БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, 48
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, 48

София . 1999 . Sofia

Macroenhancements for Calling Multiargumental Subroutines or Assemblies of them

Chavdar Korsemov, Stefan Koynov, Hristo Toshev

Institute of Information Technologies, 1113 Sofia

1. Introduction

The paper presents ways for improving the Turbo Pascal programming environment.

In nature there are lots of complex processes with tremendous lengths of the input-output vectors. At that some of the coordinate values of the vectors vary more rapidly and other values - slower. This means that some of the coordinate vectors are relatively static while the rest of them are relatively dynamic. In such cases the relatively dynamic vector coordinates seem to be a more impressive object of interest.

This formulation is supported by imposing number of examples from different aspects of the living nature and from different processes in technique, too. Next follow some of the most typical cases:

1) Monitoring and control of processes in industry, the army, the scientific research, etc.:

- systems for technological control;
- real-time systems;
- adaptive systems;
- information redirection in computer systems;

2) Macroenhancements for programmers as means for developing the friendly programming environment:

- macroassemblies;
- compiler preprocessors;
- packages for scientific research;
- developer's packages.

In the first group (monitoring and control) any unit has time-varying parameters and some of them vary more intensively than other parameters. The time-variance is a consequence from the modes of exploitation, from the unit aging and also from the design. Surely the most impressive and undoubted examples can be found in the living nature with

its most complex organisms and their higher nervous activities which can be stimulating (accelerating) and retarding (inhibiting) at the same time.

The second group concerns programming where the processes are realized as subprograms of different levels and the input-output vectors as input-output variables and/or arrays. This case is a dynamic alternative of the global static memory definitions for the different subprograms. The idea is that the relatively static variables should be default parameters in the subroutine calls and the relatively dynamic ones should be explicitely declared. At that these two groups of variables for a concrete subroutine should be with varying bounds and varying contents for the different calls. The ideal decision in such cases is when the relatively dynamic arguments are declared with key operands in the macrocalls which generate the consequent subroutine calls and the relatively static arguments in the macrocalls are omitted. The default values are indicated in accordance with the argumental type in a way which has nothing in common with the true values of the variables (for example by empty strings for the string variables and by zero values for the numeric variables). The usage of key operands in the macrocalls instead of positional adds a new degree of freedom for the programmer and at the same time it makes the program more friendly.

2. The Center procedure

The developed below approach is based on the popular text-centering procedure in friendly programs. This procedure might serve as a starting point to improve the programming environment (for example Turbo Pacal), if it is generalized for the most often used PC periphery (screen, printer and disks) and if it is consequently unified in accordance with the typical buffer lengths for the concrete output devices.

This procedure can be written in three different ways depending on the type of the output device - the screen (which is the default case), PRN or ASCII/text disk files. The expansion of the device type for the procedure thus including the printer and the text files leads to the generalization and in a way to the unification of the concrete device to a virtual device which leads in its turn to certain advantages in programming. The Turbo Pascal programming environment has different standard units for the different standard output devices—Crt for the screen, Printer for the printer and System for text files. These library units use analogical, but different subroutines (procedures and functions) for the algorithm in any of the three cases. The screen version is the most popular one concerning the "friendly" programs; the other two cases make the program decision more aesthetic in the sense that the output information is centered and thus it is symmetrical. The three versions are analyzed below and the version-specific operators are **bold**.

2.1. The screen

This is the original form of the procedure:

```
procedure Center(Line : string);
uses Crt;
begin { Center }
  GotoXY(41 – Length(Line) div 2, WhereY); writeln(Line);
end; { Center }
```

It is possible for the other two cases that the maximal string length shall be declared as a numeric constant (which can be tuned for every concrete source code) – instead of the constant 41 the programmer may use a 'type const' declaration. In addition we need a 'var i : integer;' definition for modeling the 'GotoXY(...,WhereY);' operator. So the 'GotoXY'

operator is modeled by a FOR loop as it follows:

```
GotoXY(41-Length(Line) div 2, WhereY) ≡
≡ for i := 1 to 41 - Length(Line) div 2 do write(..., ' ');
```

The first argument in the 'write' operator is of 'text' type and its name for printers (prn or lpt1) is 'Lst'.

