Mikołaj Markiewicz Warszawa, 17.04.2013r

Projekt wstępny projektu TKOM

I. Temat

XII. Zad HTML2

Napisać program scalający dwa pliki HTML, plik pierwotny oraz plik zmodyfikowany. Jeśli lokalna modyfikacja była wprowadzona w jednym z plików to zostaje wprowadzona do pliku docelowego. Przypadki konfliktowe rozstrzyga użytkownik.

II. Opis

Program ma za zadanie wczytać podane na wejście dwa pliki w formacie HTML oraz dokonać ich scalenia wg pewnej opisanej poniżej tabeli z opisanymi przypadkami konfliktowymi. W celu wykonania takiego scalenia dokonywana jest analiza leksykalna i rozbiór składniowy obydwu plików poprzez wyróżnienie tokenów oraz zbudowania na ich podstawie drzew rozbioru zadanych plików. Pliki traktowane są w podobnej skali ważności, z lekką przewagą dla pliku podanego w kolejności jako drugiego, podczas rozstrzygania połączeń.

III. Wymagania

A. Funkcjonalne

1. Pliki podane do programu dzielimy na plik ***pierwotny*** i plik ***zmodyfikowany*** podane do programu w takiej kolejności
2. Plik docelowy stworzony z podanych plików wg tabeli priorytetów konfliktów z przewagą dla pliku zmodyfikowanego
3. W plikach HTML dostępne są wszystkie znaczniki zdefiniowane w standardzie i ich atrybuty
4. Plik docelowy generowany jest składniowo poprawny - bezbłędna walidacja w http://html5.validator.nu lub http://validator.w3.org

B. Niefunkcjonalne

1. Dostarczone pliki HTML poprawnie skonstruowane składniowo. W przypadku błędu, komunikat i koniec pracy programu
2. Aplikacja udostępnia prosty, intuicyjny interfejs użytkownika
3. Prezentacja konfliktów w sposób łatwy do rozstrzygnięcia dla użytkownika - okno dialogowe

IV. Działanie programu

1. Wyodrębnienie tokenów z podanych plików wejściowych do list tokenów (2 przebiegi) (V.A.2)
2. Tworzenie drzew rozbioru podanych plików na podstawie list wyodrębnionych tokenów (2 przebiegi) (V.B.2)
3. Budowanie drzewa docelowego pliku na podstawie utworzonych drzew (1 przebieg)
4. Generacja kodu wynikowego na podstawie zbudowanego drzewa wynikowego (1 przebieg)

V. Projekt realizacji

A. Analiza leksykalna

Pierwszym zadaniem programu jest analiza leksykalna tekstu podanych na wejściu plików HTML. Tekst jest przeglądany znak po znaku w celu wyróżnienia w nim odpowiednich tokenów.

1. Tokeny

a) Struktury tokenów

```
Token {
      value      - wartość tokena
}
doctypeToken          : Token {}
startOpenTagToken     : Token {}
endOpenTagToken       : Token {}
closeTagToken         : Token {}
closeEmptyTagToken    : Token {}
attributeKeyToken     : Token {}
attributeEqualsToken  : Token {}
attributeValueToken   : Token {}
textToken             : Token {}
```

b) Rozróżniane tokeny:

Tekst	Token	Wartość
`<!doctype (.*?)>`	`doctypeToken`	`\1`
`<tag`	`startOpenTagToken`	`tag`
`</tag`	`endOpenTagToken`	`tag`
`>`	`closeTagToken`	`>`
`/>`	`closeEmptyTagToken`	`/>`
`key`	`attributeKeyToken`	`key`
`=`	`attributeEqualsToken`	`=`
`value | 'value' | "value"`	`attributeValueToken`	`value`
`[^<](.*?)<`	`textToken`	`\1`

2. Działanie

Analiza zadanego tekstu z pliku HTML. W jej trakcie budowana jest lista tokenów.

