Suggestions for an educational programming language PL/2, Alexey Podorov, 2022

Hundreds of languages are offered for teaching programming, including control of graphic elements. Therefore, the proposal of a new language for teaching requires additional justification.

I will try to describe the advantages of the proposed programming language (working title PL/2) and game programming (working title PL/2 script), and I suggest finding flaws in the approach.

The main paradigms are assumed to be the following:
Assembly code that allows you to control the processor
Logical programming, represented by Prolog, SQL languages
Functional programming
Imperative programming described by algorithms
Object programming, consisting in the transmission and processing of signals

Currently, these techniques are combined in extensions of existing programming languages. An approach is proposed to combine paradigms in a single syntax that resolves conflicts between different approaches in advance, which will reduce the time spent on learning different syntaxes.

So, the key elements of an educational programming language.

Source text

The source text file consists of arbitrary text (comments, description of the algorithm), which can include [named] fragments in various programming languages — PL/2, SQL, HTML, text, PL/2 script.

the beginning of the code insertion is indicated by the **source** keyword in the first position of the line

the end of the insertion is indicated by the sequence **/source** in the first position of the line

The PL/2 syntax includes uniform indentation requirements, in particular, the program text is shifted at least 4 positions from the beginning of the line.

Structure of the program text

The size of the shift between the nesting levels is 4 spaces. The continuation of a statement on the next line is shifted by at least 3 spaces relative to the statement level

The main level of the program is shifted by 4 spaces relative to the **source** markup character of the source file, respectively, no program character can be indented less than 4 spaces

Syntax

For different types of blocks, different types of brackets are assumed:

parentheses (...) used to describe and pass parameters when calling methods

square brackets [...] are used to describe the dimension of arrays and specify the index when accessing an array element.

curly brackets {...} are used to indicate various elements of the program structure. To localize errors, curly brackets are provided with additional characters.

{-... -} — analog of parentheses — used when describing the body of a relational class

{#... #} — used when describing the body of a function block

{$...$} — external framing brackets when describing a constant of a structural class

{*... *} — description of the class body or interface or unit

{&... &} — description of the method body

{@... @} — namespace body

{|... |} — blocks executed in parallel

{%... %} — managed block

{=... =} — responsibility class

{... } — internal block in the method body

Operators

== equality comparison
=/= comparison on inequality

:= assigning a value to a variable
++ prefix increment
-- prefix decrement

+ addition
- subtraction
* multiplication
/ division
-/- rational division (highlighted only for clarity of formulas, including in the IDE)

(? boolean ? value_true :: value_false ?) - ternary operator

Imperative programming

The following syntax for structural programming is proposed:

The operators enclosed between brackets {& and &} are executed sequentially in the order specified in the program text

Loop and branch operators are used to change the execution order

execution when condition is met

```
if (condition){
   ...
}
```

execution of the block when the first of the conditions is met and the else block when none of the conditions are met

```
decision{
   if (<condition1>)
      ...
   if (<condition2>)
      ...
   else
     ...
}
```

switch by numerical value with indication of a constant or enumeration member

```
switch (<expression>){
   case <value1>:
      ...
   case <value2>:
      ...
   else
      ...
}
```

The ternary selection operator can also be used in expressions

Loop statement

```
loop [explicit] [label]
{
   for <[type] variable [ := initializer]>;
   from <container>;
   while <condition under which the loop continues iterating>;
   next <iteration operator>;
   where <loop body execution condition>
   ...
   continue;
   ...
   break [block label];
   ...
}
```

Here:
loop - loop start operator
explicit - the loop is executed at least once, the loop exit condition is not checked before the first pass

for - list of loop variables and their initialization
while - the condition under which the loop continues to iterate
next - commands to be executed after iteration
from - the container from which elements are selected when passing through the loop
where - condition for executing the loop body
continue - operator of early transition to the next iteration
break is an operator for early exit from the current or specified loop

Error Handling

transaction and site management with the possibility of failure

```
transaction{
   finalize
   {
   }
   catch
   {
   }
   {
      throw // execute catch and finalize
      revoke // undo all changes to the program state, clear the queue of outgoing signals and execute finalize code
    }
}
```

Object-oriented programming

new — creating a new object
drop — deleting an object

working with collections

select ::= - getting information about the named objects of the collection
insert += - inserting a new object into the collection
delete -= - deleting an object from the collection
update - multiple updates in the collection

managing the rights of an object's actions over objects of the specified class or its properties/methods

grant += - assign object permissions to a class or property
deny += - assign deny rights to an object on a class or property
revoke - remove the rights of an object on a class

Class types

Classes are divided into the following types:

domain class — elementary runtime class

relation class - a relationship between the specified objects (similar to Prolog sentences, a list of function parameters), there are no methods and properties, access occurs only to public members. Similar in capability to a set returned from a relational database

structure class - a structure with public elements. The word this is used to access an element of the current object. May contain methods for processing.

system class - a system that can contain private and protected inherited members.

process class - a process to which you can pass initialization values and get the resulting values. Corresponds to the service. Any method receives not only a pointer to itself (this), but also a pointer to the calling class or object (sender). This allows you to check the authorization of the object accessing the method. Contains standard methods.

