Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

ТЕМА ЗА ПРОЕКТ

към курс “Структури от данни и програмиране”

IML

Иван си е измислил език от тагове, но няма как да пише на него. Да се напише parser за езика IML (Ivan Markup Language). Езикът съдържа 11 тага, като за всеки от тях има съответен затварящ. Някои от таговете притежават допълнителни атрибути, които указват допълнителна информация за операцията, която трябва да се извърши.

Потребителят въвежда две имена на файлове. Първото име е за входен, а второто за изходен файл, като програмата трябва да интерпретира съдържанието на входния файл и да запише резултата в изходния. Ако във входния файл участват и тагове, които не са част от езика, да се изведе съобщение за грешка. Нужно е да се направи проверка за коректност на входа.

Иван иска да се позволи коректно влагане (или “вгнездяване”, nesting) на тагове. Ако влагането не е коректно, програмата следва да изведе съобщение за грешка на стандартния изход за грешки. Идеята на Иван е да може да извършва агрегиращи (Aggregate), трансформиращи (Map) и сортиращи (Sorting) операции върху списък от сравними данни (положителни и отрицателни дробни числа). Трансформираната стойност може да бъде както списък, така и единствена стойност (например резултат от агрегираща операция). Езикът IML има три категории тагове:

1) Map: *<MAP-INC "N"> и <MAP-MLT "N"> където N е дробно число*
   a) **<MAP-INC "1">**1 2 3**</MAP-INC>** ⇒ 2 3 4
   b) **<MAP-MLT "2">**1 2 3**</MAP-MLT>** ⇒ 2 4 6

2) Aggregate: *<AGG-SUM>, <AGG-PRO>, <AGG-AVG>, <AGG-FST>, <AGG-LST>*
   a) **<AGG-SUM>**1 2 3**</AGG-SUM>** ⇒ 6
   b) **<AGG-PRO>**1 2 3**</AGG-PRO>** ⇒ 6
   c) **<AGG-AVG>**1 2 3**</AGG-AVG>** ⇒ 2
   d) **<AGG-FST>**1 2 3**</AGG-FST>** ⇒ 1
   e) **<AGG-LST>**1 2 3**</AGG-LST>** ⇒ 3

3) Sorting: *<SRT-REV>, <SRT-ORD "ARG">, където ARG е ASC или DSC, <SRT-SLC "N">, където N е положително цяло число, <SRT-DST>*
   a) **<SRT-REV>**1 2 3**</SRT-REV>** ⇒ 3 2 1 (обръща списъка)
   b) **<SRT-ORD "ASC">**3 2 1**</SRT-ORD>** ⇒ 1 2 3 (сортира във възходящ ред (при аргумент “ASC”) или в низходящ ред (при аргумент “DSC”)
   c) **<SRT-SLC "1">**3 2 1**</SRT-SLC>** ⇒ 2 1 (връща подсписък от посочения индекс нататък)
   d) **<SRT-DST>**4 8 4 3**</SRT-DST>** ⇒ 4 8 3 (премахва дубликати)

Да се извеждат подходящи съобщения за грешки при некоректно подаден вход, включително: синтактична грешка, наличие на нечислови данни вътре в таговете, некоректно записани числа, недостатъчен брой данни за операциите (например AGG-AVG, SRT-SLC), липса на задължителен параметър на някоя операция.

Примери:

Първи пример:

```
<SRT-ORD "ASC">81 3<MAP-INC "1">4 12 55<AGG-AVG>4
8</AGG-AVG></MAP-INC>22</SRT-ORD>
```

Трансформира се до:
```
3 5 7 13 22 56 81
```

Втори пример:

```
<SRT-DST><SRT-SLC "3">57 18 9<MAP-INC "-3">4 2 2</MAP-INC>5</SRT-SLC></SRT-DST>
```

Трансформира се до:
```
1 -1 5
```

Бонус:

- Да се реализира таг <LET "NAME">...<BODY/>...</LET>, който позволява свързването на име NAME с резултата от обработката на данните преди тага </BODY> и след това използването на името след тага <BODY/>. Името NAME се състои само от главни латински букви.

  Пример:

  ```
  <LET "X"><MAP-INC "1">1 2 3</MAP-INC><BODY/><SRT-REV>1 X
  5</SRT-REV></LET>
  ```

  се трансформира до

5 4 3 2 1