CS 441
Fall 2022
Programming Assignment

For this assignment, you will use Racket, a functional programming language, to write a simple parser.

Note that we're writing only a parser, not a full interpreter (although interpreters in Racket aren't that difficult[1]). Also, your program only needs to pass a verdict on the syntactical correctness of the program, not produce a full parse tree.

The standard interpreter for Racket is DrRacket, and is installed on all Flarsheim lab machines as well as available as a free download from https://racket-lang.org/.

Racket is a functional language, so it requires a different approach than what you may be used to. It's expression-oriented; functions take in parameters and return function values, with no side effects. I'd suggest spending some time getting familiar with it, perhaps working out some common algorithms in a functional form.

Your code should have a function called `parse`, which takes one parameter—the name of the file of source code to be processed:

```
(parse "source1.txt")
```

It should return a string: either "Accept", indicating the program is syntactically correct; or a message as to which line the first syntax error was found:

So output will be either:
`Accept`
or something like:
`Syntax error found on line 25`

In the case of a syntax error, printing the offending line would also be helpful. It is not necessary to continue scanning for additional syntax errors.

The grammar you are to use is listed below. This is a simplified version of '70s-era BASIC: statements are line numbered. There can be more than one statement on a line. You have selection but not iteration. You must define the FIRST, FOLLOW, and PREDICT sets.

Programming notes:

- You will need other functions besides `parse`, of course. This will be a top-down recursive-descent parser. You will need one function per nonterminal in the grammar.
- You're not limited to printing *just* a final verdict; progress messages will probably be helpful in development.
- Submit your source code (.rkt file) and a short document listing any resources you used in developing your program. (This refers to things where you copied & modified someone else's

1 There's an online book, *Beautiful Racket*, that shows how to write an interpreter for the Basic language, along with a couple of specialized languages.

code, used a workaround verbatim, or something like that. General references for Racket don't need to be listed.)

- In addition to the references at the Racket website, you may find some other programming resources helpful:
  - *Structure and Interpretation of Computer Programs* has examples in Scheme, Racket's parent language; all of its examples will run in Racket without modification. The section on symbolic data (2.3, 2.4) will have some good examples of working with abstract data types.
  - Two good general-purpose books on Racket programming are:
    - **Racket Programming the Fun Way: From Strings to Turing Machines,** by James W. Stelly (No Starch Press, ISBN-13: 978-1-7185-0082-2 (print) ISBN-13: 978-1-7185-0083-9 (ebook)); or
    - **Realm of Racket: Learn to Program, One Game at a Time,** by Bice et al. (Also by No Starch Press, ISBN-13: 978-1-59327-491-7).
    - Stelly's book draws mostly on recreational mathematics and classical computer-science problems; Bice et al. focus on game development, from simple guess-my-number games to multiplayer client-server games.

You will be given several sample files, some with syntax errors, some without. Note that lines must begin with a positive integer; there is no *syntax* requirement that the lines be listed in order or not be duplicates. In other words:

```
10 a = 5:b = 10
95 c = a + b
12 write c:write a+b-c
10 a = 1
$$
```

is syntactically correct. (In an actual interpreter, the second definition of line 10 would replace the first. Lines would still execute in numerical order unless overridden with GOTO, GOSUB, or RETURN.) Likewise, a GOTO or GOSUB to a nonexistent line number is a semantic error, not a syntax error.