Лабораторная работа №4
«MUPL»

Предварительные сведения

В ходе выполнения данной работы необходимо написать решение 16 задач на языке Haskell. Все задания связаны с реализацией обработки выражений языка программирования MUPL, описанного в следующем разделе.

Прежде чем приступать к выполнению заданий, ознакомьтесь с замечаниями, изложенными в пункте 3 данного описания.

1. Язык MUPL (Made Up Programming Language)[1]

Программы на MUPL пишутся непосредственно на Haskell. Выражения языка MUPL представляют собой вызовы конструкторов языка Haskell.

Вот описание синтаксиса языка:

- Если s — строка языка Haskell, то `Var` s — выражение языка MUPL (обращение к переменной с именем s). Например, выражение MUPL

```
Var "a"
```

  обращается к значению переменной с именем `"a"` .
- Если n — значение типа Integer (в Haskell), то `IntNum` n — выражение языка MUPL (целочисленная константа). Например, выражение MUPL

```
IntNum 5
```

  обозначает целое число 5.
- `Unit` — выражение языка MUPL (пустой элемент).
- Если e_1 и e_2 — два выражения языка MUPL, то `Add` e_1 e_2 — выражение языка MUPL (сложение). Например, выражение MUPL

```
Add (IntNum 5) (Var "a")
```

  должно возвращать сумму (в виде значения языка MUPL) значения переменной с именем `"a"` с числом 5.
- Если s_1 и s_2 — две строки языка Haskell и e — выражение языка MUPL, то `Fun` s_1 s_2 e — выражение языка MUPL (определение функции). При этом строки s_1 и s_2 задаются только для использования в теле функции: строка s_1 является именем определяемой функции для рекурсивных вызовов, а строка s_2 — имя формального параметра функции. Если s_1 — пустая строка, то определяемая функция считается неименованной. Например, выражение MUPL

```
Fun "" "x"
    (Add (Var "x") (IntNum 5))
```

  определяет неименованную функцию, прибавляющую к своему аргументу число 5.
- Если e_1 и e_2 — два выражения языка MUPL, то `Call` e_1 e_2 — выражение языка MUPL (вызов функции). Например, выражение MUPL

```
Call (Fun "" "x"
       (Add (Var "x") (IntNum 5)))
     (IntNum 3)
```

  вызывает описанную функцию с аргументом равным `(IntNum 3)` .

[1] Оригинальное описание приведено в курсе Д. Гроссмана «Языки программирования» http://courses.cs.washington.edu/courses/cse341/17sp/, https://www.coursera.org/course/proglang

- Если s — строка языка Haskell, а e_1 и e_2 — два выражения языка MUPL, то `Let` `(` s, e_1 `)` e_2 — выражение языка MUPL (конструкция, подобная `let`). Например, выражение MUPL

```
Let ("f", Fun "" "x"
            (Add (Var "x") (IntNum 5)))
  (Call (Var "f") (IntNum 3))
```

  определяет локальную переменную с именем `"f"`, которой присваивается значение — неименованная функция, и вызывает описанную функцию с аргументом равным `(IntNum 3)`.
- Если e_1, e_2, e_3 и e_4 — четыре выражения языка MUPL, то `IfGreater` e_1 e_2 e_3 e_4 — выражение языка MUPL. Это условная конструкция, которая вычисляет первые два выражения, и если значение первого строго больше, чем значение второго, то результатом всей конструкции является значение третьего выражения, а в противном случае — значение четвертого.
- Если e_1 и e_2 — два произвольных выражения языка MUPL, то `Pair` e_1 e_2 — выражение языка MUPL (Объединение значений в пару).
- Если e — выражение языка MUPL, то `Head` e — выражение языка MUPL (Извлечение первого элемента пары).
- Если e — выражение языка MUPL, то `Tail` e — выражение языка MUPL (Извлечение второго элемента пары).
- Если e — выражение языка MUPL, то `IsAUnit` e — выражение языка MUPL (проверка значения выражения на равенство `Unit`).
- `Closure` env f — значение языка MUPL, представляющее замыкание: env представляет собой список пар имя-значение для переменных окружения в замыкании, f — выражение для функции. Замыкания не являются самостоятельными выражениями языка MUPL, а могут возникать в результате вычисления объявлений функций.

Значения языка MUPL: целочисленные константы, `Unit`, замыкания, пары других значений языка MUPL.
Простые примеры выражений на MUPL можно увидеть в файле шаблона тестов.

2. Задания

В шаблоне с решением уже описан тип данных `Expr`, конструкторы которого представляют выражения языка MUPL.

Функция `valOfIntNum e` — селектор (геттер), который извлекает число из контейнера `IntNum` для программы на Haskell. По аналогии с функцией `valOfIntNum e` опишите набор селекторов для некоторых контейнеров, представляющих выражения MUPL.

Решение каждой из последующих задач — 2 строки кода.

