CERFACS Climate Modelling and Global Change Team

The PALM group

24th january 2002

A coding norm for Palm

1 Introduction

The teamwork for developing and maintaining software can greatly benefit from the adoption of an implementation standard which can make the code very easy to follow and to modify.

In addition to a well thought-out design of the algorithm and of the code architecture and to an exhaustive bench, a coding norm is required to fulfill this aim.

The purpose of a coding standard is to produce a coding style which leads to well presented source code: such code should be well structured, well documented and thus easy to understand. In particular, a norm should provide standards for the recognition of the nature and of the scope of variables, partitioning of the code within sections, and a mean of extracting documentation from the source code.

The reference languages for the PALM software are `FORTRAN 77` , `FORTRAN 90` , `C` , `Tcl` and `SHELL` and a particular relevance is given to the message passing aspects. The norm aims to be applicable to the five languages, nevertheless differences between the coding styles will be stressed out in the text.

The present norm is largely inspired by the DOCTOR norm (see Note de travail Arpège n. 4, 1988, Météo–France)

2 Coding style

Each module should begin with a set of initial comments. These should contain:

- a title

- the PURPOSE of the module
- the INTERFACE details, details of input/output parameters, arguments, etc.
- the METHOD description, principles of the algorithm
- the EXTERNALS or other routines called
- the FILES used for I/O
- a REFERENCE to further documentation
- the AUTHOR and date the routine was written, together with MODIFICATION details

The comment style will be detailed in section 3

The code body of the module should be split into sections, sub-sections, sub-sub-sections and paragraphs. The sign '+' will be used to make the distinction between the level of hierarchy. For a section, one plus should be used, for a sub-section, two plus, etc...

Each section should be separated from the previous one by a comment card followed by an asterisk (to be read as `C*` in `FORTRAN 77` as `!*` in `FORTRAN 90` , as `/*` in `C` , as `#*` in `Tcl` and in `SHELL`) and by the rest of the line of minus signs. No blank has to be put between the asterisk and the first minus sign.

Each section should begin with a section number and an underlined title whereas each sub-section should begin with a sub-section number and a title.

Each Module should be separated from the previous one by a comment card followed by an asterisk and by rest of the line of equal signs. No blank has to be put between the asterisk and the first equal sign.

```
FORTRAN 77
C
C*----------------------------------------------------------------------
C
C** + EXAMPLE OF SECTION
```

```
C
C
C** ++ EXAMPLE OF SUB-SECTION
C
C*----------------------------------------------------------------------
...
       END
C
C*======================================================================

FORTRAN 90
!
!*----------------------------------------------------------------------
!
!** + EXAMPLE OF SECTION
!
!
!** ++ EXAMPLE OF SUB-SECTION
!
!*----------------------------------------------------------------------
...
       END PROGRAM XX
!
!*======================================================================

C
  /*-----------------------------------------------------------------*/

  /** + EXAMPLE OF SECTION  **/


  /** ++ EXAMPLE OF SUB-SECTION **/

  /*-----------------------------------------------------------------*/
...

}
```

```
/*===================================================================*/

TCL
#
#*-----------------------------------------------------------------------
#
#** + EXAMPLE OF SECTION
#
#
#** ++ EXAMPLE OF SUB-SECTION
#
#*-----------------------------------------------------------------------
...

#
#*=======================================================================
```

Variables should be declared following the naming conventions of section 4 in the following order

1. arrays of exceptions (see section 4)
2. arrays of structures
3. complex arrays
4. logical arrays
5. arrays of strings
6. double arrays
7. real arrays
8. integer arrays
9. exceptions (see section 4)
10. structures
11. complex scalars

12. logicals
13. strings
14. double scalars
15. real scalars
16. integer scalars

Local declarations should follow dummy arguments declarations, global declarations, and parameter declarations. `C` and `FORTRAN 90` pointer declarations should be on separated lines.

