

# Языки программирования (осень 2018)

В начало ► Мои курсы ► ЯП 2018 ► Оценка лабораторных работ ► Лабораторная работа №5 ► Работа

## Лабораторная работа №5

### Моя работа

#### Инструкции для работы ▼

Для того, чтобы отправить работу на оценку, нажмите "Начало подготовки Вашей работы".

На открывшейся странице:

- в поле **Название** появившегося окна укажите точное название загружаемого файла (строчными буквами, с расширением, без пробелов);
- поле **Содержимое работы** оставьте пустым;
- из папки с решением перетащите загружаемый файл в поле **Приложение** или загрузите файл в это поле, используя кнопку "Добавить.." в меню этого поля;
- выполнив перечисленные пункты нажмите кнопку "Сохранить".

При необходимости, пока не окончена фаза представления работ, можно откорректировать представление работы нажав кнопку "Редактировать работу"

### lab-5.lsp

представлено: Понедельник, 19 Ноябрь 2018, 18:26 | изменено: Пятница, 23 Ноябрь 2018, 19:43

-  lab-5.lsp



#### Самооценка

от Максим Кулаков

Оценка: 100,00 из 100,00

#### Форма оценки ▼

#### Критерий 1

Функции `0?` и `1?` заключаются в том, чтобы проверить, являются ли аргумент функции числом, и если так, применить к аргументу числовое сравнение с нулем или единицей соответственно. Функции должны выглядеть так:

```
(defun 0? (n) (and (numberp n) (zerop n)))  
(defun 1? (n) (and (numberp n) (= n 1)))
```

- Следует снизить оценку на 2 балла, если в решении используется `equalp` (функция, не была упомянута среди дозволённых).

- Не следует ставить за решение более 2 баллов, если вместо функции `=` используются `eq` или/и `equal`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 1

5

### Комментарий к Критерий 1

## Критерий 2

---

Функциям `+?` и `*?` достаточно проверить, что их аргумент является списком с требуемым первым элементом.

```
(defun +? (l) (and (listp l) (eq (car l) '+)))  
(defun *? (l) (and (listp l) (eq (car l) '*)))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Не следует снижать оценку, если вместо `eq` используется `equal`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 2

5

### Комментарий к Критерий 2

## Критерий 3

---

Функции `-?` и `/?` должны дополнительно проверять наличие хотя бы одного дополнительного элемента в списке.

```
(defun -? (l) (and (listp l) (eq (car l) '-') (> (length l) 1)))  
(defun /? (l) (and (listp l) (eq (car l) '/') (> (length l) 1)))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Не следует снижать оценку, если вместо `eq` используется `equal`.
- Не следует снижать оценку, если вместо `length` используется сравнение хвостов списков (вызовов `cdr`, `cddr` и т.п.) с `nil`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 3

5

## Критерий 4

Функции `expt?`, `sqrt?`, `sin?`, `cos?`, `tan?`, `asin?`, `acos?`, `atan?` и `exp?` должны проверять количество элементов в списке с конкретным значением.

```
(defun expt? (l) (and (listp l) (eq (car l) 'expt) (= (length l) 3)))
(defun sqrt? (l) (and (listp l) (eq (car l) 'sqrt) (= (length l) 2)))
(defun sin? (l) (and (listp l) (eq (car l) 'sin) (= (length l) 2)))
(defun cos? (l) (and (listp l) (eq (car l) 'cos) (= (length l) 2)))
(defun tan? (l) (and (listp l) (eq (car l) 'tan) (= (length l) 2)))
(defun asin? (l) (and (listp l) (eq (car l) 'asin) (= (length l) 2)))
(defun acos? (l) (and (listp l) (eq (car l) 'acos) (= (length l) 2)))
(defun atan? (l) (and (listp l) (eq (car l) 'atan) (= (length l) 2)))
(defun exp? (l) (and (listp l) (eq (car l) 'exp) (= (length l) 2)))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Не следует снижать оценку, если вместо `eq` используется `equal`.
- Не следует снижать оценку, если вместо `length` используется сравнение хвостов списков (вызовов `cdr`, `cddr` и т.п.) с `nil`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 4

5

## Комментарий к Критерий 4

## Критерий 5

Функция `log?` должна проверять, что количество элементов в списке должно быть два или три:

