Projekt 2: skanery i parsery

1. Wstęp.

Celem niniejszego projektu jest napisanie prostego skanera i parsera do plików HTML, który wymusza pewne proste zasady gramatyczne, na przykład tag $< li >$ dla elementów listy może pojawić się tylko tam, gdzie występuje lista, podobnie tagi $< b >$ oraz $< /b >$ muszą być odpowiednio umiejscowione w pliku. Tak jak przy pierwszym zadaniu, naszym głównym celem jest oswojenie się z *flex*'em i z *bison*'em, nie będziemy próbować ogarnąć wszystkich subtelności języka HTML w taki sposób, w jaki powinno byłoby to być zrobione w komercyjnym programie. Zasady gramatyczne, jakich będzie musiał przestrzegać nasz parser są następujące:

Dokumentację do *lex*'a/*flex*'a znajdziecie tutaj:
`http://www.kompilatory.agh.edu.pl/pages/tk-laboratorium/flex.html`
a do *bison*'a tutaj:
`http://www.kompilatory.agh.edu.pl/pages/tk-laboratorium/bison.html`

1.1. **Opis działania programu.** Wasz program powinien pobierać dane z *stdin,* sprawdzać, że przestrzegane są odpowiednie zasady gramatyki, wyrzucać z nich wszystkie tagi HTML (patrz poprzednie zadanie) i zapisywać wynik do *stdout*. Komunikaty o błędach (patrz niżej) powinny być zapisane w pliku *stderr.*

Zasady gramatyki, których musimy przestrzegać, są następujące:

Reguły leksykalne:

Tagiem nazywamy dowolny ciąg znaków postaci $< S >$, gdzie S jest ciągiem drukowalnych znaków, który nie zaczyna się od "białychj spacji" i nie zawiera znaku $>$.

Znaki drukowalne rozumiemy w sensie, w jakim specyfikuje je funkcja *isprint()* w C, a "białe spacje" w sensie, w jakim specyfikuje je funkcja *isspace()*. Odpowiadają one klasom znaków *flex*'a definiowanym przez *[:print:]* oraz *[:blank:]*, odpowiednio.

Nasza gramatyka powinna rozpoznawać następujące tagi:

DOC_START	:	$< html >$
DOC_END	:	$< /html >$
HEAD_START	:	$< head >$
HEAD_END	:	$< /head >$
BODY_START	:	$< body >$
BODY_END	:	$< /body >$
BF_START	:	$< b >$
BF_END	:	$< /b >$
IT_START	:	$< i >$
IT_END	:	$< /i >$
UL_START	:	$< ul >$
UL_END	:	$< /ul >$
OL_START	:	$< ol >$
OL_END	:	$< /ol >$
LI_START	:	$< li >$
LI_END	:	$< /li >$

Dodatkowo, token TAG odpowiadać będzie każdemu tagowi, który nie został wypisany na powyższej liście, token TEXT będzie odpowiadał każdemu (pojedyńczemu) znakowi, który nie jest częścią tagu lub komentarza, a token SPACE będzie odpowiadał każdemu niepustemu ciągowi "białych spacji".

Reguły syntaktyczne:

Reguły syntaktyczne zbudowane są z tokenów i nieterminali. Token oznacza jeden lub więcej ciągów oznaczonych przez skaner (na przykład "identyfikator", "stała liczbowa" itp.). Nieterminal oznacza zbiór ciągów o podobnej strukturze syntaktycznej (na przykład "deklaracja", "pętla" itp.). W zasadach wypisanych poniżej tokeny wpisane są fontem `teletype,` a nieterminale fontem *pochylonym.* Symbol $\emptyset$ oznacza ciąg pusty.

Reguła syntaktyczna składa się ze strony lewej i prawej oddzielonych od siebie dwukropkiem. Lewa strona jest nieterminalem o strukturze zdefiniowanej przez regułę. Prawa strona składa się ze zbioru alternatyw oddzielonych symbolem |. Każda alternatywa to ciąg (być może pusty) tokenów i nieterminali.

Doc	:	*Wspace* `DOC_START` *Wspace Head Wspace Body Wspace* `DOC_END` *Wspace*
Head	:	`HEAD_START` *Html* `HEAD_END`
Body	:	`BODY_START` *Html* `BODY_END`
Wspace	:	`SPACE`
	|	$\emptyset$
Html	:	*Item Html*
	|	$\emptyset$
Item	:	`BF_START` *Html* `BF_END`
	|	`IT_START` *Html* `IT_END`
	|	*List*
	|	*Other*
List	:	`UL_START` *Wspace ItemList Wspace* `UL_END`
	|	`OL_START` *Wspace ItemList Wspace* `OL_END`
ItemList	:	*ItemList Wspace OneItem*
	|	*OneItem*
OneItem	:	`LI_START` *Html* `LI_END`
Other	:	`TAG`
	|	`TEXT`
	|	`SPACE`

Symbol startowy dla naszej gramatyki to *Doc.*

1.2. **Błędy syntaktyczne.** Program powinien sobie radzić z błędami w sensowny sposób i zapisywać komunikaty o błędach do pliku *stderr*. Informacja o błędzie powinna być specyficzna i zawierać numer linii tak, aby użytkownik mógł łatwo dotrzeć do błędu. Program nie musi potrafić wychodzić z błędów syntaktycznych – wystarczy, jeżeli się zatrzyma po wykryciu błędu, choć oczywiście zaimplementowanie wychodzenia z błędów syntaktycznych będzie mile widziane.

2. Wywoływanie programu.

Plik wykonywalny powinien się nazywać *myhtml2txt* i powinien czytać z pliku *stdin* i zapisywać do pliku *stdout*. Innymi słowy, "tłumaczenie" pliku *foo.html* do *foo.txt* powinno być wywołane poleceniem:

myhtml2txt < *foo.html* > *foo.txt*