

Алгоритмы и Алгоритмические Языки

Семинар #5:

1. Сравнения и логические операции.
2. Тернарная операция и операция “запятая”.
3. Приоритет операций.
4. Порядок вычисления в выражениях.

Сравнения и логические операции



Сравнения и логические операции

Операции сравнения:

< (меньше)	> (больше)
<= (меньше или равно)	>= (больше или равно)
== (равно)	!= (не равно)

Логические операции:

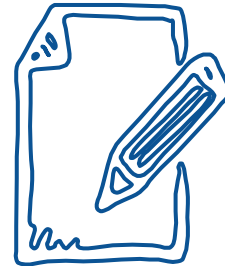
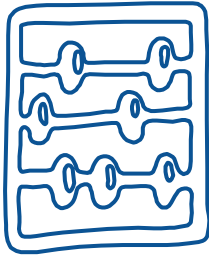
&& (логическое И) ! (не) || (логическое ИЛИ)

Правила короткой логики:

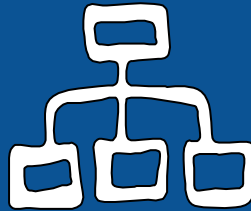
- Если для && (или ||) левая часть выражения равна 0 (или 1), то правая часть выражения **не вычисляется**.
- Левая часть **всегда вычисляется до** правой.

Сравнения: задачи

1. Записать выражение для условия: “ x делится на y без остатка”.
2. Выписать логическое выражение,
истинное **тогда и только тогда**, когда:
 - $x \in [-5, 19)$
 - $x \in (-\infty, 8] \cup [12, +\infty)$
 - $x \in (A, B] \cap [C, D)$



Тернарная операция и операция “запятая”



Тернарная операция и операция “запятая”

Оператор с тернарной операцией:

```
int x = (condition? compute_true : compute_false);
```

Близок к оператору:

```
if (condition)
{
    x = compute_true;
}
else
{
    x = compute_false;
}
```

Тернарная операция и операция “запятая”

Оператор с операцией “запятая” служит для последовательного вычисления выражений.

Конструкция:

```
printf("Comma operator return value: %c\n", (print_a(), print_b()));
```

Эквивалентна:

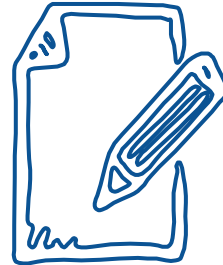
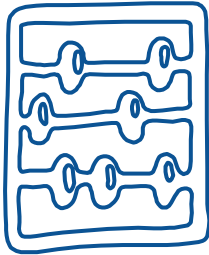
```
int print_ret = print_a();  
print_ret = print_b();  
printf("Comma operator return value: %c\n", print_ret);
```

См. пример [04_operations](#).

Тернарная операция: задачи

Выписать выражения для вычисления:

1. **Модуля** целого числа.
2. **Максимум** двух чисел.