1. Jednorazowe wykonanie funkcji ***a*** w celu znalezienia typu dokumentu
2. Wykonywanie w pętli funkcji ***b*** w celu znalezienia znaczników
3. W przypadku znalezienia znacznika jednorazowe wykonanie funkcji ***c*** dla każdego znacznika otwierającego

3. Funkcje

a) Znalezienie opisu dokumentu - wywoływana jednorazowo

1. Znajdź *'<!doctype'* na początku pliku.
2. Jeśli brak lub nie na początku
   2.1. Zakończ działanie wyświetlając komunikat o błędzie

3. Znajdź znacznik kończący '>'
4. Wyodrębnij typ dokumentu i zapisz do listy tokenów -> `doctypeToken`

b) Szukanie elementu - wywoływana w pętli

1. Znajdź najbliższy początek znacznika '<'
2. Jeżeli występuje jakiś tekst do pozycji znalezionego znacznika
   2.1. Zapisz jako token -> `textToken`
3. Jeżeli brak znaku '<'
   3.1. Zakończ działanie
4. Znajdź najbliższy koniec znacznika '>'
5. Analizuj wnętrze znacznika pomiędzy znakami '<' i '>' -> wywołanie funkcji ***c***
6. Jeżeli znaleziony element jest pusty *'... />' - {single_tag}*
   6.1. Wróć do pkt. 1
7. Znajdź najbliższy początek znacznika końcowego '</'
8. Znajdź najbliższy koniec znacznika końcowego '>'
9. Jeżeli element nie jest pusty: *'<tag></tag>'*
   9.1. Szukaj elementu pomiędzy znalezionym otwierającym a kończonym tagiem
10. Zapisz znalezione elementy jako tokeny -> `endOpenTagToken` i `closeTagToken`

c) Analiza znacznika pomiędzy znakami '<' i '>' - wywoływana przez b

1. Znajdź pierwszy biały znak
2. Jeżeli brak
   2.1. Znajdź końcowy znak znacznika '>'
3. Potraktuj tekst od początku do znalezionego znaku jako nazwę znacznika i zapisz jako token -> `startOpenTagToken`
4. Pomiń kolejne białe znaki
5. Jeżeli znaleziono koniec znacznika '>' lub '/>'
   5.1. Zakończ analizę znacznika i zapisz jako token -> `closeTagToken` lub `closeEmptyTagToken`
6. Znajdź najbliższy znak '='
7. Tekst od ostatniego białego znaku do znaku '=' potraktuj jako nazwę atrybutu i zapisz jako token -> `attributeKeyToken` i `attributeEqualsToken`
   7.1. Jeżeli brak znaku " lub '
      7.1.1. Znajdź najbliższy biały znak
      7.1.2. Jeżeli brak to znajdź koniec znacznika '>' lub '/>'
         7.1.2.1. Zapisz tekst od znaku '=' do znalezionego znaku jako token -> `attributeValueToken`
   7.2. Jeżeli znaleziono " lub '
      7.2.1. Znajdź kończący znak " lub '
      7.2.2. Zapisz tekst pomiędzy znalezionymi znakami jako token -> `attributeValueToken`
8. Wróć do pkt. 3

VI. Opis funkcjonalności

Tabela konfliktów przeznaczona do dokonania wyboru elementów w trakcie scalania przetworzonych plików. Przedstawia przykładowe możliwie ogólne przykłady fragmentów tekstu (elementów drzewa) z pliku pierwszego (pierwotnego) i drugiego (zmodyfikowanego) oraz wynik scalenia danych fragmentów. W przypadku wystąpienia niejednoznaczności opisane zostały możliwe rozwiązania konfliktów przez użytkownika.

