CSC45500 Project #2

DUE: Thursday, February 8, 2024, 11:59 PM

1 Objectives

- Building a parser
- Implementation of a grammar
- Writing a "code beautifier"

2 Problem Statement

You are to build a recursive descent parser for the programming language described by the grammar that follows. In this grammar, note that nonterminals are in *italics* and terminals (i.e. tokens) are in **boldface** as opposed to being surrounded by < and > or their absence respectively. This (hopefully) makes the grammar a little bit easier to read. The nonterminal *program* is the "starting" production.

The recursive descent parser should perform 2 tasks:

1. Print out a comment string indicating if an inputted program file contained a valid program or not *and*
2. If the program is valid (*and only if the program is valid*), it should print out a "beautified" version of the code (see below).

program	→	*declaration program* | *compound*
declaration	→	*type idlist* **SEMICOLON**
idlist	→	**ID** | **ID COMMA** *idlist*
type	→	**INTEGER** | **FLOAT** | **VOID**
compound	→	**BEGIN** *stmtlist* **END**
stmtlist	→	*stmt* | *stmt* **SEMICOLON** *stmtlist*
stmt	→	**ID** | **ID LPAREN** *exprlist* **RPAREN** |
		ID ASSIGNOP *expr* |
		IF *expr* **THEN** *compound* **ELSE** *compound* |
		WHILE LPAREN *expr* **RPAREN** *compound* |
		compound
exprlist	→	*expr* | *expr* **COMMA** *exprlist*
expr	→	*simpexpr* | *simpexpr* **RELOP** *simpexpr*
simpexpr	→	*term* | *term* **ADDOP** *simpexpr*
term	→	*factor* | *factor* **MULOP** *term*
factor	→	**ID** | **ID LPAREN** *exprlist* **RPAREN** |
		NUM_REAL | **NUM_INT** |
		LPAREN *expr* **RPAREN**

3 Beautified Code

At a *minimum*, beautified code:

- has no more than one statement on each line and has a blank line between the variable declaration header and the main code body.
- follows an indentation standard.
- places spaces before and after expressions as needed.

4 Implementation Details

You must:

- Use the `get` method from the `Token` class from the first project. If you were unable to get project 1 working, see me and I will give you a working version for the Linux machines.
- write a `main` function that reads a filename from the command line (see the example `main()` from project 1) and tries to parse it using the supplied grammar from above.
- Have your program print out a message such as:

  ```
  # successful code ============================================
  ```

  if the given code makes up a valid program. It should print out a message such as:

  ```
  # UNsuccessful code ==========================================
  ```

  if the given code does not make up a valid program. *Note: I will NOT be giving you an invalid input file, so the second comment is really only useful for you as you are debugging your code.*
- if the code was valid, your program should print out the valid code, in a beautified format.

5 What and How to Submit

You will be submitting a tgz or zip file to Canvas that contains the following files:

- all of your source code.
- a file called `read.me`, which describes your project. Partial credit abounds from information found in this file, so *do not treat this part lightly!*

6 Extra Credit

Remember, if you submit this project 48 hours or more early, you will automatically earn a free 5 points.

7 Grading Breakdown

correct submission	10%
successful compilation	20%
following directions	20%
correct execution	40%
Comments & read.me file	10%
Extra Credit	5%

8 Final Notes

- *Hint on how this will be graded:* Any beautified code should be able to be passed back into your program and recognized as a valid program (actually it should be recognized in exactly the same way!).
- START NOW !
- Do NOT try and enhance the given grammar. DO NOT WASTE TIME BY DOING UNNECESSARY WORK!
- *START NOW !*
- If you have *any* questions about *anything* regarding this project, SEE ME IMMEDIATELY !!!!!
- **START NOW !**

9 Example Run

Suppose you have the following input file, which contains some pretty ugly code:

```
#This really is ugly code.
int a,b,c;void v1,v2;float really;begin while(a+2>b*c-d)begin
calculation=a+b*c;NoParamFuncCall;if(c==b)then begin c=c-b
end else begin doNothing end;a=b+c;print(help) end;print
(awesomeness) end
```

If the above was stored in an input file called `ugly.myl` and you then your beautifier code (whose executable is called `beautifier`) as follows:

```
       ./beautifier  ugly.myl
```

The the resulting output should look similar to:

```
# successful code =========================
int a, b, c;
void v1, v2;
float really;

begin
  while ( a + 2 > b * c - d )
    begin
      calculation = a + b * c;
      NoParamFuncCall;
      if ( c == b ) then
        begin
          c = c - b
        end
      else
        begin
          doNothing
        end;
      a = b + c;
      print( help )
    end;
  print( awesomeness )
end
```