Contents

The Programming Language OJ

Status

This is the first rough draft of a new language. It need critical proof reading, correcting, and improving.

Goals

OJ is designed to be a very simple structured language with syntax not unlike C, C++ and Java. The name is yet another of the Java/Coffee puns: Orange Juice. In theory if you take a correct OJ program and embed it in a main method of an appropriately named class in a file with the correct name then if can be compiled and run as a Java program. Similar translations could handle C and C++. In practice the I/O (**in** and **out**) may have to be translated for some languages.

Note

Conventionally OJ programs are pout in files with suffix ".oj".

Example 1

```
// A program that inputs a number and outputs its square

out("Number=");

int number;

number = in();

int square;

square = number * number;

out( square );
```

Example 2

```
//Inputting and squaring many positive numbers

int x;

out("Input a series of numbers greater than 0\n");

x=in();//read ahead

while( x > 0 )

{

       out(x); out(" squared is "); out(x*x); out("\n");

       x=in();
```

```
}
```

Notation

This description uses the XBNF/MATHS notation to define and describe the OJ language. XBNF has a set of predefined terms like digit, letter, quotes, backslash, etc. plus the BNF "or" (|) and an "Any number of"(#) meta-symbol.

Lexemes

An OJ program is defined in terms of lexemes like **Strings**, **Integers**, **Variables**, and some reserved words:

```
if, while, in, out, int, else
```

and symbols:

```
= == <= >= < > != * + / % - ( ) { }
```

Ends of lines, tabs, and other extra whitespace can be used to improve the readability of an OJ program. They are ignored except as separating lexemes.

Comments

An OJ comment is optional and can be put after any statement. The copmment is terminated at the end of the line.

1. **comment**::= "//" any_thing. Comments are removed in a lexical scan and replaced by an end of line. They are not shown in the **Syntax** below.

Syntax

Programs

An OJ program is a sequence of statements. Unlike most structured languages it is not a block and so needs no special heading or ending syntax.

1. **program**::= #**statement**. An empty program does nothing of course. A program with a single statement executes it and stops. With two or more statements the first statement is executed, and when it finishes the rest of the program is executed (as if it was a program).

Statements

OJ provides the minimum set of control statements using a C/C++/Java syntax plus assignments, declarations, and output.

2. **statement**::= **control_statement** | **assignment** | **declaration** | **output_statement**.

Assignment Statements

An assignment statement changes the value of a previously declared variable to the value found be evaluating an expression.

3. **assignment**::= **variable** "=" **expression** ";".

```
square = number * number;
```

Declarations

A declaration introduces a new variable that can hold a one integer value at a time.

4. **declaration**::= "int " **variable**";"

```
int number;

int square;
```

Control Structures

There are the only two structures that you need to right a program: while and if-then-else:

5. **control_statement**::= **while_statement** | **if_statement**.

A while statement introduces a loop with a condition and a body:

6. **while_statement**::= "while(" **condition** ")" **body**.

```
while( i > 0 ) { i = i - 1 ; }
```

7. **body**::= **empty_statement** | "{" #**statement** "}".

```
;

{ out(x*x); x=in(); }
```

An if statement selects one of two branches depending on the truth-value of a condition.

8. **if_statement**::= "if(" **condition** ")" **body** "else" **body**.

```
if ( a>b) { out(a); } else { out (b); }
```

```
if (a%2 == 0){out("even");} else ;
```

9. **empty_statement**::= ";"

Expressions

Expressions are evaluated to produce integer values that can be output or stored in a predeclared variable by an assignment.

10. **expression**::= **term** #(**add_operator** **term**).

```
b*b - 4*a*c
```

```
x*x - y*y
```

11. **add_operator**::= "+" | "-".
12. **term**::= **factor** #(**mult_operator** **factor**).

```
b*b
```

```
4*a*c
```

```
(x - y)* ( x + y)
```

13. **mult_operator**::=("*" | "/" | "%").
14. **factor**::= **variable** | **integer** | "(" **expression** ")" | **input_expression**.

```
4
```

```
a
```

```
b
```

```
(b*b - 4*a*c)
```

```
(x-y)
```

Conditions

A condition compares two expressions. Its value (true or false) determines what happens next in an if_statement or a while_statement:

15. **condition**::= **expression** **relation** **expression**.
16. **relation**::= "==" | "!=" | "<=" | ">=" | "<" | ">".

Integers

The only data type is called 'int'. This is is implemented as a 16 bit signed integer. It has the operations of addition, subtraction, multiplication, division, and remainder.

17. **integer**::= digit #digit.

Input/Output

There is a special output statement and a special input expression:

18. **output_statement**::= "out(" (**expression** | **string**) ");". Sends the value of the expression, or the content of the string to the user.
19. **input_statement**::= "in()". Inputs the next integer form the user and returns the value.

. . . . . (end of section **Syntax**) <<Contents | End>>

Strings

OJ has C/C++/Java string constants but (like Algol 60) not much can be done with them. You can output them with an **output_statement**.

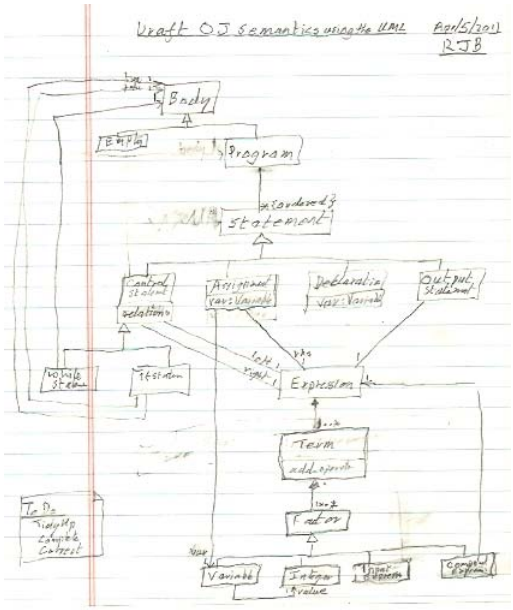
2. **string**::= **quotes** #**string_element** **quotes**.
3. **string_element**::= char~**special_character** | backslash **special_character** | **control_char**.
4. **special_character**::= backslash | quotes.
5. **control_character**::= backslash ("n" | "t"), representing a newline and tab respectively.

Variables

6. **variable**::= letter #(letter | digit).

Formal Semantics

Here is a first rough draft



[**oj.jpg**] (Full scale)

To Be Done : correct, tidy, and complete.

. . . . . (end of section **The Programming Language OJ**) <<Contents | End>>

End