TPEX: Turbo Pascal Emulation for UNIX Version 2.0 Reference Guide

GLIDER Development Group

January 2000

Contents

1 Introduction

Sun Pascal [7] [1] is an implementation of the Pascal language [4] for compatible platforms with SunSoft UNIX operating system [2]. Sun Pascal implementation includes several important extensions to the standard Pascal definition, adding capabilities that are needed or useful for developing complex applications.

The fundamental extensions are the following:

- Separated construction and compilation of *Modules* or libraries containing function and procedure declarations.
- Availability of the standard C and FORTRAN libraries.
- Feasibility for linking with libraries programmed in other languages like C, C++ or FORTRAN.

These extended facilities have been applied for building a library package for *emulating* BORLAND TURBOPascal [3] [1] Run-Time Library, including the GRAPH Library. TURBOPascal is another powerful Pascal extension by Borland International for IBM/PC and compatible platforms. TURBOPascal is partially based on ObjectPascal, the Pascal Object-Oriented extension by Apple for Macintosh. [6]

We are offering to TURBOPascal users these libraries as an easy option for converting their applications to platforms based on the SunSoft Solaris version of the UNIX operating system. That the conversion could be relatively easy does not mean that it would be immediate. There exist syntactical and implementation differences between both extensions that must be translated independently of the package use.

In particular, TURBOPascal users will miss the *interface* and *implementation* Unit concepts that lack in the SPARCPascal modules. The interface role is managed using the same system provided precompiler for C programming. The interface contents may be implemented with files for use in `#include` statements for the precompiler, as those usually named with *.h* extension by C programmers. However, the precompiler adds a benefit, the *macro* substitution capability.

The emulation is not 100% compatible in all of the included functions and procedures in the package. In some cases it was necessary to change the number of parameters and/or their types or new names have been added leaving unresolved the original ones or the behavior does not correspond exactly with the original. These anomalies are reported in the detailed documentation bellow.

[1] All SPARC trademarks are trademarks or registered trademarks of SPARC International Inc. licensed exclusively to Sun Microsystems Inc.

[2] UNIX is registered trademark of UNIX System Laboratories, Inc. Solaris and all SunSoft products are trademarks or registered trademarks of Sun Microsystems, Inc

[3] All Borland products are trademarks or registered trademarks of Borland International, Inc.

Besides the emulated set of TurboPascal functions and procedures, some useful procedures for the UNIX environment have been added. The TPEX Library, in particular GRAPH library emulation, may be used as well by C programmers for porting MSDOS/TURBO C applications.

This a first version of the emulation package. After this statement we apologize for not having a complete version accomplishing the intended purpose and for the implementation errors that could remain besides the incompatibility problems already mentioned.

Further more, no intention have been done until present for emulating the capabilities of TurboPascal as Object-Oriented language. The development of a precompiler with this purpose has been left as a future project.

2 TPEX Library components

The TURBOPascal Emulation Library for UNIX consists of the following component files:

- **ptipos.h**: File for inclusion in SPARCPascal modules and programs containing definitions of TURBOPascal types. It also contains some emulation cases that were implemented by simple macro substitution.
- **globals.h**: File for inclusion in SPARCPascal modules and programs containing declarations of constants and variables belonging to the TURBOPascal environment.
- **pturbo.h**: File for inclusion in SPARCPascal modules and programs containing the *prototypes* or headers of the functions and procedures in the **pturbo** library. The **pturbo** library is the subset of emulated routines implemented in SPARCPascal language itself. It also contains prototypes of a subset of functions and procedures belonging to standard C libraries.
- **cturbo.h**: File for inclusion in SPARCPascal modules and programs containing the *prototypes* or headers of the functions and procedures in the **cturbo** library. The **cturbo** library is the subset of emulated routines implemented in C language. [3]
- **pxgraph.h**: File for inclusion in SPARCPascal modules and programs containing the *prototypes* or headers of the functions and procedures in the **xgraph** library. The **xgraph** library is the subset of emulated routines belonging to the TURBOPascal **GRAPH** library. They were implemented in C language using the X11 library for XWindow presentation.
- **xgraph.h**: File for inclusion in C source files containing the *prototypes* or headers of the functions and procedures in the **xgraph** library.

- **ccturbo.h**: File for inclusion in C source files containing the *prototypes* or headers of the functions and procedures in the **cturbo** library.
- **pturbo.o**: Object file containing the **pturbo** library.
- **cturbo.o**: Object file containing the **cturbo** library.
- **xgraph.o**: Object file containing the **xgraph** library. This library may also be used by C programmers. The corresponding header file in this case is **xgraph.h**.

3 Using the TPEX Library

3.1 Sun Pascal Users

3.1.1 How to compile and link

- In each Sun Pascal source file *-module* or *program-* put a precompiler directive `#include` followed by the argument *component header file* for each TPEX component that has at least one reference in the source file, be a type, constant, global variable, function or procedure reference. The `#include` directive must appear any where before the first reference but for clarity it is preferable to group all of them by the beginning of the source file.

  NOTE: *ptipos.h* is already included in the files *pturbo.h* and *cturbo.h*.

- In the main file -program- put the precompiler definition directive

  ```
  #define MAIN
  ```

  as the first in the group of precompiler directives.

- In the SPARC Pascal compile command for each **module**, include the argument `-I~sananes/tpex2` for the compiler can find the TPEX Library header files.

- In the Sun Pascal compile command for the main **program** besides the -I argument, include the list of TPEX object file components used in any of the source files for the compiler can build the executable file linking the program object file with all of the referenced module object files.