1. Селектор `funName e` должен выдавать имя функции, если его аргумент — определение функции. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a function"`

2. Селектор `funArg e` должен выдавать имя формального параметра функции, если его аргумент — определение функции. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a function"`

3. Селектор `funBody e` должен выдавать выражение — тело функции, если его аргумент — определение функции. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a function"`

4. Селектор `pairHead e` должен выдавать первый элемент пары, если его аргумент — конструктор пары. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a pair"`

5. Селектор `pairTail e` должен выдавать второй элемент пары, если его аргумент — конструктор пары. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a pair"`

6. Селектор `closureFun e` должен выдавать функцию замыкания, если его аргумент — замыкание. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a closure"`

7. Селектор `closureEnv e` должен выдавать окружение замыкания, если его аргумент — замыкание. В противном случае должно выдаваться сообщение об ошибке: `"The expression e is not a closure"`

Следующие два задания связаны с организацией списков на языке MUPL. Списки в языке MUPL будем создавать с помощью конструктора пар `Pair` и значения `Unit`. Значение `Unit` будем считать пустым списком. Список с одним значением — пара, в которой первый элемент — значение в списке, а второй элемент — пустой список. Список, содержащий два элемента — пара, в которой первый элемент — голова списка, а второй элемент — хвост списка (список без головного элемента).

Объем решения для каждого из следующих двух заданий — две строки.

8. Опишите функцию `convertListToMUPL l`, где `l` — Haskell-овский список MUPL-выражений. Функция должна формировать MUPL-овский список этих же выражений.

9. Опишите функцию `convertListFromMUPL l`, где `l` — MUPL-овский список MUPL-выражений. Функция должна формировать Haskell-овский список этих же выражений.

Описанная в шаблоне решения функция `envLookUp env str` применяется в решениях последующих заданий. Эта функция осуществляет поиск значения переменной с именем `str` в окружении `env`. Окружение представляет собой список пар: имя-значение.

Функция `evalExp e` — функция, реализующая процесс вычисления выражения на языке MUPL. Она получает MUPL-выражение `e` и вычисляет его значение. Функция `evalExp` вызывает функцию `evalUnderEnv`, организующую все вычисления.

Функция `evalUnderEnv e env` — главная функция, организующая непосредственно процесс вычисления MUPL-выражения. У нее два аргумента: `e` — выражение, которое необходимо вычислить, `env` — окружение, в котором вычисляется `e`.

Первоначально (в шаблоне решения) в `evalUnderEnv` описано только вычисление выражения, представляющего собой обращение к переменной. Выполнение следующего задания должно доопределить эту функцию так, чтобы она могла вычислять любое выражение на MUPL.

10. Необходимо добавить в `evalUnderEnv` недостающие варианты вычисления выражений на MUPL. Организация вычислений должна проводиться согласно следующим правилам:
    - Результатом вычисления обращения к переменной является значение, соответствующее имени этой переменной в текущем окружении.
    - Результатом вычисления любого значения языка MUPL является само это значение (значения языка MUPL перечислены в конце раздела 1). То есть, например, результатом `evalExp (IntNum 5)` должно быть `IntNum 5`.
    - При MUPL-овском сложении должны вычисляться по очереди два подвыражения, и если оба их значения — MUPL-овские целые числа, то должно генерироваться MUPL-овское целое число для их суммы.
    - Выражение `IfGreater e1 e2 e3 e4` должно вычислять по очереди первые два подвыражения. Оба их значения должны быть MUPL-овскими целыми числами. Они сравниваются, и если значение первого выражения строго больше второго, то вычисляется значение третьего выражения, которое и выдается в результате. В противном случае вычисляется значение четвертого выражения.
    - Выражение `Pair e1 e2` должно вычислять два подвыражения и формировать из них пару значений, которая и является значением всего выражения.
    - Выражение `Head e` должно вычислять сначала значение подвыражения `e`, которое должно быть парой. Первый элемент этой пары должен являться значением всего выражения.
    - Выражение `Tail e` должно вычислять сначала значение подвыражения `e`, которое должно быть парой. Второй элемент этой пары должен являться значением всего выражения.
    - Выражение `IsAUnit e` должно вычислять сначала значение подвыражения `e`. Если вычисленное значение — `Unit`, то результатом должно быть `IntNum 1`, иначе `IntNum 0`.
    - В выражении `Let (str, e1) e2` должно сначала вычисляться значение `v1` подвыражения `e1`, после чего формируется окружение для вычисления `e2`: в текущее окружение помещается пара `(str, v1)`. Результат вычисления `e2` в новом окружении является результатом `Let`.
    - Результатом вычисления выражения `Fun name arg e` должно быть замыкание, состоящее из текущего окружения и данного выражения.
    - При вычислении `Call e1 e2` сначала должно быть вычислено значение `v1` выражения `e1`, которое должно быть замыканием. Должно быть вычислено значение `v2` выражения `e2`, после чего сформировано окружение для вычисления значения функции. Оно должно быть составлено из окружения из замыкания `v1`, в которое помещена пара `(arg, v2)`, где `arg` — имя формального параметра функции в замыкании, и, если функция имеет не пустое имя `name`, пара `(name, v1)` (на случай рекурсивных вызовов). В получившемся окружении нужно вычислить значение функции из замыкания и выдать его в качестве результата.

Описание ни одного из случаев не превышает 7 строк.

