CompSci 334 **Project 1** **Spring 2021**

Project Due: Tuesday, Feb. 23, 11:59pm
34 points

The company SUPERPETS has hired you to write an interpretor for a simple programming language called CATANDMOUSE for simulating cat/mouse games. This language allows the programmer to specify the starting positions and movements of cats and mice. For valid CATANDMOUSE programs, you will visualize the motion of the cats and mice (with rules given in project 3).

The CATANDMOUSE programming language has a program definition (shown first) and seven types of statements. Note v is a variable, i and j are integers, d is a direction (north, south, east or west) and *stmts* represents 1 or more valid statements.

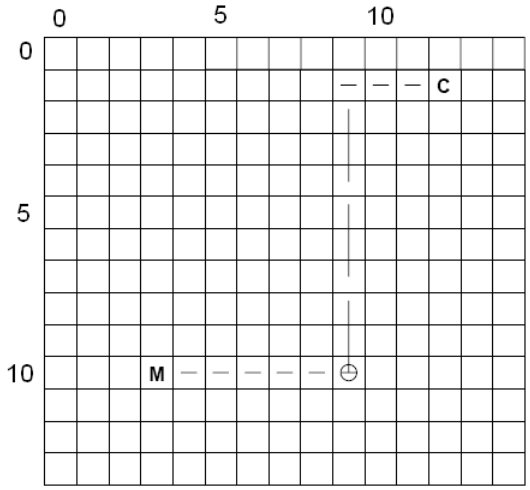
Statement	Meaning
size i j begin *stmts* halt	program definition - defines height i and width j of the room.
cat v i j d ;	draw a cat at position (i, j) in direction d
mouse v i j d ;	draw a mouse at position (i, j) in direction d
hole i j ;	draw a hole at position (i, j)
move v ;	move the critter one space in its current direction
move v i ;	move the critter i spaces in its current direction
clockwise v ;	turn the critter v 90 degrees clockwise
repeat i *stmts* end ;	execute *stmts* i times

Here is a sample CATANDMOUSE program.

```
size 40 60
begin                                   // room has height 40 and width 60
      cat charlotte 12 1 west ;         // a cat, charlotte, at (12,1) heading west
      mouse 1A 3 10 east ;              // a mouse, 1A, (3,10) heading east
      hole 9 10 ;                       // a hole at (9,10)
      repeat 3
            move charlotte ;            // charlotte moves one step west
            move 1A ;                   // 1A moves one step east
      end ;
      clockwise charlotte ;
      clockwise charlotte ;
      clockwise charlotte ;             // charlotte now heading south
      repeat 3
            move 1A ;                   // 1A moves one step east
            move charlotte 3 ;          // charlotte moves three steps west
      end ;
halt
```

This sample program draws a cat, mouse, and hole and then moves the critters around the room. We will assume that all rooms are a grid of points (x,y) with width w and height h with x from 0 to width and y positions from 0 to height. The upper left point is point (0,0). A comment is anything from // to the end of the line.

A picture showing a portion of the room and the critters might look like the following picture. The starting position of the cat is indicated by C, the starting position of the mouse is indicated by M, the hole is indicated by a circle, and the movement of critters is indicated by a line.



The interpretor for CATANDMOUSE will be built in three parts, the first part is this project. For this part, you will write a scanner that will identify the elementary parts (tokens) of a CATANDMOUSE program and store these parts for later use. In project 2, you will write a parser that will identify syntactically correct CATANDMOUSE programs. Project 3 will further extend the parser into an interpretor that will execute a CATANDMOUSE program and generate an animation of cats and mice.

DESCRIPTION OF THE SCANNER

Given a sample CATANDMOUSE program, your first task is to write a scanner to identify all its *parts* (or *tokens*).

The purpose of the scanner is to find the next token in your program, enter its value into a data structure (called a symbol table) that handles searches and insertions, and return 1) a reference to the tokens location in the data structure, and 2) a unique symbol, called the *token type*, which indicates the type of the token. Not every token is entered into the symbol table, but for those that are, make sure that there is only one copy of each. Thus, upon encountering a token, search the symbol table first to see if it is already there. If so, return a reference to its location. If it is not in the symbol table, insert it, and then return the reference to its location.

Your program will repeatedly compute the next token type and the reference to its location in the symbol table. For this project, you will print the token type and its values in the symbol table, and then request the next token (thus losing the information about the previous token). Although we are throwing away this information now, we will use it in projects 2 and 3.

Be sure to keep track of the program line number for reporting where errors occur.

TOKENS

The tokens of a CATANDMOUSE program consist of keywords, variable names, integer constants, and punctuation symbols. Tokens are separated by blanks, end-of-line, and end-of-file.

Not all tokens are entered into the symbol table. If they are to be entered, then a character value and an integer value are inserted for each token.

The tokens of the CATANDMOUSE programming language and their associated types are:

Keywords: Keywords are **not** entered into the symbol table. They have no value. For each keyword found, return its type and NULL for its value. Keywords are only formed using lowercase letters. The uppercase and lowercase of the same letter should be treated as the same, so beGIN is the same as begin.

KEYWORD	TYPE
begin	begin
halt	halt
cat	cat
mouse	mouse
clockwise	clockwise
move	move
north	north
south	south
east	east
west	west
hole	hole
repeat	repeat
size	size
end	end

Variables: Variables are entered into the symbol table. Valid variable names may contain any number of letters and digits. If it contains a digit and is length 3 or less, then it must have at least one letter (1A and 99a3 are valid variable names). The uppercase and lowercase of the same letter should be treated as the same, so SUm is the same variable as sum. The type of a variable is *variable*. The character value associated with a variable is the name of the variable. with all uppercase letters converted to lowercase. Its integer value is set to 0 for now (it is not needed until project 3). (Exceptions: Keywords are not variables. So, *east* is a keyword, not a variable!)

