## Робота №4

XML

The E**X**tensible **M**arkup **L**anguage (розширена мова розмітки або *XML*), один із стандартів для зберігання та передачі даних.

XML документ містить точно один XML елемент, котрий має ім’я і, може мати, атрибути і компоненти: текстові поля та інших «дочірні» XML елементи. Кожний XML елемент з іменем *name* має вид - *<name attr1 attr2 … attn> компоненти </name>* .

* *<name attr1 attr2 … attn>*- початковий тег,
* кожний атрибут *attri* має вид - *nameAtt = “value”*
* *тестове поле* – довільна послідовність символів крім ‘<’ і ‘>’
* *</name>* - заключний тег.

Ім’я XML елемента *name* і атрибута *nameAtt* – це рядок, що починається з букви і може містити довільну комбінацію букв, цифр, крапки (.) та тире (-). Ім’я не може містити проміжки.

Значення атрибуту *“value”*- це рядок в подвійних лапках, що не містить подвійних лапок.

Проміжки між елементами початкового тегу та компонентами атрибуту (*nameAtt, =, “value”*) – не мають ніякої ваги, тобто НЕ являються частиною даних.

Проміжки в *тестовому полі* являються частиною даних.

Далі Мал.1 наведено XML-документ, котрий виглядає так як він з’являється в текстовому файлі. При обробці такі документи мають формат Haskell-рядка String. Цей рядок містить явні ‘лапковані’ символи (ескейп-символи) такі як ‘\n’ та ‘\”’ (Див. рядок з іменем films в файлі-заготовка).

Цей документ має один елемент з іменем filmlist, що включає три «дочірні» XML елементи з даними, що пов’язані з фільмами. Кожний з цих елементів має назву film і один атрибут з іменем title і значенням - назва відповідного фільму. Елементи не мають ніякого додаткових умов відносно елементів, котрі вони містять, або порядку в якому вони повинні з’являтися. Єдина вимога - XML повинен бути синтаксично вірним.

<filmlist>

<film title = "Rear Window">

<director>Alfred Hitchcock</director>

<composer>Franz Waxman</composer>

<year>1954</year>

</film>

<film title = "2001: A Space Odyssey">

<director>Stanley Kubrick</director>

<composer>Richard Strauss</composer>

<composer>Gyorgy Ligeti</composer>

<composer>Johann Strauss</composer>

<year>1968</year>

</film>

<film title="Lawrence of Arabia" >

<duration>228</duration>

<director>David Lean</director>

<composer>Maurice Jarre</composer>

</film>

</filmlist>

*Мал.1: Список фільмів в XML документі.*

Типи даних в Haskell, що задають XML-документ і використовуються в роботі:

***type*** Name = String

***type*** Attributes = [(String, String)]

***data*** XML = Text String | Element Name Attributes [XML]

deriving (Eq, Show)

***type*** Stack = [XML]

Ще один приклад XML документу – інформація про фільм «Касабланка»:

* Вид як у текстовому файлі

<film title="Casablanca">

<director>Michael Curtiz</director>

<year>1942</year>

</film>

* Як Haskell-рядок String

"<film title=\"Casablanca\">\n <director>Michael Curtiz</director>\n \

\<year>1942</year>\n</film>\n\n\n"

* + Текстовий рядок в Haskell може займати декілька ліній, для зручності читання використовуються символи з ‘\’ , щоб відмітити кінець одної лінії і її продовження на наступній.
* Внутрішнє представлення – об’єкт типу XML

Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n ",

Element "year" [] [Text "1942"],

Text "\n"]

Головне завдання роботи – реалізувати синтаксичний аналіз рядка, що містить коректне тестове представлення XML-документу, і побудувати його внутрішнє представлення.

XML аналізатор читає XML документу, що заданий як рядок (String), і генерує внутрішнє представлення документу як об’єкт типу XML. В роботі припускається, що всі XML – документи синтаксично вірні в тому сенсі, що:

* Всі XML-елементи повністю вкладаються (гніздяться) і кожний початковий тег має відповідний заключний тег.
* Імена елементів та атрибутів вірно побудовані і не містять проміжків.
* Імена атрибутів та їх значення розділяються символом ‘=’ і значення атрибутів розташовані в парних подвійних лапках.

В процесі аналізу необхідно підтримувати стек (Stack) частково оброблених елементів: елемент заноситься в стек, коли зустрічається його початковий тег і вилучається зі стеку, коли зустрічається його заключний тег.

На початку аналізу стек містить один початковий елемент, Element “” [] [].

Пропустивши довільні проміжки до першого символу ‘<’, виконуються наступні правила аналізу (зверху вниз):

1. Якщо вхідний рядок порожній (“”) , то результат - це *перший син* елементу на вершині стеку.
2. В іншому випадку застосовуємо наступні правила *в порядку їх запису*:
   1. Якщо перший символ є ‘<’ і другий є ‘/’, то - початок заключного тегу.