A. Konflikty

1. Legenda

```
Text            - ciąg znaków nie zawierający znaczników
Text1 ≠ Text
<tag>           - jakiś znacznik
<tagX> ≠ <tag> ≠ <tagY>, X ≠ Y
...             - tekst lub znaczniki
...1 ≠ ...2
```

2. Tabela

Lp.	Plik pierwotny	Plik zmodyfikowany	Plik docelowy
1	Znacznik typu dokumentu wybierany z przewagą dla drugiego pliku		
	`<!doctype Text>`	`<!doctype Text1>`	`<!doctype Text1>`
2	W pierwszym pliku brak atrybutu/atrybutów, w drugim występują. Atrybuty są sumowane.		
	`<tag>`	`<tag a="b" ..>`	`<tag a="b" ..>`
3	W pierwszym pliku występują atrybuty, w drugim brak. Atrybuty są sumowane.		
	`<tag a="b" ..>`	`<tag>`	`<tag a="b" ..>`
4	W obydwu plikach występują różne atrybuty. Atrybuty są sumowane.		
	`<tag a="b" ..>`	`<tag c="d" ..>`	`<tag a="b" c="d" ..>`
5	W obydwu plikach występują atrybuty. Jeżeli nazwy atrybutów się powtarzają to wybierane są z pliku drugiego.		
	`<tag a="b" ..>`	`<tag a="c" ..>`	`<tag a="c" ..>`
6	Jeżeli tagi są takie same, algorytm scalania (VI.C.b) przepisuje znacznik i ewentualnie weryfikuje atrybuty wg tabeli oraz następnie sprawdza kolejno wnętrze tegoż znacznika		

	`<tag ..1>...1</tag>`	`<tag ..2>...2</tag>`	`<tag ..3>`
			`  // Wewnętrzna analiza`
			`</tag>`

7	Jeżeli tagi są takie same i zawierają tylko tekst. Teksty są różne, to użytkownik ma do wyboru możliwe rozwiązania przedstawione poniżej.		

	`<tag>Text</tag>`	`<tag>Text1</tag>`	**`Konflikt, wybór:`**
			`1. <tag>Text</tag>`
			`2. <tag>Text1</tag>`
			`// Konkatenacja tekstu:`
			`3. <tag>TextText1</tag>`
			`4. <tag>Text1Text</tag>`

8	Jeżeli tagi są różne, ale zawierają tylko tekst, który jest równy to wynikiem jest suma tagów z zadanym tekstem.		

	`<tag>Text</tag>`	`<tag1>Text</tag1>`	`<tag>`
			`  <tag1>Text</tag1>`
			`</tag>`

9	Jeżeli porównywany jest sam tekst z nim samym ale otagowanym pojedyńczym znacznikiem to wynikiem jest tenże otagowany tekst.		

	Text	<tag>Text</tag>	<tag>Text</tag>
10	Jeżeli porównywany jest tag zawierający jakieś elementy z elementem tekstowym, to wynikiem jest konkatenacja tagu i elementu tekstowego		
	<tag>...</tag>	Text	<tag>...</tag> Text
11	Jeżeli porównywane jest wnętrze znacznika i w jednym z plików nie ma znaczników a w drugim są lub w jednym jest więcej elementów niż w drugim to znaczniki są sumowane		
	<tag></tag> -------------------------np.- <tag> <tag1></tag1> </tag>	<tag> ... </tag> -------------------------np.- <tag> <tag1></tag1> ... </tag>	<tag> ... </tag> -------------------------np.- <tag> <tag1></tag1> ... </tag>
12	Jeżeli porównywane elementy to różne tagi które zawierają jakieś elementy, to użytkownik ma do wyboru możliwe rozwiązania przedstawione poniżej:		
	<tag> ...1 </tag>	<tag1> ...2 </tag1>	**Konflikt, wybór:** 1. <tag>...1</tag> <tag1>...2</tag1> 2. <tag>...1</tag> 3. <tag1>...2</tag> 4. Wybór *tag1* z pliku drugiego i rozstrzyganie konfliktu pomiędzy *tag* z pliku pierwszego i wnętrzem *tag1* pliku drugiego
12.1	Przykład do pkt. 11: Do pliku wynikowego dodawany jest znacznik otaczający *tag1* i następnym elementem do rozstrzygnięcia jest ten sam znacznik pliku pierwszego *tag* i wnętrze znacznika pliku drugiego *tag1 -> tag2* …3		
	<tag> ...1 </tag>	<tag1> <tag2> ...2 </tag2> ...3 </tag1>	**Cd. opis:** Dodaj <tag1> do pliku docelowego i rozstrzygaj konflikt pomiędzy: <tag>...1</tag> i wnętrzem <tag1> <tag2>...2</tag2>

