Assignment 4 - Simple Interpreter
Due Friday, November 24, 2023

Write an interpreter which reads a program file and then executes the code stored in the file.

1. The Language
The program file has the format:
  <lineNumber> <command> <arg1> <arg2> <arg3>
The number of arguments depends on the command. There can be 0, 1, 2, or 3 args included on each line.

There are no lines without a <lineNumber> and a <command> with the associated arguments.

The language uses the following keywords:

int <var> -declare a variable as an int with the name <var>
set <var> # -set the variable <var> equal to #
begin -start program, after all declarations, there is one begin command
end -last line of the program, there is one end command

add <var> # -these functions perform arithmetic operations using
sub <var> # <var> and #, for example: add <var> 3 means
mult <var> # <var> = <var> + 3
div <var> #

print <var1> <var2> string -if the program is using the graphics mode it prints the string at row == <var1> and column == <var2>
-if not in graphics mode it prints the contents of <var1>, then <var2>, then the string to stdout.
-the string cannot have any spaces within it

goto <lineNumber> -jump to the <lineNumber> and begin executing there
-you cannot jump before the begin line
-goto can jump anywhere between the begin and end commands, including both of those commands

if <var> <op> <var> -<op> can be eq, ne, gt, gte, lt, lte
-evaluates the expression
-if true then execute the next line
-if false then jump over next line

The begin and end commands perform no operations. Think of them as brackets which define a block of code.

Sample Program - pgm1

```
1 int a
3 int b
5 begin
10 set a 5
12 add a 2
15 set b 7
40 if a eq b
50   print a b a==b
100 end
```

Without curses the output will be:

```
7 7 a==b
```

Sample Program - pgm3

```
1 int a
2 int b
5 begin
8 set a 5
9 set b 0
50 add b 1
55 print  b b *
60 if a ne b
70   goto 50
100 end
```

Without curses the output will be:

```
1 1 *
2 2 *
3 3 *
4 4 *
5 5 *
```

There are two steps to implementing the interpreter. They are writing reading the file and loading it into memory and interpreting the program.

2. Reading the Program File

Store the contents of the file in memory. The file structure can be thought of as multiple columns:

line # command arg1 arg2 arg3

The line # is an integer. All of the other values are strings or integers.
It may be easier to treat all of the args as strings and convert the
ones which are integers into ints when needed.
There can be 0, 1, 2, or 3 args. For example:

-begin -has zero arguments
-goto 50 -has one argument
-add a 2 -has two arguments
-if a gt b -has three arguments

The command determines they type of the args. For example:
int string
set string int
add string int
print int int string
goto int
if string string string

Variable names will always be strings. Variable values will always be ints.

You can use any data structures you wish to store the program. For simplicity
they are shown as arrays in the following example.

Example: Internal representation of the code.

array index	lineNumber[]	command[][]	arg1[][]	arg2[][]	arg3[][]
0	1	int	a		
1	3	int	b		
2	5	begin			
3	10	set	a	5	
4	12	add	a	2	
5	15	set	b	7	
6	40	if	a	eq	b
7	50	print	a	b	a==b
8	100	end			

3. Interpreting the Code
You will need to store the declared variables in memory. Both the name and
the value of the variable must be stored. The internal representation of the
variables can be created when the program is read from the file or when it
is being interpreted. There can be up to 1000 variables with names of up to 10 characters long.

This discussion refers to the previous example of the code when it is stored in memory. Your program should start interpreting the code on the begin line. The program will need to maintain a program counter which keeps track of which line of code it currently being interpreted. This was the array index in the previous example. The program counter for the begin statement is set equal to 2. The begin statement does not perform any actions so the only operation to perform is to increment the program counter to 3.

The next commands is on lineNumber of 5 and the command is set. The set command requires two arguments, the first is a variable name==a, the second is an int==5. The program must find the variable named a in memory and store the value 5 in it.

```
intNames[][]   intValues[]
a              5
b
```

Once this is done the program counter is incremented to 3 and the next line is interpreted.

There are two commands which directly change program counter.
The goto # command causes the program to move to line with the number which follows the goto command (arg1). To do this the program needs to locate the lineNumber which matches the goto statement and change the program counter to that line.

```
program   line
counter   number
0         1            int a
1         2            begin
2         5            set a 0
3         12           add a 1
4         20           goto 40
5         30           print a a *
6         40           end
```

The goto 40 command will make execution jump to line 40. This will jump over line 30 and end the program. The interpreter must search the line numbers and find line 40. It will then set the program counter to the index value for line 40, which will be 6 in the above example.

The if command can also affect the program counter. When the expression in the if statement is true then the program counter increments by 1. When the expression is false the program counter is incremented by 2.

Depending on implementation, the program counter can be the same as the array index.

4. Output

The output depends on if the curses version of the code is being used or if it is the non-curses version.

The curses program will draw output using the locations provided by the print statement in arg1 and arg2.

The non-curses program will print the output to stdout.

5. Starting Code

The starting code compiles into two programs. One uses curses the other does not. Both programs should work correctly with your code.
The curses version of the code is named a4, the non-curses version is named a4ng.

The non-graphics program ends when the program finishes interpreting the code. If it is stuck in an infinite loop then it wont end. The graphics version of the code exits when q is pressed.

Coding Practices

Write the code using standard stylistic practices. Use functions, reasonable variable names, and consistent indentation.
If the code is difficult for the TA to understand then you will lose marks.

Do not use goto statements or global variables unless they are provided in the starting code.

As usual, keep backups of your work using source control software.

Submitting the Assignment

Submit only the source code a4.c, the makefile, and readme.

Include a readme file.
If there are any parts of your program that don't work as specified then describe them in the readme.