushl $5            # 1
  pushl $2            # 2
  popl %eax
  addl %eax, -4(%ebp) # обратимся ко второму элементу в стеке

  pushl a
  popl %eax           # 3
  subl %eax, -4(%ebp)
  popl %eax

  leave   # вернём стек обратно на адрес в ebp
  ret
```

ebp == 0xFFFF0A08

esp == 0xFFFF0A0C

Адрес	Значение
0xFFFF0A0C	XXX
0xFFFF0A08	XXX
0xFFFF0A04	XXX
0xFFFF0A00	XXX

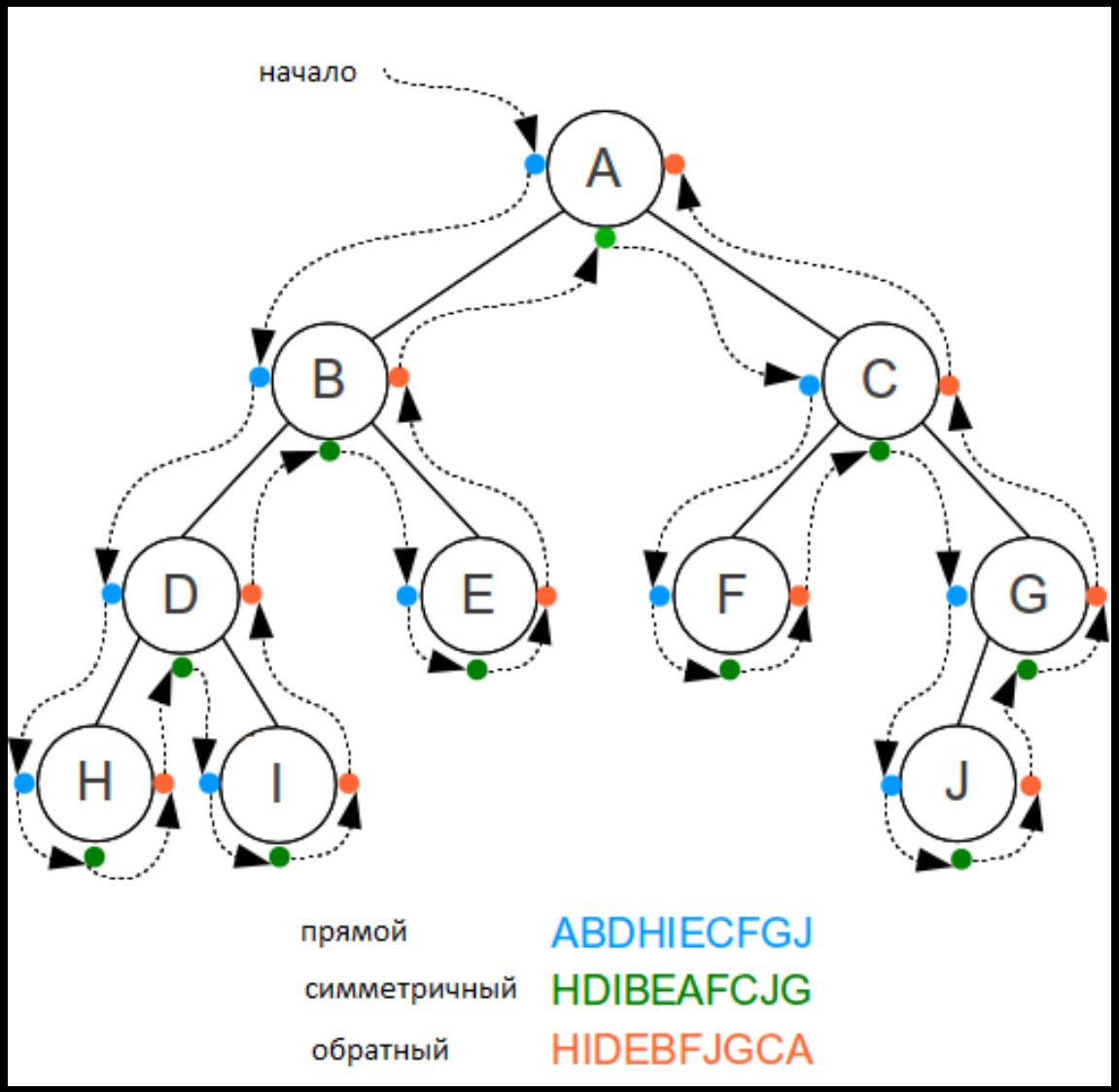