```
(defun log? (l)
  (and (listp l) (eq (car l) 'log)
    (let ((len (length l)))
      (and (> len 1) (< len 4))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Не следует снижать оценку, если вместо `eq` используется `equal`.
- Не следует снижать оценку, если вместо `length` используется сравнение хвостов списков (вызовов `cdr`, `cddr` и т.п.) с `nil`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 5

## Комментарий к Критерий 5

## Критерий 6

Функции `make+` и `make*` должны получать произвольное количество аргументов и конструировать из них соответствующее выражение. Функции могут выглядеть так:

```
(defun make+ (&rest l) (cons '+ l))
(defun make* (&rest l) (cons '* l))
```

Для конструирования выражений можно использовать конструкции, обычно применяемые при написании макросов:

```
(defun make+ (&rest l) `(+ ,@l))
(defun make* (&rest l) `(* ,@l))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 6

5

## Комментарий к Критерий 6

## Критерий 7

Функции `make-` и `make/` должны получать один обязательный аргумент и произвольное количество дополнительных аргументов. Функции могут выглядеть так:

```
(defun make- (a &rest l) (cons '- (cons a l)))
(defun make/ (a &rest l) (cons '/ (cons a l)))
```

Для конструирования выражений можно использовать конструкции, обычно применяемые при написании макросов:

```
(defun make- (a &rest l) `(- ,a ,@l))
(defun make/ (a &rest l) `( / ,a ,@l))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 7

5

## Критерий 8

Функции `makeexpt`, `makesqrt`, `makesin`, `makecos`, `maketan`, `makeasin`, `makeacos`, `makeatan` и `makeexp` должны получать точное количество аргументов. Функции могут выглядеть так:

```
(defun makeexpt (a b) (list 'expt a b))
(defun makesqrt (l) (list 'sqrt l))
(defun makesin (l) (list 'sin l))
(defun makecos (l) (list 'cos l))
(defun maketan (l) (list 'tan l))
(defun makeasin (l) (list 'asin l))
(defun makeacos (l) (list 'acos l))
(defun makeatan (l) (list 'atan l))
(defun makeexp (l) (list 'exp l))
```

или, например, так:

```
(defun makeexpt (a b) `(expt ,a ,b))
(defun makesqrt (l) `(sqrt ,l))
(defun makesin (l) `(sin ,l))
(defun makecos (l) `(cos ,l))
(defun maketan (l) `(tan ,l))
(defun makeasin (l) `(asin ,l))
(defun makeacos (l) `(acos ,l))
(defun makeatan (l) `(atan ,l))
(defun makeexp (l) `(exp ,l))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 8

5

### Комментарий к Критерий 8

## Критерий 9

Функция `makelog` должна получать один обязательный аргумент и один необязательный. Функция может выглядеть так:

```
(defun makelog (a &optional b)
  (if b (list 'log a b) (list 'log a)))
```

или, например, так:

```
(defun makelog (a &optional b)
  (if b `(log ,a ,b) `(log ,a)))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Решение заслуживает не более 1 балла, если вместо `&optional` второй параметр объявлен с помощью `&rest`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 9

5

### Комментарий к Критерий 9

## Критерий 10

Реализации `+-normalize` и `*-normalize` должны иметь примерно следующий вид

```
(defun +-normalize (expr)
  (cond ((null (cdr expr)) 0)
        ((null (cddr expr)) (normalize (cadr expr)))
        (T (make+ (normalize (cadr expr))
                   (normalize (cons '+ (cddr expr)))))))
(defun *-normalize (expr)
  (cond ((null (cdr expr)) 1)
        ((null (cddr expr)) (normalize (cadr expr)))
        (T (make* (normalize (cadr expr))
                   (normalize (cons '* (cddr expr)))))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 10

5

### Комментарий к Критерий 10

## Критерий 11

Реализации `--normalize` и `/-normalize` должны иметь примерно следующий вид

