INFORMATICA, 1995, Vol. 6, No. 2, 233–243

# THE COMPARISON OF STANDARD PASCAL AND TURBO PASCAL 7.0

Vladas TUMASONIS

Department of Computer Science, Vilnius University
2006 Vilnius, Naugarduko St. 24, Lithuania

**Abstract.** The newest version of Turbo Pascal – Turbo Pascal 7.0 – is concentrately described in comparing with optimal language for programming teaching – Standard Pascal. Data types, control structures, procedures and functions, parameters, new directions of development are classified and discussed.

**Key words:** programming languages, Pascal, Turbo Pascal, data types, control structures, procedures and functions, units, objects.

**1. Introduction.** The computer programming language Pascal was designed by Niklaus Wirth to satisfy two principal aims:

a) to make available a language suitable for teaching programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language;

b) to define a language whose implementations could be both reliable and efficient on then available computers.

However, it has become apparent that Pascal has attributes that go far beyond these original goals. It is now being increasingly used commercially in the writing of both system and application software.

In 1977 a working group was formed within the British Standard Institution (BSI) to produce a standard for Pascal. In 1982 the standard for programming language Pascal (Standard Pascal) was accepted. This standard helps to promote the portability of Pascal programs between data processing systems and to compare different versions and implementations of Pascal.

Pascal version for IBM PC (Turbo Pascal) was designed and implemented by Borland International. The newest version Turbo Pascal 7.0 has appeared in 1992. In this paper systematic comparison of Standard Pascal and Turbo Pascal 7.0 (data types, control structures, parameters and so on) is described.

**2. Data types.** There are five predefined integer types in Turbo Pascal 7.0: *Integer*, *Shortint*, *Byte*, *Word*, and *Longint*. Each type denotes a specific subset of the whole numbers (represented in 8, 16 and 32 bits). The type of an integer constant is the predefined integer type with the smallest range that includes the value of the integer constant. For a binary operator, both operands are converted to their common type before the operation. The common type is the predefined integer type with the smallest range that includes all possible values of both types.

There are five kinds of predefined real type: *Real*, *Single*, *Double*, *Extended*, and *Comp*. The real types differ in the range and precision they hold ( 4, 6, 8 and 10 bytes).

There are four predefined boolean types. *Boolean* and *Bytebool* variables occupy one byte, a *Wordbool* occupies two bytes, a *Longbool* ocupies four bytes. *Boolean* is the preferred type and uses the least memory. *Bytebool*, *Wordbool* and *Longbool* exist primaly to provide compatibility with other languages and the Windows environment.

In Turbo Pascal it is possible to declare untyped files – file types without component types:

**var** *f*: **file;**

Why do we need untyped files when we don't know the structure of their components (the logical structure of the file) and we can't read, write and process in usual way? Untyped files are low level input/output channels primarily used for direct access to the disk regardless of component type and structuring. The portion of read/write is a physical record (block). Instead of standard procedures *Read* and *Write*, two procedures *BlockRead* and *BlockWrite* are used for high speed data transfers.

In addition to usual pointer types Turbo Pascal allows untyped pointers. The predefined type *Pointer* denotes un untyped pointer; that is a pointer that doesn't point to any specific type. Variables of type *Pointer* can't by dereferenced. Generic pointers, however, may be typecast to allow dereferencing. Like the value denoting by the word **nil**, values of type *Pointer* are compatible with all other pointer types.

There are three new data types: string type, object type and procedural type.

The newest ideas of object-oriented languages are reflected and imple-

mented in Turbo Pascal through object type. An object type is a structure consisting of a fixed number of components. Each component is either a field, which contains data of a particular type, or a method, which performs an operation on the object. Similar to a variable declaration, the declaration of a field specifies data type and an identifier that names the field. Similar to a procedure or function declaration, the declaration of a method specifies a procedure, function, constructor or destructor heading.

Contrary to other types, an object type can be declared only in a type declaration part in the outermost scope of a program or unit. Therefore, an object type can't be declared in a variable declaration part or within a procedure, function, or method block. The component type of a file can't be an object type, or any structured type with an object-typed component.

Standard Pascal regards procedures and functions as program parts that can be executed through procedures and functions calls. Turbo Pascal allows procedures and functions to be treated as entities that can be assigned to variables and passed as parameters. Such actions are made possible through procedural types. A procedural type declaration specifies the parameters and, for a function, the result type.

