

# Задание №17

## Двойная диспетчеризация

### 1. Общая постановка задачи

Для выполнения задания вам понадобятся файлы `task17-math.rb`, `task17-NN.rb`. Скачайте их с портала. Данные файлы представляют собой работоспособную программу (описание см. ниже). Для выполнения задания необходимо разобраться с содержанием кода в файле `task17-math.rb`.

Переименуйте файл `task17-NN.rb`, заменив `NN` на номер вашего варианта.

Добавьте код в файл `task17-NN.rb`, где `NN` — номер вашего варианта, реализующий изменения в функционировании программы, описанные в задании.

Файл `task17-NN.rb`, где `NN` — номер вашего варианта, загрузите на портал в качестве выполненного задания.

### Описание оригинального функционала

В целом функционал приложения подобен функционалу интерпретатора языка MUPL, разработанного в лабораторной работе №4. Только в этот раз команды MUPL представляют собой вызов конструкторов соответствующих классов, описанных на языке Ruby.

Программа предназначена для моделирования обработки математических выражений, содержащих числовые литералы (`Number`) и имена переменных (`Variable`). Математические выражения в оригинальном варианте программы могут содержать бинарные операции сложения (`Add`) и умножения (`Multiply`), операцию смены знака (унарный минус, `Negate`), операцию взятия производной (`Derivative`) и функцию экспоненты (`Exp`). Кроме этого в выражении могут быть использованы вспомогательные обозначения для подвыражений (`Let`) и конструкторы подвыражений (функции, аргументами которых являются выражения и значениями являются выражения — аналог макросов, `MyFunc`).

Выражение может быть представлено в обработанной и необработанной форме. Обработанное выражение — выражение, в котором произведены возможные подстановки и вычисления.

Сформировать выражение можно последовательно создав соответствующие объекты. Например, сформировать выражение

$$((2x + 3y - z)(5 + x))'_x$$

можно с помощью выражения

```
1 Derivative.new(  
2   Multiply.new(  
3     Add.new(  
4       Add.new(  
5         Multiply.new(Number.new(2),  
6           Variable.new("x")),  
7         Multiply.new(Number.new(3),  
8           Variable.new("y")),  
9       Negate.new(Variable.new("z")),  
10    Add.new(Number.new(5),  
11      Variable.new("x")),  
12  "x")
```

С помощью конструкции `Let` можно определять подвыражения, например, выражение

$$(2x + 3y)(2x + 3y)(2x + 3y)$$

можно определить с помощью следующего выражения

```
1 Let.new("a", Multiply.new(Number.new(2),  
2   Variable.new("x")), # a = 2x  
3   Let.new("b", Multiply.new(Number.new(3),  
4     Variable.new("y")), # b = 3y  
5   Let.new("e", Add.new(Variable.new("a"),
```

```

6         Variable.new("b")),      # e = a + b
7     Multiply.new(Multiply.new(Variable.new("e"),
8         Variable.new("e")),
9         Variable.new("e")))))

```

С помощью MyFunc можно определять функции, формирующие выражения. Например, для формирования выражения

$$(2x + 3y)(2x + 3y)(2x + 3y) + (3y)(3y)(3y)$$

можно применить следующее выражение

```

1 Let.new("a", Multiply.new(Number.new(2), Variable.new("x")),      # a = 2x
2     Let.new("b", Multiply.new(Number.new(3), Variable.new("y")), # b = 3y
3         Let.new("cube",
4             # определение функции cube(e) = e * e * e
5             yFunc.new(nil, "e",
6                 Multiply.new(Multiply.new(Variable.new("e"),
7                     Variable.new("e")),
8                         Variable.new("e"))),
9             # вычисление cube(a + b) + cube(b)
10            Add.new(Call.new(Variable.new("cube"),      # вызов cube(a + b)
11                Add.new(Variable.new("a"),
12                    Variable.new("b"))),
13            Call.new(Variable.new("cube"),      # вызов cube(b)
14                Variable.new("b")))))

```

Обработать выражение можно с помощью метода eval с указанием окружения. Метод eval\_exp (без аргументов) запускает метод eval с пустым окружением.

Окружение представляет собой массив, элементами которого являются пары (массивы из двух элементов), первый элемент которых строка — имя переменной, а второй элемент — значение переменной.