```
(defun --normalize (expr)
  (if (null (cddr expr)) (make* -1 (normalize (cadr expr)))
      (make+ (normalize (cadr expr))
              (make* -1 (normalize (cons '+ (cddr expr)))))))
(defun /-normalize (expr)
  (if (null (cddr expr)) (make/ (normalize (cadr expr)))
      (make* (normalize (cadr expr))
              (make/ (normalize (cons '* (cddr expr)))))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 11

5

### Комментарий к Критерий 11

## Критерий 12

Реализации `expt-normalize`, `sin-normalize`, `cos-normalize`, `asin-normalize`, `acos-normalize`, `atan-normalize` и `exp-normalize` похожи: сначала нормализуется аргумент, а затем из него (из них)конструируется выражение:

```
(defun expt-normalize (expr)
  (makeexpt (normalize (cadr expr))
            (normalize (caddr expr))))
(defun sin-normalize (expr)
  (makesin (normalize (cadr expr))))
(defun cos-normalize (expr)
  (makecos (normalize (cadr expr))))
(defun asin-normalize (expr)
  (makeasin (normalize (cadr expr))))
(defun acos-normalize (expr)
  (makeacos (normalize (cadr expr))))
(defun atan-normalize (expr)
  (makeatan (normalize (cadr expr))))
(defun exp-normalize (expr)
  (makeexp (normalize (cadr expr))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 12

5

### Комментарий к Критерий 12

## Критерий 13

---

Реализация `sqrt-normalize` и `tan-normalize` преобразуют полученные выражения в вызовы других функций

```
(defun sqrt-normalize (expr)
  (makeexpt (normalize (cadr expr)) 1/2))
(defun tan-normalize (expr)
  (let ((arg (normalize (cadr expr))))
    (make* (makesin arg) (make/ (makecos arg)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 13

5

### Комментарий к Критерий 13

## Критерий 14

---

Реализация `log-normalize` должна выражать полученное выражение через натуральный логарифм

```
(defun log-normalize (expr)
  (let ((ln (makelog (normalize (cadr expr)))))
    (if (null (cddr expr)) ln
        (make* ln (make/ (makelog (normalize (caddr expr))))))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 14

5

### Комментарий к Критерий 14

## Критерий 15

---

Функция `simplify+` должна выглядеть следующим образом



```
(defun simplify+ (expr)
  (let ((1st-arg (simplify (cadr expr)))
        (2nd-arg (simplify (caddr expr))))
    (cond
      ((0? 1st-arg) 2nd-arg)
      ((0? 2nd-arg) 1st-arg)
      ((numberp 1st-arg) (simplify+-aux1 2nd-arg 1st-arg))
      ((numberp 2nd-arg) (simplify+-aux1 1st-arg 2nd-arg))
      ((+? 1st-arg) (simplify+-aux2 2nd-arg 1st-arg))
      ((+? 2nd-arg) (simplify+-aux2 1st-arg 2nd-arg))
      (T (make+ 1st-arg 2nd-arg)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 15

5

### Комментарий к Критерий 15

## Критерий 16

Функция `simplify+-aux1` должна выглядеть следующим образом

```
(defun simplify+-aux1 (expr num)
  (cond ((numberp expr) (+ expr num))
        ((+? expr)
         (let ((1st-arg (cadr expr)))
           (if (numberp 1st-arg)
               (let ((num-sum (+ num 1st-arg)))
                 (if (zerop num-sum)
                     (cons '+ (caddr expr))
                     (cons '+ (cons num-sum (caddr expr)))))
               (cons '+ (cons num (cdr expr))))))
        (T (make+ num expr))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 16

5

### Комментарий к Критерий 16

## Критерий 17

Функция `simplify+-aux2` должна выглядеть следующим образом

