COMP 520 Compiler Design
Individual Assignment #2
Language Specifications

Overview:

Given the lexical and syntactic choices from the first assignment, your second assignment is to implement the relevant semantics for each language construct. Note that as you implement your compiler, some decisions may be more difficult to implement than you first thought - or we might have missed out a key detail you need. In these cases, bring them up in class and we can discuss possible changes. By the end of this assignment you will have produced your first full compiler!

Specifications

A program in `minilang` consists of a list of variable declarations followed by a list of statements.

Declarations

Variable declarations must follow the following 2 rules or a compile time error "INVALID..." with explanation must be thrown.

- Variable identifiers must be defined before being used

```
var a : int;
b = 5; // INVALID: "b" is not declared
```

- Variable identifiers must not be redeclared (regardless of type)

```
var a : int;
var a : int; // INVALID: "a" is already declared
```

For this assignment, we will be using `int, float` from C to represent integer and floating point types.

Variable Initialization

Variables are initialized with the following default values:

- **int:** `0`
- **float:** `0.0`
- **string:** `""` (the empty string)

Statements

- **Read** into a variable according to C `scanf` semantics

  ```
  read x;
  ```

- **Print** an expression according to C `printf` semantics

  ```
  print x * x;
  ```

- **Assignment** into a variable. Assignment compatibility is as follows:

  ```
  int := int
  float := float
  float := int
  string := string
  ```

  Note that `floats` may not be assigned to `ints`.

- **If statement**, with optional else branch. The condition `<expression>` must be an integer

  ```
  if <expression> then
      <stmts>
  [else
      <stmts>]
  endif
  ```

- **While loop**. The condition `<expression>` must be an integer

  ```
  while <expression> do
      <stmts>
  done
  ```

Expressions

Literals

Literals in `minilang` have their corresponding types. i.e. an integer literal is of type `int`, etc.

Binary Operations

Given a binary expression `<expr1> <op> <expr2>` where `<op>` is one of `+, -, *, /`:

- `int <op> int` is VALID and results in type `int`
- `float <op> float` is VALID and results in type `float`
- `int <op> float` (and vice-versa) is VALID and results in type `float`
- `string + string` is VALID and results in type `string`. The semantics of this operation are string concatenation.
- `string * int` (and vice-versa) is VALID and results in type `string`. The semantics of this operation are string repetition.

  ```
  "derp" * 3 -> "derpderpderp"
  ```

  Note that a runtime check to ensure the integer is ≥ 0 will be required. A value of 0 results in the empty string. A negative value produces a runtime exception.

Unary Operations

Given a unary minus expression `- <expression>`

- `<expression>` must be either of type `int` or of type `float`
- the resulting expression has the same type as `<expression>`