2 esp == 0xFFFF0A00

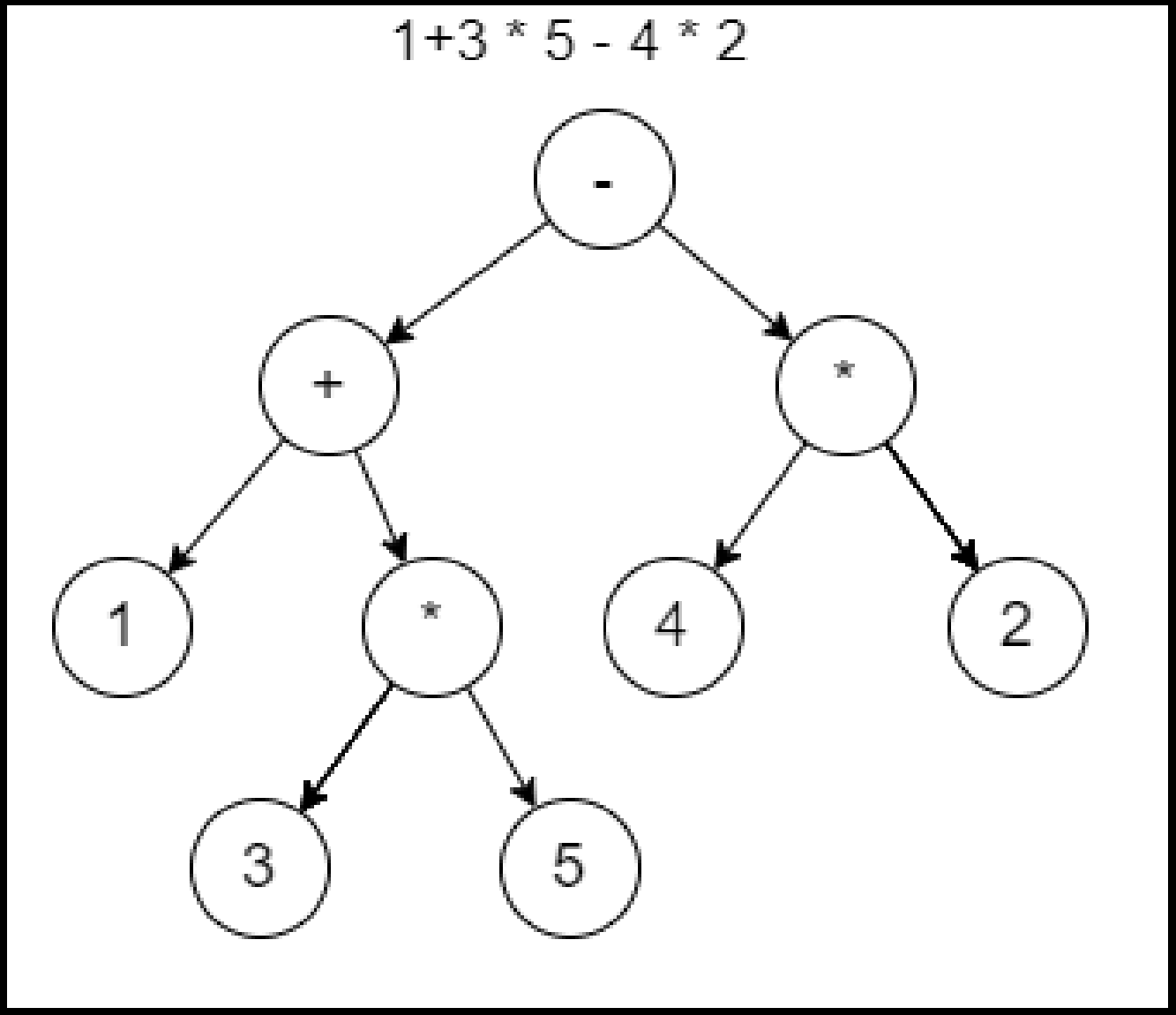
0xFFFF0A0C	XXX
0xFFFF0A08	0
0xFFFF0A04	5
0xFFFF0A00	2
0xFFFF09FC	XXX

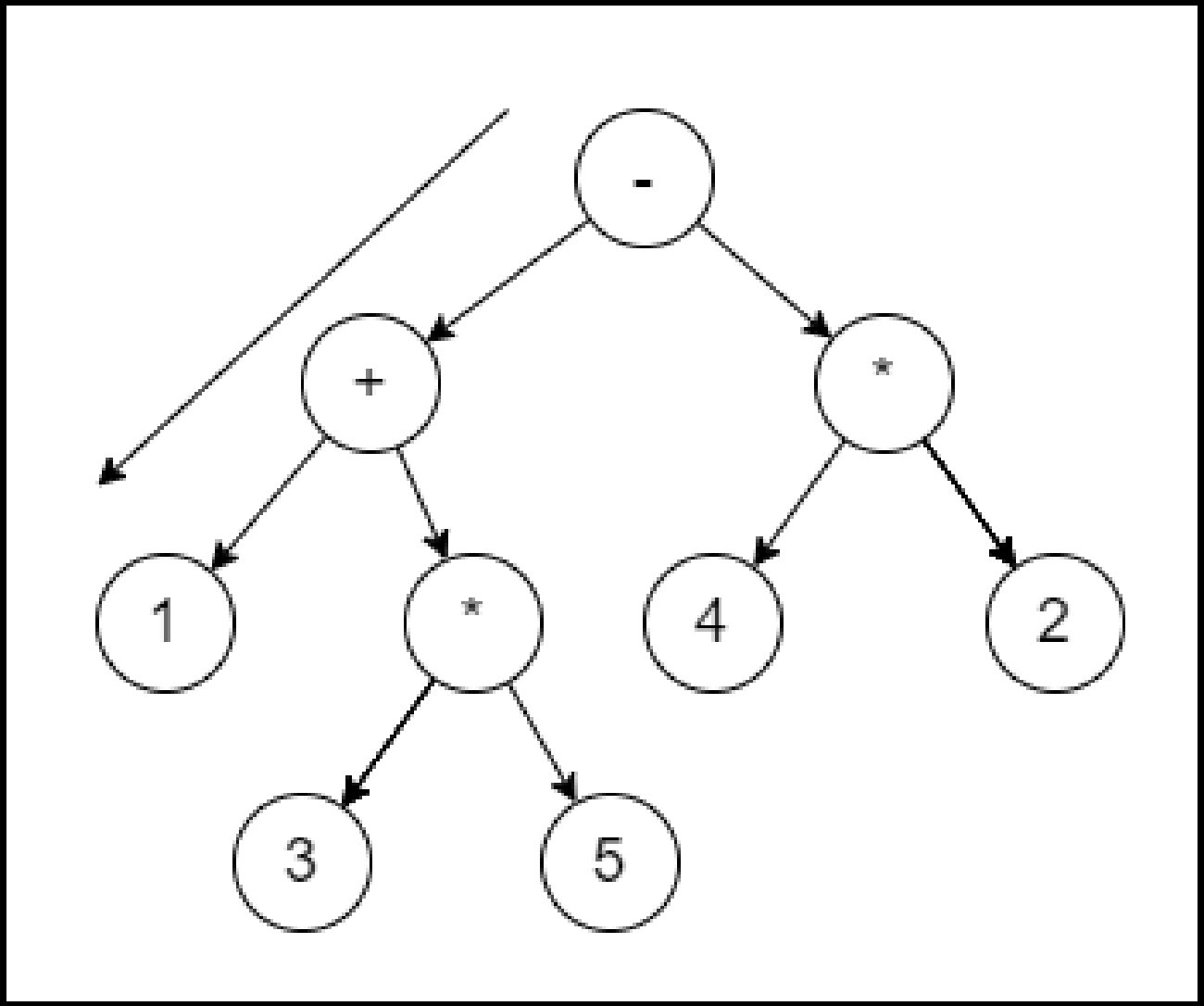
3 esp == 0xFFFF0A04

0xFFFF0A0C	XXX
0xFFFF0A08	0
0xFFFF0A04	7
0xFFFF0A00	XXX
0xFFFF09FC	XXX

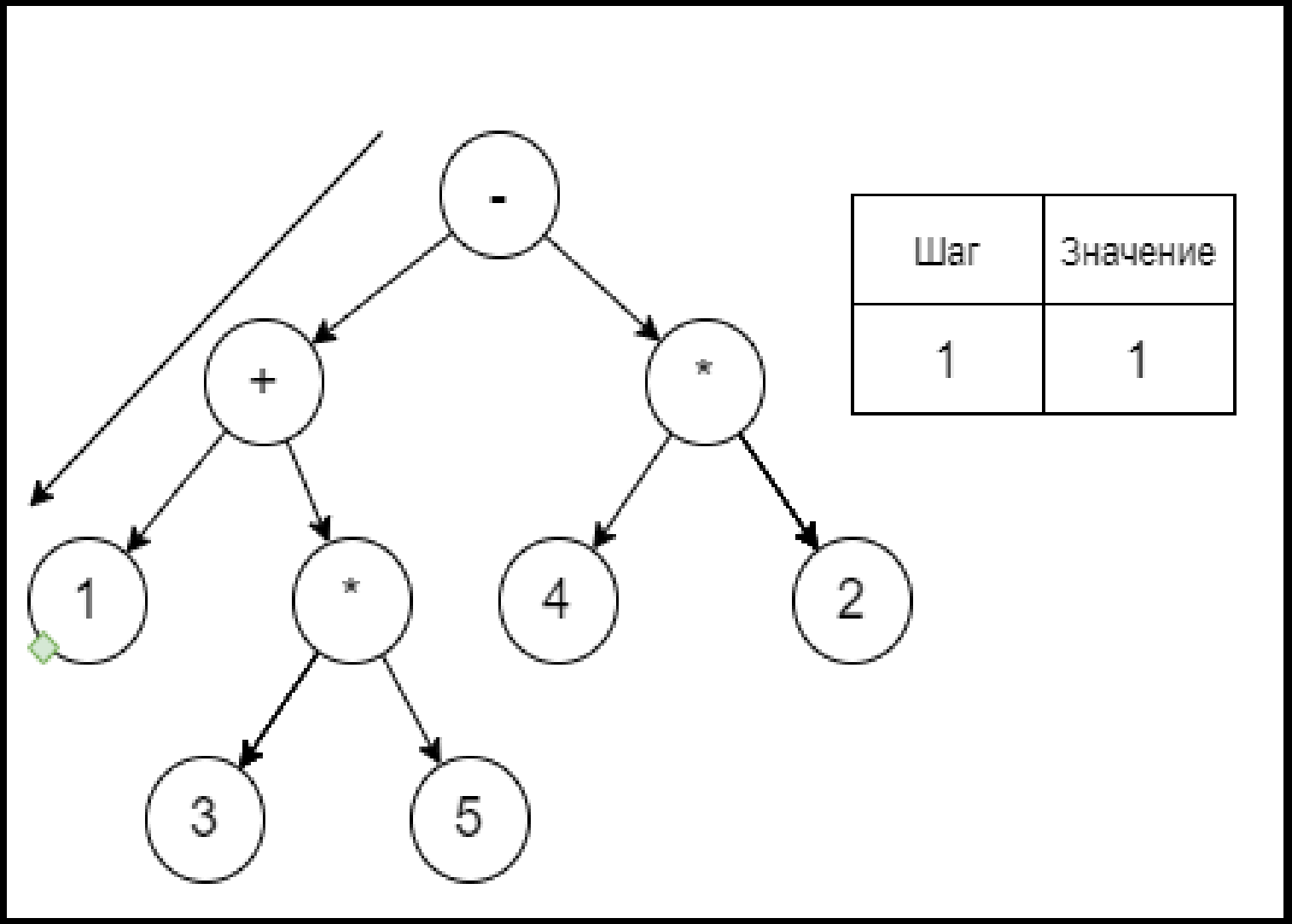
Обратный обход дерева выражений (post-order)




1+3 * 5 - 4 * 2
-
+
*
1
*
4
2
3
5

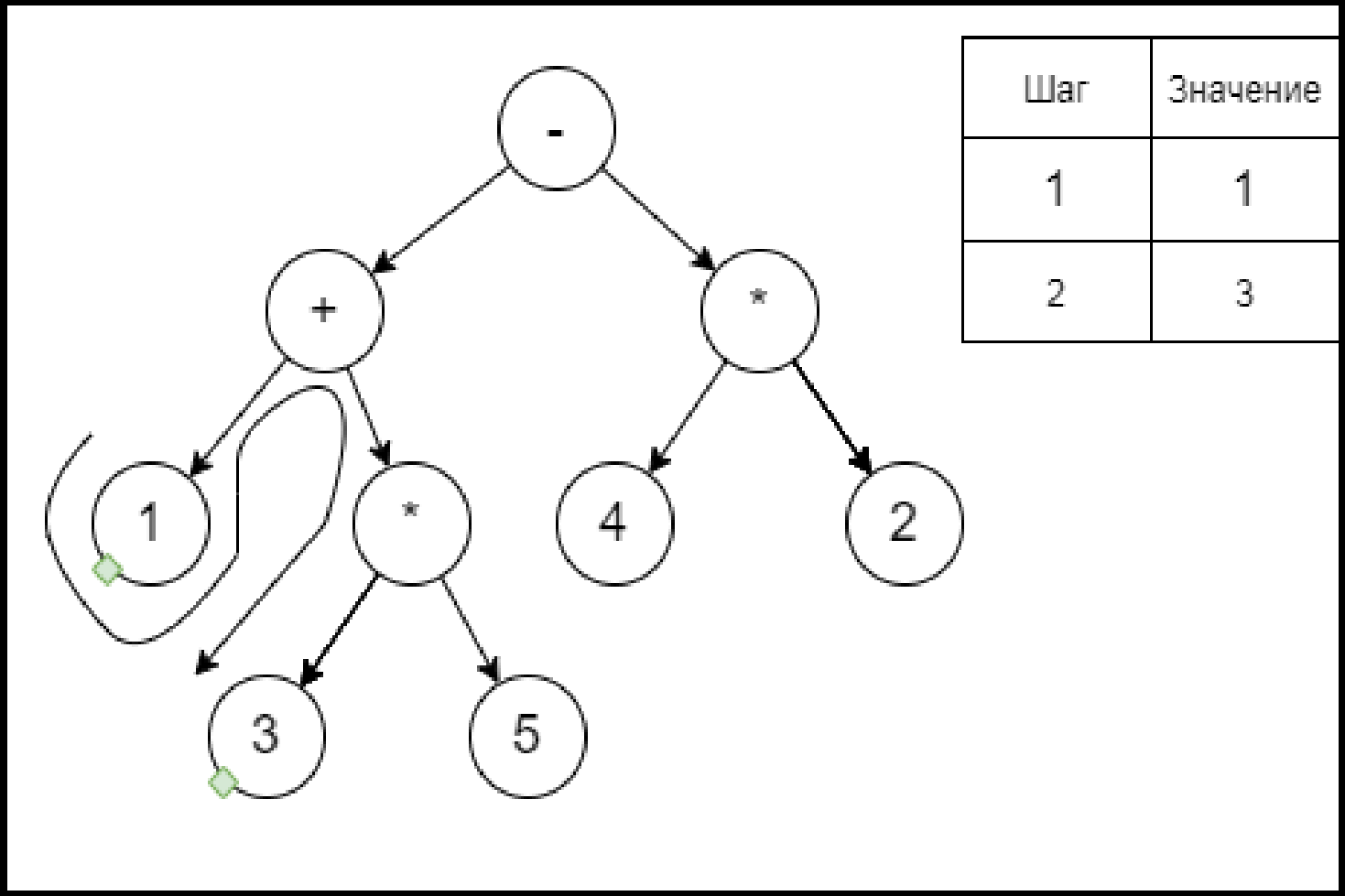