All data types are presented in the Table 1.

**3. Strings.** Only static strings there are in Standard Pascal (declared as **packed array** $[1 \ldots n]$ **of** *Char*, $n \geqslant 1$). A character string of $n$ characters may be assigned to the variable of such type.

In Turbo Pascal dynamic strings (that is of variable length) are implemented – **string** strings. A **string** type value is a sequence of characters with a dynamic length attribute (depending on the actual character count during program execution) and a constant size attribute from 1 to 255.

Turbo Pascal supports a class of character strings called null-terminated strings. Such string has no length byte; instead it consist of a sequence of non-null characters followed by NULL (#00) character. The upper limit length is 65 535 characters. Null-terminated strings are stored as an array of the form

**array** $[0 \ldots n]$ **of** *Char*;

Turbo Pascal has a predefined type, *Pchar*, to represent a pointer to a null-terminated string:

**type** $Pchar = \hat{}Char$;

There are no built routines specifically for null-terminated string handling.

**Table 1.** Classification of data types

| Data type | | Pascal | Turbo Pascal 7.0 |
|---|---|---|---|
| Simple type | *Integer* | + | + |
| | *Shortint* | − | + |
| | *Byte* | − | + |
| | *Word* | − | + |
| | *Longint* | − | + |
| | *Real* | + | + |
| | *Single* | − | + |
| | *Double* | − | + |
| | *Extended* | − | + |
| | *Comp* | − | + |
| | *Boolean* | + | + |
| | *Bytebool* | − | + |
| | *Wordbool* | − | + |
| | *Longbool* | − | + |
| | *Char* | + | + |
| | subrange type | + | + |
| | enumerated type | + | + |
| Structured type | array type | + | + |
| | record type | + | + |
| | set type | + | + |
| | string type | − | + |
| | file type | + | +* |
| | object type | − | + |
| pointer type | | + | +** |
| procedural type | | − | + |

* Also untyped files
** Also untyped pointers

Instead such functions are in the *Strings* unit. You can perform main operations with strings: to assign (copy), connect, compare, search and so on.

All types of strings are presented in the Table 2.

**Table 2.** Classification of strings

| The kind of string | Pascal | Turbo Pascal 7.0 |
|---|---|---|
| Simple strings (type **packed array** $[1 \ldots n]$ **of** *Char*) | + | + |
| **string** strings (type **string**) | − | + |
| Null-terminated strings (type *Pchar* = ^*Char*) | − | + |

**4. Control structures.** Traditionally used control structures (**if**, **case**, **while**, **repeat**, **for**, **goto** statements in Standard Pascal) in Turbo Pascal 7.0 are extended by flow control procedures *Exit*, *Halt*, *Break* and *Continue*.

Exit exits immediately from the current block. Executed in a subroutine (procedure or function), *Exit* causes the subroutine to return. Executed in the statemant part of a program, *Exit* causes the program to terminate. A call to *Exit* is analogous to a **goto** statement addressing a label just before the end of a block. *Halt* procedure stops program execution and returns to the operating system. Note that *Halt initiates* execution of any *Exit* procedures.

*Break* and *Continue* are used within a **for**, **repeat**, or **while** statements. *Break* procedure terminates the innermost enclosing **for**, **repeat**, or **while** statement. *Break* is analoguos to a **goto** statemant addressing a label just after the end of the innermost enclosing repetitive statemant. *Continue* procedure causes the innermost enclosing **for**, **repeat**, or **while** statement to immediately proceed with the next iteration.

The Turbo Pascal compiler will report an error if a call to *Break* or *Continue* is not enclosed by a **for**, **repeat**, or **while** statement.

All types of control structures are presented in the Table 3.

**5. Procedure and function parameters.** Constant parameter (it is declared with reserved word **const**, for example, **function** *f*(**const** *i*: *Integer*): *Integer*) is local variable which value can't be changed during the execution of

**Table 3.** Classification of control structures

| Control structures | Pascal | Turbo Pascal 7.0 |
|---|---|---|
| **If** statement | + | + |
| **While** statement | + | + |
| **Repeat** statement | + | + |
| **For** statement | + | + |
| **Case** statement | + | + |
| **Goto** statement | + | + |
| *Exit* procedure | − | + |
| *Halt* procedure | − | + |
| *Break* procedure | − | + |
| *Continue* procedure | − | + |

a procedure or function. Thus the behaviour of constant parameter is similar to value parameter. But assignements to a formal constant parameter aren't allowed. Thus the compiler can generate more efficient code when constant parameters are used instead value parameters.