2.2. Prn (Lpt1)

This is the simpler expansion of the original Center procedure:

```
procedure Center(Line : string);
uses Printer;
var i : integer;
begin { Center }
  for i := 1 to 41 – Length(Line) div 2 do write(Lst, ' ');
 writeln(Lst, Line);
end; { Center }
```

2.3. ASCII/text disk files

Now we need that the declarations 'Fail : text' and 'FileName : string' must be in the procedure body but the 'assign', 'rewrite' and 'close' statements must be out of the procedure. Also we see that this version becomes an equivalent of the printer version, if FileName := 'prn'|'lpt1'.

```
procedure Center(Line : string);
var i : integer;
begin { Center }
  for i := 1 to 41 – Length(Line) div 2 do write(Fail, ' ');
 writeln(Fail, Line);
end; { Center }
```

All the three versions of the text-centering procedure lead us to the conclusion that it is possible to unify them in some general procedure which will count for the device type. Enriched by some text processing, this unification naturally leads to the presented below "universal" procedure WriteF.

3. The Center procedure unified for the basic types of output devices for IBM PC and the compatibles with them

In cases when there are even two different types of devices then the programmer is stressed additionally also by the specific features of any of them. As a consequence this complicates additionally the dialog with the user. On the other hand the proposed method can be treated as a new decision of integrated programming environment. The WriteF procedure is an unification of the Center procedure with respect to the possible instantaneous type of the output device (which can be the screen, the printer or any text file).

The unified text-centering procedure is designed easy to redirect the output information to the most often used output devices. This means that WriteF takes into account the specific features and the limitations for the different output devices in Pascal programs. Any Pascal programmer knows that the Crt unit controls the keyboard input and/or the screen output, that the Printer unit is responsible for the printer (with system names prn or lpt1,

but the system printer name is Lst) and that the System unit handles the operations with text files. In cases when there are output text files ('con' for the screen and 'prn' for the printer are of the same text type) it is possible to use just one procedure with a kind of a virtual output text file which at any instant can be the screen, the printer or a disk text file. Now we may use WriteF for such virtual output files instead of three different versions of the Center procedure. WriteF may be "included" by an 'uses' statement or by the $I directive. So the new procedure substitutes the three standard Pascal units counting for the correspondent buffer lengths; in addition it is possible to include elements of text processing (centering and left-right-alignments).

(Surely the specific for the device operations may be handled by the standard compiler units. Still these specific operations are an object of interest below in chapter 5. The application of the proposed approach is mainly in dialogous programs with at least two of the exposed device types, i.e. con+prn or con+text files, or prn+text files.)

The result of the proposed approach is that the generalization of the Center procedure also for printers and for disk output text files which is properly preprocessed makes the control of the output text devices and files pleasant and convenient moreover because this output text periphery is popular for IBM PC computers and compatible with them.

In Fig. 1a, b, c (Appendix) is shown the length of the source code and of the Tpu-realization of the unit Output in the form of the accessible by the programmer interface comment section of the unit with the WriteF procedure.

The next chapter is a convenient illustration for an usage of WriteF properly upgraded with the output macro. For this reason this chapter gives the correspondence between the WriteF parameters and the corresponding to them key operands of the output macro.

4. Turbo Pascal source code macrogenerator: independence on the subroutine call format

The good programming style presents the essence of the thought / the program algorithm as clearly and quickly as possible. The first obstacle though comes from the unalterable subroutine call format. Even if the programmer tends to alter the argument sequence in the format. Or if she/he is keen on skipping the default parameters – the relatively static parameters. This double obstacle is overcome by the macroprocessors. They guarantee the subroutine call format sequence allowing at the same time the plasticity of the true source code of the programmer: the programmer is allowed to skip the static arguments and at the same time she/he may declare the dynamic arguments in any arbitrary sequence (if the operands are of a key type). As a result the source code for the compiler is generated by the macroprocessor which rearranges the arguments in the subroutine calls in the due sequence at the same time including the skipped parameters (or including indications that these parameters are defaults). Such preprocessings are orders quicker than the time necessary for the programmer to follow the subroutine call formats "by hand".

