PARROT VIRTUAL MACHINE AS A PEDAGOGIC TOOL

C. Aperghis-Tramoni, J. Guizol

Universite de la Mediterranee

For about 20 years now, Perl programming language did develop itself up to the actual release which is 5.

Former releases were :

Perl 1 : January 1988

Perl 2 : June 1988

Perl 3 : October 1989

Perl 4 : In 992

Perl 5 : Since August 1997

Lets say first some words on how releases are numbered.

Three elements have to be considered : Revision. Version. Subversion.

It means that, if we consider Perl 5.6.1, Revision is 6, Version is 5 and subversion is 1.

Even versions are maintenance one (5.6.x)

Odd versions are versions for development (5.7.x)

Before reaching 5.6.0, subversions _01 à _49 will have been reserved for debugging.

Subversions _50 à _99 are considered as unstable versions.

Perl contains a magical variable ($]) that contains the number of the release. This coding uses the composite scheme of numbering in floating point mode.

Value is computed as : Revision + Version/1000 + Subversion/10000

Up to now, all these evolutions have never been revolutions, language did remains an interpreted one, only release 5 did propose objects as a new feature.

On contrary, the last one, number 6, will bring a radical change in Perl design, it will no more be interpreted but compiled in a byte codes that will be carried out on a virtual machine, the parrot machine..

The simplicity of this machine, the power of its assembly language did decide us to use it as a support for assembly language courses.

Teaching "machine language" is, unfortunately, closely dependent from the platform (namely the processor on which it will be based). In addition an assembler (or a macro assembler) reflects quasi perfectly the machine for which it is destined to generate the code.

For us, it is a constraint, since great number of basic concepts must precede such a course to present the machine, its structure and all the constraints related to its design, before being able to write the first lines of an assembly program.

And worse, we must deal with the fact that writing of assembly language programs is never obvious.

Lets take for example the program any beginner will try to realize for his first experiment,

The famous and well known one that prints “Hello World.”. Realized on a PC platform, four instructions are used to fulfill the desired function, whereas 22 additional one are necessary to prepare the environment

We so lose ourselves among a great number of principles to initially forget the fundamentals of what it is judicious to teach, an approach of a given programming, use of elementary operations chained together to carry out a more complex functions, great rigor in the reasoning, judicious distribution of the different components of the program, split a problem in functions and a strict definition of interfaces between the various modules.

A virtual machine escapes itself from all these problems since, unlike a real platform confronted with constraints of the reality, it can take freedom regarding number and type of its registers and considering the complexity of basic instructions its users will dispose.

Thus parrot virtual machine is built around four sets of 32 registers, each unit being intended to treat a type of specific data (integer, string, floating numbers, PMC)

Since its instructions reproduce main functions of Perl 5 programming language, it allows us to build narrow connection between lesson between the assembly course and the Perl module which takes place in parallel.

The algorithms students did formerly write within Perl course will be used as basis for new realization in parrot machine language.

For instance, lets see how a program that prints “Hello World.” Will be written in assembly language on the parrot virtual machine. It will look like :

```
set     S1, "Hello World.\n"  # Load the string in the string register number 1.
print   S1                    # Print the content of the string register number 1.
```

It means, only the minimum instructions necessary to describe what we want to do.

Parrot is a “register oriented machine”. This implies that realization of a program must be as optimal as possible in term of register resources. Their number and diversity provide great facilities to use them, and since conversion of type is automatic when transferring data from a register of one set to register of a different set, users may concentrate themselves on realizing their algorithm, disregarding all environment it would be essential to manage on a real machine.

Parrot machine does include in its definition several stacks mechanisms. One of these stacks belongs to the user and can be used to store, whenever he wills, the content of any of his registers. The notion of private (or local) variable, as it exists in some high level languages, is easy to put evidence.

Let us see how can be described, in perl for instance, the traditional program that computes recursively the factorial on an integer number.

```
sub fact {                          # Declaring the name of the function.
  my ($y) = $_[0];                  # Declaring a private variable $y.
  return ($y == 1)?1:$y * fact($y - 1);# Effective calculus of the factorial.
}                                   # End of the function.
print "Give me a number : ";
$v = <STDIN>;                               # Getting the value.
$r = fact ($v);                     # Calling the function.
print "Result : $r.\n";             # Printing the result.
```

The same thing can be written in parrot assembly language as :

```
        getstdin      P0
        getstdout     P1
        print         P1, " Give me a number ?:\n"
        readline      S0, P0                # Getting the value.
        set           I0, S0                # Register I0 is used to transmit value to function.
        bsr            _fact                # Calling the function.
        print                 P1, "Result : "
        print                 P1, I1
        print         P1, "\n"
        end

_fact:                                      # Declaring the name of the function.
        gt            I0,1, CALC            # Checking end of the calculus.
        set           I1, 1                 # Result is transmitted using I1.
        ret
CALC:
        save          I0                    # Stack is used to set data in I0 private .
        dec           I0
        bsr           _fact                 # Recursive call of the function
        restore       I0                    # Getting back private value
        mul           I1, I1, I0                    # Effective calculus of factorial.
        ret
```

These some lines show how easy it is to use parrot assembly language, and point out the fact we must be as carefull as possible on constraints of a such assembly programming.

On another point of view, this virtual machine offers a nice particularity we can note in the formerly presented program, a special feature called PMC (Parrot Magic Cookies). These are structures that will be used to represent high level data structures as lists, hashes, scalars or objects.

PMC's are also used as file managers to define descriptors as we did see in our factorial program.

```
getstdin     P0              # Gets the definition of <STDIN> in register PMC0.
getstdout    P1              # Gets the definition of <STDOUT> in register PMC1.
print        P1, "mess :\n"  # Writes in the device described in PMC0.
readline     S0, P0          # Reads on the device described in PMC1.
```

This facility allows us, as we can do it in Perl, t redirect without any problem standards descriptors (<STDIN>, <STDOUT> and <STDERR>) on any file only by changing the description of the related PMC.

```
getstdin     P0, "data.txt"  , "<"     # Defining an input file.
getstdout    P1, "file.txt"  , ">"     # Defining an output file.
print        P1, "Hello World :\n"     # Writing in the output file.
readline     S0, P0                    # Lreading from the input file.
```

In conclusion, such a machine is perfectly adapted to machine language learning, it makes possible to concentrate all of the teaching on the assembly characteristics disregarding all that connects it to the reality of an existing machine. Variety of instructions, their power and the fact that they quite all come from Perl 5 functions give all our students the possibility to realize again, easily, in assembly, the programs, often complex, they already wrote and checked in Perl.

REFERENCES :

The parrot reference site. http://www.parrotcode.org/
The perl reference site. http://www.perl.org/
The CPAN organization. http://www.cpan.org/
The mongers organization. http://www.pm.org/
The French community of Perl. http://www.mongueurs.net/