-
+
*
1
*
4
2
3
5

Обратная польская запись (ОПЗ):
1



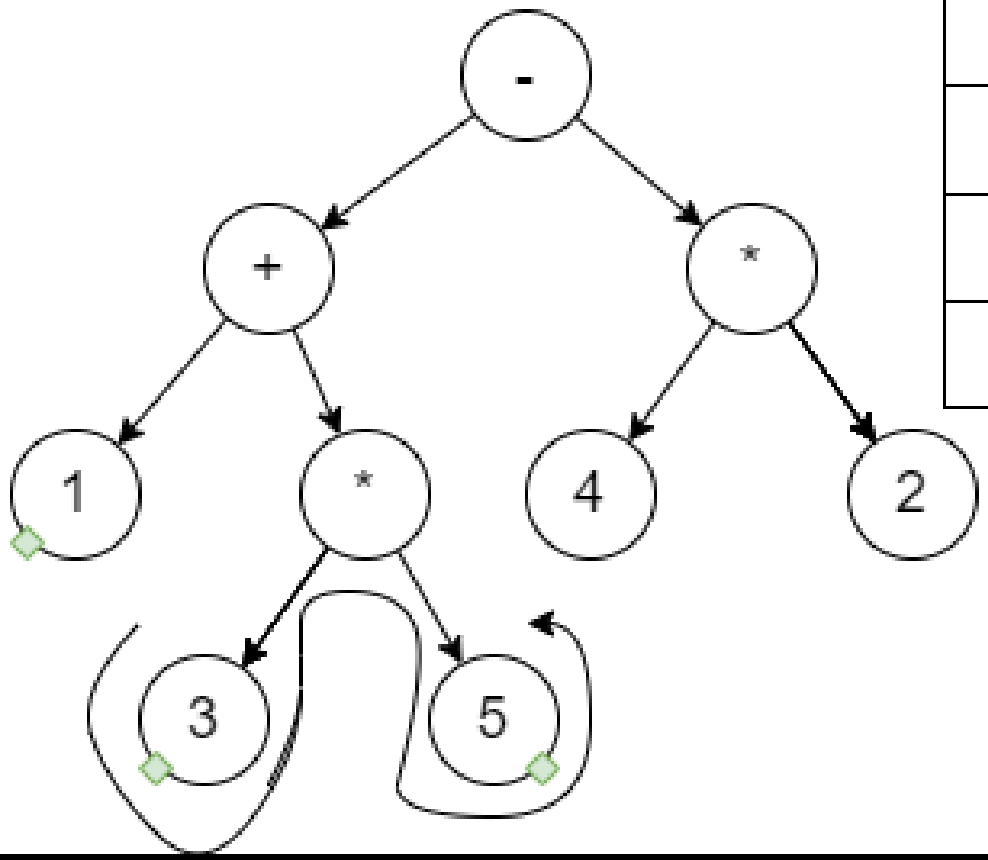
Шаг	Значение
1	1

Обратная польская запись (ОПЗ):
1 3



Шаг	Значение
1	1
2	3

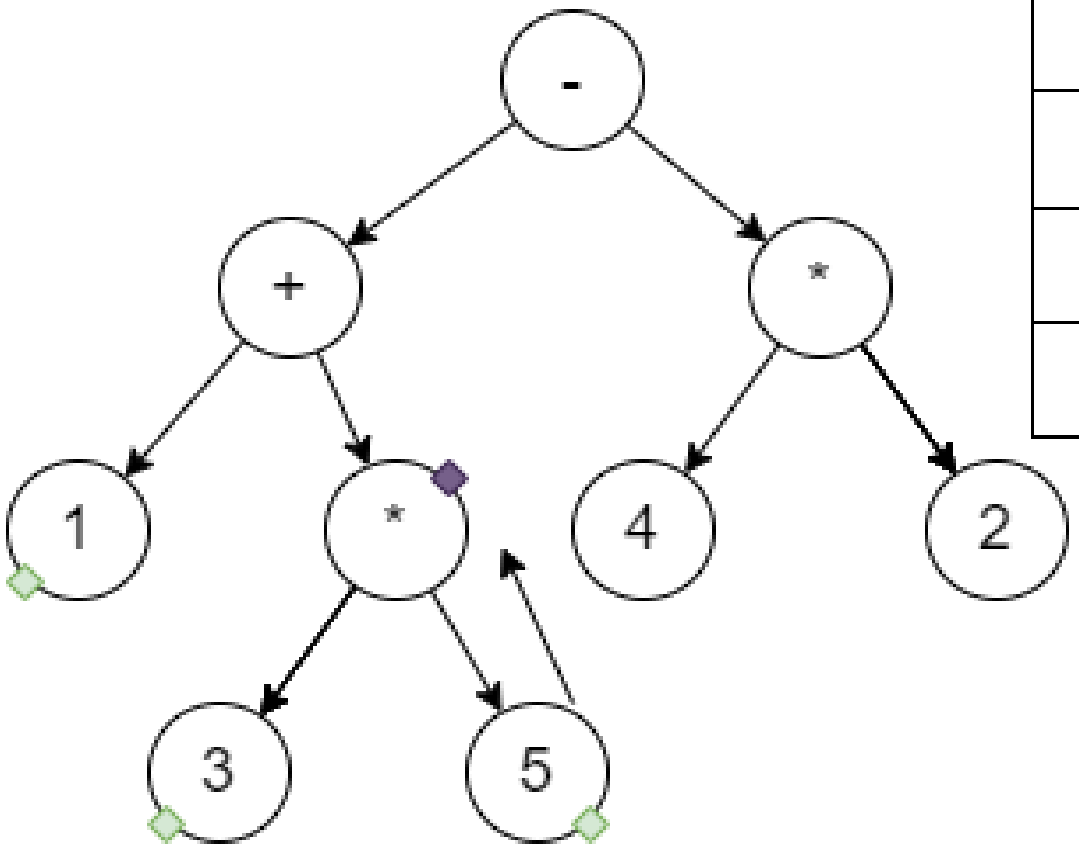
Обратная польская запись (ОПЗ):
1 3 5



Шаг	Значение
1	1
2	3
3	5

Обратная польская запись (ОПЗ):

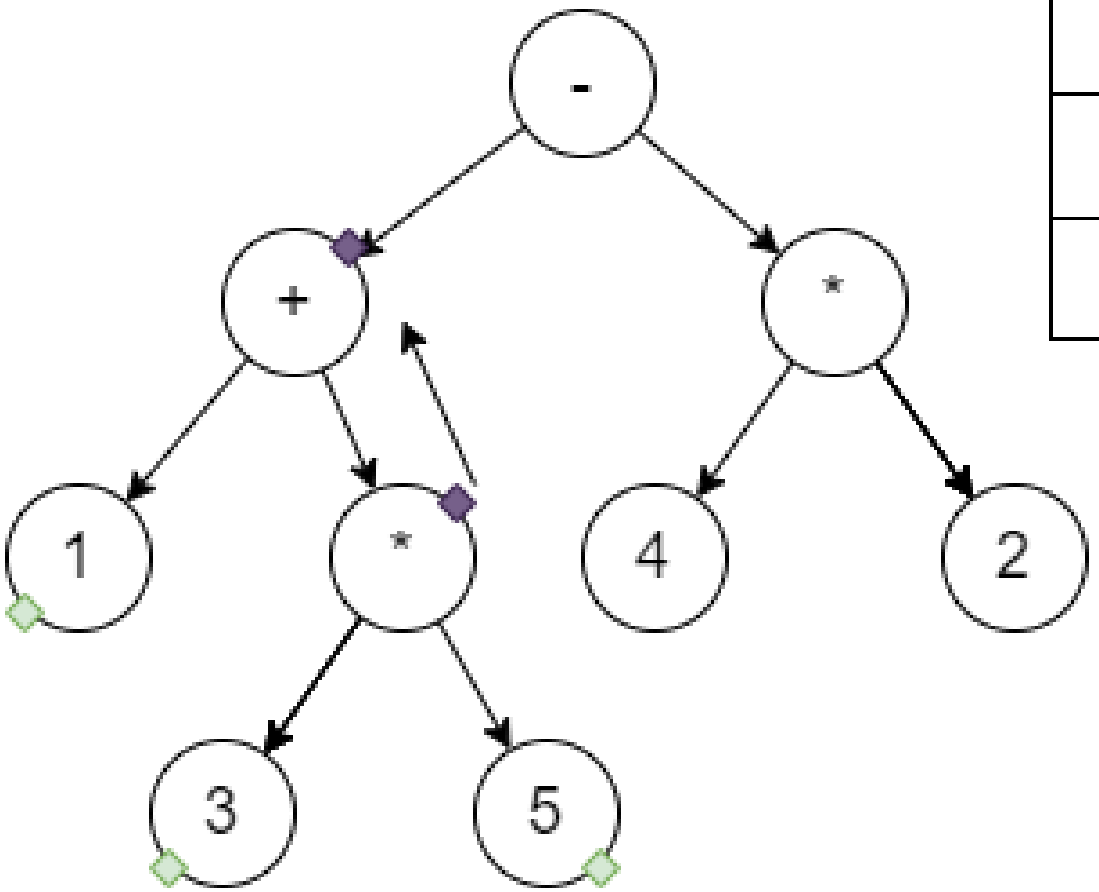
```
1  3  5  *
```



Шаг	Значение
1	1