`FORTRAN 90` declarations style

```
type, attribute, ... ::  name
```

should be preferred to the `FORTRAN 77` style

```
type name
attribute name
```

The declaration of crucial variables has to be followed by a comment explaining the meaning of the variables (see section 3.4)

Indentation should undergo the following rules:

`FORTRAN 77`
: Normal lines start in column 7
  Lines inside a loop should be indented 2 columns
  Lines inside a conditional statement should be indented 4 columns

`FORTRAN 90`
: `PROGRAM` and `END PROGRAM` start in column 1
  Normal lines start in column 3
  Lines inside a loop should be indented 2 columns
  Lines inside a conditional statement should be indented 4 columns
  Lines inside a case statement should be indented 4 columns

`C`
: Normal lines should be indented 2 columns w.r.t. the { delimiter starting the block
  The { delimiter starting a loop block should be on a separate line and intented 2 columns. Lines inside the block should be indented 2 columns w.r.t. the { delimiter starting the block

The { delimiter starting an `if` block should be on a separate line and intented 2 columns. Lines inside the block should be indented 2 columns w.r.t. the { delimiter starting the block
The { delimiter starting a `switch` block should be on a separate line and intented 2 columns. The `case` statements should be aligned with the { delimiter starting the switch

`Tcl`

Normal lines start in column 1
Lines inside a loop should be indented 3 columns
Lines inside a conditional statement should be indented 4 columns

In `FORTRAN 77` and `FORTRAN 90` keywords should be uppercase and variable names lowercase. In `C` , `Tcl` and `SHELL` the case of keywords is imposed. Variable and function names should be lowercase. Preprocessor constant and macros should be uppercase.

3 Comments

The comments within the codes can fulfill two purposes: on one hand they can be extracted to form the basis of an external documentation, on the other hand they represent the means for communication among the development team. A triggering code has to be established to make the extraction and the search of the comments easy and unambiguous.

Comments can start in any column if the language allows the indentation. It is up to the user to choose between following the code indentation and always starting in column 1.

Six levels of comments are supported:

1. Overview
2. External comments
3. Internal structure
4. Variables description and inclusion explanation
5. Working comments
6. Section and module separators, as well as empty line comments

3.1 Overview

Overview documentation comprises the title, the purpose and the interface of each module. It is triggered by a comment card followed by 4 asterisks on the first comment line. It has to be understood as `/****` in `C` , as `#****` in `Tcl` and `SHELL` , as `C****` in `FORTRAN 77` , as `!****` in `FORTRAN 90` . The trigger must contain the title. The overview is terminated in `FORTRAN` `Tcl` and `SHELL` by the first non comment line or by the first different trigger. In `C` it is terminated by `****/`.

3.2 External comments

External comments comprise, in addition to the overview

- the METHOD description
- the EXTERNALS or other routines called
- the FILES used for I/O
- a REFERENCE to further documentation
- the AUTHOR and DATE the routine was written, together with MODIFICATION details

The external comments are triggered by a comment card followed by 3 asterisks on the first comment line. It has to be understood as `/***` in `C` , as `#***` in `Tcl` and `SHELL` , as `C***` in `FORTRAN 77` , as `!***` in `FORTRAN 90`. They are terminated in `FORTRAN`, `Tcl` and `SHELL` by the first non comment line or by the first different trigger. In `C` they are terminated by `***/`.

3.3 Internal structure

The internal structure of the codes can be outlined by the means of comments containing, in addition to the external ones

- section titles
- sub-section titles
- details of branches referring to section and sub-section branched to

- any appropriate additional comments.

The internal comments are triggered by a comment card followed by 2 asterisks on the first comment line. It has to be understood as `/**` in `C` , as `#**` in `Tcl` and `SHELL` , as `C**` in `FORTRAN 77` , as `!**` in `FORTRAN 90`. They are terminated in `FORTRAN` and `Tcl` by the first non comment line or by the first different trigger. In `C` they are terminated by `**/`.

3.4 Variables description, inclusion explanation, loops labelling and conditional branches laballing

The most important variables should be described at their declaration. In `FORTRAN 77` and `Tcl` the descriptions should be on separate lines following the declaration and triggered by `C*V` or `#*V` starting in column 1. The description must contain the name of the variable.

In `FORTRAN 90` and in `C` the description should follow the declaration on the same line and should be triggered by `!*V` or `/*V` respectively. The description must contain the name of the variable. In `C` the comments are terminated by `*/`.

`C*V`, `!*V`, `/*V` or `#*V` comments can be used on a separate line to separate different groups of declarations.

The use of `include` or `USE` directives, although very useful for modular programming, can affect the readability of the code. Inclusion should be followed by a short remind of the purpose of the included module.

In `FORTRAN 90` and in `C` the description should follow the inclusion statement on the same line and should be triggered by `!*I` or `/*I` respectively. In `C` the comments are terminated by `*/`.