```
(defun simplify+-aux2 (expr +expr)
  (cond ((numberp expr) (simplify+-aux1 +expr expr))
        ((+? expr)
         (let ((1st-arg-of-1st (cadr expr))
               (1st-arg-of-2nd (cadr +expr)))
           (cond
            ((numberp 1st-arg-of-2nd)
             (simplify+-aux1 (append expr (cddr +expr))
                              1st-arg-of-2nd))
            ((numberp 1st-arg-of-1st)
             (simplify+-aux1 (append +expr (cddr expr))
                              1st-arg-of-1st))
            (T (append expr (cdr +expr))))))
        (T (append +expr (list expr)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Обратите внимание на то, что второй аргумент функции `append` в последней строке функции дополнительно оборачивается в список, чтобы слагаемое сформировалось корректно. Если такого оборачивания нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 17

5

### Комментарий к Критерий 17

## Критерий 18

Функция `simplify*` должна выглядеть следующим образом

```
(defun simplify* (expr)
  (let ((1st-arg (simplify (cadr expr)))
        (2nd-arg (simplify (caddr expr))))
    (cond
      ((0? 1st-arg) 0)
      ((0? 2nd-arg) 0)
      ((1? 1st-arg) 2nd-arg)
      ((1? 2nd-arg) 1st-arg)
      ((numberp 1st-arg) (simplify*-aux1 2nd-arg 1st-arg))
      ((numberp 2nd-arg) (simplify*-aux1 1st-arg 2nd-arg))
      ((*? 1st-arg) (simplify*-aux2 2nd-arg 1st-arg))
      ((*? 2nd-arg) (simplify*-aux2 1st-arg 2nd-arg))
      ((and (/? 1st-arg) (equal (cadr 1st-arg) 2nd-arg)) 1)
      ((and (/? 2nd-arg) (equal (cadr 2nd-arg) 1st-arg)) 1)
      ((+? 1st-arg)
       (simplify (cons '+ (simplify*-through+ (cdr 1st-arg) 2nd-arg))))
      ((+? 2nd-arg)
       (simplify (cons '+ (simplify*-through+ (cdr 2nd-arg) 1st-arg))))
      (T (make* 1st-arg 2nd-arg)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Обратите внимание на то, что после того, как получено выражение с помощью `simplify*-through+`, от него снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не более одного балла.
- Возможна несколько отличная реализация, в которой, когда один из множителей является обратным элементом второго, их сравнение проводится до их упрощения. Несмотря на то, что в задании такое упрощение не предполагалось, снижать оценку за такое решение не следует.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 18

5

### Комментарий к Критерий 18

## Критерий 19

Функция `simplify*-aux1` должна выглядеть следующим образом

```
(defun simplify*-aux1 (expr num)
  (cond ((numberp expr) (* expr num))
        ((=? expr)
         (let ((1st-arg (cadr expr)))
           (if (numberp 1st-arg)
               (let ((num-prod (* num 1st-arg)))
                 (if (= num-prod 1)
                     (cons '* (cddr expr))
                     (cons '* (cons num-prod (cddr expr)))))
               (cons '* (cons num (cdr expr))))))
        ((+? expr)
         (simplify (cons '+ (simplify*-through+ (cdr expr) num))))
        (T (make* num expr))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Обратите внимание на то, что после того, как получено выражение с помощью `simplify*-through+`, от него снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 19

5

## Комментарий к Критерий 19

## Критерий 20

Функция `simplify*-aux2` должна выглядеть следующим образом

```
(defun simplify*-aux2 (expr *expr)
  (cond ((numberp expr) (simplify*-aux1 *expr expr))
        ((=? expr)
         (let ((1st-arg-of-1st (cadr expr))
               (1st-arg-of-2nd (cadr *expr)))
           (cond
            ((numberp 1st-arg-of-2nd)
             (simplify*-aux1 (append expr (cddr *expr))
                              1st-arg-of-2nd))
            ((numberp 1st-arg-of-1st)
             (simplify*-aux1 (append *expr (cddr expr))
                              1st-arg-of-1st))
            (T (append expr (cdr *expr)))))
         (T (append *expr (list expr)))))
        ((+? expr)
         (simplify (cons '+ (simplify*-through+ (cdr expr) *expr))))
        (T (append *expr (list expr)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Обратите внимание на то, что после того, как получено выражение с помощью `simplify*-through+`, от него снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не

более одного балла.

- Обратите внимание на то, что второй аргумент функции `append` в последней строке функции дополнительно оборачивается в список, чтобы множитель сформировался корректно. Если такого оборачивания нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 20

5

### Комментарий к Критерий 20

## Критерий 21

---

Функции `simplify*-through+` и `simplify/-through*` достаточно просты и должны иметь вид

