Tabletop Script Compiler

Description of the Compiler Project for the course Formal Languages and Compilers 2024/25

The Compiler's functionality

The compiler processes input programs and executes C code based on the program's syntax. The Tabletop Script language provides custom syntax with special features for tabletop games.

The compiler supports three data types: integers, floating-point decimals, and strings. Integer and decimal types default to 0, while strings initialize as "undefined," similar to C's NULL.

A hash table-based symbol table with scopes tracks variables. Variables must be declared in a single line and assigned values in subsequent lines.

The compiler handles basic mathematical operations, number and string comparisons, and string concatenation using the '+' operator. Operations follow a specific precedence: parentheses first, then unary minus, followed by equality and comparison operators, multiplication and division, and finally addition and subtraction. If both operands are integers, the result is an integer; if one is a decimal, the result is a decimal.

Two built-in functions, **prt()** and **prtln()**, print values to the console. **prtln()** appends a newline, while **prt()** does not.

Another way of assigning integer values is the use of the dice notation. The compiler interprets inputs in the form of ***N*d*M*** as rolling of N dice with M faces. Modifiers **adv** and **dadv** roll two dice instead of one and take the better or worse result respectively. Dice rolls can be combined with other integer or decimal values and assigned to variables.

Simple **if**-conditional statements without "else" conditions are supported. Successful conditions (evaluated to 1) execute the subsequent block, executing in a new scope. Comparisons use '>', '<', '>=', '<=', '==', and '!=', returning 0 (false) or 1 (true).

Errors raised during program execution, or lexer interpretation the compiler result in descriptive error messages, including the line number of the problematic code.

The language grammar

```
<program>          → <block>
<block>            → { <stmt_list> }
<stmt_list>        → <statement>
                   |  <statement> <stmt_list>
                   |  <block>
                   |  <block> <stmt_list>
<statement>        → <declaration> ;
                   |  <assignment> ;
                   |  <function_exec> ;
                   |  if ( <expression> ) <block>
<declaration>      → int ID
                   |  dec ID
                   |  str ID
<assignment>       → ID = <expression>
<expression>       → INT_LITERAL
                   |  DEC_LITERAL
                   |  STR_LITERAL
                   |  DICE
                   |  DICE adv
                   |  DICE dadv
                   |  ID
                   |  ( <expression> )
                   |  - <expression>
                   |  <expression> == <expression>
                   |  <expression> != <expression>
                   |  <expression> > <expression>
                   |  <expression> >= <expression>
                   |  <expression> < <expression>
                   |  <expression> <= <expression>
                   |  <expression> + <expression>
                   |  <expression> - <expression>
                   |  <expression> * <expression>
                   |  <expression> / <expression>
<function_exec>    → prt ( <expression> )
                   |  prtln ( <expression> )
```

The following regular definitions are used in the Grammar above:

```
INT_LITERAL        → [0-9]+
DEC_LITERAL        → [0-9]+.[0-9]+
STR_LITERAL        → "[^"]*"
DICE               → [0-9]+[dD][0-9]+
ID                 → [a-zA-Z_][a-zA-Z0-9_]*
```

Description of Compiler Inputs

Sample programs are included in the “**samples**” folder provided with the project. Compiler Inputs are also described in more detail in the **README.md** file.
In order to run Tabletop Script programs on the compiler, a new file containing the program must be created. The file will then be passed to the compiler in order to execute it.

Instructions on running the Compiler

The compiler is compiled using the provided Make file. It uses flex and bison in order to create an executable compiler. You can build the compiler using the **make** command inside the folder that contains the “**Makefile**”.

If you wish to execute the compiler with additional debug logging, you can set the variable **DEBUG_MODE** in the **tabletop-script-compiler.y** file to **1**. Make sure that you compile the compiler again in order to apply the changes.

In order to run a program on this compiler, it is necessary to pass the program as an input to the compiler. This can be done, on unix based systems, through the command:

```
./tabletop-script-compiler.o < program.tts
```

The **program.tts** file is the program you wish to be executed by the compiler.