Inclusion is not a standard feature of `FORTRAN 77` nevertheless it is a widespread implementor extension. The description should be on a separate line following the inclusion and triggered by `C*I` starting in column 1.

3.5 Section and module separators

Each section should be separated from the previous one by a comment card followed by an asterisk (to be read as `C*` in `FORTRAN 77` , as `!*` in `FORTRAN 90`, as `/*` in `C` and as `#*` in `Tcl` and `SHELL`) and by the rest of the line of

	INT	REAL	DOUBLE	CHAR	LOG	COMPL	TYPE	MPI		FILE	EXC
	int	float	double	char			struct	mpi	void	file	exc
Global	ig_	rg_	dg_	cg_	lg_	yg_	sg_	mg_	vg_	fg_	eg_
Local	il_	rl_	dl_	cl_	ll_	yl_	sl_	ml_	vl_	fl_	el_
Dummy	id_	rd_	dd_	cd_	ld_	yd_	sd_	md_	vd_	fd_	ed_
Parameter	IP_	RP_	DP_	CP_	LP_	YP_	SP_	MP_	VP_	FP	EP_
Function	if_	rf_	df_	cf_	lf_	yf_	sf_	mf_	vf_	ff_	ef_
Loop	ib[_]										

Table 1: Prefixes indicating nature and scope of scalar variables

minus signs. No blank has to be put between the asterisk and the first minus sign.

Each section should begin with a plus and an title whereas each subsection should begin with two plus and a title.

Each module should be separated from the previous one by a comment card followed by an asterisk and by rest of the line of equal signs. No blank has to be put between the asterisk and the first equal sign.

4 Naming conventions

The purpose of the following naming convention is to convey, through prefix and suffix letters, the scope, the nature and possibly the message passing use of all variables within the program. This norm is compatible with the four languages for the common types of variables.

The structure of the name of a variable is

`ns[a]_`*name*`[_db]`

where `n` is a letter indicating the nature of the variable, `s` is a letter indicating the scope of the variable, and *name* is a mnemonic name not exceeding 10 letters. The convention for `n` and `s` is detailed in table 1. The letter `a` is added to the prefix of the arrays.

The loop index `ib` will be enough for one-level loops. In case of nested loops, indexes should be named `ib_`*name* where *name* is a mnemonic key (e.g. `ib_lon`, `ib_lat`). For the sake of compatibility with the DOCTOR

`_sb`	Variable to send. Blocking.
`_sn`	Variable to send. Non blocking.
`_rb`	Variable to receive. Blocking.
`_rn`	Variable to receive. Non blocking.
`_bb`	Variable to receive, treat and send. Blocking on send and receive.
`_bn`	Variable to receive, treat and send. Non blocking.
`_bs`	Variable to receive, treat and send. Blocking only on send.
`_br`	Variable to receive, treat and send. Blocking only on receive.
`_tn`	Transit variable. Never blocking.

Table 2: Suffixes indicating message passing nature of variables

coding style, the loop index can also start by `j`.

The last column "EXC" of table 1 concerns all the variables the nature of which does not fit one of the other classes. It is, for instance, the case of unsigned variables in `C` or of variables with a non standard `KIND` in `FORTRAN 90`. These variables should be fully commented at the declaration (see section 3.4)

The facultative suffix is used only for variables involved in message passing. `d` is a letter indicating the direction of the communication whereas `b` indicates if the communication should be blocking or non blocking. The convention used for the suffix is detailed in table 2.

5 Comments in Latex

The user could want to write some comments using latex functionnalities. That could be possible using the two keys \begin{itemize} (and \end{itemize}) and beginlatex: or latex: (and endlatex: or xetal:).
In the first case, the user will be able to enumerate different items whereas in the second case, he could use all the latex functionnalities.

6 Extraction tools

The adoption of this norm allows the use of automatic tools for extracting selected comments. It will be possible to extract any of the levels of comment and to search the code for particular working comments or declarations. This tools will be part of the PALM development tools.

7 Examples

A sample code is written in `FORTRAN 77`, in `FORTRAN 90`, and in `C` to illustrate the above mentioned norms.

