Finish Early this weekend

CS 4323
PROGRAMMING ASSIGNMENT #2
(PARSER)

Due: April 26, 2018

rules changed from HW #1, now it is LL(1)

Refer to the lexical definition of TrumpScript++ Language (A1), the syntax rules for TrumpScript++ language (A2), the LL(1) parse table for TrumpScript++ (A3), and the general structure of the LL(1) parser (A4). Add to your compiler a procedure/class PARSER() which, when given an arbitrary input program, generates its parse output, using the syntax rules for TrumpScript++ and the LL(1) parse table.

In order to write PARSER(), the first thing you have to do is to transform the syntax rules for TrumpScript++ (given in A2) into a PDA, which is nondeterministic in general. The transformation algorithm was given in class. Now, with the lookahead of one token allowed (note that a token is viewed as a single symbol here) and the LL(1) parse table (given in A3) added to the finite control, the PDA is a deterministic LL(1) parser.

PARSER() should be called *once* from the main body of your program. Once it is called from the main body, the whole parsing must be done in PARSER().

SCANNER() needs some modification now. First, it will not be called from the main body (see A4); it will be called from PARSER() when a lookahead token is needed. Second, your previous SCANNER() *printed* a token as it identified one; here, it must *pass* it to PARSER(). SCANNER() will pass it in the form of a pair (token, token type), where token means the actual token identified (i.e., lexeme) and token type means the classification of tokens into those represented in the syntax rules. Finally, SCANNER() need not print out the scanning information now, since you know that it does the correct job (if you correctly constructed SCANNER() in the first programming assignment).

It is convenient to use *integer codes* to classify token types. We shall use integers 1..3 for [id], [const] and [string], integers 4..26 for the 23 keywords, and integers 27..33 for the 7 special symbols. 37

It is also convenient to define the stack of the parser using an array of integers; an array of size 100 should be enough for our purpose. Note that the stack stores nonterminals and terminals. For terminal tokens, use the integer codes 1..33 passed from SCANNER(). Use integers 34..53 for the 20 nonterminals in the grammar. Use 0 for the stack bottom marker. 20

More details will be discussed in class, with examples. For the required output, the following informations must be printed in a well-documented form:

- Print out the source program to be parsed (given on the next page), exactly as you stored in your input file. This must be done before parsing begins.
- Print out the parse output, i.e., the sequence of actions taken by the parser together with supporting informations (stack-top symbols and lookhead tokens in their original names and integer codes). An example of the parse output will be given in class; it is important to follow the output format given in this example.
- Print out the SYMTAB content. This must be done in the main body of your compiler after parsing has been completed.

identifier and const → 2 · cstring

parser did not need to know what exactly it is

(scanner will be called use boolean flag to see if the lookahead is consumed or not)

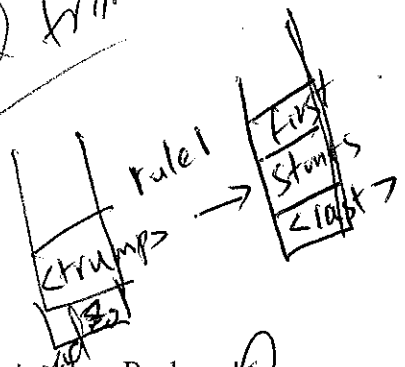
comment

Source program:

edited according to the new rules.

changed from before

```
Make programming great again
# main body begins
Make x number; make   y1 z2zz w number;
make a b Boolean  ;
X is 1000001; y1 is 2000000; z is 123456789;
A is fact; b   is lie;
As long as, fact or lie ;
:
Tell x y1 z2zz ; say "continue";
If , more x plus (y ) times 2000000 z? ; : tell a b; say "stop"; ! else make c Boolean; !;
C is not not not fact and less x z ?;
Tell a b c x y z ;
Say "done";  # say done
!
America is great
```



non_terminal need lookahead

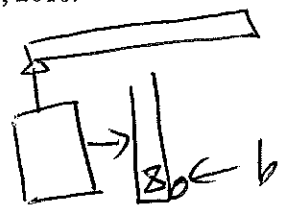
$z_0 \rightarrow 0$

make not consumed yet, so not need to call the scanner.

Submit a hard copy of your program and output on the due date in class. An incomplete work claiming partial credit (of no more than half the full credit) should include on the first page of the submitted work a description of what troubles you had. Failure to include this information may result in zero credit on the work. Penalty for late submission: 10% per calendar day.