In this chapter the authors propose a macroprocessor of this type applied for a small example. The example is an upgrade of the already discussed text-centering procedure enlarged to include printers and disk text files. For example the macrocall

```
&output text='This is a macroprocessor test';
```

will generate the following subroutine call (the zero and the empty strings indicate the default parameters):

```
{+}WriteF('This is a macroprocessor test','',0,'');
```

The only limitation comes from the Turbo Pascal source code line length (version at least 5.5 and above). The generated program code is embedded between the standard Pascal source code; the macrogenerator output code is marked at the beginning with '{+}'.

This macroprocessor may be additionally develop to include arguments of the array type. This can be done if the arrays defining the key words and their values are bidimensional. The second dimension coinsides with the "dynamic" index of the vector argument.

The second example is an illustrative program which is an input for the macroprocessor; after this program piece follows the macrogenerated Pascal source code:

```
program MacroTest; {Turbo Pascal macrogenerator test file}
uses Output; {Unit Output includes the WriteF procedure}
var s: array[1..3] of string;
  l: byte;
begin
      {Dialog setting the macro information}
  write('Input the device name: '); readln(s[1]);
  write('Input the maximal line length: '); readln(l);
  write('Input the alignment ([left], center, right): '); readln(s[2]);
  write('Input the text for output: '); readln([3]);
  &output file=s[1], mll=l, align=s[2], text=s[3];
     {'On-line' macrocall}
  &output text='This is a SCREEN sample test';
end.
```

Correspondence between the WriteF parameters and the &output operands. The WriteF procedure calls require the following sequence of the arguments (this limitation is overcome by the &output macro format):

I parameter: string for output. This is the only obligatory parameter. In the &output macro the correspondent operand is 'text=...;';

II parameter: name of the output device / disk text file. Legal are the system names con, prn and any disk text file names. The macrogenerator analyzes the file type (for disk text files). If the file is hidden, 'system', a disk volume identificator or a directory name, the generation is aborted with a warning; only if the file is read-only, then the attribute is reset to read-and-write. The default for this parameter is the system name con and it is indicated by the macrogenerator with an empty string. In the &output macro the correspondent operand is 'file=...;';

III parameter: maximal line length. In fact this is the output device 'buffer' length and it naturally depends on the given name (the second WriteF parameter). The default for this parameter is zero. Next come the maximal line lengths for the screen ('con'), the system printer ('prn') and for disk text files:

con – 79 characters;
prn – 136 characters;
text files – 255 characters.

In the '&output' macro the correspondent operand is 'mll=...;';

IV parameter: text alignment. This parameter resembles the same property of textprocessing systems except for the text justification. So the possible values may be: 'left', 'center' and 'right'. The default for this parameter is 'left' and it is indicatd by the macrogenerator with an empty string. In the '&output' macro the correspondent operand

is 'align=...;'.

Next follows the generated Pascal source code for our second example. Fig. 1 is a copy of the unit Output interface comment section which gives all the necessary information for the programmer when calling the WriteF procedure.

```
program MacroTest; { Turbo Pascal macrogenerator test file }

uses Output; { Unit Output includes the WriteF procedure }

var s: array[1..3] of string;
  l: byte;

begin

          { Dialog setting the macro information }

 write('Input the device name: '); readln(s[1]);
 write('Input the maximal line length: '); readln(l);
 write('Input the alignment ([left], center, right): '); readln(s[2]);
 write('Input the text for output: '); readln([3]);

  {+}WriteF(S[3], S[1], L, S[2]);

{ 'On-line' macrocall}

  {+}WriteF('This is a SCREEN sample test', '', 0, '');

end.
```

In Fig. 2 (Appendix) the file sizes for the two macrogenerator versions are presented. The first version works in the "statement-per-line" mode which is necessary for program codes with too long statements (with too long string constants or with too long comments). The second version is the "normal" one. It has the '{+}'-prefix in the beginning of the generated Pascal program lines.

5. Principal classification of the standard procedures and functions in Turbo Pascal

The authors made a classification for the subroutines of the most intensively used Turbo Pascal units (i.e. Crt, Graph, System and Dos). The shortest classification concerns the Crt unit and the longest – the Dos unit.

Next follow these classifications in details.