Решения последующих заданий должны представлять собой функции на Haskell, которые являются своеобразными расширениями языка MUPL, функционируют как макросы языка MUPL. Функции должны быть описаны только средствами языка MUPL.

При решении следующих заданий нельзя использовать функции `evalExp`, `evalUnderEnv` и конструктор `Closure`.

11. Опишите функцию `ifAUnit e1 e2 e3`, являющуюся условной конструкцией языка MUPL, выдающую значение `e2`, если значение `e1` — `Unit`, и `e3` — в противном случае.

    Объем решения — 1–2 строки.

12. Опишите функцию `mLet l e`, где `l` — Haskell-овский список пар «строка-выражение MUPL». Действие `mLet` должно быть таким же как и у `Let`, но не с одним предварительным связыванием, а со всеми, перечисленными в списке `l`.

    Типовое решение состоит из двух строк.

13. Опишите функцию `ifEq e1 e2 e3 e4`, являющуюся условной конструкцией языка MUPL, выдающую значение `e3`, если значения `e1` и `e2` равны, значение `e4` — в противном случае. Предполагаем, что нигде в выражениях `e1`, `e2`, `e3`, и `e4` не упоминаются имена переменных `"_x"` и `"_y"` (т. е. эти имена можно использовать как имена системных MUPL-овских переменных).

    Объем типового форматированного решения — 7 строк.

14. Обозначьте `mMap` функцию на MUPL, являющуюся аналогом Haskell-овской функции `map`. То есть это должна быть функция, определенная в каррированной форме: ее аргументом является функция, результатом — другая функция. Функция-аргумент — функция, которая должна применяться к каждому элементу списка. Функция-результат — функция, которая применяется к списку и результатом которой должен быть список.

    Типовое форматированное решение состоит из девяти строк.

15. Обозначьте через `mMapAddN n` макрос, в котором `n` — MUPL-овское число, а результатом макроса должна быть функция от списка, добавляющая к каждому элементу списка заданное число.

    Типовое решение — две строки.

16. Предположим, что в MUPL рассматриваются только неотрицательные целые числа.

    Свяжите имя `fact` с MUPL-овским определением для функции вычисления факториала заданного MUPL-овского числа `n` обычным рекурсивным алгоритмом.

    Для решения понадобится описать вспомогательную MUPL-овскую функцию для умножения и, возможно вспомогательную функцию для вычитания единицы из числа.

    Объем форматированного решения — порядка 20–25 строк.

Общий объем решения, включая строки, предварительно заданные в файле, должен составить порядка 190–200 строк.

3. Замечания по выполнению заданий

3.1. Необходимый минимум

Для выполнения работы потребуются сведения о следующих функциях, операциях и конструкциях:

- конструкция `data...=...|...|...deriving(...)` для определения типа данных
- конструкция `case...of...|...`
- операция `++` для объединения списков или конкатенации строк
- строки в Haskell являются списками символов. Поэтому проверить, является ли строка пустой, можно, сравнив ее на равенство с пустым списком

Кроме того, могут оказаться полезными функции `foldr`, `foldl`.

Информацию об этих и других конструкциях можно найти в конспекте, приведенном на портале.

3.2. Ограничения

При выполнении данной лабораторной работы нужно соблюдать следующие ограничения:

1. При описании функций не должны быть явно описаны типы аргументов и тип результата;
2. Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, не описанная в разделе 3.1 и не представленная на форумах «Лабораторная работа №3» и «Лабораторная работа №4» — задайте вопрос на форуме «Лабораторная работа №4»;
3. НЕЛЬЗЯ описывать какие-либо ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ, за исключением тех, о которых явно говорится в задании.

3.3. Предостережения насчет решения

Решением каждой задачи должна быть функция с указанным именем и возвращающая значение в той форме, в которой спрашивается в задании. Прежде чем отправить решение на проверку проводите сравнение сигнатуры написанной вами функции с соответствующей сигнатурой, приведенной в разделе 3.4. В частности, обратите внимание, что функции многих аргументов задаются в каррированной форме.

В решениях заданий 11–16 именам языка Haskell должны быть сопоставлены MUPL-овские выражения с той целью, чтобы можно было использовать эти имена в дальнейшем в программах на MUPL в целях сокращения кода.

Не следует делать предположений насчет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Лабораторная работа №4».

3.4. Результат

Запуск корректно выполненного решения должен привести к следующим определениям:

```haskell
funName :: Expr -> [Char]
funArg :: Expr -> [Char]
funBody :: Expr -> Expr
pairHead :: Expr -> Expr
pairTail :: Expr -> Expr
closureFun :: Expr -> Expr
closureEnv :: Expr -> [(String,Expr)]
convertListToMUPL :: [Expr] -> Expr
convertListFromMUPL :: Expr -> [Expr]
evalUnderEnv :: Expr -> [([Char],Expr)] -> Expr
ifAUnit :: Expr -> Expr -> Expr -> Expr
mLet :: [(String,Expr)] -> Expr -> Expr
ifEq :: Expr -> Expr -> Expr -> Expr -> Expr
mMap :: Expr
mMapAddN :: Expr -> Expr
fact :: Expr
```