Задание №9
Функторы

1. Общая постановка задачи

На языке Standard ML опишите реализацию функтора, для получения модуля, обобщающего операции со структурами данных в соответствии с Вашим вариантом задания 8. Решением должен являться функтор, интерфейсная часть результата которого задана сигнатурой модуля из решения задания 8, в которую внесены следующие изменения:

- в сигнатуру добавляется определение типа `item` — типа элементов той структуры данных, которую Вы определяете (тип `real` в или `int` Вашего задания 8);
- везде в сигнатуре (за исключением определения функции `fromReal` / `fromInt`) следует изменить использование типа `real` / `int` на тип `item`.

На вход функтор должен получать модуль, интерфейсная часть которого должна определяться сигнатурой `ITEM`.

С помощью описанного функтора получите модуль, предложенный Вам в качестве Вашего варианта задания 8. Тестовые примеры запусков Вашего решения задания 8 должны срабатывать при загрузке полученного модуля.

С помощью описанного функтора получите два модуля или более, реализующих работу со структурами данных, соответствующими Вашему варианту задания 8, но определёнными над типами данных, отличными от `real` / `int`. В частности, одна из реализаций модуля должна быть определена над типом `int` / `real` (тип `real` для варианта 2 и тип `int` для остальных вариантов).
С помощью набора тестовых примеров продемонстрируйте работу всех функций, доступных в полученных модулях.

В функции ни одно выражение (подвыражение) не должно вычисляться дважды. В случае необходимости такого вычисления нужно связать значение вычисленного выражения с некоторым локальным именем для дальнейшего использования.

Реализация функции должна предполагать, что в ходе вызова параметры заданы корректно (не следует добавлять реализацию «защиты от дурака»).

Файлу с программой дайте имя `task9-NN.sml`, где вместо `NN` — номер вашего варианта в задании 8. Полученный файл загрузите на портал в качестве решения задания.

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

2. Пример выполнения задания

0 (текст задания 8)**.** Опишите модуль `Interval` реализующий возможности работы с интервалами.

Опишите тип данных `interval`, определяющий интервал, элементами которых являются значения типа `real`. Инфиксный конструктор `:-:` должен принимать значения концов интервала в виде пары элементов типа `real * real`.

Необходимо описать вспомогательную функцию `iNormalize` типа `interval -> interval`, возвращающую нормализованный интервал: первый элемент — начало интервала, второй — конец (на случай, если начало интервала лежит за его концом). В дальнейшем, когда с интервалом будем проводить какие-то действия, будем предполагать, что интервал нормализован.

Необходимо реализовать селекторы (геттеры) `iStart`, `iEnd` — функции типа `interval -> real`, извлекающие из интервала значение его начала и конца, соответственно.

Опишите инфиксную функцию `<~` типа `real * interval -> bool`, определяющую принадлежит ли заданный элемент заданному интервалу.

Опишите вспомогательную функцию `iMap` типа `(real -> real) -> interval -> interval`, применяющую заданную функцию к интервалу и возвращающую нормализованный интервал результатов.

Опишите функцию `isEq : interval * interval -> bool`, сравнивающую на равенство два интервала, используя свойство, что два интервала равны, когда равны между собой их стартовые и концевые значения.

Опишите функцию `toString : interval -> string`, превращающую интервал в строку. Интервал должен выводиться как два вещественных значения, перечисленных через точку с запятой в квадратных скобках.

Опишите функции `add`, `mul` типа `interval * interval -> interval` сложения и умножения двух интервалов, используя следующие соотношения.

$$[a_1; b_1] + [a_2; b_2] = [a_1 + a_2; b_1 + b_2]$$
$$[a_1; b_1] \cdot [a_2; b_2] = [\mathsf{min}\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\}; \mathsf{max}\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\}]$$

Опишите функции `negate`, `abs`, `sign` типа `interval -> interval` для смены знака, вычисления модуля и вычисления знака интервала. Результат функций `negate` и `sign` следует реализовать как результат применения к исходному интервалу функции `iMap` с функцией смены знака или функцией вычисления знака вещественного числа соответственно. Функцию вычисления модуля интервала следует реализовать используя следующее соотношение:

$$||[a; b]|| = \begin{cases} [\mathsf{min}\{|a|, |b|\}; \mathsf{max}\{|a|, |b|\}], & 0 \notin [a; b]; \\ [0; \mathsf{max}\{|a|, |b|\}], & 0 \in [a; b]. \end{cases}$$

Опишите функцию `sub` типа `interval * interval -> interval` для разности двух интервалов используя соотношение

$$a - b = a + (-b).$$

Опишите функцию `fromReal` типа `real -> interval` для получения интервала из вещественного числа, воспринимая вещественное число как интервал у которого начало и конец совпадают:

$$n = [n; n].$$

Опишите функцию `divide` типа `: interval * interval -> interval` деления интервалов, используя соотношение:

$$[a_1; b_1]/[a_2; b_2] = [\mathsf{min}\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}; \mathsf{max}\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}].$$