B. Analiza składniowa

W trakcie analizy składniowej dokonywany jest rozbiór składniowy, budowane jest drzewo rozbioru dla danych plików wejściowych.

1. Budowa drzewa rozbioru

Budowa drzewa na podstawie listy wyróżnionych tokenów wg następującej gramatyki:

a) Znaczenie

<...> - element gramatyki "..." - element terminalny {...} - zdefiniowane symbole

b) Składnia

```
<root>            ::= <doctype> <rootElement>
<doctype>         ::= "<!doctype" <whitespaces> <text> ">"
<rootElement>     ::= <openTag> <elements> <closeTag>
<elements>        ::= <element> | <element> <elements>
<element>         ::= <openTag> <content> <closeTag> | <emptyElement> | <textElement>
<emptyElement>    ::= <openCloseTag>
<textElement>     ::= <text>
<openTag>         ::= "<" {tag} <optWhitespaces> ">" |
                      "<" {tag} <whitespaces> <attributes> <optWhitespaces> ">"
<closeTag>        ::= "</" {tag} <optWhitespaces> ">"
<openCloseTag>    ::= "<" {single_tag} <optWhitespaces> <openCloseEndTag> |
                      "<" {single_tag} <whitespaces> <attributes> <optWhitespaces>
                      <openCloseEndTag>
<openCloseEntTag> ::= "/>" | ">"
<content>         ::= <elements> | <optWhitespaces>
<text>            ::= <character> | <character> <text>
<character>       ::= {char} | <whitespace>
<attributes>      ::= <attribute> | <attribute> <whitespaces> <attributes>
<attribute>       ::= <key> "=" <value>
<key>             ::= <alphas>
<value>           ::= <word> | '"' <val> '"' | "'" <val> "'"
<val>             ::= <text> | <optWhitespace>
<word>            ::= {char} | {char} <word>
<alphas>          ::= <alpha> | <alpha> <alphas>
<alpha>           ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                      | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" |
                      "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" |
                      "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "-"
<whitespaces>     ::= <whitespace> | <whitespace> <whitespaces>
<whitespace>      ::= " " | "\t" | "\n" | "\r"
<optWhitespaces>  ::= <optWhitespace> | <optWhitespace> <optWhitespaces>
<optWhitespace>   ::= "" | <whitespace>
```

c) Tagi:

```
{char}       - znaki alfanumeryczne i znaki specjalne z wyjątkiem '<'
{tag}        - a, abbr, acronym, address, applet, article, aside,
               audio, b, bdi, bdo, big, blockquote, body, button, canvas, caption, center, cite,
               code, colgroup, command, datalist, dd, del, details, dfn, dialog, dir, div, dl,
               dt, em, fieldset, figcaption, figure, font, footer, form, frame, frameset, head,
               header, hgroup, h1, h2, h3, h4, h5, h6, html, i, iframe, ins, kbd, label, legend,
               li, map, mark, menu, meter, nav, noframes, noscript, object, ol, optgroup,
               option, output, p, pre, progress, q, rp, rt, ruby, s, samp, script, section,
               select, small, span, strike, strong, style, sub, summary, sup, table, tbody, td,
               textarea, tfoot, th, thead, time, title, tr, tt, u, ul, var, video
{single_tag} - area, base, basefont, br, col, embed, hr, img, input, keygen, link, meta, param,
               script, source, track, wbr, !--
```

