CS305 – Programming Languages

Spring 2019-2020

Extra Homework 1

Due date: April 22, 2020 @ 23:55

1 Introduction

In this homework you will implement a scanner, parser and semantic analyzer for a simple language. The following sections provide extensive information for the homework. Please read the entire document carefully before starting homework. Also, an executable that works as desired is shared with you. You can compare output of your program with the executable output.

2 Keywords

Below is the list of keywords that will be implemented. The list corresponds to the complete list of keywords.

int	string	return	print

tINT, tSTRING, tRETURN, tPRINT token names can be used for int, string, return and print keywords respectively.

3 Operators & Punctuation Symbols

Below is the list of operators and punctuation symbols to be implemented, together with the corresponding token names.

Lexeme	Token	Lexeme	Token
(	tLPAR	)	tRPAR
,	tCOMMA	%	tMOD
=	tASSIGNM	-	tMINUS
+	tPLUS	/	tDIV
*	tSTAR	;	tSEMI
{	tLBRAC	}	tRBRAC

4 Identifiers & Integers & Strings

You need to implement identifiers, integers and strings. The rule for an identifier is the following:

- An identifier consists of any combination of letters, digits and underscore character. However, it cannot start with a digit.
- Integer can be any positive or negative integer, for example 1, -2, and 10.
- String can be anything that is defined between two double quotation mark. For example:

  ”this_is a str1ng...”

- You can use tIDENT, tINTVAL, tSTRINGVAL as a token name for an identifier, integer value and string literal respectively.

5 Language Definition

The grammar you will write will generate the CVD19 language as described below. Here is an example program in this language to give you an idea how a CVD19 program looks like. Below is the detailed syntactic features of the CVD19 language:

- A CVD19 program consists of a non-empty sequence of items where each item is either a function definition, or a variable declaration, or an assignment, or a print statement.
- A variable declaration is given by the type of the variable, followed by an identifier (i.e. the name of the variable), followed by an equality sign (i.e. the character =), followed by an expression, and followed by a semicolon.
- The type can either be integer or string.
- An expression can be a value, or a function call, or two expressions being applied to a binary operator.
- Binary operators are $*,+,-,/$, %.
- A value can be an identifier, or an integer value, or a string value.
- A function call starts with an identifier, followed by a left parenthesis, followed by zero or one or more comma separated values (these are the actuals) and followed by a right parenthesis.
- An assignment statement is given by a variable name, and an equality sign (i.e. the character =), and then an expression, finally followed by a semicolon.
- A function definition is given by a type (which is the return type of the function), followed by an identifier (this is the name of the function), followed by a left parenthesis, followed by zero or one or more comma separated parameters (these are the formal parameters), followed by a right parenthesis, followed by a opening curly braces, followed by the body of the function, lastly followed by closing curly braces.
- The body of a function is a sequence of variable declarations, assignments, print statements. These declarations, assignment and print statements can be given in any order. However the last statement of the body of a function must be a return statement.
- A return statement is given by the keyword "return", followed by an expression, and followed by a semicolon.

- A formal parameter of a is given by a type, followed by an identifier.
- A print statement starts with the keyword ”print”, followed by a left parenthesis, followed by an expression, followed by a right parenthesis, followed by ”;”.

In Figure 1, an example program is given.

```
int a = 5;
string str = "This is a string";

int sum(int a, int b)
{
    int c = 5;
    return x+y;
}

int m = 4;

string hello(string n)
{
    int c = 6;
    return "Hello" + name;
}

int x = 5;
int y = 10;
int z = sum(x,y);
string n = "Yusuf";
print(hello(n));
n = "Ali";
print(hello(n));
print(a);
print(sum(5,1000/2));
print("End Of Program");
```

Figure 1: An example program

6 Semantic Checks

1. A variable should be defined before it's used. If a variable is used without being defined then you have to print the following error.

   ```
   LineNO Undefined variable
   ```

   where LineNO is the line number where the variable is located.

   The variables declared at the global level are visible globally. In other words, when a variable is declared, it is visible in any statement following this declaration. A variable declared at the global level is also visible within the body of a function coming after this declaration. However, a variable declared globally is not visible in the body of a function if this function is defined before the variable.

   For example, in Figure 1 we have the following declaration:

   ```
   int m = 4;
   ```

   The variable `m` is defined at the global level. The variable `m` is visible in any part of the program that comes after this line on which `m` is defined. The variable `m` is also visible within the body of the function `hello` since this function is defined after the definition of the variable `m`. However, the variable `m` is not visible within the body of the function `sum`, since this function `sum` is defined before the definition of the variable `m`.

   A variable defined inside a function (i.e. a local variable of a function) is only visible within the body of the function. Similarly, the variable is visible only after its definition. The formal parameters of a function are to be considered as locally defined within that function, and they are visible in the entire body.

2. Variables that are already defined before can't be defined again. If that's the case you should print the following error.

   ```
   LineNO Redefinition of variable
   ```

where LineNO is the line number on which the second variable declaration is located.

Note again that scope of a variable of a function (which can be a local variable of the function or the formal parameter of the function) is restricted to the body of the function. If a variable defined earlier at the global level is redefined inside a function (as a formal parameter or as a local variable), this will cause an error. If a variable defined inside a function (as a formal parameter or as a local variable) is redefined inside the same function (as a formal parameter or as a local variable), this will cause an error. However, if a variable is defined inside function (as a formal parameter or as a local variable) and it is also defined inside *another* function (as a formal parameter or as a local variable), then this will not cause an error. For example consider the variable `c` defined both in the function `sum` and in function `hello`. There will be no error produced for `c`.

There can be multiple errors in a program. Your analyzer will simply print out the first error and stop.

There may not any semantic errors (due to undefined variable or multiple definition as explained above) in the program, but the program can simply have a syntax error (i.e. the program being analyzed can be grammatically incorrect). In this case your analyzer should simply print out `ERROR` and stop.

If there are no syntax errors and there are no semantic errors, then you should simply write `OK`.

7 Submission

Submit your scanner and parser through SuCourse. You can also submit your header file if you create. The name of your files must be:

```
username-ex1.flx
username-ex1.y
username-ex1.h (optional, if any)
```

where username is your SuCourse username.

8 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important**: Make sure you submit correct files. If we are not able to compile your code **grade will be zero for this homework.**
- Make sure your output is exactly as it is supposed to be.
- No homework will be accepted if it is not submitted using SUCourse.
- Since the grading will be done automatically on the **flow.sabanciuniv.edu** machine, we strongly suggest you to at least test your code on flow before submitting it.
- Late submission is allowed subject to the following conditions:
  - Your homework will be multiplied by a "submission time factor (STF)".
  - If you submit on time (i.e. before the deadline), your STF is 1.
  - If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
  - We will not accept any homework later than 500 mins after the deadline.
  - SUCourse's timestamp will be used for STF computation.
  - If you submit multiple times, the last submission time will be used.