MAJAN programming language
User manual

Version 0.0.1

Overleaf

February 11, 2024

Contents

1 Introduction

This document is a user manual for MAJAN programming language. MAJAN was created as a project for "Formal languages and compilers" course at Warsaw University of Technology, by: Jan Janiszewski and Maciej Zalewski. In below sections you can find information about language syntax, how the instructions work and what are their limitations.

2 Language elements

This section describes all elements of the MAJAN language broken down into keywords, operators, special symbols, identifiers and numbers.

2.1 Keywords

1. int	**2.** float	**3.** bool	**4.** string
5. if	**6.** else	**7.** while	**8.** write
9. read	**10.** true	**11.** false	**12.** length
13. function	**14.** return		

2.2 Operators

1. = (assignment)	**2.** + (adding)	**3.** - (subtracting)	**4.** $*$ (multiplying)
5. / (dividing)	**6.** == (is equal)	**7.** > (is greater)	**8.** >= (is greater or equal)
9. < (is smaller)	**10.** <= (is smaller or equal)	**11.** ! (boolean negation)	**12.** and (boolean and)
13. or (boolean or)	**14.** xor (boolean xor)		

2.3 Special symbols

1. ;	**2.** ,	**3.** (	**4.**)
5. {	**6.** }		

2.4 Identifiers and numeric/text values

Identifier is a character string consisting of letters and numbers starting with a letter. Both upper and lower case letters are allowed and are distinguishable. A character string that is a keyword or operator cannot be used.

Int value is a sequence of digits, a leading zero is only allowed if it is the only digit. A number sign (**+** or **-**) may precede the sequence of digits.

Float value is a sequence of numbers denoting the whole part of a number, a decimal point and a sequence of digits denoting the fractional part of a number. All these elements are mandatory. A number sign (**+** or **-**) may precede the sequence of digits denoting whole part of a number.

Text value is a character string starting and ending with double quote symbol (").

2.5 Ignored elements

The following elements may also be part of programs written in <name>; they are ignored by the compiler:

Single line comment - starts with hash symbol (**#**), after it any sequence of characters can be placed

Indendation - leading spaces, either tabs (**\t**) or whitespace .

3 Program structure

Program structure is important and it should be as follows:

- global variables declarations
- function declarations
- main code

Example:

```
int a, b;

function int sum(int a, int b)
{
    int c;
    c = a + b;
    write(c);
    return 0;
}

a = 3;
b = 2;
sum(a, b);
```

Code outside of function declarations is treated as a main function code. Every identifier that is used must be declared before usage.

IDs work in order: local variables > global variables > functions. If there are global variable and local variable with the same ID, local variable has priority.

4 Declarations

4.1 Variables

All variables must be declared before any value can be assigned. One can declare many variables at a time, but only with the same type. Variables are declared by writing type and ID. There cannot be two global (or two local) variables with the same ID.

```
int a;
float b, c, d;
string s;
bool tr;
```

Variables declared outside of the function declaration are treated as global variables that can be accessed inside functions. They should be declared at the very beginning of the program. Variables cannot be declared inside if statements and loops.

4.2 Functions

Functions should be declared after global variables. The order of declaration matters, if one wants to nest functions, the nested one should be declared

first. Function declaration must have **function** keyword followed by return type (**int** or **float**), then ID and in parentheses one can pass parameters by specifying type (**int** or **float**) and ID. Code of the function must be between curly brackets and at the end one must specify returned value explicitly.

```
function float sum(int a, float b)
{
    float c;
    c = a + b;
    return c;
}

function int myWrite(int a, float b)
{
    float c;
    c = sum(a, b);
    write(c);
    return 0;
}
```

Variables declared in the function are visible only inside it.

5 Instructions

The language allows the following instructions to be included:

5.1 Expression

An expression statement is any expression followed by a semicolon. Examples:

```
n = 1;
x+y;
(a>= 0 and a<10) or b >3;
677.4343;
```

As can be seen above, it is possible to add brackets to expressions to split them into sub-expressions. Both values and predefined variables can be used in expressions. The keywords **true** and **true** represent the respective boolean values.

5.1.1 Arthmetic operators

Arthmetic operators can only be used on integer and float values, but is possible to mix them:

```
6 + 0.6;
```

The value of this expression will be of type float. Exception to the rule mentioned at the beginning of this section is string type. It is possible to add string values together, result of such operation is concatenation of these strings.

5.1.2 Comparison operators

Comparison operators can only be used on matching types. String type is not supported.

5.1.3 Boolean operators

Boolean operators can only be used on boolean values. Short-circuit boolean evaluation is implemented.

5.2 If instruction

The **if** statement has two forms with and without **else**.

5.2.1 If with else

It starts with the **if** keyword followed by a boolean expression enclosed in brackets. This is followed by a sequence of instructions enclosed in braces, then the keyword **else** and further instructions enclosed in brackets. If condition is evaluated to **true** then instructions after **if** are executed. Example:

```
int a;
if(2 > 4){
a = 6;
} else{
a = 8;
}
```

5.2.2 If without else

The operation and syntax of this form is analogous except that the **else** keyword and the second set of instructions are absent. Example:

```
int a, b;
a=6;
if(a > 0){
b= a +1;
}
```

5.3 Loop instruction

A loop statement begins with a **while** keyword followed by a boolean expression enclosed in brackets, followed by a sequence of statements enclosed in curly brackets. The operation of the loop is standard: as long as the expression is true, the instructions are repeated.

Example:

```
int a;
a=6
while(a <10){
a= a +1;
}
```

5.4 Read instruction

The read instruction starts with the keyword **read** followed by an identifier enclosed in brackets and then a semicolon. As a result of executing the instruction, the variable specified by the identifier receives the value loaded from the input stream. Example:

```
int a;
read(a);
```

Reading the values of boolean variables is not supported.

5.5 Write instruction

The write instruction starts with the keyword **write** followed by an expression enclosed in brackets. As a result of executing the instruction, value of provided expression is displayed on output stream. Example:

```
write(2 + 5 * 3 + 3);
```

6 Compiling source files

In order to compile your source file to LLVM code following command should be executed:

```
python compile.py <path_to_your_source_file>
```

In the same directory where compile.py is located file named output.ll sould appear.