Language reference Star

BCS Project

Implementation of a compiler

Teacher: Dr. Jasmin Blanchette
Student: F.D. van Emden

Contents

1 Introduction

Star is a toy programming language which contains basic data types. The file extension of programs written in Star is .star.

The language Star has the following scoping rules :

- Variables inside a function or function block are called local variables, and these local variables can only be referenced from within that function or function block, or a function or function block nested inside the scope the variable was declared in.
- A new declaration of a variable with the same name as an already existing variable which resides in a higher scope is not allowed.

Comments are supported in the form of single line comments and multiple line comments. Single line comments are achieved by inserting a // somewhere on a line, making the compiler ignore everything after the // on that line. Multiple line comments can be constructed by putting the text which is to be commented between /* and */.

2 Types

The supported data types in the Star language are listed below.

- bool : 1-byte value representing true or false
- int : 4-byte signed integer value
- char : 1-byte signed character
- float : 4-byte signed floating point value

3 Syntax

The language Star contains a relatively simple syntax, mainly consisting of the non-terminal symbols such as program, statements and expressions, which can be further reduced to terminal symbols such as a constant floating point number type. Important to note is that control flow statements such as return start with a capital letter in Star.

The following regular expression rules apply :

- x* means 0 or more instances of x
- x+ means 1 or more instances of x
- x! means exactly 1 instance of x
- [x] means x is optional
- <0-9a-z>+ means we have 1 or more instances of an alpha numeric character
- <A-F>* means we have 0 or more capital letters of the character set A,B,C,D,E,F

In the syntax rules the non-literals are written in capital letters, literals are bold, character **|** should be interpreted as OR and **[x]** represents that x is optional.

3.1 Program

Programs written in Star consist only of functions. Functions can return a value, but do not have to. Void indicates that a function returns no value. There needs to be one int start() function without parameters which is the first function which is executed when the program starts. Also, function definitions may not contain more than 6 parameters.

Program :
FUNCTION+

Function :
RETURNTYPE IDENTIFIER (**[** PARAMETERS **]**) **{** STATEMENTS* **}**

Returntype :
TYPE | void

Parameters :
TYPE IDENTIFIER [, TYPE IDENTIFIER]*

3.2 Statements

A statement can be :
(1) a function block, which must be enclosed with curly brackets
(2) definition of a variable
(3) an assignment
(4) an expression ended by a semicolon
(5) a return call
(6) control flow

Function block :
{ STATEMENT* **}**

Variable definition :
TYPE IDENTIFIER = EXPRESSION **;**

Assignment :
VARIABLE = EXPRESSION **;**
| VARIABLE MODIFIER EXPRESSION **;**

Expression semicolon :
EXPRESSION **;**

Modifier:
+= | -= | *= | /= | %=

Variable :
IDENTIFIER

Return :
Return [EXPRESSION]**;**

Control Flow :
If EXPRESSION **{** [STATEMENT]* **}** [**Else {** [STATEMENT]* **}**]
For VARIABLE_DEF EXPRESSION**;** ASSIGNMENT **{** [STATEMENT]* **}**
While EXPRESSION **{** [STATEMENT]* **}**
Do { STATEMENT **} While** EXPRESSION**;**

IDENTIFIER : (<a-zA-Z>)+(<a-zA-Z0-9>)*, which is a regular expression indicating that the identifier has to start with 1 character which is a letter, followed by 0 or more characters which are alphanumeric.

3.3 Expressions

An expression can be reduced to one of the following parsing rules :
(1) unary_operator expression
(2) expression bitwise_operator expression
(3) **(**expression**)**
(4) a function call
(5) a variable
(6) a constant

Unary operator :
! | **-**

Bitwise operator :
ARITHMETIC_OPERATOR | RELATIONAL_OPERATOR
| LOGICAL_OPERATOR | EQUALITY_OPERATOR

Arithmetic operator :
+ | **-** | ***** | / | %

Relational operator :
< | > | <= | >=

Equality operator :
== | **!=**

Logical operator :
&& | **||**

Function call :
IDENTIFIER **(** [EXPRESSION [, EXPRESSION]*] **)**

Constant :
INTEGER_CONSTANT | FLOAT_CONSTANT
| BOOLEAN_COSTANT | CHARACTER_CONSTANT

INTEGER_CONSTANT : <0-9>+
FLOAT_CONSTANT : (<0-9>)+**.**(<0-9>)+ | **.**<0-9>+
BOOLEAN_COSTANT : **true** | **false**
CHARACTER_CONSTANT : **'**<0-9a-fA-F>!'

4 Example programs

In this section some example programs are showed which should be correct according to the language parsing rules and syntax, and some which are not. The correct programs should be compiled by the Star compiler without problems into x86-64 Assembly, and the compilation of the incorrect programs should give appropriate errors.

Program 1 :

```
/* Testing data types. This program should give no compilation errors.*/
int start() {
   bool b = false;
   int num = 9999;
   char c = 'x';
   float f1 = 1212.234;
   float f2 = .02923;

   Return 0;
}
```

Program 2 :

```
/* Testing conditionals. This program should give no compilation errors. */
int start() {
   If 5 > 3 {
      char c = 'c';
   }
   Else {
      char x = 'x';
   }

   For int i = 0; i < 10; i+=1; {
      int temp = i*i;

      If i % 2 == 0 {
          temp -= 1;
      }
   }
   int num = 0;

   While num < 10 {
      num += 1;
   }

    Return num + 3;
}
```

Program 3 :

```
/* Testing operators. This program should give no compilation errors. */
int start() {
   bool b = !true;
   int five = 5;

   If 10 > 5 && false == !true && 30 <= 30 || 10 == 10 || 90 != 100 {
      int reachable = 123;
      five *= 2;
   }
   Else {
      five = 999;
   }

   Return five;
}
```

Program 4 :

```
/* Missing int start() function, therefore the program shouldn't compile. */
int main() {
    int a = 10 - 4;
    Return a;
}
```

Program 5 :

```
/* This program declares unsupported global variables, therefore it should not
    compile. */
int global_int = 1337;

int start() {
    If global_int % 1000 == 337 {
        Return 1;
    }
    Else {
        Return 0;
    }
}
```

Program 6 :

```
/* This program has an unparsable expression, namely + < 5, therefore it
    should not compile. */
int start() {
    bool b = true;
    b = false;
    b = + < 5;
    Return 0;
}
```