2	3
3	5

Обратная польская запись (ОПЗ):

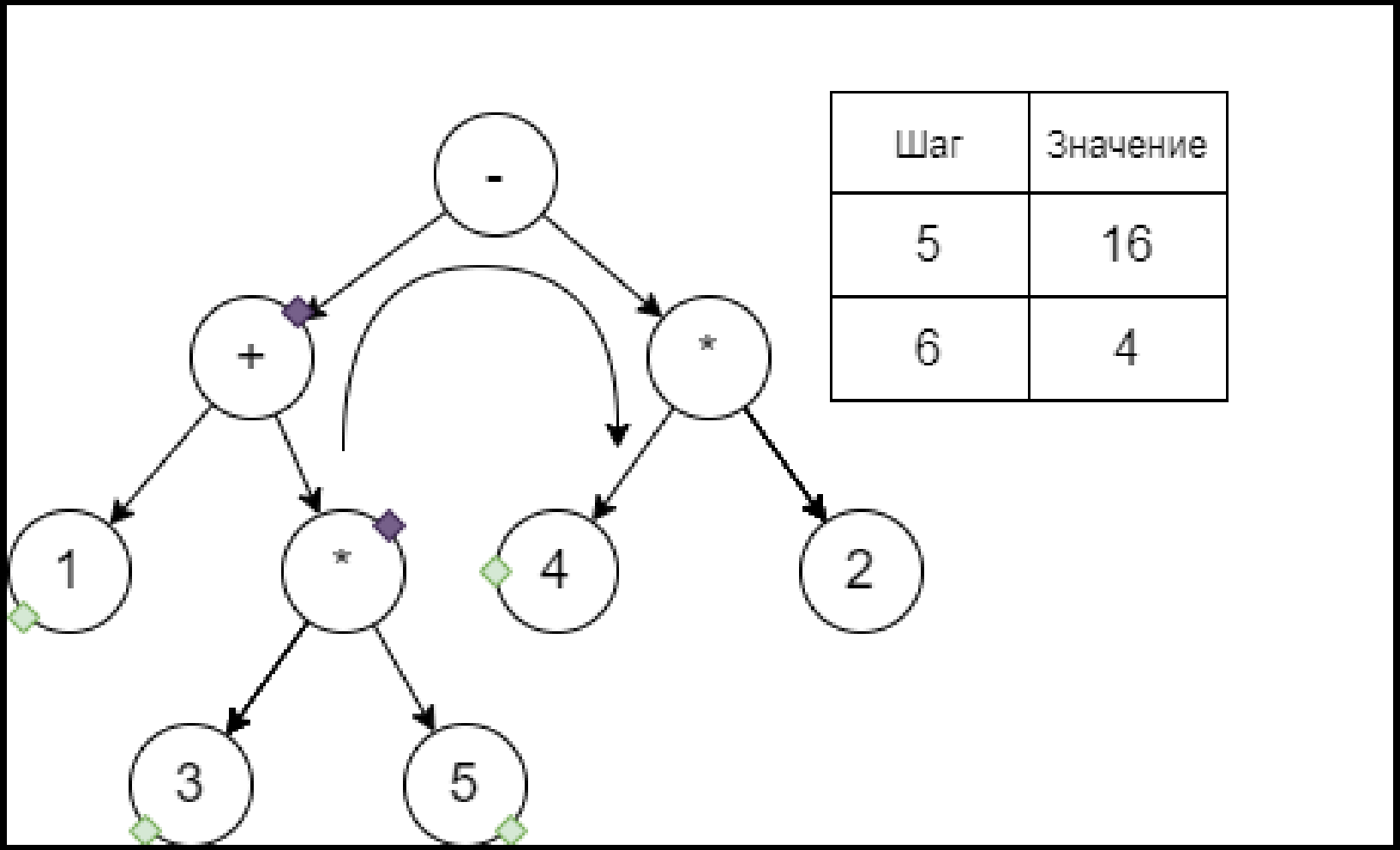
```
1  3  5  *  +
```



Шаг	Значение
1	1
4	15

Обратная польская запись (ОПЗ):

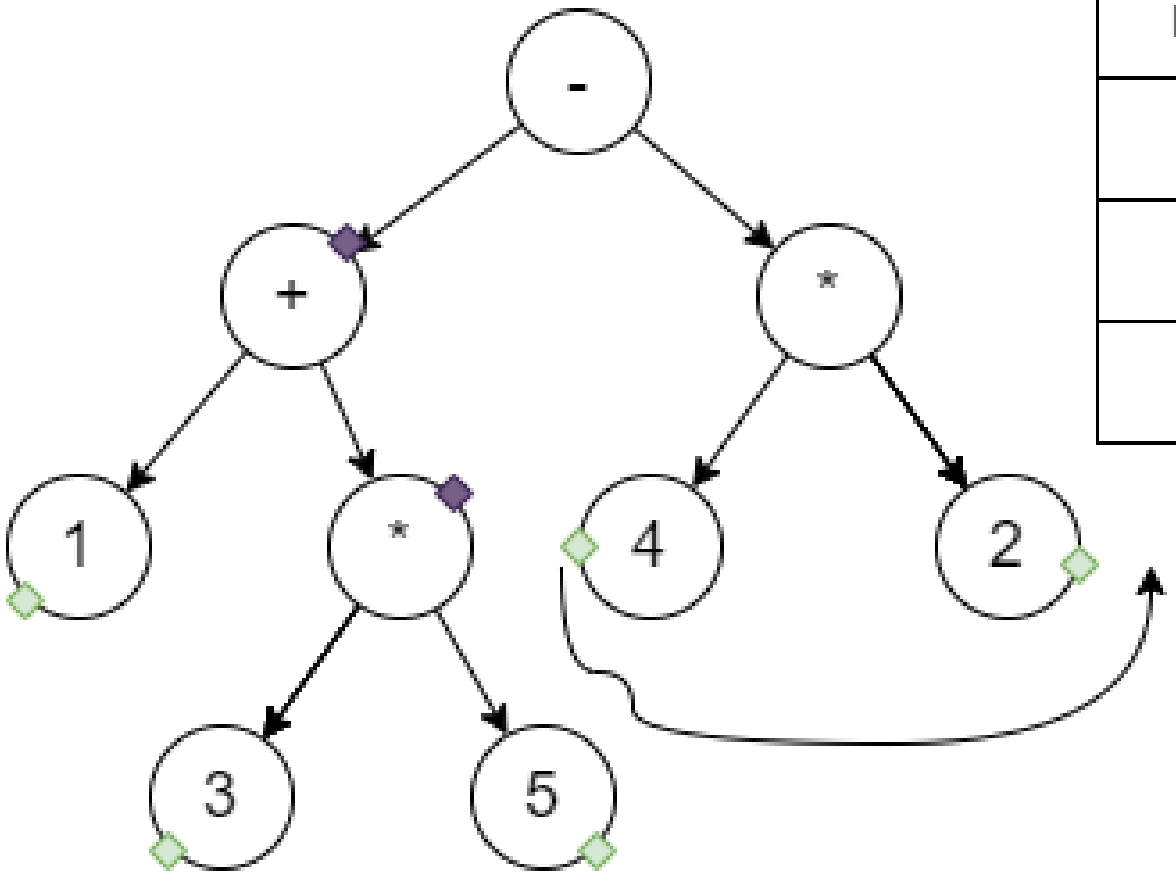
```
1  3  5  *  +  4
```



Шаг	Значение
5	16
6	4

Обратная польская запись (ОПЗ):

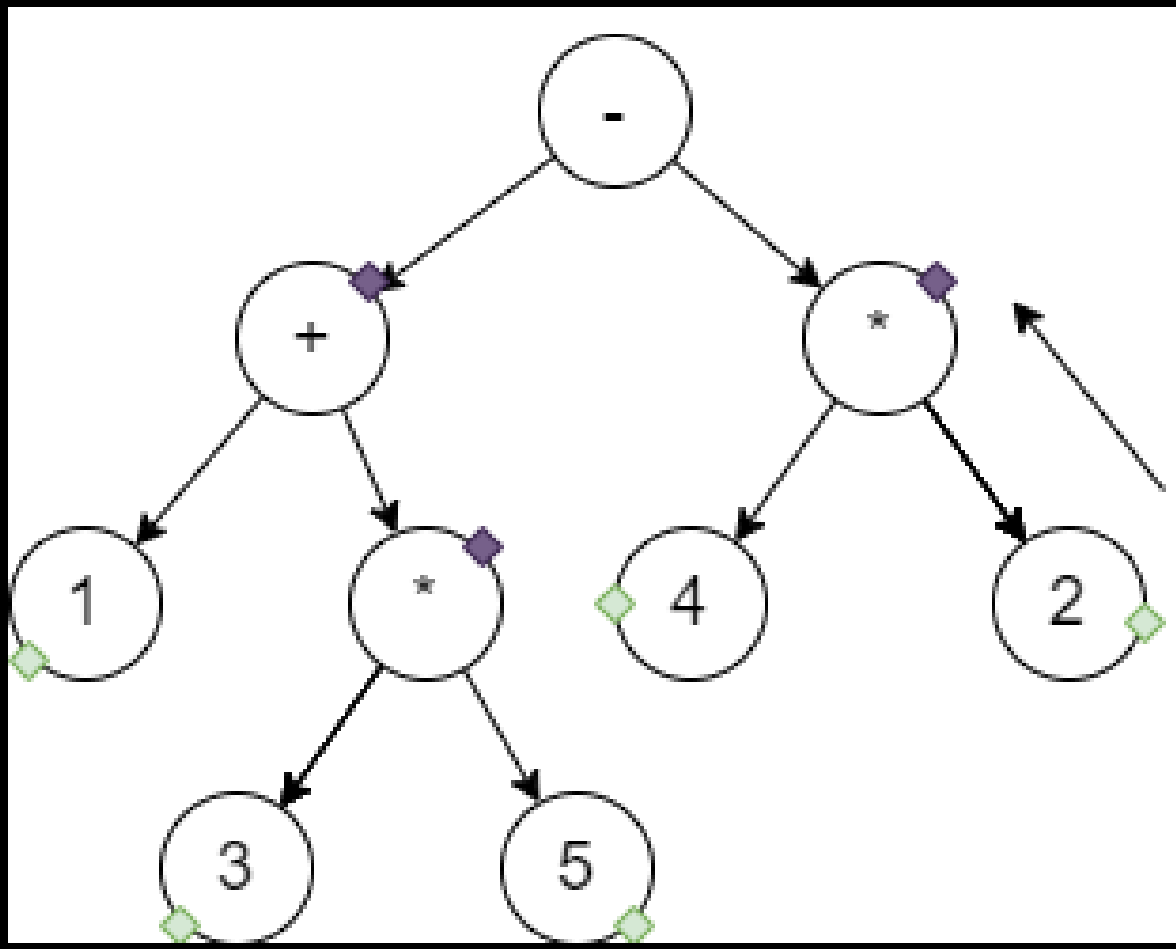
1 3 5 * + 4 2



Шаг	Значение
5	16
6	4
7	2

Обратная польская запись (ОПЗ):