Приоритет операций



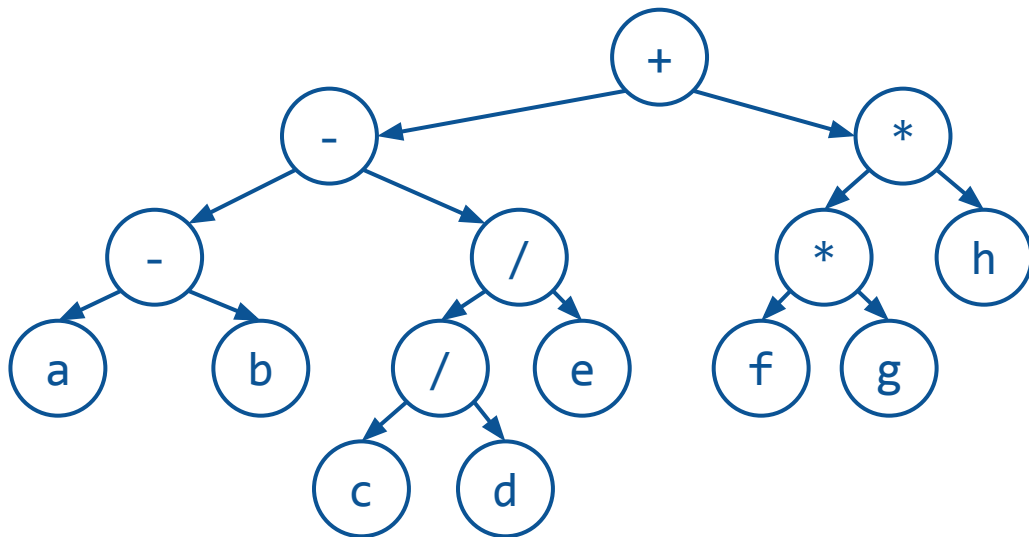
Интерпретация выражений

Рассмотрим выражение языка Си:

$a - b - c / d / e + f * g * h$

Что нужно знать, чтобы однозначно расставить скобки?

$((a - b) - ((c / d) / e)) + ((f * g) * h)$



Приоритеты операций

Ответ: приоритеты и ассоциативности всех операций!

Precedence	Operator	Description	Associativity
1	<code>++ --</code> <code>()</code> <code>[]</code> <code>.</code> <code>-></code> <code>(type){ list}</code>	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer Compound literal(c99)	Left-to-right
2	<code>++ --</code> <code>+ -</code> <code>! ~</code> <code>(type)</code> <code>*</code> <code>&</code> <code>sizeof</code> <code>_Alignof</code>	Prefix increment and decrement ^[note 1] Unary plus and minus Logical NOT and bitwise NOT Cast Indirection (dereference) Address-of Size-of ^[note 2] Alignment requirement(c11)	Right-to-left

Приоритеты операций

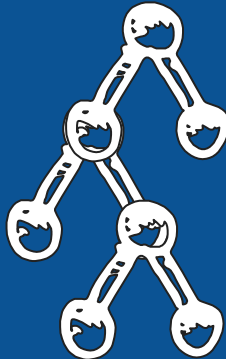
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10		Bitwise OR (inclusive or)	
11	&&	Logical AND	
12		Logical OR	
13	?:	Ternary conditional ^[note 3]	Right-to-left
14 ^[note 4]	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= =	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Приоритеты операций: задачи

Убрать из записи выражений избыточные скобки:

1. `i = ((i++) + (++i))`
2. `d ^= ((a & b) | c)`
3. `x = (((a < b)? (a + 1) : b) - 1)`
4. `(x >> (y & 0xF)) & 0xF`
5. `(a & 0x0F) | (b & 0xF0)`

Порядок вычисления в выражениях



Структура вычисления выражения

Каков будет результат вычисления набора операторов?

```
int i = 5;  
int x = i++ + ++i;  
printf(“%d\n”, x);
```

Структура вычисления выражения

Каков будет результат вычисления набора операторов?

```
int i = 5;  
int x = i++ + ++i;  
printf("%d\n", x);
```

Вычисление
значения
`i++ + ++i`

Вычисление
результата
операции `+`

Вычисление
результата
операции `_++`

Вычисление
результата
операции `++_`

=

Побочный эффект
`i++ + ++i`

Побочный эффект
`i++`

Побочный эффект
`++i`

Как взаимно упорядочены эти коробочки?

Отношение следования вычислений

Отношение sequenced-before (отношение “**следования до**”).

Опр. Для вычислений A и B верно $A \rightarrow B$, если и только если вычисление A заканчивается до начала вычисления B .

1. Если $A \rightarrow B$ и $B \rightarrow C$, то $A \rightarrow C$.
2. Если $A \rightarrow B$, то $B \nrightarrow A$.
3. Если $A \nrightarrow B$ и $B \nrightarrow A$, то:
 - Либо A и B вычисляются в произвольном порядке.
 - Либо инструкции вычисления A и B перемешаны.
4. Для если между выражениями \mathbb{A} и \mathbb{B} находится точка следования, то для любых вычислений значений или побочных эффектов A и B верно $A \rightarrow B$.