2. Parser

Po wyodrębnieniu listy tokenów następuje *parsowanie* tejże listy w celu zbudowania struktury drzewa pasującej do zadanej gramatyki. Tworzony jest obiekt przechowujący tę strukturę i w pętli uruchamiany jest algorytm parsera. Tokeny czytane są po kolei tworząc odpowiednie elementy wynikowego obiektu.

a) Algorytm parsera

1. Wczytaj token
2. Jeżeli `doctypeToken`
   2.1. Wpisz do struktury drzewa
3. Jeżeli brak
   3.1. Zgłoś błąd braku typu dokumentu
4. Wczytaj kolejny token
5. Jeżeli `startOpenTagToken`
   5.1. Utwórz element w aktualnym węźle drzewa
   5.2. Wczytaj kolejny token
   5.3. Jeżeli `closeEmptyTagToken`
      5.3.1. Wróć do pkt. 5
   5.4. Jeżeli `closeTagToken`
      5.4.1. Przejdź do utworzonego elementu i wróć do pkt. 5
   5.5. Jeżeli `attributeKeyToken`
      5.5.1. Dodaj atrybut identyfikowany danym kluczem do węzła
      5.5.2. Wczytaj token
      5.5.3. Jeżeli nie `attributeEqualsToken`
         5.5.3.1. Zgłoś błąd i zakończ
      5.5.4. Wczytaj token
      5.5.5. Jeżeli nie `attributeValueToken`
         5.5.5.1. Zgłoś błąd i zakończ
      5.5.6. Zapisz wartość atrybutu
      5.5.7. Wróć do pkt. 6.2
6. Jeżeli `endOpenTagToken`
   6.1. Wejdź poziomy wyżej w drzewie
   6.2. Wróć do pkt. 5
7. Jeżeli `textToken`
   7.1. Utwórz element tekstowy w drzewie
   7.2. Wróć do pkt. 5
8. Jeżeli brak
   8.1. Zakończ budowę drzewa

C. Tworzenie pliku docelowego

Plik docelowy powstaje z połączenia dwóch plików podanych na wejście programu. Łączenie plików następuje wg tabeli konfliktów poprzez sprawdzanie ich zawartości w zbudowanych drzewach rozbioru poszczególnych plików. W wyniku łączenia powstaje nowe drzewo pliku docelowego.

a) Struktury tworzące drzewo

```
Element {
      Tag         - tekst określający aktualny element
      isEmpty     - flaga czy jest to znacznik bez zawartości
}

TagElement : Element {
      Attributes - mapa atrybut-wartość atrybutów znacznika
      Elements    - lista elementów pomiędzy znacznikiem otwierającym a
                  kończącym
}

TextElement : Element {
      Tag         = '#text'
      isEmpty     = true
      Content     - zawartość tekstowa znacznika = plain text pomiędzy
                  znacznikami otwierającym a kończącym
}

CommentElement : TextElement {
      Tag         = '#comment'
}
```

b) Budowa docelowego drzewa

1. Przechodź równolegle po kolejnych elementach (w głąb) obydwu zbudowanych drzew rozbioru na podstawie podanych na wejściu plików HTML
   1.1. Jeżeli elementy są takie same
      1.1.1. Utwórz taki element w drzewie docelowym
   1.2. Jeżeli znaczniki są takie same a atrybuty nie
      1.2.1. Rozwiąż konflikt atrybutów wg tabeli konfliktów
   1.3. Jeżeli znaczniki są różne
      1.3.1. Rozwiąż konflikt wg tabeli konfliktów
      1.3.2. Jeżeli konflikt wymaga decyzji użytkownika
         1.3.2.1. Wyświetl okno do decyzji jak rozwiązać konflikt
2. Analizuj wewnętrzne elementy w rozważanym elemencie wg algorytmu pkt. 1
3. Jeżeli brak elementów wewnętrznych
   3.1. Przejdź do następnego elementu i wykonuj algorytm wg pkt. 1
   3.2. Jeżeli brak następnego elementu to koniec

