**1. Определение уравнения**

lisp

Копировать код

(setq meEquation '(A / x + B = C))

Здесь создается список meEquation, который представляет уравнение в виде списка символов и операций. Это уравнение:

Ax+B=C\frac{A}{x} + B = CxA​+B=C

где:

* AAA — переменная или константа.
* xxx — переменная.
* BBB — константа.
* CCC — результат уравнения.

**2. Функция left**

lisp

Копировать код

(defun left(e op)

(if e

(let

((f (car e)))

(if (eq f op)

nil

(cons f

(left (cdr e) op)))))

nil))

Эта рекурсивная функция выделяет **левую часть уравнения** до символа оператора op. Она работает следующим образом:

* Если список e не пуст, она берет первый элемент списка (car e) и проверяет, является ли он оператором op.
* Если текущий элемент списка — это оператор op, то функция завершает рекурсию и возвращает nil.
* В противном случае, она добавляет текущий элемент в результат и рекурсивно продолжает искать до тех пор, пока не встретит оператор op.

Пример работы:

* Для уравнения A / x + B = C и оператора =, она вернет левую часть уравнения до знака равенства: (A / x + B).

**3. Функция right**

lisp

Копировать код

(defun right(e op)

(if e

(let

((r (cdr e)))

(if (eq (car e) op)

r

(right r op))))

nil))

Эта рекурсивная функция выделяет **правую часть уравнения** после оператора op.

* Если список не пуст, она проверяет первый элемент списка (car e) на соответствие оператору op.
* Если это оператор, функция возвращает остаток списка (cdr e), который является правой частью.
* Если это не оператор, функция рекурсивно продолжает искать в оставшемся списке.

Пример работы:

* Для уравнения A / x + B = C и оператора =, она вернет правую часть уравнения: (C).

**4. Функция res**

lisp

Копировать код

(defun res(e)

(let

((l (left e '=)))

(if l

(let

((expr\_c (car (right e '=)))

(expr (left l '=)))

(let

((expr\_left (left expr '+)))

(list

(car (right expr\_left '/))

`=

(car (left expr\_left '/))

`/

(list

expr\_c

`-

(car (right expr '+))

)))))

nil))

Эта функция решает уравнение. Основной принцип — разбить уравнение на части и решить его шаг за шагом:

1. **Разделение уравнения на части:**
   * l — левая часть уравнения до знака равенства.
   * expr\_c — правый операнд (в данном случае C).
   * expr — левая часть уравнения до знака равенства в левой части.
2. **Решение уравнения:**
   * Внутри функции происходит решение уравнения для переменной xxx, используя простые алгебраические преобразования:
     + Из уравнения Ax+B=C\frac{A}{x} + B = CxA​+B=C, можно выразить xxx как x=AC−Bx = \frac{A}{C - B}x=C−BA​.
     + В коде это делается с помощью рекурсивного разбиения на части и манипулирования операциями деления и вычитания.

**5. Функция res\_init**

lisp

Копировать код

(defun res\_init(e)

(let

((r (res e)))

(if r

r

"Ошибка")))

Эта функция служит для обработки ошибки при решении уравнения. Если функция res не может вернуть результат (например, если уравнение задано неверно), она возвращает строку "Ошибка".

**6. Вывод результата**

lisp

Копировать код

(print (res\_init meEquation))

Здесь выводится результат решения уравнения. Если уравнение правильно задано, будет выведен результат в виде списка, который описывает решение. Если в процессе решения возникает ошибка, будет выведено сообщение "Ошибка".

**Пример работы кода**

Для уравнения (A / x + B = C) результат работы кода будет следующего вида:

css

Копировать код

(A / (C - B)) = x

где x=AC−Bx = \frac{A}{C - B}x=C−BA​.

**Заключение**

Этот код реализует базовое решение линейного уравнения с операциями деления и сложения. Он разбивает уравнение на компоненты, извлекает нужные части, а затем решает его для переменной xxx.