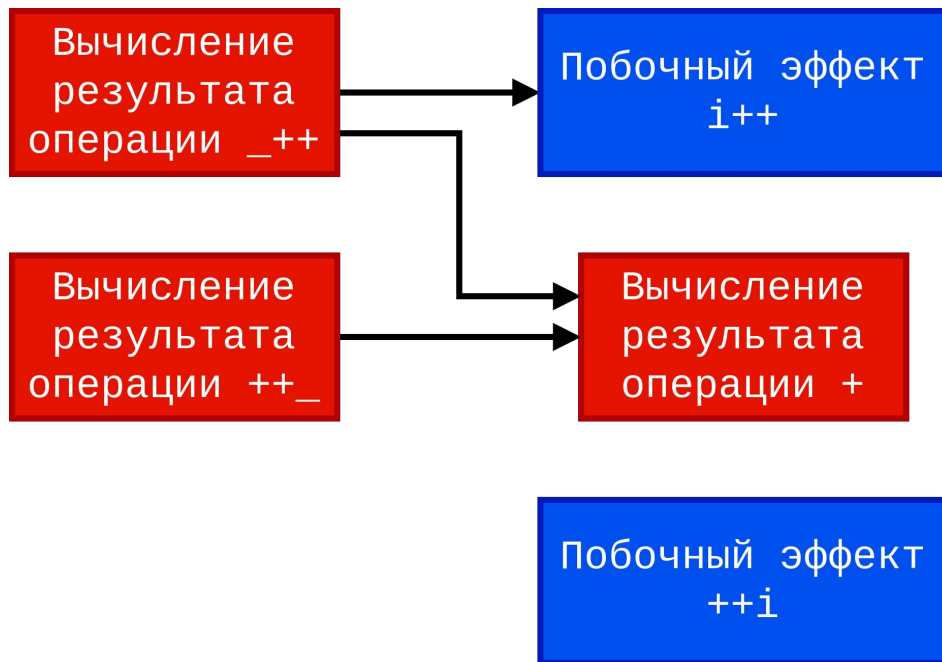
Но когда между \mathbb{A} и \mathbb{B} находится точка следования?

Точки следования

1. Некоторые операции задают точки следования для операндов:
 - В выражениях $A \ \&\& \ B$ и $A \ || \ B$, A, B будет $A \rightarrow B$.
 - В выражении $A? B : C$ будет $A \rightarrow B$ и $A \rightarrow C$.
2. Вызов функции задаёт точку следования:
В выражении $F(A, B)$ верно:
 $A \rightarrow \langle \text{вызов ф-ии} \rangle, B \rightarrow \langle \text{вызов ф-ии} \rangle, F \rightarrow \langle \text{вызов ф-ии} \rangle$
3. Между операторами идёт точка следования:
Для двух последовательных операторов: $A; B$ будет $A \rightarrow B$.
4. Точка следования находится после вычисления управляющего выражения в `if/for/while/do`.
5. Для операций $x++$ и $x--$ вычисление значения следует до побочного эффекта.
6. Для операции $A = B$ будет $(\text{вычисление } B) \rightarrow (\text{запись в } A)$.

Точки следования

Отношение следования для вычислений выражения $i++ + ++i$.



Undefined Behavior и следование

Undefined Behavior возникает тогда, когда:

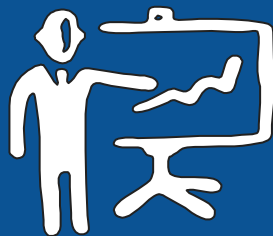
1. Вычисления A и B ($A \nrightarrow B$ и $B \nrightarrow A$) выполняют побочные эффекты над одной и той же переменной.
`i = ++i + i++; // undefined behavior`
`i = i++ + 1; // undefined behavior`
`f(++i, ++i); // undefined behavior`
`f(i = -1, i = -1); // undefined behavior`
2. Вычисление A – это побочный эффект объекта, а B – использование объекта, A и B не упорядочены ($A \nrightarrow B$, $B \nrightarrow A$).

Порядок вычисления: задачи

Указать выражения, содержащие UB:

1. `i = i + 1`
2. `i = i++`
3. `i++ * i++`
4. `i++ && i++`
5. `a[i] = i++`
6. `a[i++] = i`
7. `i = f(i++)`
8. `i += i`
9. `a ^= b ^= a ^= b`
10. `a ^= b, b ^= a, a ^= b`
11. `i++ ? i++ : ++i`

Вопросы?



Красивые иконки взяты с сайта handdrawngoods.com