Project 2

Summary:
Download the file

`www.math.us.edu.pl/~pgladki/teaching/2016-2017/tk_lab2.zip`

Run **sml** and compile the code using **CM.make "sources.cm";**

This assignment consists of 2 parts. They are independent of each other, so it doesn't matter which one you work on first.

Part A: abstract syntax

Translate (by hand) a short Fun program into abstract syntax constructors.

(1) Read and understand the following files: absyn.sml test.sml. *Notice that in test.sml there are five Fun programs translated by hand into abstract syntax constructors.*
(2) After your CM.make, execute **Test.run();** from the sml interactive prompt. This will pretty-print and evaluate all five programs.
(3) Read the following files, but you're not (yet) responsible for understanding them in detail: heap.sig heap.sml eval.sig eval.sml funpp.sig
(4) Glance through the following files, but most likely you will never be responsible for understanding them in detail: table.sig table.sml funpp.sml
(5) **Finish myfun.sml by translating the given program into abstract syntax constructors.**

Part B: lexical analyzer:

Build a lexer for the Fun language. In order to do this, you will need to figure out all of the tokens that you need to lex, by reading

`www.math.us.edu.pl/~pgladki/teaching/2016-2017/tk_fun.html`

You will be building your lexer using ML-LEX. There is online documentation on

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

There is also some help in your textbook (Appel, chapter 2).

(1) Read and understand the following files: sources.cm, errormsg.sml, tokens.sig, tokens.sml, fun.lex, runlex.sml. Only these files are relevant to this part of the assignment.
(2) Edit the code in **fun.lex**. You will have to remove some of the sample code and add a lot of your own.
- The tokens are declared in tokens.sig. Here is a list of symbols that will appear in the source along with the tokens they should be associated with:

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

where i is a nonnegative integer without leading 0's: 0,1,...
Fun keywords should be represented using tokens with the same name. The end-of-file token should be represented using the token EOF.

Each token takes two integers: the line number and column number of the beginning of the token. These are used for error reporting. In the example below, x is on the second line in column 7. Notice, the first row is row 1 and the first column is column 1 (as opposed to 0).

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

- UseErrorMsg.error: ErrorMsg.pos2 $->$ string $->$ unit to report errors. It takes two arguments: a pair of file-positions (the beginning and end of the erroneous text, measured in characters from the beginning of the file), and the error message to print out. You should keep the ErrorMsg module informed of where the newlines occur (by calling newLine) so that it can translate these file-positions to line numbers. The make_pos function is a convenient way to convert the things ML-Lex knows (yypos and yytext) into the file positions of the beginning and end of the token.
- Be careful with your syntax in lex files. Remember that each lex definition must end with ";" and each lexing rule must also end with ";". If you forget ";" then ML-Lex will complain.
- **type** is a reserved keyword that we may use for later language extensions. For now, the easiest way to handle it is to simply enforce that programs don't contain it. Since there's no token type for it in the current files, this can't be done in the parser, so you may want to do it in the lexer. But it's also fine if you don't handle it at all - simply assume that programs don't contain **type.** But strings like **typexx** should still be lexed as identifiers.
- To test your code, run **RunLex.runlex "test.fun";** It will output a sequence of tokens along with the line and column number of each token. As always, a single test is far from complete. You will want to write your own test cases and thoroughly test your lexer.

(3) **Nested comments.** This assignment would be a lot easier if the Fun language didn't have nested comments. I recommend that you first get everything working except nested comments. Then add that feature as your time permits.

That's it for now. Send the files myfun.sml, fun.lex and readme over email, where, in particular, you should describe all the decisions you have made while designing your software. The deadline for this assignment is **November 21st, 2016**.