5.1. The Crt unit subroutine classification

This unit contains 20 subroutines. With respect to the macrogenerator we grouped them in two basic groups:

- Text processing subroutines (TextColor, TextBackground, TextMode);
- Video mode subroutines (LowVideo, NormVideo, HighVideo).

5.2. The Graph unit subroutine classification

This unit contains 79 subroutines. Fig. 3 (Appendix) presents the classification for these procedures and functions.

5.3. The System and Dos units subroutine classification

Unit System contains 28 disk operation subroutines and unit Dos contains 43 subroutines.

In Fig. 4 of the Appendix a possible classification of the standard procedures and functions for disk operations in Turbo Pascal for these two units is shown.

6. Macrodecisions for Turbo Pascal compiler standard subroutines

This chapter proposes an illustrative macro-covering of the already viewed Turbo Pascal standard subroutines. The overall estimation shows that at the average one macro covers about 8,95 compiler subroutines. And this is in addition to the basic enhancements to use default parameters and explicit parameters in any arbitrary sequence for the generated subroutine calls!

Next follow the lists of the possible macro-coverings of the Turbo Pascal standard subroutines from the previous chapter. These lists begin right after the headlines. The reader will note that the macro key operands usually reflect the subroutine names.

6.1. Possible coverings for the Crt unit subroutines (see also 5.1.)

Textprocessing macro:

```
&text Color=..., Background=..., Mode=...;
```

– Videomode macro:

```
&video mode=...;
```

6.2. Possible coverings for the Graph unit subroutines (see also 5.2. and Fig. 3)

Drawing and filling of graphical primitives:

```
&line /LineFrom=..., LineTo=...\, style=...;
    \LineRel=...      /

&poly draw=..., fill=...;
&ellipse StAngle=..., EndAngle=..., XRadius=...,
    YRadius=..., fill=..., /PieSlice=...\
                          \Arc=...   /
```

Graphical subsystem setup macros:

```
&GraphColor BackGround=..., Color=..., Pallete=...,
         AllPallete=...;

&fillpattern=..., style=..., flood=...;

&GraphPar BufSize=..., mode=...;
&GraphText justify=..., style=...;
&GraphSys type=...;
&GraphCurs or move=...;
&GraphDev clear=....
```

6.3. Possible coverings for the disk operation subroutines (see also 5.3. and Fig. 4)

```
&env type=..., count=..., str=..., var=...;
&common DosVer=..., CBreak=..., verify=...;
&disk type=...;
&dir type=...;
```

```
&file assign=..., mode=..., event=..., i/o=..., param=...,
 DOS=..., change=..., env=..., find=..., FAttr=...;
&DateTime type=..., FType=..., P/UTime=...;
&int type=..., no.=..., MsDos=...;
```

Now it is evident why the authors obtained from a total of 170 Turbo Pascal standard subroutines for the units Crt, Graph, System and Dos (only the procedures and the functions for disk operations are counted from the last two units) only 19 covering macros. Or why a single macro covers at the average 170/19=8,95 subroutines.

7. Conclusions and trends for future research

The paper presents a solution for improving the working environment when programming in Turbo Pascal (and for other programming languages in analogy). The starting point for this is the popular text-centering procedure on the computer screen. Amplified with other properties now this subroutine is fertile with ideas for a Turbo Pascal macroprocessor which allows the subroutine call arguments to be skipped or to be defined in any arbitrary sequence.

The first step in this direction leads to the unification of the upper bound for the buffer length which is device-oriented and it may be at most 79 characters for the screen, up to 136 characters for the system printer and up to 255 characters for disk text files. Next it is possible to include textprocessing elements; textprocessing may be centering or alignment (left or right) of the string. The three upper bounds for the buffer length are intimately connnected with the usage of the system names 'con' and 'prn' thus making possible the program-mode output redirection. The result of this step is the enlarged text-centering device-dependent procedure WriteF with its personal unit Output. The new version of the Center procedure may serve as a kind of a standard for a "virtual" text output for IBM PC and compatibles with them.

The second step leads to the usage of default (optional) parameters by simply skipping them. The explicit parameters may be defined in any arbitrary sequence. This splitting the list of the subroutine arguments to default and to explicit but defined in an arbitrary sequence is available by a respective program which rearranges them in the due order and indicating the presence of default parameters. This program is a macroprocessor the output of which is an input for the Turbo Pascal compiler; two versions of this macrohandler are presented.