Integers: Integers are entered into the symbol table. Valid integers may contain up to three digits (0-9). If it starts with 0 then it must be of length 1. The type of an integer is *integer*. Integers are read in as character strings. Store the character value and convert the character string to an integer, and also store the integer value (it will not be used until project 3).

Punctuation Symbols: These are not entered into the symbol table. They do not have values. For each symbol found, return its type and NULL for its value. There is only one punctuation symbol in the CATANDMOUSE language.

SYMBOL	TYPE
;	;

Comments are not tokens! In addition to tokens, your program may contain comments. A comment begins with // and includes everything to the end of that line. All comments are to be ignored. When a comment is encountered, ignore everything up until the end of the comment. Since comments are not tokens, there is no type associated with a comment.

INPUT

A data file consists of one CATANDMOUSE program. Sample data files `pX.mc` (where X=1,2,3,...) will be available soon. These are not necessarily the data files that your program will be tested on. To ensure your program runs correctly, you should also create your own data files for testing. A sample data file called *p1.mc* is:

```
// program 1
size 30 40
begin
      cat charlotte 20 21 east ;
      move charlotte 4 ;
halt
```

OUTPUT

For each CATANDMOUSE program print out the following information for each token in three columns: the type of token, the character value, and the integer value. If the token is not entered into the symbol table, then the character and integer values are left blank.

Possible output for the file *p1.mc* above might be:

OUTPUT FOR PROGRAM

TYPE	CH VALUE	INT VALUE
====	========	=========
size		
integer	30	30
integer	40	40
begin		
cat		
variable	charlotte	0
integer	20	20
integer	21	21
east		
;		
move		
variable	charlotte	0
integer	4	4
;		
halt		

ERROR HANDLING

Your program should handle files that contain invalid tokens. When an invalid token is found, report it as an error, list the line number the error occured on, and continue processing.

For example, consider the following CATANDMOUSE program.

```
cat charlotte 20 21 east ;
mouse 98 5 6 north+1 ;
hole 5874 8; // Wow * * *
move charlotte east - 3 ;
```

This program is loaded with syntactic errors, however, for project 1 identify only invalid tokens. The syntactic errors will be caught in project 2.

The invalid tokens above are:

- In line 2: north+1 is not a valid variable name
- In line 3: note that 5874 is NOT an integer and is NOT an error, it is a valid variable name.
- In line 3: 8; (there is no separator between "8" and ";", so they are treated as one token, which is invalid.)
- In line 4: − is an invalid token.

When an invalid token is encountered in the scanner, print an error message indicating the error and the line number, and the token, then continue scanning for the next token.

THE PROGRAM AND ITS SUBMISSION

REQUIREMENTS:

- Your program can be written in Java or Python.
- If written in Java, the name of the file with main should be called `Project1.java`. If written in Python, the name of the file with main should be called `Project1.py`.
- Your program should prompt the user for the name of the CATANDMOUSE program to test. If that file is in the same folder as your code, then your program should run on that file.
- Submit your program as a .zip file under **Project1Files** in Sakai under assignments.
- You must also submit a video that is 3 minutes or less long where you explain and show us the modules/methods you wrote and what each one does, and run your program on at least three examples. The video can be submitted one day late with no penalty. Submit the video in Sakai under **Project1Video**.

  You can make the video however you want. One way is to use zoom.
- In addition to submitting your program, you must fill out the REFLECT form on the assignment page.

GRADING

Your program will be graded on style as well as content. Style will count for 15-20% of your grade. Appropriate style for this course includes:

- *Modularity* - Your program should be divided into multiple methods and/or classes. Comments should describe each part of the methods/class(es).
- *Liberal use of comments* - In addition to the comment for each module, each nontrivial section of code should have a comment describing its purpose. Comments should not merely echo the code.
- *Readability* - Your program should use the indentation and spacing appropriately to make it easily readable. Your comments should be clearly distinguishable from the code.
- *Appropriate variable names* - Give variables names that describe their function.
- *Understandable output* - Your program should indicate its input as well as its output in a clear and readable manner. Remember, the output from your program is the only indication that it works!

The remaining part of your grade is based on meeting the specifications of the assignment. If you do not get your program correctly running, for a small amount of partial credit you may generate output that identifies which part of your program is correctly working. This output must also be clearly understandable or no credit will be given!

Late Policy

Programs not submitted by the due date are penalized 10% up to three days late and 20% if four or more days late (Sunday does not count as a late day). You must meet with Prof. Rodger if your program is not turned in one week after the deadline.

EXTRA CREDIT (2 pts)

For extra credit, if a word is not a valid token, assume that it contains tokens that are not separated by whitespace, try to identify the tokens, print a warning message, and then return the tokens one at a time as valid tokens.

For any part that cannot be identified, report an error, discard the invalid token, and check the next token.

Examples:

The word `one15;` is actually two tokens: one15 and ;. Split this into two tokens, "one15" first and then ";". You should either print one warning message for both tokens, or two separate warning messages.

For example, `Me3*98` would be returned as the variable `Me3`, discard the `*`, and later return the integer `98`.

The word `begin`$*$ should be reported as the keyword begin and an invalid token $*$.

The words `starmove` and `movestar` are valid variable names. Even though movestar contains the keyword move, this is a valid variable name, so don't assume that an error has been made.