**</ім’я-елементу> залишок тексту …**

*пропустити це рекурсія на цьому*

Необхідно

* + - обробити його до першого ‘>’ включно;
    - на вершині стеку повністю сформований елемент, котрий виділяється і додається до синів наступного верхнього елементу стеку;
    - обробити рекурсивно текст, що залишився.
  1. Якщо перший символ є ‘<’ (без наступного ‘/’), то - початку нового тегу.

**<ім’я-елементу атрибути > залишок тексту …**

*читати ім’я читати атрибути**рекурсія на цьому*

Необхідно

* + - прочитати ім’я елементу та всі атрибути, включаючи наступний ‘>’;
    - занести в стек новий частково сформований елемент з атрибутами;
    - обробити рекурсивно текст, що залишився.
  1. Перший символ – початок нового текстового поля.

**деякий текст символи <залишок тексту …**

*прочитати тест рекурсія на цьому*

Необхідно

* + - прочитати його до першого символу ‘<’ (не включаючи його)
    - додати тестове поле до синів елементу на вершині стеку
    - обробити рекурсивно текст, що залишився

Оскільки XML документ має правильну структуру, то *аналіз списку атрибутів* має вид:

* Пропустити всі проміжки і перевірити перший символ – не проміжок
* Якщо цей символ ‘>’, то кінець аналізу. Результат – пара: порожній список атрибутів і текст після ‘>’.
* Інакше є по крайній мірі один атрибут, котрий можна прочитати:
  + Прочитати *ім’я атрибуту* і, можливі, проміжки за ним
  + Прочитати пропускаючи символ ‘=’ і, можливі, проміжки за ним
  + Прочитати пропускаючи відкриваючий символ ‘”’
  + Прочитати *значення атрибуту* - весь текст до наступного символу ‘”
  + Прочитати пропускаючи закриваючий символ ‘”’
* *Ім’я атрибуту* і *значення атрибуту* утворюють пару, що є першим атрибутом в списку, що повертається.
* Інші атрибути отримуються рекурсивно на тексті, що залишився.

Аналіз XML-документу, що заданий в рядку *s*, може виконує функція *parse s*, що за допомогою функції *skipSpace* вилучає всі початкові проміжки до першого першого символу ‘<’, формує початкове значення стеку *[sentile]* і використовує допоміжну функцію *parse’.*

parse :: String -> XML

parse s = parse’ (skipSpace s) [sentinel]

parse’ :: String -> Stack -> XML

parse’ = undefined

Процес аналізу (*parse casablanca*) демонструє стан вхідного рядка та стеку, коли аналізується XML документ про фільм «Касабланка», котрий наведено раніше.

* Стек показано після застосування іменованого правила.
* Символи вхідного рядка, що обробляються виділені підкресленням.
* На початку стек містить тільки один елемент - Element “” [] [].

*Вхід* : "<film title=\"Casablanca\">\n <director>Michael Curtiz</director>..."

*Правило*: 2b

*Новий стек*: [Element "film"

[("title","Casablanca")]

[],

Element "" [] []]

*Вхід*: "\n <director>Michael Curtiz</director>\n <year>1942</year>..."

*Правило*: 2c

*Новий стек*: [Element "film"

[("title","Casablanca")]

[Text "\n "],

Element "" [] []]

*Вхід*: "<director>Michael Curtiz</director>\n <year>1942</year>..."

*Правило*: 2b

*Новий стек*: [Element "director" [] [],

Element "film"

[("title","Casablanca")]

[Text "\n "],

Element "" [] []]

*Вхід*: "Michael Curtiz</director>\n <year>1942</year>\n</film>\n\n\n""

*Правило*: 2c

*Новий стек*: [Element "director" [] [Text "Michael Curtiz"],

Element "film"

[("title","Casablanca")]

[Text "\n "],

Element "" [] []]

*Вхід*: "</director>\n <year>1942</year>\n</film>\n\n\n"

*Правило*: 2a

*Новий стек*: [Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"]],

Element "" [] []]

*Вхід*: "\n <year>1942</year>\n</film>\n\n\n"

*Правило*: 2c

*Новий стек*: [Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n "],

Element "" [] []]

*Вхід*: "<year>1942</year>\n</film>\n\n\n"

*Правило*: 2b

*Новий стек*: [Element "year" [] [],

Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n "],

Element "" [] []]

*Вхід*: "1942</year>\n</film>\n\n\n"

*Правило*: 2c

*Новий стек*: [Element "year" [] [Text "1942"],

Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n "],

Element "" [] []]

*Вхід*: "</year>\n</film>\n\n\n"

*Правило*: 2a

*Новий стек*: [Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n ",

Element "year" [] [Text "1942"]],

Element "" [] []]

*Вхід*: "\n</film>\n\n\n"

*Правило*: 2c

*Новий стек*: [Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n ",

Element "year" [] [Text "1942"],

Text "\n"],

Element "" [] []]

*Вхід*: "</film>\n\n\n"

*Правило*: 2a

*Новий стек*: [Element ""

[]

[Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n ",

Element "year" [] [Text "1942"],

Text "\n"]]]

*Вхід*: "\n\n\n"

*Правило*: 2c

*Новий стек*: [Element ""

[]

[Element "film"

[("title","Casablanca")]

[Text "\n ",

Element "director" [] [Text "Michael Curtiz"],

Text "\n ",

Element "year" [] [Text "1942"],

Text "\n"],

Text "\n\n\n"]]

В допоміжному файлі-заготовка, котрий включає визначення типів, допоміжних функцій і даних для тестування, надати визначення наступних функцій.

