UDC 004:37

PECULIARITIES OF CREATING PROGRAMMING ENVIRONMENT WITH UNIVERSAL SUPPORT OF INTERPRETERS OF PROGRAMMING LANGUAGES SUBSETS

Lavrik A.

Kherson State University

The article considers principles of extraction subsets of programming languages, sufficient for performing computational experiments in the certain class of problems on example of sorting algorithms.

Keywords: *computer science, programming, sorting algorithms, videointerpreter.*

INTRODUCTION

Within scientific research on using educational software in the pedagogical and methodical process in Kherson State university a new version of educational programming environment for visual demonstration algorithms of sorting and searching in arrays and comparing of their efficiency, also known as "Videointerpreter", is being developed.

The environment has complex structure and includes the following components:

- parser and interpreter of high-level programming language subsets (Pascal, C or Java, further called target languages);
- sorting and searching procedures visualizer;
- smart program code editor;
- results statistics processing unit.

This article will consider principles of creating high-level languages parser and interpreter. The target languages will be compared for extracting their most common features, which can provide ability of sorting and searching algorithms implementing. Syntax and semantics of target programming languages and of tools for their describing will be the object of research.

Requirements for programming languages subsets implementation

Sorting algorithms are based on comparing and swapping operations. Namely their quantity depending on sorted elements' count defines the algorithm's efficiency — it's complexity. Simple algorithms, in particular bubble and insert sorting, have $O(n^2)$ complexity in the worst case, and may have $O(n \cdot \log n)$ in general case; effective ones, such as quick sorting, have complexity $O(n \cdot \log n)$ in the worst case. But this doesn't exclude possibilities of outstripping effective sorting algorithms by simple ones on some data sets, for example on short sequences. Such cases can become objects of research, in particular for what "Videointerpreter"system is being developed.

Let's generalize requirements for target programming languages implementation. Extracted subsets should provide sorting algorithms describing. The following language's properties and elements will be required for that:

1. procedural programming paradigm — Pascal and C are procedural languages, Java can imitate procedurability with a help of classes' static methods;
2. standard integral data types (for using as array indeces and loop parameters) and boolean type (for using in conditional statements) and operations with them — present in all target languages:

Pascal: integer, boolean;
C, Java: int, bool[1];

3. additional abstract data type *data* and comparing operations for it — it's missing in

1 Though in C usually integral values are used as boolean ones (0 — false, other numbers — true), for the C99 standard bool type with vlues true and false also presents;

target languages and is the environment innovation;

4. constants declaration:

Pascal: **const** name *[* : type *]* = value ;
C: **const** type name = value ;
Java: **final** type name = value ;

5. variables declaration:

Pascal: **var** name1 *[*, name2*…]* : type;
C, Java; тип name1 *[*= value*][*, name2 *[*= value] *...]*;

6. sequences (one-dimensional arrays) declaration:

Pascal: **var** name1 *[*, name2 *...]* : **array** [lower..upper] **of** type;
C: type name[length] *[*= { *[*value *[, ...]] }][, ...]* ;
Java: type[] name *[*, *...]*;

7. subroutines declaration:

Pascal: **procedure** name *[* (*[***var***]* name *[*, *…]* : type *[*; ... *]*) *]* ;
function name *[* (*[***var***]* name *[*, *…]* : type *[*; ... *]*) *]* : type;
C: *{***void**|type*}* name (type *[&]*name *[*= value*] [*, ...*]*
Java: *{***void**|type*}* name (*[* type name *[*, ...*]]*);

8. algebraic expression notation is present in all languages. In different languages operations can have different priority (for example comparing operations are performed before logical ones in Pascal, and after them in C and Java) and designation, difference in which is submitted in the table:

Operation	Pascal	C and Java
Integer division	**div**	/*
Remainder of division	**mod**	%
Logical operations “and”, “or”, “exclusive or”, “not”	**and**, **or**, **xor**, **not**	&&, ||, ^, !
Bitwise operations “and”, “or”, “exclusive or”, “not”	**and**, **or**, **xor**, **not**	&, |, ^, ~
Bitwise left or right shift operations	**shl**, **shr**	<<, >>
Operations “equal” and “inequal”	= , <>	==, !=
Conditional choice operation	missing	? :
Type casting operations	missing	(type name)

9. value assignment — main operation of any imperative languages:

Pascal: name := expression;
C, Java: name *[*operator*]* = expression;

where operator — + , – , or another operator designation, expression — algebraic expression (in C and Java another assignment statement may perform as expression);

10. comparing operations < , > , ≤ , ≥ , = , ≠ , both for type *Data*, and integral types:

Pascal: < , > , <= , >= , = , <>;
C, Java: < , > , <= , >= , == , !=

11. conditional branching statement:

Pascal: **if** expression **then** … *[* **else** ... *]*;
C, Java: **if** (expression) … *[* **else** ... *]*;

12. loop organizing statement:

- with precondition:

Pascal: **while** expression **do** ... ;
C, Java: **while** (expression) ... ;

- with postcondition:

* If integral values are used as operands.