Обработка (вычисление) каждого подвыражения происходит по следующим правилам:

- Значением числовой константы (Number) является сама константа;
- Чтобы вычислить значение переменной (Variable) просматривается окружение, и если в окружении не найдется такой переменной, то в качестве значения выдается сама переменная. Если же в окружении присутствует переменная с таким же именем, то извлекается выражение, соответствующее этому имени в окружении, вычисляется и вычисленное значение выдается в качестве результата;
- Для вычисления суммы (Add) вычисляются сначала слагаемые, после чего происходит попытка добавить одно слагаемое к другому. Если такая попытка завершилась неудачей, то результатом является объект класса Add;
- Для вычисления произведения (Multiply) сначала вычисляется каждый множитель, после чего происходит попытка перемножить полученные результаты. Если перемножение невозможно, то в результате выдается объект класса Multiply.
- Для вычисления смены знака выражения сначала вычисляется подвыражение, а затем происходит попытка сменить его знак. Если такая попытка завершилась неудачей, то результатом является объект класса Negate;
- При обработке функции (MyFunc) формируется замыкание (Closure), состоящее из самой функции и окружения, в котором эта функция была определена. Сформированное замыкание выдается в качестве результата;
- Значением замыкания является само замыкание;
- Вычисление вызова функции (Call) происходит в следующем порядке: в заданном окружении вычисляется функциональное выражение (замыкание) и значение фактического параметра. Формируется окружение для вычисления значения функции: окружение состоит из элементов окружения, взятого из вычисленного замыкания, к которым добавляется имя формального параметра в паре с вычисленным значением фактического параметра. Если функция имеет имя, то в окружение помещается еще и имя функции в паре с ее замыканием (для возможных рекурсивных вызовов). В полученном окружении вычисляется тело функции и результат вычисления выдается как результат всего выражения.
- Вычисление производной начинается с вычисления выражения, стоящего под знаком производной. Затем результат обрабатывается по правилам вычисления производной.

Стоит отметить, что результат обработки не должен содержать элементы Let, MyFunc, Derivative.

## 2. Предварительные замечания

Изменять содержимое файла `task17-math.rb` не следует. В файле `task17-NN.rb` должно быть описано доопределение и переопределение функционала, определенного в файле `task17-math.rb`.

Ваша задача изменить правила функционирования программы в соответствии с вашим заданием.

**ВАЖНО!!!** Для выполнения своего задания следует пользоваться двойной диспетчеризацией, полагаться на динамическое переопределение классов. При выполнении задания нельзя пользоваться методами `is_a?`, `instance_of?` или собственными методами с подобным функционалом.

Все реализованные элементы решения должны быть подробно прокомментированы. Отсутствие комментариев к решению снижает оценку на 50%.

Решение может быть засчитано только если задание выполнено на 100%.

Не следует делать предположений на счет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Язык Ruby».

## 3. Варианты заданий

1. Внесите следующие изменения в функционирование программы:

- а) Определите класс `PI` для константы  $\pi$ .
- б) Определите классы `Sin` и `Cos` для синуса и косинуса.
- в) Обеспечьте корректное вычисление результатов, там, где такой результат может быть получен.

2. Внесите следующие изменения в функционирование программы:

- а) Определите класс `Divide` для деления.
- б) Определите класс `Log` для логарифма. Логарифм должен задаваться двумя параметрами: выражения для вычисления основания и выражения под знаком логарифма.
- в) Обеспечьте корректное вычисление результатов, там, где такой результат может быть получен.

3. Внесите следующие изменения в функционирование программы:

- а) Определите класс `Divide` для бинарной операции деления.
- б) Определите класс `Expt` для бинарной операции возведения в степень.
- в) Обеспечьте корректное вычисление результатов, там, где такой результат может быть получен.

4. Внесите следующие изменения в функционирование программы:

- а) Определите класс `Div` для бинарной операции деления нацело.
- б) Определите класс `GreaterThanZero` для сравнения своего аргумента с нулем. Результат вычисления такой операции — число один, если аргумент — число больше нуля, число ноль — если аргумент число меньшее или равное нулю. Если результат вычислить невозможно, то значение — объект класса `GreaterThanZero`.
- в) Обеспечьте корректное вычисление результатов, там, где такой результат может быть получен.

5 (бонус 40%). Внесите следующие изменения в функционирование программы:

- а) Определите класс `Minus` для бинарной операции вычитания (операция не должна определяться как частный случай сложения и смены знака).
- б) Определите класс `Abs` для вычисления абсолютного значения.
- в) Переопределите расстановку скобок в выражении при переводе значений объектов в строку, так, чтобы не было лишних скобок. Скобки считаем лишними, если их удаление из выражения не меняет его результат.
- г) Обеспечьте корректное вычисление результатов, там, где такой результат может быть получен.