The so designed Turbo Pascal macrohandler is easy to expand with new macros which will be processed in the same way for the presence of parameters which may be default and explicit but in an arbitrary order. Every macro may correspond to a single subroutine call or to an assembly of such calls. The first possibility is demonstrated with the expanded text-centering decision WriteF. The second possibility is an object of interest for the subroutines of the most often used units Crt, Graph, Dos and the disk operations of the System unit.

The principal advantage of this approach is in cases of processes which are rich of events (i. e. macros) for applications of intensive modeling when the events will have parameters varying to different extents (i. e. default and explicit). The paper proves that a total of 170 Turbo Pascal standard procedures and functions may be covered by some 19 macros for the units Crt, Graph, Dos and System (with rrespect to the disk operations for the last). Or that a single macro covers at the average about 179/19=8,95 subroutine calls.

This approach can be applied to any programming language thus reducing the stress of the programmer in a way which liberates him from the tyranny of the usual subroutine call format. This is valid during the whole cycle of input and tuning the object program. The efficiency of the approach is as greater as the variety of subroutine calls or their

combinations rise. The macrohandler rearranges the arguments in the due sequence for orders quicker than the programmer will do that "by hand". And now she/he has the possibility of true skipping the relatively static parameters. And the main enhancement is that this approach makes the program source codes more elegant, more distinct and better readable.

8. Appendix

```
                        unit Output;
                          interface
                         uses Dos;

    procedure WriteF(Line,                { Empty | nonempty  }
            FileName    : string;    { 'con' | 'prn' | text file }
       MaxLineLength: byte;     { device-dependent }
        Align       : string);  { 'left'|'center'|'right'  }

{        Directs the output to output text devices or files to        }
{     con, prn (lpt1) or disk new/old archive [read-only] files       }
{                                                                     }
{ MaxLineLength default & range- according to the device type and     }
{   set to the possible maximal device-dependent length.             }

{                    RANGES of the arguments:                         }
{                                                                     }
(* FileName      ={'con', 'prn' or 'lpt1', disk file names}          *)
(* MaxLineLength = { [1..79], [1..136], [1..255]       }            *)
{                                                                     }
{ (*) Admits empty strings and zero integer MaxLineLength as default  }
{    (wildcard) arguments.                                           }

{                 WILDCARD & DEFAULT arguments:                       }
{                                                                     }
{  FileName = '      : FileName = 'con'                               }
(* MaxLineLength = 0: MaxLineLength = {79, 136, 255} *)              }
{  Align = ''         : Align = 'left'                                }
{                                                                     }
{ (*) Disk file type checked. For output on hidden/system files, Volume }
{   IDentifiers or DIRectories the procedure is aborted.              }
{                                                                     }
{ n Line is aligned to the left, centered or to the right.            }

{                        ALIGN data set:                              }
{                                                                     }
(*              Align = { 'left, 'center, 'right }                   *)
{                                                                     }
```

Fig. 1. Unit **Output**; body of the INTERFACE comment section

```
{                          SUMMARY TABLE:                          }
{                                                                  }
{  Topic     Line   FileName  MaxLineLength        Align          }
{                                                                  }
{  Wildcard   ''      ''           0                 ''           }
{                                                                  }
{                              'con' - 79                          }
{  Default    ''     'con'       'prn' - 136         'left'       }
{                              file - 255                          }
{                                                                  }
{                  'con'   -    [1..79]                            }
{  Range         'prn'   -   [1..136]     'left','center','right'  }
{                  file    -   [1..255]                           }
{                                                                  }
  end.
```

Fig. 1. Unit **Output**; INTERFACE comment section summary table

```
 Output Pas  6307 23.04.98 13:36
 Output Tpu  3760 23.04.98 13:36


MacPas1 Pas  5668 23.03.98 18:11
MacPas1 Exe  7008 23.03.98 18:12
MacPas2 Pas  8181 26.03.98 17:14
MacPas2 Exe  6720 26.03.98 17:15
```

Fig. 1. Unit **Output**; Unit file parameters

1. Drawing and filling of graphical primitives

a) **Lines** (Line, LineTo, LineRel, SetLineStyle, SetWriteMode);
b) **Polygons** (Arc, Circle, Ellipse, FillEllipse, PieSlice).