Non-imperative programming

operators for working with lambda expressions and lists

::= — definition of expression disclosure
|> — pipeline for all items from the left collection, specified by the right expression

quantifiers of existence and universality on ordered lists

&> — a collection of all elements from the left collection satisfying the right expression
!> — pipeline for all items from the left collection that do not satisfy the specified right expression
%> — collection of the first element from the left collection that satisfies the right expression (arithmetically, the remainder of the division)
~> — all elements of the list, except the first, satisfying the right condition
$> — a collection of all elements from the left list, except the last one satisfying the specified right expression
?> — collection of the last element from the left list that satisfies the specified right expression

for example,

head ::= collection %> (\true),
tail ::= collection ~> (\true)

Assembler

The processor class is used, the arguments of the methods of which are the arguments of assembler commands, and the properties are the registers of the processor core.

This approach makes it possible to use inheritance as a transition to a new generation of the processor line, as well as to use full command names instead of short mnemonics and the possibility of an equal description in the help systems.

Comments

// inline comment
/* … */ - block comment
/% … %/ - identifier comment

Hallo, world

hallo.pl2
When using **program**, the code is placed in the default.Default.Main() method

```
source mode ="pl2" name="example2"
   using user.OS.IO; // using OS Console interface

   program (String arguments[] /% execution time command line parameters%/ )
   {&
      Console.write_line("Hallo, world!");
   &}
/source

*** hallo.pl2 ***
```

PL/2 script

The PL/2 script language provides the ability to describe scripts and extend classes while describing a variable for game programming.

Instead of a functional variable, it is proposed to use a string that (as in Excel) begins with an equal sign.

At the same time, it is convenient to store the description of classes and objects in XML, and use the code for active processing.

Class Description:

```
<?xml version="1.0" encoding="windows-1251"?>
<pl2script>
    <type id="type.class.Scene" save="true">
      <help id="class.Scene.help">pl2script/game/scene.htm</help>
      <system>
        <class id="class.Scene" domain="type" save="true">
          <name>
            <ascii>Scene</ascii>
            <local>Stage</local>
          </name>
          <tip id="class.Scene.tip">
            <ascii>Game class as a main class for a game</ascii>
            <local>Class for describing the context of the game</local>
          </type>
        </class>
        <property id="class.Scene..." domain="string" save="true">
          <name>
            <ascii>prop ...</ascii>
            <local>prop ...</local>
          </name>
          <tip id="class.Scene..tip">
            <ascii>tip ...</ascii>
            <local>Message when ...</local>
          </type>
        </property>
```

```
        </system>
      </type>
    </pl2script>
```

Description of the object:

```
    <?xml version="1.0" encoding="windows-1251"?>
    <pl2script>
      <scene id="game.">
        <name>Noosphere</name>
        <description>Big world</description>
        <message>You have entered the big world</message>
        <way id="game.universe" path="Universe/index" enabled="true">Universe</way>
        <way id="game.solar" path="Solar/index" enabled="true">Solar System</way>
        <way id="game.earth" path="Earth/index" enabled="true">Earth</way>
        <way id="game.country" path="country/index" enabled="true">By Country</way>
        <way id="game.body" path="body/index" enabled="true">To yourself</way>
        <way id="game.nature" path="nature/index" enabled="true">Nature</way>
        <way id="game.school" path="school/index" enabled="true">To School</way>
        <way id="game.plant" path="plant/index" enabled="true">To Plant</way>
        <way id="game.university" path="university/index" enabled="true">To
university</way>
      </scene>
    </pl2script>
```

I ask you to speak about the problems possible when using this syntax

Original text: https://habr.com/ru/post/219419/
Discussions: https://pl2-en.livejournal.com
Sources:
https://github.com/palexisru/pl2_rus,
https://github.com/palexisru/pl2_international,
https://github.com/palexisru/HTML.NET.Table