OCamlToe User Manual

Programming Language created by James Buck and Lyubomir Vasilev

Introduction

OCamlToe is a domain specific stream processing language designed to be minimalistic in its syntax. It intentionally provides a limited number of essential operations which are sufficient for its purposes. This manual will go through the functions available to the programmer, the syntax, exception handling and typical program structure together with some finer details that need to be taken into account.

Functions and Operators

OCamlToe is an imperative language that works with variables and provides a For and If statements together with arithmetic, Boolean, some stream operations and writing to output. These will now be discussed in greater detail and the syntax will be explained in the next section.

I. **Variables**

The programmer can assign and retrieve variables. A variable's value can only be an integer or an arithmetic expression (since it evaluates to an integer). A variable can be defined in terms of itself, i.e. a variable can have an arithmetic expression assigned to it that has the same variable as one of its arguments.

II. **For and If statements**

The For and If statements are generally straightforward and it is assumed that the user is familiar with their generic use and concepts. One detail to note about the For statement is that the programmer can only specify the total number of iterations, i.e. the statement is designed to start counting from one and the user only specifies where to count to. It is important to remember that counting starts from one and not zero (as a lot of programmers might be used to). This stripped down loop, although it offers less control, is more intuitive and syntactically more efficient. The problems that OCamlToe aims to solve do not require a more advanced loop.

III. **Arithmetic and Boolean expressions**

These, again, are quite self-explanatory. All the standard arithmetic operations are allowed: addition, subtraction, multiplication, division, modulo and exponentiation. One feature to point out in the language is that arithmetic operations can be performed on variables or functions since they all evaluate to integers (integer is the only primitive type). There is nothing to note about Boolean expression apart from the fact that they can take arithmetic expressions as arguments.

IV. **Stream operations**

The programmer can call a function to get the length of the stream and can also get a particular element from each stream.

V. Writing to output

The user has the ability to write any variable to an internal list structure that OCamlToe manages. When a program is compiled and executed it will automatically print out the output in a suitable manner. This is convenient for the programmer as he does not need to be concerned with printing out the output manually.

Syntax

This section will explain the syntax for each of the above operations illustrated with some code examples.

Operation	Examples
Variables	
Assignment	`$i=0` `$i = $i + 1`
Retrieval	`$first`
For and If statements	
For statement	`for 5 do $i=$i + 1 endfor`
If-then statement	`if $i>1 then $i=$i + 1 endif`
If-then-else statement	`if $i % 2 == 0 then $i=$i + 1 else $i=$i - 1 endif`
Arithmetic Expressions	`5+5` `$i % 2` `$i ^ $y`
Boolean Expressions	`$i != 0` `$i > 5 +5`
Stream operations	
Stream length	`stream_length`
Stream element retrieval	`get_stream 1 5` `get_stream 2 $i`
Writing to output	`write get_stream 1 $i`

Table 1 OCamlToe Syntax

The syntax should be quite clear from this table. One operation that might be need some explanation is the stream element retrieval – get_stream is a function that has two arguments; the first one is the number of stream that the user wishes to extract an element from and the second is the position of the desired element in the stream (this argument can be an integer, variable or a valid arithmetic expression). It works similar to an array. The syntax of OCamlToe is generally easy to use but the user needs to follow certain rules when it comes to sequences, which could need some getting used to. These rules are explained in the next section.

Program Example

```
$i=0;
for stream_length do
    if $i % 2 == 0
        then write get_stream 1 $i
    endif;
    $i = $i + 1
endfor
```

Figure 1 OCamlToe Program Example

The program example shows a computation on a single stream that will print every element in an odd numbered position in the list, starting at 1. To do this the program uses a variable $i to store the stream index, starting at 0. In the for loop it checks to see if $i is an even number, corresponding to an odd position in the stream, and if so it will write the element at the stream index to the output. It does this for every element that the if statement holds for.

Semicolons ';' are used to separate statements from each other. If and For loops are considered statements and so a semicolon is required after the endif / endfor respectively if another statement is to follow. Functions and arithmetic operations are also statements, therefore a semicolon is required after if another statement is to follow. A program must end in a statement without a trailing semicolon. If and For loops also take statements as part of their procedure and so these rules must be followed when writing statements inside them. There should never be a semicolon directly before the endif / else / endfor keywords.

Exception Handling

OCamlToe provides three types of exceptions which should cover most of the mistakes the programmer could make. When an exception is caught when compiling a relevant error message will be displayed:

I. **Syntax error** – occurs when the user has entered an invalid input; message displayed: "Syntax Error. Please check your syntax."
II. **Out of bounds exception** – thrown when the programmer has tried to access a variable from a stream that doesn't exist; message displayed: "Stream number does not exist."
III. **Undefined variable exception** – thrown when the user tries to use a variable that hasn't been declared; message displayed: "Access to an undefined variable."