Pascal: **repeat** ... **until** expression;
C, Java: **do** { … } **while** (expression);

- with parameters:

Pascal: **for** name := start *{***to**|**downto***}* finish **do** …;
C, Java: **for** (
*[{*declaration|assignement*}]*;
*[*condition_expression*]*;
*[*modifying_expression*]*
)
…;

13. subroutine call with parameters, including recursive (for using in quick sorting methods):

Pascal: subroutine_name *[*(parameter1*[*, ...*])]*;
C, Java: subroutine_name (*[*parameter1*[*, ...*]]*);

14. return from subroutine statement:

Pascal: **exit**;
C, Java: **return** [expression];

15. comments:

Pascal: { multi-line }
C, Java: // single-line
/* multi-line */

Software development

As a tool for implementing the educational programming environment for visual demonstration creating project we chose Mono — cross-platform Microsoft .NET and C# language implementation. It was preferred before other tools for the following reasons:

- rich and expressive C# language features;
- wide set of classes in Mono/.NET libraries;
- ability to use Moonlight tool — Microsoft Silverlight analogue for animated objects creating;
- convenient integrated development environments Visual Studio and Mono Develop;
- team members' experience of work in Microsoft .NET.

For successful completing of works' part on target languages subsets parser and interpreter development it's ought to use already existing solutions, namely parser creating tools. Among the most popular tools ANTLR, lex + yacc, bison and so on can be marked. Each solution has it's advantages and disadvantages. But it was decided, that from them all namely ANTLR mostly satisfies our needs, because it:

- has convenient and understandable syntax for describing parsing rules, which is send to the program input;
- on the output it generates classes for lexical parsing an AST-tree constructing and walking through on high-level language, role of which can play Java, C++ and, what's the most important, C# (most of other instruments generate code on C/C++/Java);
- has special development and debug environment ANTLRWorks — multi-windowed editor, which supports rules syntax highlighting, autocompletion, visual displaying of grammacy, which is built in real time during text input, debugger, refactoring tools etc.

Language elements extracting

Syntax elements of describing languages can be divided into groups:

- insignificant syntax elements — that is comments and whitespace;
- declarative — which declare program objects;
- executable — statements, that will be executed during the program running. In the target languages they are placed in functions', procedures' and main program's (in Pascal) bodies.

Within the problems set before us, declarations of the following objects may be included into the declarative group: variables, constants, arrays, subroutines. Declarations of variables and constants may be nested into subroutines. Also the declaration of header object — **program** in Pascal and the main static class in Java – may be classified as declarative object.

The following kinds of statements will belong to executable:

- expressions (as part of all other types of instructions);
- assignment;
- branching;
- loops;
- loop continuation;
- loop interrupting;
- subroutine calls;
- returning from subroutine.

So the next table for program elements classes appears:

Class	Derived	Description
ProgramElement		Base class for all program elements
Insignificant	ProgramElement	Base class for all insignificant program elements
Whitespace	Insignificant	Whitespace
Comment	Insignificant	Comments
TypedElement	ProgramElement	Base class for typed program elements
Expression	TypedElement	Expression
Constant	TypedElement	Constant
Variable	Constant	Variable
Subroutine	TypedElement	Subroutine
Statement	ProgramElement	Statement
Assignement	Statement	Assignment value to a variable
Branching	Statement	Branching
Loop	Statement	Loop
LoopContinue	Statement	Loop continuation
LoopExit	Statement	Loop interrupting
SubroutineCall	Statement	Call of subroutine
SubroutineExit	Statement	Returning from subroutine

Now implementation of target languages subsets consists of describing above-stated language elements in ANTLR syntax. According to this description a set of classes on the selected language (C# in our case) for lexical parsing source code on Pascal language and algorithm tree constructing is generated. After analogical describing of C and Java language grammacy we'll get an ability to create trees with same structure from the source codes of different languages.

For tree walking a special class Walker is used. In environment which is developed it will provide interaction and data exchange between interpreter and other program components on the base of events subscribing and handling mechanism. For example calculation of exchanges quantity in statistics processing unit can be implemented by subscribing on the event Swap, animation of comparings — subscribing on the event Comparing.

From the above-stated the conclusion can be made, that chosen way of extracted programming languages subsets implementation provides uniformed processing of program structural trees, and the most rational and flexible architectural construction of the project. In perspective it will give an ability to extend supported languages subsets and include new imperative languages to their number in way of describing their syntax by the shown scheme. To it's users multilingual environment also gives an ability as visually compare and learn rapidly syntax and semantics of different programming languages, so develop skills of algorithmic thinking, not fixed on the features of concrete language.

REFERENCES

1. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка. – М.: Финансы и статистика, 1982. – с. 250.
2. Б.В. Керниган, Д.М. Ричи, А.Фьюер. Язык программирования Си. Задачи по языку Си. М.: Финансы и статистика, 1992. – с. 320.
3. Мейнджер Джейсон. Java: Основы программирования / Пер. с англ. С.Бойко под ред. Я.Шмидского. – К.: BNV, 1997. – с. 460.
4. Роберт Седжвик. Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск – СПб.: ДиаСофтЮП, 2003. – с. 680.
5. Terrence Parr. The Definitive ANTLR Reference: Building Domain-Specific Languages — Oxford Associated Press, 2008. – с. 740