It is possible to use formal parameters without its types – so called untyped parameters (parameter $x$ in declaration procedure $p$(**var** $x$; $c$ : *Char*)). The corresponding actual parameters can be any variable, regardless of its type. Within the procedure ar function, the untyped parameter is typeless; that is incompatible with variables of all other types. It means that it is impossible to use such parameter. This a little strange situation is corrected: the variable can get a specific type through variable typecast.

Another new sort of formal parameters in Turbo Pascal 7.0 – open-array parameters. They are, in some sense, similar to Standard Pascal conformant array parameters. Open-array parameters are declared using reserved words **array**, **of** and the type of array elements (**function** $S(m$ : **array of** *Real*): real;). Actual parameter must be an array of any index range variable whose elements type is real. Within the procedure or function the formal parameter behaves as if it was declared as

**array**$[0..n-1] =$ **of** *Real*,

where $n$ is the number of elements in the actual parameter. The index range of the actual parameter is mapped onto the integers 0 to $n-1$. The *High* standard

function returns the index of the last element in the actual parameter. Thus if we have the function call $S(a)$ and the actual array

**var** $a$ : **array** $[p..q]$ **of** *Real*;

corresponds to formal open array $m$, then $m[i]$ means $a[p+i]$, $i = 0, 1, \ldots,$ $high(m)$; $high(m) = p - q$.

The same situation can be modelled in Standard Pascal. Function heading

**function** $S(m$ : **array** $[u..v$: *Integer*] of *Real*): *Real*;

has two extra parameters $u$ and $v$. After the call $S(a)$ these parameters have the value of $p$ and $q$. Thus the formal conformant array parameter $m$ corresponds to the actual array $a$ and $m[i]$ means $a[i]$, $i = u, u+1, \ldots, v$.

Open-string parameters are declared with standard word (not reserved!) *OpenString* (**procedure** $k(var\ s$: *Openstring*)). The actual parameter can be a variable af any string type.

Notice, that there not procedural and functional parameters in Turbo Pascal 7.0. But such parameters could be expressed by procedural type variables in following way.

In Standard Pascal:

```
procedure P(n: Integer;
            function f(i: Integer): Integer;
            var m: Integer);
function ff1(n: Integer): Integer;
```

The call to the procedure $P(5,\ ff1,\ s)$.

In Turbo Pascal 7.0:

```
type Func = function(i: Integer): Integer;
var fvar: Func;
function ff1(n: Integer): Integer; far;
procedure P(n: Integer;
              f: Func;
            var m: Integer);
```

The call to the procedure $fvar := ff1;$

$$P(5,\ fvar,\ s)$$

or $P(5,\ ff1,\ s)$.

All types of parameters are presented in the Table 4.

**Table 4.** Classification of parameters

| Kind of parameters | Pascal | Turbo Pascal 7.0 |
|---|---|---|
| Value parameters | + | + |
| Variable parameters | + | + |
| Procedural parameters | + | − |
| Functional parameters | + | − |
| Conformant array parameters | + | − |
| Constant parameters | − | + |
| Untyped parameters | − | + |
| Open-array parameters | − | + |
| Open string parameters | − | + |

**6. Procedures and functions.** Turbo Pascal supports two procedure and function call models: near (short two bytes address is used) and far (long four bytes address is used). The near call model is more efficient, but near procedures and functions can only be called from within the module they are declared in. The compiler automatically selects the correct call model based on a procedure and function declaration.

The Turbo Pascal run-time library and the code generated by the compiler are fully interruptible. Also, most of the run-time library is reentrant, which allows to write personal interrupt service routines in Turbo Pascal – **interrupt** procedures.

**External** declarations let to interface with separately compiled procedures and functions.

With **assembler** declaration it is possible to write procedures and functions in assembly language. The **inline** directive enables generate machine code instruction instead of procedure or function call.

All types of declarations are presented in the Table 5.