7.1 Fortran 77

```
      PROGRAM example
C
C**** EXAMPLE - The title of the module
C
C     Purpose:
C       Purpose of the module: this program is written
C       using the coding norm for PALM, describing:
C \begin{itemize}
C \item the norm,
C \item the extraction tools.
C \end{itemize}
C
C     Interface:
C       Options and input/output parameters for the main module
C
C***  Method:
C       The principles of the algorithm
C \begin{latex}
C \epsfig{file=Image.eps,height = 1cm}
C \end{latex}
C
C     Externals:
C       And other routines called
```

```
C
C     Files:
C       Used for I/O
C
C     References:
C       To further documentation
C
C     History:
C     --------
C       Version   Programmer     Date        Description
C       ---------------------------------------------
C       1.0       Mister PALM    1969/04/18  created
C       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
C
C*-----------------------------------------------------------------
C
C** + DECLARATIONS
C
C** ++ Include files
C
C*I example.inc : a sample include file for parameter declaration
      INCLUDE 'example.inc'
C
C** ++ Global declarations
C
      INTEGER ig_ex
      REAL rg_ex
      DOUBLE PRECISION dg_ex
C*V    explanation for the content of dg_ex
      CHARACTER*4 cg_ex
      LOGICAL lg_ex
      COMPLEX yg_ex
      INTEGER*2 eg_ex
      INTEGER iga_tab(ip_ndim)
      REAL rga_tab(ip_ndim)
      DOUBLE PRECISION dga_tab(ip_ndim)
      CHARACTER*8 cga_tab(ip_ndim)
      COMMON /EXAMP/ ig_ex, rg_ex, dg_ex, lg_ex, yg_ex, iga_tab, rga_tab
```

```
C
C** ++ Local declarations
C
      INTEGER il_ex
      REAL rl_ex
      DOUBLE PRECISION dl_ex
C*V    explanation for the content of dl_ex
      CHARACTER cl_ex
      LOGICAL ll_ex
      COMPLEX yl_ex
      INTEGER ila_tab(ip_ndim)
      REAL rla_tab(ip_ndim)
      DOUBLE PRECISION dla_tab(ip_ndim)
      CHARACTER cla_tab(ip_ndim)
C
C*----------------------------------------------------------------------
C
C** + EXAMPLE OF SECTION
C
C** ++ Example of sub-section
C
      DO 114 ib=1,ip_ndim
        print*,'Beginning of the code'
 114  CONTINUE
C
C
C*----------------------------------------------------------------------
C
C** + EXAMPLE OF SECTION
C
      IF (il_ex .lt. ig_ex) THEN
          WRITE(*,*) 'if statement'
      ENDIF
C
C*----------------------------------------------------------------------
C
      END
C
```

```
C*=================================================================
C
      INTEGER FUNCTION if_example(id_param)
C
C**** IF_EXAMPLE - The title of the module
C
C     Purpose:
C       Purpose of the module: this function is written
C       using the coding norm for PALM
C
C     Interface:
C       if_example(id_param)
C
C     Input:
C       id_param     input parameter
C
C     Output:
C       if_example   value of the function
C
C***  Method:
C       The principles of the algorithm
C
C     Externals:
C       And other routines called
C
C     Files:
C       Used for I/O
C
C     References:
C       To further documentation
C
C     History:
C     --------
C       Version   Programmer     Date        Description
C       ---------------------------------------------
C       1.0       Mister PALM    1969/04/18  created
C       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
C
```

```
C*----------------------------------------------------------------
C
C** + DECLARATIONS
C
      INTEGER il_param
C
C*----------------------------------------------------------------
C
C** + EXAMPLE OF SECTION
C
      if_example=0
C
C*----------------------------------------------------------------
C
      RETURN
      END
C
C*=================================================================
C
      SUBROUTINE sample(id_input,rd_output)
C
C**** SAMPLE - The title of the module
C
C     Purpose:
C       Purpose of the module: this subroutine is written
C       using the coding norm for PALM
C
C     Interface:
C       CALL sample(id_input,rd_output)
C
C     Input:
C       id_input     input parameter
C
C     Output:
C       rd_output    output parameter
C
C***  Method:
C       The principles of the algorithm
```

```
C
C     Externals:
C       And other routines called
C
C     Files:
C       Used for I/O
C
C     References:
C       To further documentation
C
C     History:
C     --------
C       Version   Programmer     Date        Description
C       ------------------------------------------------
C       1.0       Mister PALM    1969/04/18  created
C       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
C
C*-----------------------------------------------------------------
C
C** + DECLARATIONS
C
C** ++ Input parameters
C
      INTEGER id_input
C
C** ++ Output parameters
C
      REAL rd_output
C
C** ++ Local variables
C
      INTEGER il_local
C*V  explanation of the contents of il_local
C
C*-----------------------------------------------------------------
C
C** + EXAMPLE OF SECTION
C
```

```
      rd_output = FLOAT(id_input + il_local)
C
C*----------------------------------------------------------------
C
      RETURN
      END
C
C*=================================================================
```