Опишите функцию `recip` типа `interval -> interval` для нахождения обратного элемента используя соотношение

$$a^{-1} = \frac{1}{a}$$

Решение: Содержимое файла `task9-00.sml`:

```
(* сигнатура входного модуля для функтора
 * сигнатура для типа набора операций, необходимых для
 * определения интервалов над этим типом
 *)
signature ITEM = sig
  type item
  val min : item * item -> item
```

```
  val max : item * item -> item
  val isLT : item * item -> bool
  val isGT : item * item -> bool
  val isEq : item * item -> bool
  val toString : item -> string
  val add : item * item -> item
  val mul : item * item -> item
  val negate : item -> item
  val sign : item -> item
  val abs : item -> item
  val divide : item * item -> item
  val fromReal : real -> item
end

(* Модуль, удовлетворяющий сигнатуре ITEM,
 * собравшийий в себя необходимые функции
 * для работы с концами вещественнозначного интервала *)
structure RealItem = struct
  type item = real
  val min = Real.min
  val max = Real.max
  val isLT = Real.<
  val isGT = Real.>
  val isEq = Real.==
  val toString = Real.toString
  val add = Real.+
  val mul = Real.*
  val negate = Real.~
  val sign = real o Real.sign
  val abs = Real.abs
  val divide = Real./
  fun fromReal x = x
end

(* сигнатура для модуля интервала *)
signature INTERVAL = sig
  (* тип данных *)
  type interval
  type item
  (* конструктор *)
  val :-: : item * item -> interval
  val iStart : interval -> item
  val iEnd : interval -> item
  val <~ : item * interval -> bool
  val isEq : interval * interval -> bool
  val toString : interval -> string
  val add : interval * interval -> interval
  val mul : interval * interval -> interval
  val negate : interval -> interval
  val sign : interval -> interval
  val abs : interval -> interval
  val sub : interval * interval -> interval
  val fromReal : real -> interval
  val divide : interval * interval -> interval
  val recip : interval -> interval
end

(* функтор для определения модуля для работы с интевалами
 * входной модуль обозначен через Item
 * к операциям над концами интервала будем обращаться через этот модуль *)
functor IntervalFn (Item : ITEM) : INTERVAL = struct

(* объявляем, что конструктор будет инфиксным  внутри модуля *)
infix :-:
(* тип данных в виде контейнера *)
datatype interval = :-: of Item.item * Item.item

(* тип значений концов интервала *)
type item = Item.item

(* последующие определения повторяют определения из решения задания 8,
 * но с заменой опреаций над типом real на операции над типом item,
 * т.е. те операции, которые выполнялись над концами интервала или
 * их производными должны быть определены в модуле Item *)

(* раскладываем интервал по шаблону и строим новый интервал из составляющих *)
```

```
fun iNormalize (x :-: y) = Item.min (x, y) :-: Item.max (x, y)


fun iStart i = let val x :-: _ = iNormalize i
               in x
               end

fun iEnd i = let val _ :-: x = iNormalize i
             in x
             end

(* объявляем функцию инфиксной *)
infix <~

(* так как функция объявлена инфиксной, то заголовок функции описываем в
 * соответствующем формате *)
fun x <~ i =
  let
    val istart = iStart i
    val iend = iEnd i
  in
    (Item.isGT (x, istart) orelse Item.isEq (x, istart))
    andalso (Item.isLT (x, iend) orelse Item.isEq (x, iend))
  end

fun iMap operation i  =
  iNormalize (operation (iStart i) :-: operation (iEnd i))

fun isEq (i1, i2) =
  Item.isEq (iStart i1, iStart i2)
  andalso Item.isEq (iEnd i1, iEnd i2)

fun toString i =
  "[" ^ Item.toString (iStart i) ^ "; "
      ^ Item.toString (iEnd i) ^ "]"

fun add (i1, i2) =
  Item.add (iStart i1, iStart i2) :-: Item.add (iEnd i1, iEnd i2)

fun mul (x1 :-: y1, x2 :-: y2) =
  let
    val x = Item.mul (x1, x2)
    val xs = [Item.mul (x1, y2), Item.mul (y1, x2), Item.mul (y1, y2)]
    val a = foldr Item.min x xs
    val b = foldr Item.max x xs
  in a :-: b
  end

fun negate i = iMap Item.negate i

fun sign i = iMap Item.sign i

fun abs (x :-: y) =
