Compilers project specification: Tiny

Introduction

Tiny is a small language based on a subset of C modified to obtain a better fit with the JVM.

Tokens

- A **NAME** is a string starting with a letter, followed by 0 or more letters, digits or underscores.
- A **NUMBER** is a string of digits.
- A **QCHAR** is a character between single quotes.
- A comment starts with `//` and continues until the end of the line.
- The other tokens:

INT	**int**	IF	**if**	ELSE	**else**	NEQUAL	**!=**
RETURN	**return**	LPAR	(	RPAR	)	LBRACE	{
RBRACE	}	LBRACK	[	RBRACK	]	ASSIGN	=
SEMICOLON	;	COMMA	,	PLUS	+	MINUS	-
TIMES	*	DIVIDE	/	EQUAL	==	CHAR	**char**
WRITE	**write**	READ	**read**	GREATER	>	LESS	<
NOT	**!**	LENGTH	**length**	WHILE	**while**		

Syntax

Conventions:

- Terminal symbols (tokens) are in upper case.
- ``[string]+'' means one or more occurrences of ``string'', where ``string'' is a sequence of symbols.
- ``[string]*'' means zero or more occurrences of ``string'', where ``string'' is a sequence of symbols.
- Otherwise, the rules are as in yacc/bison specifications.

```
program         : [declaration]+
                ;

declaration     : fun_declaration
                | var_declaration
                ;

fun_declaration : type NAME LPAR formal_pars RPAR block
                ;

formal_pars     : formal_par [ COMMA formal_par ]*
```

```
                |      // empty
                ;

formal_par      : type NAME
                ;

block           : LBRACE var_declaration* statements RBRACE
                ;

var_declaration : type NAME SEMICOLON
                ;

type            : INT
                | CHAR
                | type LBRACK exp RBRACK // array type
                ;

statements      : statement [ SEMICOLON statement]*
                |
                ;

statement       : IF LPAR exp RPAR statement
                | IF LPAR exp RPAR statement ELSE statement
                | WHILE LPAR exp RPAR statement
                | lexp ASSIGN exp
                | RETURN exp
                | NAME LPAR pars RPAR          // function call
                | block
                | WRITE exp
                | READ lexp
                ;

lexp            : var
                | lexp LBRACK exp RBRACK       // array access
                ;

exp             : lexp
                | exp binop exp
                | unop exp
                | LPAR exp RPAR
                | NUMBER
                | NAME LPAR pars RPAR          // function call
                | QCHAR
                | LENGTH lexp                  // size of an array
                ;

binop           : MINUS
                | PLUS
                | TIMES
                | DIVIDE
                | EQUAL
                | NEQUAL
                | GREATER
                | LESS
                ;

unop            : MINUS
```

```
                | NOT
                ;

pars            : exp [COMMA exp]*
                |
                ;

var             : NAME
```

Semantics

Data types

Tiny supports two primitive data types: char, and int. The only type constructor is the array type.

```
int             a1;
int[32]         a2;     // an array of 32 integers
int[10][2]      a3;     // an array of 2 arrays of 10 integers
```

Passing paramemters

As in C and java: primitive data types are always passed by value, arrays are passed by address.

There is an automatic conversion between integers and characters.

```
char     c;
c = 10;  // c is the newline character
```

Libraries

There is no support for external functions, file inclusion etc.

I/O

Since there is no support for libraries, I/O is built in.

- The primitive `read` reads either a single character or a single integer (depending on the type of its argument) from standard input.

- The primitive `write` writes its argument, which must have a primitive type, to the standard output.

Expressions

Expressions are standard. As in C, there is no boolean type and an integer value of 0 stands for false, any other integer for true.

Functions

Every function must be declared with a return type but this type may be ignored when called outside an expression. Array parameters need not match exactly. The size of an array can be tested using the `lenth` built-in function.

```
  int f(int[1] a)
  {
  ..
  }

  ..
  int[10] b;
  f(b);   // OK
```

Main function

When executed, the program starts up by executing the function `tiny`.

```
int tiny()
```

```
        {
        ..
        }
```

Memory management

Follows the JVM, so there is no need to e.g. deallocate local array variables.