7.2 Fortran 90

```
PROGRAM example
!
!**** EXAMPLE - The title of the module
!
!     Purpose:
!       Purpose of the module: this program is written
!       using the coding norm for PALM
! \begin{itemize}
! \item the norm,
! \item the extraction tools.
! \end{itemize}
!
!     Interface:
!       Options and input/output parameters for the main module
!
!***  Method:
!       The principles of the algorithm
! \begin{latex}
! \epsfig{file=Image.eps,height = 1cm}
! \end{latex}
!
!     Externals:
!       And other routines called
!
!     Files:
!       Used for I/O
!
!     References:
!       To further documentation
!
!     History:
!     --------
!       Version   Programmer     Date        Description
!       ------------------------------------------------
!       1.0       Mister PALM    1969/04/18  created
!       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
```

```
!
!*-----------------------------------------------------------------
!
!** + DECLARATIONS
!
!** ++ Include files and use of modules
!
!*I declaration: a sample module for parameter and global declarations
  USE declaration
!
!** ++ Local declarations
!
  INTEGER :: il_ex
  REAL :: rl_ex
  DOUBLE PRECISION :: dl_ex !*V explanation for the content of dl_ex
  CHARACTER(LEN=ip_nlen) :: cl_ex
  LOGICAL :: ll_ex
  COMPLEX :: yl_ex
  TYPE (object) :: sl_obj
  INTEGER(KIND=2) :: el_ex
  INTEGER, DIMENSION(ip_ndim) :: ila_tab
  REAL, DIMENSION(ip_ndim) :: rla_tab
  DOUBLE PRECISION, DIMENSION(ip_ndim) :: dla_tab
  CHARACTER(LEN=ip_nlen), DIMENSION(ip_ndim) :: cla_tab
!
!*-----------------------------------------------------------------
!
!** + EXAMPLE OF SECTION
!
!** ++ Example of sub-section
!
  DO ib=1,ip_ndim   !*L label: sample loop
    PRINT*,'Beginning of the code'
  ENDDO !*LEND label
!
!*-----------------------------------------------------------------
!
!** + EXAMPLE OF SECTION
```

```
!
  IF (il_ex .lt. ig_ex) THEN
      PRINT*,'if statement'
  ENDIF
!
!*-----------------------------------------------------------------
!
END PROGRAM example
!
!*==================================================================
!
INTEGER FUNCTION if_example(id_param)
!
!**** PRIVATE
!
!**** IF_EXAMPLE - The title of the module
!
!     Purpose:
!       Purpose of the module: this function is written
!       using the coding norm for PALM
!
!     Interface:
!       if_example(id_param)
!
!     Input:
!       id_param     input parameter
!
!     Output:
!       if_example   value of the function
!
!***  Method:
!       The principles of the algorithm
!
!     Externals:
!       And other routines called
!
!     Files:
!       Used for I/O
```

```
!
!     References:
!       To further documentation
!
!     History:
!     --------
!       Version   Programmer     Date        Description
!       ------------------------------------------------
!       1.0       Mister PALM    1969/04/18  created
!       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
!
!*--------------------------------------------------------------------
!
!** + DECLARATIONS
!
  INTEGER il_param
!
!*--------------------------------------------------------------------
!
!** + EXAMPLE OF SECTION
!
  if_example=0
!
!*--------------------------------------------------------------------
!
  RETURN
END FUNCTION if_example
!
!*=====================================================================
!
SUBROUTINE sample(id_input,rd_output)
!
!**** SAMPLE - The title of the module
!
!     Purpose:
!       Purpose of the module: this subroutine is written
!       using the coding norm for PALM
!
```

```
!     Interface:
!       CALL sample(id_input,rd_output)
!
!     Input:
!       id_input     input parameter
!
!     Output:
!       rd_output    output parameter
!
!***  Method:
!       The principles of the algorithm
!
!     Externals:
!       And other routines called
!
!     Files:
!       Used for I/O
!
!     References:
!       To further documentation
!
!     History:
!     --------
!       Version   Programmer     Date        Description
!       ------------------------------------------------
!       1.0       Mister PALM    1969/04/18  created
!       x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
!
!*-----------------------------------------------------------------
!
!** + DECLARATIONS
!
!** ++ Input parameters
!
  INTEGER id_input
!
!** ++ Output parameters
!
```

```
  REAL rd_output
!
!** ++ Local variables
!
  INTEGER il_local
!*V  explanation of the contents of il_local
!
!*----------------------------------------------------------------
!
!**   1. EXAMPLE OF SECTION
!
  rd_output = FLOAT(id_input + il_local)
!
!*----------------------------------------------------------------
!
  RETURN
END SUBROUTINE sample
!
!*=================================================================
```

