Курсови проекти по Функционално програмиране

СУ-ФМИ, Специалност „Компютърни науки“, 2 курс

Зимен семестър 2013/2014

Обща информация за проектите

Курсовите проекти описани в този файл трябва да бъдат разработени на езика Haskell.

Предаване на курсовите проекти

След като приключите с работата по проекта, трябва да качите решението си в системата moodle.openfmi.net. За целта на страницата на курса ще бъде публикуван специален формуляр.

- Решението на един проект трябва да бъде оформено в един файл с разширение .hs.
- Програмата ви трябва да може да работи на Haskell Platform (http://www.haskell.org/platform/) версия 2013.2.0.0 или по-нова.

Важно: При предаване на курсовите си работи, изпращайте само и единствено файла съдържащ изходния код на решението. Моля, не предавайте други файлове (например .bak файлове, файлове с описание на заданието на проекта и т.н.). Файлът трябва да бъде качен директно, **без да се компресира.**

Защита и оценяване на курсовите работи

По време на защитата проектите се оценяват по няколко критерия, вкл. и следните:

- Дали предаденото решение работи коректно.
- Дали предаденото решение отговаря на всички изисквания, вкл. на тези в настоящия раздел.
- Дали кодът е добре оформен и дали е добре документиран.
- Дали решението е било тествано. (Проверява се как точно е било тествано решението и как е проверено поведението му в различни ситуации).
- Дали по време на защитата можете да внесете корекции в кода.
- Дали добре разбирате и можете да обясните как работи решението.

Всеки от вас трябва да може да отговори на различни въпроси свързани с решението, а също и да бъде в състояние да внесе малки корекции в кода на място.

Ако искате, по време на защитата можете да донесете и да работите на личния си лаптоп, вместо на компютрите в залите.

Съвместна работа / използване на чужд код

Проектите са предвидени за реализация от един човек и трябва да бъдат разработвани самостоятелно.

В рамките на проекта е допустимо да използвате код написан от някой друг (напр. готова библиотека или помощ от ваш приятел/колега), като това трябва ясно да се обяви:

1. При предаването на проекта, под формата на бележка, която ще оставите в системата
2. При защитата на проекта, като още в началото посочите коя част от проекта сте разработили самостоятелно.

Когато в даден проект се използва чужд код се оценяват и (1) способността ви за внедряване на този код във вашето решение (напр. в случаите, когато се използва външна библиотека) и (2) дали добре разбирате какво прави чуждият код.

Оценката на всеки от проектите се формира от онази негова част, която е била самостоятелно разработена от вас.

В случаите на плагиатство се поставя оценка слаб 2,00 за целия курс.

3: Обработка на израз

Даден е текстов файл, в който на един ред е записан аритметичен израз. Изразът може да включва:

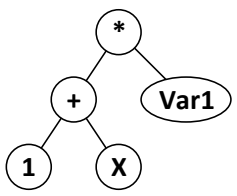
1. Цели или дробни числа (например 123, -10, 5.67)
2. Променливи – поредици от букви и/или цифри, които започват с буква (например `var1`, `SomeReallyLongNameForAVariable`, `x`)
3. Инфиксни операции събиране (+), изваждане (-), умножение (*), деление (/) и степенуване (^);
4. Скоби
5. Произволен брой празни символи (whitespace)

По-долу са дадени примерни валидни изрази:

```
100
3 + 5
(X + 11) ^ 123.4
X * Y
```

Напишете функция `loadExpr path`, която (1) зарежда израза, който е съхранен във файла с път path, (2) проверява дали той е валиден и след това (3) построява дърво, което представя израза. Функцията трябва да връща построеното дърво. За представянето на дървото трябва да се използва създаден от вас тип. По-долу е даден пример за израз и съответстващото му дърво.

```
(1 + X) * Var1
```



Ако в израза има грешка или файлът е празен, функцията трябва по подходящ начин да укаже на извикващия, че е срещнала проблем в работата си. За целта измислете подходящ механизъм.

Напишете и функция `calculate expr args`, която получава израз конструиран от `loadExpr` и пресмята неговата стойност. Ако в израза НЕ участват променливи, аргументът `args` на функцията може да се игнорира. Ако в израза участват променливи, `args` трябва да се използва, за да се укаже какви трябва да бъдат техните стойности. Например за дадения на схемата пример, ако X = 1, Var1 = 2, стойността на израза ще бъде 4. Вие трябва да измислите от какъв тип да бъде `args` и как да го използвате, за да предадете стойностите на аргументите.

4: Работа с двоична пирамида

Двоична пирамида (heap) ще наричаме двоично дърво, което е пълно и освен това за него е в сила едно от следните две свойства:

- **Max:** За всеки връх A е изпълнено, че стойността в A е по-голяма или равна на стойностите в наследниците на A;
- **Min:** За всеки връх A е изпълнено, че стойността в A е по-малка или равна на стойностите в наследниците на A.

Изберете подходящо представяне за двоична пирамида. Представянето трябва да отговаря на следните изисквания:

1. Да позволява да се създават пирамиди съдържащи различни типове данни във възлите си (напр. пирамида съдържаща `Integer`, пирамида съдържаща `String` и т.н.). Ще считаме, че типовете, които се използват са в класа `Ord`;
2. Да пази информация за това от какъв тип е пирамидата (min или max);
3. Да позволява да се представя празната пирамида.

След това напишете следните функции:

- Функция `heapTop H`, която връща стойността на върха на пирамидата H (коренът на дървото).
- Функция `heapAdd H elem`, която добавя нов елемент `elem` в пирамидата H.
- Функция `heapDelete H`, която премахва корена на пирамидата H.

В работата на някои от горе-описаните функции е възможно да възникнат проблеми (например прилагане на функцията `heapDelete` върху празна пирамида). Изберете подходящ начин, по който те ще бъдат третирани.

В най-лошия случай сложността на функциите `heapAdd` и `heapDelete` трябва да бъде O(log N).

Като резултат от работата си, `heapAdd` и `heapDelete` трябва да връщат новополучената пирамида.