

Zadanie domowe 2

Podsumowanie:

Ściągnijcie plik

www.math.us.edu.pl/~pgladki/teaching/2012-2013/tk2_lab2.zip

Uruchomcie **sml** i skompilujcie kod przy użyciu **CM.make "sources.cm"**;

To zadanie składa się z dwóch części. Są w zasadzie całkowicie niezależne jedna od drugiej, możecie więc pracować nad nimi w dowolnej kolejności.

Część A: abstrakcyjny syntaks:

Przetłumaczcie (ręcznie) krótki program w Fun na konstruktory abstrakcyjnego syntaksu.

- (1) Przeczytajcie i zrozumcie następujące pliki: `absyn.sml` oraz `test.sml`. *Zwróćcie uwagę, że w pliku `test.sml` jest w sumie pięć programów w Fun przetłumaczonych ręcznie na konstruktory abstrakcyjnego syntaksu.*
- (2) Po wykonaniu `CM.make` uruchomcie `Test.run()`; z poziomu `sml-a`. To polecenie szczegółowo opisze i oceni wszystkie pięć programów.
- (3) Przeczytajcie pliki `heap.sig`, `heap.sml`, `eval.sig`, `eval.sml`, `funpp.sig`. Póki co nie musicie rozumieć w stu procentach, o co w nich chodzi.
- (4) Na podobnej zasadzie przejrzyjcie pliki `table.sig`, `table.sml` i `funpp.sml`
- (5) **Dokończcie `myfun.sml` tłumacząc dany program na konstruktory abstrakcyjnego syntaksu.** m [aghtk2]. Termin oddania zadania domowego mija w **piątek, 5 kwietnia**.

Część B: analizator leksykalny:

Zbudujcie lexera dla języka Fun. Aby to zrobić, musicie dowiedzieć się jakich tokenów będziecie potrzebowali do lexa. W tym celu musicie zapoznać się z definicją Funa:

www.math.us.edu.pl/~pgladki/teaching/2012-2013/tk2_fun.html

Będziecie budować własnego lexera przy użyciu ML-LEXa. Dokumentację online znajdziecie tutaj:

<http://www.smlnj.org/doc/ML-Lex/manual.html>

Możecie też znaleźć trochę pożytecznych informacji w podręczniku Appela, w rozdziale 2.

- (1) Przeczytajcie i zrozumcie następujące pliki: `sources.cm`, `errmsg.sml`, `tokens.sig`, `tokens.sml`, `fun.lex`, `runlex.sml`. Tylko te pliki mają coś wspólnego z tą częścią zadania.
- (2) Edytujcie kod w **`fun.lex`**. Będziecie musieli usunąć część przykładowego kodu i dodać sporo własnego.
 - Tokeny są zadeklarowane w `tokens.sig`. Oto lista symboli, jakie pojawiają się w źródle razem z tokenami, z którymi powinny być związane:

<code>- ></code>	ARROW	<code>!</code>	BANG
<code>:=</code>	ASSIGN	<code>)</code>	RPAREN
<code>(</code>	LPAREN	<code> </code>	OR
<code>&</code>	AND	<code>=</code>	EQ
<code>></code>	GT	<code><</code>	LT
<code>*</code>	TIMES	<code>-</code>	MINUS
<code>+</code>	PLUS	<code>;</code>	SEMICOLON
<code>,</code>	COMMA	<code>:</code>	COLON
<code># i</code>	PROJ		

gdzie `i` jest nieujemną liczbą całkowitą nie zaczynającą się od ciągu zer.

Słowa kluczowe Funa powinny być reprezentowane tokenami o tej samej nazwie. Token end-of-file powinien być reprezentowany przez EOF.

Każdy token potrzebuje dwóch liczb: numeru linii i numeru kolumny z początkiem danego tokena. Będą one używane do zgłaszania błędów. W poniższym przykładzie `x` jest w 2 linii i 7 kolumnie. Zauważmy, że linie i kolumny zaczynamy liczyć od 1 a nie od 0:

```
if true then
let x = ...
```

- Użycie `ErrorMsg.error: ErrorMsg.pos2 - > ciąg - > jednostka` to zgłaszania błędów. Potrzebne będą dwa argumenty: pary opisujące lokalizację w pliku (początek i koniec błędnego tekstu, liczony w znakach od początku pliku) oraz wiadomość o błędzie do wydrukowania. Powinniście zadbać o to, aby moduł `ErrorMsg` był właściwie informowany o tym, gdzie pojawiają się nowe linie (przez wywołanie `newLine`) tak, aby odpowiadające im lokalizacje zostały właściwie przetłumaczone na numery linii.

Funkcja `make_pos` jest wygodnym sposobem na konwertowanie rzeczy znanych `ML-Lexowi` (`yypos` oraz `yytext`) na lokalizacje w pliku odnoszące się do początku i końca tokena.

- Uważajcie na syntaks w plikach `lexa`. Pamiętajcie, że każda definicja `lexa` musi kończyć się średnikiem, tak samo jak każda reguła.
- **type** jest odwróconym słowem kluczowym jakiego możemy użyć do późniejszych rozszerzeń języka. Póki co najprościej będzie się z nim obejść wymuszając, aby programy go nie zawierały. Jako że póki co nie ma tokena `type` w dostarczonych plikach, nie można tego zrobić w parserze, możecie więc zechcieć użyć w tym celu `lexera`. Zwróćcie uwagę, że mimo to musicie zlexować ciągi takie jak **typexx** jako identyfikatory.
- Aby wypróbować Wasz kod, uruchomcie **RunLex.runlex "test.fun"**; W rezultacie otrzymacie ciąg tokenów wraz z numerami linii i kolumn dla każdego tokena. Jak zwykle, jeden test to za mało, żeby przetestować program. Powinniście sami napisać kilka programów do testowania.

- (3) **Zagnieżdżone komentarze.** Zadanie byłoby o wiele prostsze, gdyby `Fun` nie miał zagnieżdżonych komentarzy :) Sugeruję, abyście w pierwszym podejściu opanowali wszystko poza zagnieżdżonymi komentarzami, a następnie zajęli się nimi na sam koniec.

Tym razem to wszystko. Prześlijcie mailem pliki `myfun.sml`, `fun.lex` oraz `readme`, w którym w szczególności powinniście wyjaśnić wszelkie decyzje dotyczące wykonania zadania, jakie podjęliście. Termin zadania mija w **piątek, 19 kwietnia**. Proszę pamiętać o oznaczeniu maila tagiem `[aghtk2]`.