Assignment handed out on April 3, 2018.

stack + lookahead



is symbol table. Expand left to right

flag

Sample output

LL(1) parse output	stack top symbol	look ahead	Actions
0	z_0 (0)	—	push(Trump)
1	Trump (34) (Trump call Parse scanner)	Make	use Rule(1)
2	<first> (not call scanner, make not consume yet)	Make	Rule(2)
3	make ( )	make	match, pop
4	programming	programming	consume stack, match ✓
5	great	great	match
6	again	again	match
7	<stmts> ;	Make.	match
8	<stmt>	make	rule 7
9	<decl>.	make	rule(12)
10	make	make	match
11	<ids>	✗ (only stored id)	rule(20)
12	[id]	[id]	match
13	<more_ids>.	number	32

200 steps +

A1. Lexical definition of TrumpScript++ Language

- **Keywords**: make, programming, great, again, america, is, else, number, boolean, if, as, long, tell, say, fact, lie, not, and, or, less, more, plus, times.
- **Identifiers**: any letter followed optionally by digits and/or letters.
- **Constants**: any sequence of digits whose corresponding value is greater than 1,000,000.
- **Strings**: any sequence of characters in a pair of " and ".
- **Special symbols**: „ ; : ! ? ( ).

A2. Syntax Rules for TrumpScript++ Language

[1] <Trump> -> <first> <stmts> <last>
[2] <first> -> Make programming great again
[3] <last> -> America is great
[4] <stmts> -> <stmt> ; <more-stmts>
[5-6] <more-stmts> -> <stmt> ; <more-stmts> | \epsilon
[7-11] <stmt> -> <decl> | <asmt> | <cond> | <loop> | <output>
[12] <decl> -> make <ids> <type>
[13-14] <type> -> number | boolean
[15] <asmt> -> [id] is <expr>
[16] <cond> -> if , <bool> ; : <stmts> ! else : <stmts> !
[17] <loop> -> as long as , <bool> ; : <stmts> !
[18-19] <output> -> tell <ids> | say [string]
[20] <ids> -> [id] <more-ids>
[21-22] <more-ids> -> [id] <more-ids> | \epsilon
[23-24] <expr> -> <bool> | <arith>
[25-28] <bool> -> fact <bool-tail> | lie <bool-tail> | not <bool> | <test> <arith> <arith> ?
[29-31] <bool-tail> -> and <bool> | or <bool> | \epsilon
[32-34] <test> -> less | is | more
[35-37] <arith> -> [id] <arith-tail> | [const] <arith-tail> | (<arith>) <arith-tail>
[38-40] <arith-tail> -> plus <arith> | times <arith> | \epsilon

A3. LL(1) Parse Table for TrumpScript++ Language

Productions		FIRST$_1$	FOLLOW$_1$	LOOKAHEAD$_1$
<Trump>	1	Make	ε	Make
<first>	2	Make	FIRST$_1$(<stmt>)	Make
<last>	3	America	ε	America
<stmts>	4	FIRST$_1$(<stmt>)	America !	FIRST$_1$(<stmt>)
<more-stmts>	5	FIRST$_1$(<stmt>)	America !	FIRST$_1$(<stmt>)
	6	ε		America !
	7	make		make
	8	[id]		[id]
<stmt>	9	if	;	if
	10	as		as
	11	tell say		tell say
<decl>	12	make	;	make
<type>	13	number	;	number
	14	boolean		boolean
<asmt>	15	[id]	;	[id]
<Cond>	16	if	;	if
<loop>	17	as	;	as
<output>	18	tell	;	tell
	19	say		say
<ids>	20	[id]	number boolean ;	[id]
<more-ids>	21	[id]	number boolean	[id]
	22	ε	;	number boolean ;
<expr>	23	FIRST$_1$(<bool>)	;	FIRST$_1$(<bool>)
	24	FIRST$_1$(<arith>)		FIRST$_1$(<arith>)

A3. LL(1) Parse Table (Continued)

Productions		FIRST$_1$	FOLLOW$_1$	LOOKAHEAD$_1$
	25	fact		fact
<bool>	26	lie	;	lie
	27	not		not
	28	less is more		less is more
	29	and		and
<bool-tail>	30	or	;	or
	31	ε		;
	32	less		less
<test>	33	is	[id] [const] (	is
	34	more		more
	35	[id]	; [id] [const] (?)	[id]
<arith>	36	[const]		[const]
	37	(		(
	38	plus	; [id] [const] (?)	plus
<arith-tail>	39	times		times
	40	ε		; [id] [const] (?)

A4. Deterministic LL(1) Parser