7.3 C

```
/**** PROLOGUE - Title of the module

  Purpose:
    this prologue is written using the coding norm for PALM       ****/

/*-----------------------------------------------------------------*/

/** + DECLARATIONS  **/

/** ++ Include files  **/

/*I Inclusion of standard libraries */
#include <stdio.h>
/*I Inclusion of custom files   */
#include "examplec.inc"

/** ++ Global declarations **/

int ig_ex;
int *ig_pointer; /*V ig_pointer: pointer to an integer*/
float rg_ex;
double dg_ex;
char cg_ex;
unsigned char eg_ex;

int iga_tab[ip_ndim];
float rga_tab[ip_ndim];
double dga_tab[ip_ndim];
char cga_tab[ip_ndim];

struct object
{
  int ig_obj;
  float rg_obj;
  char cga_obj[ip_ndim];
};
```

```
typedef struct object object;
object sg_obj;

/*=================================================================*/

main()
{
  /**** EXAMPLE - The title of the module

    Purpose:
      Purpose of the module: this program is written
      using the coding norm for PALM
      \begin{itemize}
           \item the norm,
      \item the extraction tools.
      \end{itemize}

    Interface:
      Options and input/output parameters for the main module    ****/

  /***
    Method:
      The principles of the algorithm
      \begin{latex}
             \epsfig{file=Image.eps,height = 1cm}
      \end{latex}

    Externals:
      And other routines called

    Files:
      Used for I/O

    References:
      To further documentation

    History:
    --------
```

```
    Version   Programmer     Date        Description
    -------------------------------------------------
    1.0       Mister PALM    1969/04/18  created
    x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
                                                                     ***/

/*-------------------------------------------------------------------*/

/** + DECLARATIONS  **/

/** ++ Local variables  **/

int ib; /*V loop index*/
int il_ex;
float rl_ex;
double dl_ex;   /*V il_ex : content of dl_ex */
char cl_ex;
unsigned char el_ex; /*V el_ex unsigned char : index */

int ila_tab[ip_ndim];
float rla_tab[ip_ndim];
double dla_tab[ip_ndim];
char cla_tab[ip_ndim];

/*-----------------------------------------------------------------*/

/** + EXAMPLE OF SECTION  **/

/** ++ Example of sub-section  **/

/*-----------------------------------------------------------------*/

for (ib=1; ib<ip_ndim; ib++) /*L label: sample loop */
  {
    printf("Beginning of the code\n");
  }

/*-----------------------------------------------------------------*/
```

```
  /** + EXAMPLE OF SECTION  **/

  il_ex=0;
  ig_ex=0;
  if (il_ex<ig_ex)
    {
      printf("if statement\n");
    }

  /*------------------------------------------------------------------*/

}
/*==================================================================*/

int if_example(int il_param)
{
  /**** IF_EXAMPLE - The title of the module

    Purpose:
      Purpose of the module: this function is written
      using the coding norm for PALM

    Interface:
      if_example(id_param)

    Input:
      id_param     input parameter

    Output:
      if_example   value of the function                         ****/

  /***
    Method:
      The principles of the algorithm

    Externals:
      And other routines called
```

```
    Files:
      Used for I/O

    References:
      To further documentation

    History:
    --------
      Version   Programmer     Date        Description
      ---------------------------------------------
      1.0       Mister PALM    1969/04/18  created
      x.y.z...  Name           yyyy/mm/dd  modified or added + explanation
                                                                   ***/
  /*-----------------------------------------------------------------*/


  /** 1. EXAMPLE OF SECTION  **/

  return(0.);

  /*-----------------------------------------------------------------*/
}

/*=================================================================*/
```