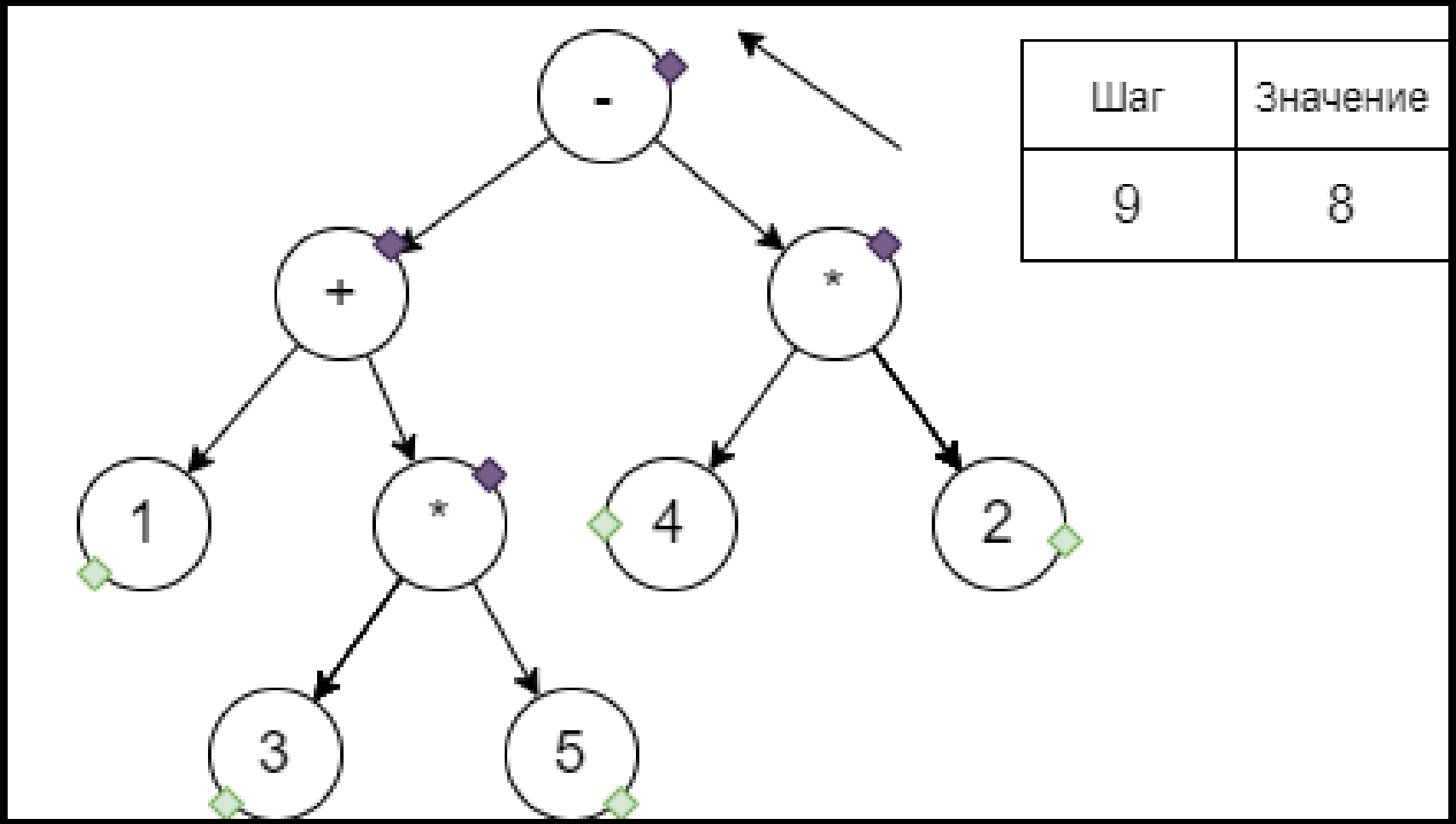
1 3 5 * + 4 2 *



Шаг	Значение
5	16
8	8

Обратная польская запись (ОПЗ):

1 3 5 * + 4 2 * -



Шаг	Значение
9	8

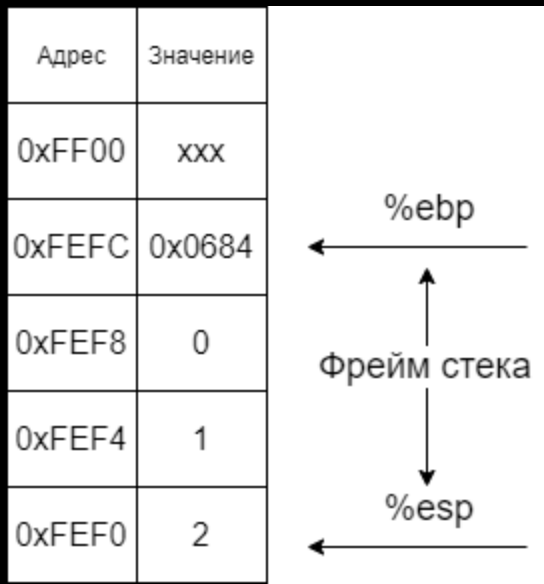
Подпрограммы

Подпрограмма – вызываемый метод/функция/процедура. Используются для избегания дублирования кода, введения абстракции, структуризация кода.

Соглашения о вызове - это методы, **с помощью которых** вызывающая и вызываемая функции **согласовывают**, **как параметры** и возвращаемые значения **должны передаваться** между ними, **и как стек используется** самой функцией.

Фрейм стека – отдельный «**кусок**» **памяти**, **выделяемый из стека** во время исполнения (runtime) , **при каждом вызове функции**. Используется для хранения локальных переменных.

По сути, это область между адресом в %esp и %ebp.



Подпрограммы

Соглашения о вызове:

cdecl (see-deck_ll) — стек очищается вызывающим. Аргументы, меньше 4х байт, расширяются до 4х байт.

stdcall — стек очищается вызванным;

fastcall — стек очищается вызванным; Аргументы передаются в регистрах.

thiscall — стек очищается вызванным; В регистр *ecx* записывается указатель на объект для которого вызывается метод.

Google it: ABI, ISA (instruction set architecture)

__cdecl