  let
    val ax = Item.abs x
    val ay = Item.abs y
    val zero = Item.fromReal 0.0
  in
    if Item.isEq (Item.sign x, Item.sign y) then iNormalize (ax :-: ay)
    else zero :-: Item.max (ax, ay)
  end

fun sub (i1, i2) = add (i1, negate i2)

fun fromReal x =
  let
    val realItem = Item.fromReal x
  in realItem :-: realItem
  end

fun divide (x1 :-: y1, i2 as x2 :-: y2) =
  if Item.fromReal 0.0 <~ i2 then raise Div
  else
    let
      val x = Item.divide (x1, x2)
      val xs = [ Item.divide (x1, y2)
```

```
               , Item.divide (y1, x2)
               , Item.divide (y1, y2) ]
      val a = foldr Item.min x xs
      val b = foldr Item.max x xs
    in
      a :-: b
    end

fun recip i = divide (fromReal 1.0, i)

end


(* определяем модуль для работы с вещественнозначными интервалами *)
structure Interval = IntervalFn (RealItem)

(* РАБОТАЕМ С ВЕЩЕСТВЕННОЗНАЧНЫМИ ИНТЕРВАЛАМИ *)
(* создаем более короткий псевдоним имени типа для интервала *)
type interval = Interval.interval
(* вне модуля конструктор :-: префиксный. Чтобы использовать его в более
 * короткой форме, введем для него псевдоним, но прежде этот псевдоним тоже
 * объявим инфиксным *)
infix :-:
val (op :-:) = Interval.:-:
(* То же самое проделываем с функцией <~ *)
infix <~
val (op <~) = Interval.<~

val i1 = ~12.0 :-: ~4.0
val i2 = 16.0 :-: 8.0
val i3 = ~10.0 :-: 5.0
(* вывод значений первых двух интервалов *)
val i1String = Interval.toString i1
val i2Start = Interval.iStart i2
val i2End = Interval.iEnd i2
val i2String = Interval.toString i2
val i3String = Interval.toString i3
(* операции над интервалами *)
val zeroIni1 = 0.0 <~ i1
val zeroIni3 = 0.0 <~ i3
val sevenIni1 = 7.0 <~ i3

(* арифметические операции  над интервалами *)
val i1Plusi2 = Interval.toString (Interval.add (i1, i2))
val i1Minusi3 = Interval.toString (Interval.sub (i1, i3))
val i3Muli1 = Interval.toString (Interval.mul (i3, i1))
val i3Divi1 = Interval.toString (Interval.divide (i3, i1))
val i1negate = Interval.toString (Interval.negate i1)
val i1abs = Interval.toString (Interval.abs i1)
val i3abs = Interval.toString (Interval.abs i3)
val i3Sign = Interval.toString (Interval.sign i3)
(* смешанные арифметические операции, подключающие fromReal *)
val i1Div4 = Interval.toString (Interval.divide (i1, Interval.fromReal 4.0))
val i1Recip = Interval.toString (Interval.recip i1)


(* определяем модуль для работы с целочисленными интервалами *)

(* определим структуру для целочисленного значения *)
structure IntItem = struct
  type item = int
  val min = Int.min
  val max = Int.max
  val isLT = Int.<
  val isGT = Int.>
  val isEq = op =
  val toString = Int.toString
  val add = Int.+
  val mul = Int.*
  val negate = Int.~
  val sign = Int.sign
  val abs = Int.abs
  val divide = Int.div
  val fromReal = trunc
end
```

```
structure IntInterval = IntervalFn (IntItem)

(* РАБОТАЕМ С ЦЕЛОЧИСЛЕННЫМИ ИНТЕРВАЛАМИ *)
(* создаем более короткий псевдоним имени типа для интервала *)
type iinterval = IntInterval.interval

infix :-:
val (op :-:) = IntInterval.:-:
(* То же самое проделываем с функцией <~ *)
infix <~
val (op <~) = IntInterval.<~

val ii1 = ~12 :-: ~4
val ii2 = 16 :-: 8
val ii3 = ~10 :-: 5
(* вывод значений первых двух интервалов *)
val ii1String = IntInterval.toString ii1
val ii2Start = IntInterval.iStart ii2
val ii2End = IntInterval.iEnd ii2
val ii2String = IntInterval.toString ii2
val ii3String = IntInterval.toString ii3
(* операции над интервалами *)
val izeroIni1 = 0 <~ ii1
val izeroIni3 = 0 <~ ii3
val isevenIni1 = 7 <~ ii3

(* арифметические операции  над интервалами *)
val ii1Plusi2 = IntInterval.toString (IntInterval.add (ii1, ii2))
val ii1Minusi3 = IntInterval.toString (IntInterval.sub (ii1, ii3))
val ii3Muli1 = IntInterval.toString (IntInterval.mul (ii3, ii1))
val ii3Divi1 = IntInterval.toString (IntInterval.divide (ii3, ii1))
val ii1negate = IntInterval.toString (IntInterval.negate ii1)
val ii1abs = IntInterval.toString (IntInterval.abs ii1)
