ОБЕКТНО-ОРИЕНТИРАНО РАЗШИРЕНИЕ НА SCHEME ИЛИ HASKELL

Да се реализира йерархия от един базов клас и два негови наследника, които да бъдат имплементирани със свои подходящи конструктори, оператори и методи, както и да бъде реализиран полиморфизъм между тях.

Целта на този проект е да се реализират три класа в стил C++ на избран език за Функционално програмиране, като всички „методи“ на този клас да бъдат извиквани и използвани със синтаксис, подобен на този в C++. Нека за пример имаме клас **Point** за точка в двуизмерното пространство. В C++ бихме имали три конструктора за него – по подразбиране, задаващ координати (0;0), по двойка зададени координати и копи-конструктор. Реализацията на Scheme тогава би позволила следните дефиниции:

```
(define p1 (Point '()))      ; създава точката (0;0)
(define p2 (Point '(2 3)))   ; създава точката (2;3)
(define p3 (Point '(p2)))    ; създава още една точка с координати (2;3)
```

Други подходящи функции за този клас биха били умножение на координатите със скалар, събиране с друга точка (взимайки ги като двумерни вектори), както и намиране на разстояние до друга точка. Можем да имаме и метод за принтиране на стандартния изход като форматиран стринг. Всичко това би позволило и следните извиквания:

```
(p2 'print '())                    ; ⇔ p2.print();
→ "Point: [2;3]"

((p3 * 2) 'print '())              ; ⇔ (p3 * 2).print();
→ "Point: [4;6]"

(define p4 (Point '(-2 0)))        ; дефинираме функцията distanceTo
((p1 '+ p2) 'distanceTo '(p4))     ; като метод на класа Point.
→ 5                                ; (не е задължително)
```

При тази реализация можем да имаме и оператори над обектите от класа, както е показано на последния пример, стига да имат смисъл за конкретно имплементирания клас. Една примерна имплементация би използвала асоциативен списък с ламбда функции, по които да извършва подходящите действия на всяко извикване. Заради принципа на immutability в Scheme и Haskell ще

приемаме, че веднъж създадените обекти не могат да се променят и всяко извикване на някой оператор би създавал нов обект от класа.

Изискванията за проекта са да бъдат реализирани поне три класа – един базов и два или повече наследяващи го, които да поддържат множество от конструктори, методи и операции. Изисква се наследяването да бъде „смислено“ и да бъде имплементиран колкото се може по-широк полиморфизъм между класовете с ***поне 5*** виртуални метода. Не е задължително те да бъдат чисто виртуални. Един пример за такава йерархия е базов клас **Number**, наследен от класовете **Rational** и **Complex**. Всички числа могат да бъдат събирани, изваждани, умножавани и делени на свои подобни (няма нужда да имплементираме събиране на рационално с комплексно число). Други операции с тези числа биха били повдигане на степен, коренуване, отново форматирано извеждане на стандартния изход и т.н.. Можете да имплементирате точно тези три класа, може и някакви други – стига те да имат същата йерархична зависимост и да спазват условието за полиморфизъм. Полезно е всеки обект да съдържа и етикет, указващ от кой точно клас е, или да съществува метод, който да връща такъв уникален идентификатор – бил той стринг, число, или друго.

По-богата или дълбока йерархия от класове не е необходима, но ще бъде поощрена с бонус точки. Друга допълнителна функционалност като събирането на различни типове числа в примера по-горе също ще носи бонус. Този проект не изисква промяна на езика или дефиниране на нови синтактични правила – всички изучавани и използвани досега правила за извикване и оценяване важат, а конструкторите на тези класове са просто обикновени функции в средите, в които работим.