```
(defun simplify*-through+ (exprs n)
  (if (null exprs) '()
      (cons (make* n (car exprs))
            (simplify*-through+ (cdr exprs) n))))
(defun simplify/-through* (exprs)
  (if (null exprs) '()
      (cons (make/ (car exprs))
            (simplify/-through* (cdr exprs)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 21

5

### Комментарий к Критерий 21

## Критерий 22

---

Функция `simplify/` может иметь вид

```
(defun simplify/ (expr)
  (let ((arg (simplify (cadr expr))))
    (cond
      ((numberp arg) (/ arg))
      ((/? arg) (cadr arg))
      ((*? arg) (simplify (cons '* (simplify/-through* (cdr arg)))))
      (t (make/ arg)))))
```

- Обратите внимание на то, что после того, как получено выражение с помощью `simplify/-through*`, от него снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 22

5

## Комментарий к Критерий 22

## Критерий 23

Функция `simplifyexpt` должна рассматривать достаточно большое количество случаев упрощения

```
(defun simplifyexpt (expr)
  (let ((1st-arg (simplify (cadr expr)))
        (2nd-arg (simplify (caddr expr))))
    (cond
      ((0? 2nd-arg) 1)
      ((1? 2nd-arg) 1st-arg)
      ((0? 1st-arg) 0)
      ((1? 1st-arg) 1)
      ((and (numberp 1st-arg) (numberp 2nd-arg)) (expt 1st-arg 2nd-arg))
      ((expt? 1st-arg) (simplify (makeexpt
                                   (cadr 1st-arg)
                                   (make* 2nd-arg (caddr 1st-arg)))))
      ((exp? 1st-arg) (simplify (makeexp (make* 2nd-arg (cadr 1st-arg)))))
      (t (makeexpt 1st-arg 2nd-arg)))))
```

- Оценка должна быть снижена на два балла, если не используются функции `0?` и `1?` при сравнении с нулем и единицей.
- Оценка должна быть снижена до 1 балла, если для сравнения с нулем и единицей используются функции `eq`, `equal` или `equalp`.
- Обратите внимание на то, что после того, как преобразуются выражения возведения в степень показательной функции (два предпоследних случая), от полученного выражения снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 23

5

## Комментарий к Критерий 23

## Критерий 24

simplify-fun-1

МОЖЕТ ИМЕТЬ ВИД

```
(defun simplify-fun-1 (expr)
  (let* ((arg (simplify (cadr expr)))
        (newexpr (list (car expr) arg)))
    (if (numberp arg)
        (eval newexpr)
        newexpr)))
```

- Это единственная функция, в которой желательно использовать вызов `eval`. Если такой вызов отсутствует, оценку следует снизить на 1 балл.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 24

5

### Комментарий к Критерий 24

### Критерий 25

simplifyexp

```
W simplifylog
```

могут иметь вид

```
(defun simplifyexp (expr)
  (let ((arg (simplify (cadr expr))))
    (cond
      ((numberp arg) (exp arg))
      ((log? arg) (cadr arg))
      (T (makeexp arg)))))

(defun simplifylog (expr)
  (let ((arg (simplify (cadr expr))))
    (cond
      ((numberp arg) (log arg))
      ((exp? arg) (cadr arg))
      ((expt? arg) (simplify
                     (make* (caddr arg)
                           (makelog (cadr arg)))))
      (T (makelog arg)))))
```

- Обратите внимание на то, что в функции `simplifylog`, после преобразования логарифма степени в произведение (в предпоследнем случае), от него снова запускается функция `simplify`. Если такого вызова нет, решение заслуживает не более одного балла.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 25