**7. Standard procedures and functions.** First of all we should like to notice the different status of standard procedures and functions in Standard Pascal and Turbo Pascal. In Standard Pascal they are the part of language. There is no need to declare them. You can imagine that they are declared in the imaginary block surrounding the program. It is possible to redefine the same

**Table 5.** Classification of declarations

| Kind of declaration | Pascal | Turbo Pascal 7.0 |
|---|---|---|
| Procedure and function | + | + |
| **Forward** | + | + |
| **Far** | − | + |
| **Near** | − | + |
| **External** | − | + |
| **Assembler** | − | + |
| **Inline** | − | + |
| **Interrupt** | − | + |

identifier within the program. The user of Standard Pascal only uses standard procedures and functions. There is no need for him to know where are these procedures and functions, how to search them and so on. In fact we don't care about of implementation of **while** statement, for example.

The situation in Turbo Pascal is quite different. A lot of usefull procedures and functions are in different a priori created units so called standard units. All these units are in the run-time library. Standard procedurs and functions of Standard Pascal are in the *System* unit (read and write procedures, arithmetic functions, floating point and string handling and so on).

The *Dos* and *WinDos* units implement a number of very usefull operating system and file handling routines. None of the routines in these units are defined by Standard Pascal, so they have been placed in their own modules.

The *Crt* unit permits to write programs that send their screen directly to the BIOS or to video memory. The result is increased speed and flexibility.

The *Printer* unit lets send standard Pascal output to printer using *Write* and *Writeln*.

The *Overlay* unit enables to reduce program's total run-time memory requirements. In fact, you can write programs that are larger than total available memory because only part of your program will reside in memory at any given time.

With Turbo Pascal extended syntax and the *Strings* unit programs can use null-terminated strings, so that they are more compatible with any Windows programs.

**Table 6.** Classification of development directions

| Development direction | New notions and facilities |
|---|---|
| Units | Unit, interface part, implementation part, initialization part, uses clause, modular programming, standard units |
| Objects | Objects-data and actions, private part, public part, methods, inheritance, constructors, destructors, dynamic objects, virtual methods, virtual objects, polymorphism, object-oriented programming |
| Low level facilities | Absolute variables, interruptions, interrupt procedures, *Intr* and *MsDos* procedures, CPU registers declaration, direct access to memory and ports, assembler code and machine code inside Pascal programs. |

The *Graph* unit supplies a set of fast, powerfull graphics routines. It implements the device independent graphic handler. Notice that some units represent their own notions system, own environment. You can study these possibilities independently of the language.

The *System* unit is used automatically by any unit or program. Other units are not used automatically. You must include them in the uses clause.

**8. Development directions.** Main Pascal development directions, realized in Turbo Pascal 7.0:

1. Units;
2. Objects;
3. Low level programming facilities.

All development directions are presented in the Table 6.

## REFERENCES

*Specification for Computer Programming Language Pascal.* (1982). British Standards Institution.

Tumasonis, V. (1993). *Paskalis ir Turbo Paskalis 7.0.* Ukas Publishing Company, Vilnius. 382 pp. (in Lithuanian).

*Turbo Pascal. Version 7.0. Language Guide.* (1992). Scotts Valley: Borland International.

*Turbo Pascal. Version 7.0. User's Guide.* (1992). Scotts Valley: Borland International.

*Turbo Pascal. Version 7.0. Programmer's Reference.* (1992). Scotts Valley: Borland International.

Wirth, N. (1971). The programming language Pascal. *Acta Informatica,* 1(1).

Wirth, N. (1972). *The Programming Language Pascal: Revised Report..* Zurich: ETH.

Received January 1995

**V. Tumasonis** is an associate professor of Department of Computer Science at Vilnius University. He received the Degree of Candidate of Physical and Mathematical Sciences from Moscow University in 1972. His research interests include programming languages, symbolic computation system, computer algebra.

## PASKALIO KALBOS STANDARTO IR TURBO PASKALIO LYGINAMOJI ANALIZĖ

Vladas TUMASONIS

Straipsnyje lyginamas Paskalio kalbos standartas, kurį 1982 metais priėmė Britanijos Standartizacijos institutas (British Standard Institution), su Turbo Paskaliu 7.0. Pastarasis turi didžiuli programavimo priemonių rinkinį (modulius, objektus, papildomas eilučių apdorojimo priemones ir kt.), kuris gali tenkinti net ir reikliausią programuotoją. Lentelių forma sistemingai pateikiamos visos papildomos Turbo Paskalio 7.0 priemonės (duomenų tipai, valdymo struktūros, procedūros ir funkcijos, parametrų rūšys, naujos išplėtimo kryptys), gretinant su Paskalio standartu. Toks gretinimas turėtų padėti geriau pajusti Paskalio ir Turbo Paskalio dvasią.