```
.data
fmt:
  .string "Some long text, size: %d\n"
fmt_length:
   .long . - fmt  # . возвращает текущий адрес

fmt2:
  .string "Also please print %d\n"

.text
.globl _main
_main:
  # пролог
  pushl %ebp
  movl  %esp, %ebp
  ##################
  pushl fmt_length
  pushl $fmt
  call _printf

  pushl $1337
  pushl $fmt2
  call _printf

  # эпилог
  movl %ebp, %esp  # можно заменить
  popl %ebp        #  оператором leave
  ret
```

__stdcall

```
.data
fmt:
  .string "Some long text, size: %d\n"
fmt_length:
   .long . - fmt  # . возвращает текущий адрес

fmt2:
  .string "Also please print %d\n"

.text
.globl _main
_main:
  # пролог
  pushl %ebp
  movl  %esp, %ebp
  ##################

  pushl fmt_length
  pushl $fmt
  call _printf # стек очищается вызванной
               # подпрограммой
  ret
```

gcc -S

```
void main() {
  int a = 0;
  int b;
  b = a + 3 - 1 * 2;
}
```

gcc -m32 -o - -S main.c

```
# (MinGW, Windows 10)
andl $-16, %esp
subl $16, %esp
call ___main

movl $0, 12(%esp)
movl 12(%esp), %eax
addl $1, %eax
movl %eax, 8(%esp)
nop
```

```
        .file   "main.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp

        movl    $0, -8(%ebp)
        movl    -8(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -4(%ebp)
        nop

        leave
        ret
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.2.0-
ubuntu3.1~16.04.york0) 7.2.0"
        .section .note.GNU-stack,"",@progbits
```

case

```
  movl  $30, %eax     # получить в %eax
                      # некоторое интересующее
                      # нас значение

  cmpl  $5, %eax
  je    case_5

  cmpl  $30, %eax
  je    case_30

  cmpl  $120, %eax
  je    case_120

case_default:
  movl $100, %ecx
  jmp switch_end

case_5
  movl $5, %ecx
  jmp switch_end

case_30
  movl $30, %ecx
  jmp switch_end

case_120
  movl $120, %ecx
  jmp switch_end
```

```
val := 30;
case val of:
    5:  c := 5;
    30: c := 30;
    120: c := 120;
end;
```

Логические (побитовые) операторы

```
and  источник, приёмник
or   источник, приёмник
xor  источник, приёмник
not  операнд
test операнд_2, операнд_1
```

Работают аналогично операторам в Си:
`&` (and), `|` (or), `^` (xor), `~` (not)

`test` – команда логического сравнения. Позволяет проверить наличие определённых битов. Например:

```
  testl %eax, %eax
  je is_zero  # if (ZF == 1) jump to is_zero

  testl $0x10, %eax   # eax != 0
  je isnt_exist
  nop                 # если в eax есть бит 0001 0000

isnt_exist:           # если в eax нет бита 0001 0000
  nop
```

Google it

sal/sar, shl/shr, rol/ror, rcl/rcr
rep, repe/repz, repne/repnz
movs, cmps, scas
lods, stos
cld, std
setcc

Try It

```
program math;
var a : integer = 4 + 2 - 3 * 1;
    c : integer;
const b : integer = 3;
begin
  c := 20 div a;
  a := a - (a * 2 + 3) + c * 2;
end.
```

And it

```
program an;
  var d, a : integer = 1 * 2 - 3 + 1;
  const  b : boolean = false;
begin
  a := d;
  d := a * 1 + a * (2 + d) * 3 - 4 + 523;
end.
```

And it

```
program hard;
var a : integer;
    b : boolean = false;
begin
  a := 123 xor 321;
  if a > 1 then
  begin
    b := true;
    a := a xor a * 3 + 2;
  end
  else
     a := a + a * a div a + a;
end.
```

Compile time

```
program test_me;
var i : integer;
const a : integer = 2;
  x : array [-0..3] of integer = (0, 1, a, 3);
label _end;
begin
  for i := 0 to 3 do
    writeln(x[i]);

  x[0] = 5;
end.
```

Links

- про соглашения вызовов (ру вики);
- ещё больше соглашений и информации (en wiki);
- про cdecl и stdcall;
- и ещё про соглашения вызовов;
- тут можно подсмотреть перечень jump’ов;
- онлайн парсер обратной польской записи;