5

## Критерий 26

Функции `+-deriv`, `*-deriv` и `/-deriv` реализуют широко известные формулы вычисления производных и могут иметь вид

```
(defun +-deriv (expr var)
  (make+ (deriv (cadr expr) var)
        (deriv (caddr expr) var)))
(defun *-deriv (expr var)
  (let ((1st-arg (cadr expr))
        (2nd-arg (caddr expr)))
    (make+ (make* (deriv 1st-arg var) 2nd-arg)
            (make* (deriv 2nd-arg var) 1st-arg))))
(defun /-deriv (expr var)
  (let ((arg (cadr expr)))
    (make* -1 (make* (deriv arg var) (make/ (make* arg arg))))))
```

- Функции `+-deriv` и `*-deriv` определяются однозначно (с точностью до перемены мест слагаемых или множителей). А при реализации функции `/-deriv` возможны варианты. Функция не обязана генерировать выражение в нормализованном виде, поэтому возможен, например, следующий вариант:

```
(defun /-deriv (expr var)
  (let ((arg (cadr expr)))
    (make- (make/ (deriv arg var) (makeexpt arg 2))))))
```

Снижать оценку за подобную реализацию не следует.

- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 26

5

### Комментарий к Критерий 26

## Критерий 27

Функции `sin-deriv`, `cos-deriv`, `exp-deriv` и `log-deriv` могут иметь вид



```
(defun sin-deriv (expr var)
  (let ((arg (cadr expr)))
    (make* (makecos arg)
            (deriv arg var)))))
(defun cos-deriv (expr var)
  (let ((arg (cadr expr)))
    (make* -1
            (make* (makesin arg)
                    (deriv arg var))))))
(defun exp-deriv (expr var)
  (make* (deriv (cadr expr) var)
          expr))
(defun log-deriv (expr var)
  (let ((arg (cadr expr)))
    (make* (deriv arg var)
            (make/ arg)))))
```

- Решение заслуживает не более 1 балла, если оно сложнее, чем приведенное выше.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

### Оценка для Критерий 27

5

### Комментарий к Критерий 27

## Критерий 28

Функции `expt-deriv`, `asin-deriv`, `acos-deriv` и `atan-deriv` реализуют менее популярные, но не более сложные формулы для взятия производных. Эти функции могут иметь вид

```

(defun expt-deriv (expr var)
  (let ((base (cadr expr))
        (order (caddr expr)))
    (make+
      (make* order
        (make* (makeexpt base (make+ order -1))
          (deriv base var)))
      (make* expr
        (make* (makeexp log base)
          (deriv order var))))))
(defun asin-deriv (expr var)
  (let ((arg (cadr expr)))
    (make/ (deriv arg var)
      (makesqrt (make- 1 (makeexpt arg 2))))))
(defun acos-deriv (expr var)
  (make- (deriv (makeasin (cadr expr)) var)))
(defun atan-deriv (expr var)
  (let ((arg (cadr expr)))
    (make/ (deriv arg var)
      (make+ 1 (makeexpt arg 2)))))

```

- Не следует снижать оценки в случае если `acos-deriv` не использует `asin-deriv`.
- Ваша оценка должна быть беспристрастной. Вполне нормально, если решение отличается от приведенных выше вариантов. Вы проверяете правильность решения и его стиль, а не степень совпадения решения с приведенными вариантами.

## Оценка для Критерий 28

5

## Комментарий к Критерий 28

## НАВИГАЦИЯ



В начало

■ Личный кабинет

Страницы сайта

Мои курсы

Графика Осень 2018

ЯП 2018

Участники

 Значки

 Компетенции

 Оценки

Общее


Форумы курса


Материалы по тематике курса


Лабораторные работы

Оценка лабораторных работ

 Лабораторная работа №0

 Лабораторная работа №1


 Лабораторная работа №2

 Лабораторная работа №3

 Лабораторная работа №4

 **Лабораторная работа №5**

■ Моя работа

 Лабораторная работа №6

Раздел 1. Standard ML

Раздел 2. Haskell

Раздел 3. LISP

Раздел 4. Ruby

Раздел 5. PROLOG

Реферат

Аттестация

Бонусы

Напоминалки

ИКБ

On-line

---

Вы зашли под именем Максим Кулаков (Выход)

ЯП 2018