2. Graphical subsystem setup

a) **Modes of filling** (FloodFill, SetFillPattern, SetFillStyle);
b) **Graphical parameters** (SetGraphBufSize, SetGraphMode);
c) **Graphical text processing** (SetTextJustify, SetTextStyle);
d) **Graphical subsystem total control** (InitGraph, DetectGraph, CloseGraph);
e) **Graphical cursor motion** (MoveTo, MoveRel);
f) **Graphical periphery control** (ClearDevice, ClearViewPort).

Fig. 2. The two versions of the Turbo Pascal macrogenerator

Subroutine	1	2	3	4	5	6	7	8	9	A	B	C
append assign		• •	•					• •		•		
Block Read Block Write					• •							• •
ChDir close	• •	• •						• •				
EOF EOLn erase				• •			• •		•		• •	
FilePos FileSize flush					• •	• •			• •	• •		
GetDir	•							•				
IOResult					•				•			
MkDir	•							•				
read ReadLn rename reset rewrite RmDir	• •		• •		• •		• •	• • • •		• •		
seek SeekEOF SeekEOLn SetTextBuf		• •		• •	•			•		• •	• • •	
truncate					•			•				
write WriteLn					• •					• •		

Legend

Authors' classification:

1 – dir operations
2 – attachments and deattachments
3 – processing modes
4 – checks for events
5 – input/output operations
6 – file parameters
7 – analogical to DOS commands

Classification according to [1]:

8 – input/output procedures
9 – input/output functions
A – procedures for text files
B – functions for text files
C – procedures for untyped files

Fig. 3. Illustrative classification of the unit **Graph** subroutines in accordance with the fields of application and from the point of view of their grouping in macros

1. Date and Time procedures

GetDate, GetFTime, GetTime, PackTime, SetDate, SetFTime, SetTime, UnpackTime

2. INT handling procedures

GetIntVec, Intr, MsDos, SetIntVec

3. Disk status functions

DiskFree, DiskSize

4. File processing procedures and functions

a) File processing procedures and functions for DOS

FExpand, FSearch, FSplit

b) File processing procedures and functions for Windows

FileExpand, FileSearch, FileSplit

c) File processing procedures and functions for both environments

FindFirst, FindNext, GetFAttr, SetFAttr

5. Other common procedures and functions

DosVersion, GetCBreak, GetVerify, SetCBreak, SetVerify

6. Procedures and functions only for DOS mode

a) Functions for interaction with the DOS environment

EnvCount, EnvStr, GetEnv

b) Process handling procedures

DosExitCode, Exec, Keep, SwapVectors

7. Procedures and functions only for Windows mode

a) Subdirectory handling procedures and functions

CreateDir, GetCurDir, RemoveDir, SetCurDir

b) Functions for interaction with the environment of the operational system

GetArgCount, GetArgStr, GetEnvVar

Fig. 4. Turbo Pascal file subroutines

References

1. Faronov, V. V. Basis of Turbo Pascal. Moscow, MHTU, Festo Didactic, 1991 (in Russian).
2. Deriev, I., S. Tokar. Borland Pascal with Objects 7.0 Procedures and Functions Reference Guide. Kiev, Dialectica, 1993 (in Russian).
3. Mizrohi, S. V. Turbo Pascal and Object-oriented Programming. Moscow. Finansi i Statistica, 1992 (in Russian).
4. Ruetten, T., G. Franken. Turbo Pascal 6.0. Sanct Petersburg, Griffon, 1992 (in Russian).

Макрогенераторы, улесняющие обращения к многопараметричных программ или их ансамблям

Чавдар Корсемов, Стефан Койнов, Христо Тошев

Институт информационных технологий, 1113 София

(Резюме)

Обсуждается улучшение рабочей среды при программировании в языке Турбо Паскал, а по аналогии – и в других языках. Применяется идея включения макросредств для удобного обращения к многопараметричных программам или ансамблям из них.

Этот подход облегчает труд программиста и отличается большей эффективности при настройке программ.