1. Функція *skipSpace s*, котра пропускає всі ведучі проміжки в рядку *s*. Можна використати функцію *isSpace*. Наприклад:
   * skipSpace "\n \n\nsome \n \n text” = "some \n \n text”
2. Функція *getAttribute s xml,* котра шукає значення атрибуту *s* в XML елементі *xml*. Якщо такого атрибуту не існує, то результат повинен бути “”. Наприклад:
   * getAttribute "x" x2 = “1”
   * getAttribute "x" (Text "t") = “”
3. Функція *getChildren s xml*, що повертає список синів XML елементe *xml*, котрі мають ім’я *s*. Наприклад:
   * getChildren "b" x2 = [Element "b" [] [Text "A"],Element "b" [] [Text "B"]]
   * getChildren "c" x2 = []
4. Функція *getChild s xml*, що повертає першого сина XML елементу *xml*, що має ім’я *s*. Якщо такого сина не існує, то результат повинен бути Text “”. Наприклад
   * getChild "b" x2 = Element "b" [] [Text "A"]
   * getChild "c" x2 = Text ""
5. Функція *addChild xml1 xml2*, котра додає нового сина *xml1* до існуючих синів XML елементу *xml2*. Нового сина потрібно додається в самий правий кінець існуючого списку. Передумова - другий аргумент *xml2* буде завжди побудований конструктором Element. Наприклад.
   * addChild (Text "B") (Element "a" [] [Text "A"]) =

Element "a" [] [Text "A",Text "B"]

1. Функція *getValue* *xml*, котра повертає значення елементу *xml*. Значення XML елементу - конкатенація всіх текстових полів в середині елементу, включаючи (рекурсивно) значення текстових полів його синів. Результуючий String потрібно «загорнути» в конструктор Text. Наприклад:
   * getValue x1 = Text "A"
   * getValue x2 = Text "AB"
2. Функція *addText s st*, котра додає текстове поле, чиє значення є рядок *s*, до дітей елементу на вершині стеку *st*. Передумова - є по крайній мірі один елемент Element в стеку *st*. Наприклад.
   * addText "B" [Element "a" [] [Text "A"], sentinel] =

[ Element "a" [] [Text "A",Text "B"], Element "" [] []]

1. Функція *popAndAdd* *st,* котра вилучає елемент на вершину стеку *st* і додає його до дітей елементу, що знаходився нижче в тому самому стеку. Передумова - є по крайній мірі два елемента Element в стеку *st.* Наприклад:
   * popAndAdd [Element "a" [] [Text "A"], Element "ab" [("a","1")] [], sentinel] = [Element "ab" [("a","1")] [Element "a" [] [Text "A"]], Element "" [] []]
2. Функція *parseAttributes s*, котра розпізнає на початку рядка *s* список атрибутів і повертає список атрибутів і залишок тексту. (Можна використати функцію *parseName s*, що визначається в допоміжному файлі-заготовка для аналізу імені атрибуту, вона повертає ім’я та текст, що слідує за ним, як пару String.) Наприклад:
   * parseAttributes "x=\"7\">rest of text" = ([("x","7")],"rest of text")
   * parseAttributes "a = \"0\" b = \"1\" >rest of text" =

([("a","0"),("b","1")],"rest of text")

1. Функція *parse’ s st*, котру використовує функція *parse s,* що виконує синтаксичний аналіз XML документу в рядку *s,* згідно приведеного алгоритму. (Можна використати функцію *parseName s* для аналізу імені елементу.) Якщо функція *parse’* побудована вірно, то:
   * parse s1 = Element "a" [] [Text "A"]
   * parse films = filmsParsed

skipSpace :: String -> String

getAttribute :: String -> XML -> String

getChildren :: String -> XML -> [XML]

getChild :: String -> XML -> XML

addChild :: XML -> XML -> XML

getValue :: XML ->XML

addText :: String -> Stack -> Stack

popAndAdd :: Stack -> Stack

parseAttributes :: String -> (Attributes, String)

parse` :: String -> Stack -> XML

Зауваження:

Назва файлу Family04.hs (Family – прізвище студента). Файл включає модуль Family04 і створюється на основі файла-заготовки HWP04.hs