c) Generacja kodu wynikowego

Na podstawie zbudowanego docelowego drzewa przeglądając rekurencyjnie jego elementy generowany jest kod wynikowy - plik docelowy. Znaczniki są wypisywane do pliku z odpowiednimi atrybutami zawartymi w strukturze drzewa.

VII. Przykłady testowe

Przykładami testowymi będą proste lub bardziej skomplikowane kody źródłowe stron internetowych oraz ich odpowiedniki z wprowadzonymi lokalnymi modyfikacjami. Oba pliki powinny przejść walidację przez jeden z dostępnych w internecie walidatorów takich jak np. http://html5.validator.nu lub http://validator.w3.org

A. Przykładowe pliki

Zielone kwadraty oznaczają konflikty w pliku pierwotnym, zmodyfikowanym i wynikowym. Litera oznacza danym przypadek konfliktowy w kolejnych plikach, a liczba oznacza numer (lp.) z tabeli konfliktów danego konfliktu.

1. Plik pierwotny

```
<!doctype html>
<html>
  <head>
  </head>
<body>
  <section cl      av>
    <article>
      <b>Ipsum</b>
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      <br /><br />
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      <hr />
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    </article>

    <article>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    </article>
  </section>

  <section class=cont>
    <article>
      <h2>Description</h2>
    </article>

    <article>
      <h2>Photos</h2>
      <span>(Click to do)</span>
      <ul class=phot>
        <li>
          <a href="#" target="_blank">
            <img src="logo.png" alt="photo">
          </a>
        </li>
      </ul>
    </article>
  </section>

</body>
</html>
```

2. Plik zmodyfikowany

```
<!doctype html>
<html>
  <head>
    <title>Title</title>
  </head>
<body bgcolor="#0F0">
  <section class=nav>
    <article>
      <u><b>Ipsum</b></u>
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
    </article>
```

```
    <article style="color: #00F;">E2
      consectetur adipiscing elit. Lorem ipsum dolor sit amet.F7
    </article>
  </section>

  <section class=cont>
    <article>
      <h3>Description</h3> G8
    </article>

    <article>
      <h2>sotohP</h2>H7
      (Click to do) I9
      <ul class=phot>
        <li>
          <a href="#" target="_self">J5
            <img src="logo.png" alt="photo updated"> K5
          </a>
        </li>
        <li>Txt</li>
      </ul>
    </article>
  </section>

  <div>
    123.345
    <i>.00</i>
  </div>

</body>
</html>
```

L11

M11

3. Plik wynikowy

```
<!doctype html>
<html>
  <head>
    <title>Title</title>  A1
  </head>
<body bgcolor="#0F0"> B2
  <section class=nav>
    <article>
      <u><b>Ipsum</b></u> C8
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
       - Lorem ipsum dolor sit amet.
      <br>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      <br /><br />
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      <hr />
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    </article>

    <article style="color: #00F;">E2
      consectetur adipiscing elit. Lorem ipsum dolor sit amet. F7
    </article>
  </section>

  <section class=cont>
    <article>
      <h2><h3>Description</h3></h2> G8
    </article>

    <article>
      <h2>sotohP</h2> H7
      <span>(Click to do)</span> I9
      <ul class=phot>
        <li>
          <a href="#" target="_self">J5
            <img src="logo.png" alt="photo updated"> K5
          </a>
        </li>
        <li>Txt</li>
      </ul>
```

D11

L11

```
    </article>
  </section>

  <div>
    123.345
    <i>.00</i>
  </div>

</body>
</html>
```

M11