val ii3abs = IntInterval.toString (IntInterval.abs ii3)
val ii3Sign = IntInterval.toString (IntInterval.sign ii3)
(* смешанные арифметические операции, подключающие fromReal *)
val ii1Div4 = IntInterval.toString (IntInterval.divide (ii1, IntInterval.fromReal 4.0))
val ii1Recip = IntInterval.toString (IntInterval.recip ii1)



(* определяем модуль для работы с интервалами целочисленных интервалов *)

(* определим структуру для интервального значения *)
structure IntervalItem = struct
  type item = IntInterval.interval
  fun isLT (i1, i2) =
    IntInterval.iEnd i1 < IntInterval.iEnd i2
  fun isGT (i1, i2) =
    IntInterval.iStart i1 > IntInterval.iStart i2
  val isEq = IntInterval.isEq
  fun min (i1, i2) =
     if isLT (i1, i2) then i1 else i2
  fun max (i1, i2) =
     if isGT (i1, i2) then i1 else i2
  val toString = IntInterval.toString
  val add = IntInterval.add
  val mul = IntInterval.mul
  val negate = IntInterval.negate
  val sign = IntInterval.sign
  val abs = IntInterval.abs
  val divide = IntInterval.divide
  val fromReal = IntInterval.fromReal
end

structure IntervalInterval = IntervalFn (IntervalItem)

(* РАБОТАЕМ С ИНТЕРВАЛАМИ ЦЕЛОЧИСЛЕННЫХ ИНТЕРВАЛОВ *)
(* создаем более короткий псевдоним имени типа для интервала *)
type ininterval = IntervalInterval.interval

infix :-:
val (op :-:) = IntervalInterval.:-:
(* То же самое проделываем с функцией <~ *)
infix <~
val (op <~) = IntervalInterval.<~
```

```
(* воспользуемся ранее определенными целочисленными интервалами
 * и организуем интервалы из них *)
val ini1 = ii1 :-: ii2
val ini2 = ii2 :-: ii3
val ini3 = ii1 :-: ii3
(* вывод значений первых двух интервалов *)
val ini1String = IntervalInterval.toString ini1
val ini2Start = IntervalInterval.iStart ini2
val ini2End = IntervalInterval.iEnd ini2
val ini2String = IntervalInterval.toString ini2
val ini3String = IntervalInterval.toString ini3
(* операции над интервалами *)
(* так как мы определили операцию :-: для интервальных интервалов
 * целочисленный интервал проще получить из вещественного числа *)
val inzeroIni1 = IntInterval.fromReal 0.0 <~ ini1
val inzeroIni3 = IntInterval.fromReal 0.0 <~ ini3
val insevenIni3 = IntInterval.fromReal 7.0 <~ ini3
val inii1Ini3 = ii1 <~ ini3
val inii2Ini3 = ii2 <~ ini3
val inii3Ini3 = ii3 <~ ini3

(* арифметические операции  над интервалами *)
val ini1Plusi2 = IntervalInterval.toString (IntervalInterval.add (ini1, ini2))
val ini1Minusi3 = IntervalInterval.toString (IntervalInterval.sub (ini1, ini3))
val ini3Muli1 = IntervalInterval.toString (IntervalInterval.mul (ini3, ini1))
val ini1negate = IntervalInterval.toString (IntervalInterval.negate ini1)
val ini1abs = IntervalInterval.toString (IntervalInterval.abs ini1)
val ini3abs = IntervalInterval.toString (IntervalInterval.abs ini3)
val ini3Sign = IntervalInterval.toString (IntervalInterval.sign ini3)
(* смешанные арифметические операции, подключающие fromReal *)
val ini1Div4 =
  IntervalInterval.toString
    (IntervalInterval.divide (ini1, IntervalInterval.fromReal 4.0))
```

Текст примера (файл `task9-00.sml`) можно загрузить с портала.

3. Необходимый минимум

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

- конструкции **`fun`** и **`val`** для определения функций и переменных
- конструкции **`structure`** и **`struct...end`** для описания модулей
- конструкции **`signature`** и **`sig...end`** для определения сигнатур модулей
- конструкции **`functor`** для получения модулей
- конструкции **`type`** и **`datatype`** для описания типов
- конструкция **`if...then...else...`**
- конструкция **`let...in...end`**
- конструктор кортежа `( , )`
- стандартные арифметические и логические операции, стандартные операции сравнения
- функции модуля `Math`, `Real`, `Int`, `String`, `List`.

Нельзя использовать конструкции и функции, не перечисленные в этом разделе (за исключением функций собственного сочинения). Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, отсутствующая в перечислении, то задайте вопрос на форуме «Язык Standard ML»;