  If the **xgraph** component is used, then include also the argument **-lX11** for linking also with the **xlib** Library.

3.1.2 Compile and link example

Suppose you have a program comprising the program or main module and two libraries modules, named **myexample.p, module1.p** and **module2.p**, containing calls to TPEX routines and all of them syntactically correct. The following UNIX commands will compile and link the modules to build the executable, named **myexample**:

```
pc -c -w -P -I~sananes/tpex2 module1.p
pc -c -w -P -I~sananes/tpex2 module2.p
pc -w -P -I~sananes/tpex2 -lX11 ~sananes/tpex2/cturbo.o \
                                ~sananes/tpex2/pturbo.o \
                                ~sananes/tpex2/xgraph.o \
        module1.o module2.o -o myexample myexample.p
```

3.2 C Users

C users have to follow the same steps as Pascal users, but including in their source files the C versions of the heading or prototype files instead of the Pascal versions. The compile command is **cc** instead of **pc**, the -P option does not apply (it has a different meaning for the cc compiler).

NOTE:

The use of the pturbo.o component has not yet been tested in C programs

4 Porting a Turbo Pascal program to SPARC Pascal

A list of possible changes to be made in the source modules follows:

- Be sure all Pascal keywords are written in lower case and that all SP identifiers are written exactly as they are documented in the SPARC Pascal Compiler Reference Manual.
- Be sure all TPEX identifiers (for types, constants, variables, function and procedures) are written exactly as they are documented in the section **TPEX Library Reference**.
- All type *string* declarations have to be changed into

  `varying[<length>] of char`

  because *string* type in SP is a fixed length array of 80 characters.
- String variables used to store file names for arguments in *assign, open, reset* or *rewrite* must be declared of type SP *string* (remember that SP string type is a fixed character array of size 80).

- Calls to routine *val* have to be changed to conform with the number and type of the arguments in SP version. In SP version the third parameter is a code for the numeric type of the second argument.

- Calls to routine *str* have to be changed to conform with the number and type of the arguments in SP version. In SP version the second argument is a constant or expression string of type *t_formato* (see content of **ptipos.h** in section **TPEX Library Reference**) that must be a C-type format for output. The TPEX functions *mkeformat* and *mkfformat* may be used to build the desired format for floating point variables.

- TP typed constants: In SP there aren't Typed constants. Instead, SP allows for initialization of variables in any *var* declaration. For variable initialization, the assignment operator *:=* must follow the type specification and then it must follow the value, simple or structured. The character pair for delimiting structured values are [] instead of **()**. Values for record fields do not require the names but have to be specified according to the record declaration order, separated by commas.

- To prevent program crashes due to input/output errors, instead of using the TP directives for enabling/disabling error trapping, SP provides the following mechanism:

  Near the beginning of the main program or anywhere before the statement that could cause an input or output (IO) error, call the SP procedure:

  `set_ioerr_handler(addr(IOerr_f))`

  IOerr_f is a TPEX function that the execution program will invoke in case of an IO error occurrence. **IOerr_f** sets the value of the global variable **IOResult** to the error code of the IO error and then, control returns to the statement following the one that caused the error. The value returned in **IOResult** is an integer corresponding to the relative position -order- of the error in the SP type **IOerror_codes**. This is an enumerated type, the list of its values may be found in the SP Reference Manual. A zero value means that no error occurred.

  For this mechanism to work properly, the user must reset the global variable **IOResul** to zero before the IO statement that could cause error.

  If you want to disable the error trapping mechanism, call again the

  `set_ioerr_handler`

  procedure but with the *nil* value as argument. After that, the program will crash in case of any IO error.

- Direct access processing of binary -not text- files, may be converted to the TPEX implemented abstract type *tpFile*. SP doesn't have a procedure equivalent to the TP procedure *seek* for positioning a file at a specified

record number for reading or writing. The user may also use the C file processing functions and procedures whose prototypes are included in the *pturbo.h* header file, in particular, the *fseek* function. If the file has been already opened for SP, the corresponding SP file variable need to be converted to a C file pointer variable using the SP function *getfile*.

- TP units requires special work.
  - Duplicate the unit into two files, one with *.h* extension and the other with *.p* or *.pas* extension.
  - In the .h file remove all of the TP implementation part.
  - In the .h file add the attribute `external;` to each procedure or function declaration in the ex-interface part.
  - In the .p or .pas file remove all the TP interface part. In its place put the compiler directive

    ```
    #include "<unit_name>.h"
    ```

  - In the .p file change the TP reserved word *unit* for the equivalent SPARC Pascal *module*.
  - TP *Uses* clausule: For each unit name in a *uses* clausule, put a corresponding

    ```
    #include "<unit_name>.h"
    ```

    . This rule is valid also for *uses* clauseles contained in the *implementation* part of an unit.
  - Global variables: All types, variables and constants declared in a .h file and included in any module and in the main or *program* module, are global. But initialized variables must be declared as *extern* for the modules. The extern declaration requires to precede the type declaration with the reserved word *extern*. This difference may be resolved setting a precompiler variable with the precompiler statement `#define` having no value for the main or program module and the value *extern* for the rest of the modules. Accordingly, a precompiler variable, **MAIN** for instance, must be defined in the program module and undefined in the rest of the modules for proper compilation.
- Procedural types and variables: In SP procedural types are defined as pointer types. For example, instead of this TP piece of code:

```
type funcion = function(x: real):real;
function escala1(x: real): real;
  ....
end;
function escala2(x: real): real;
  ....
end;
....
var  unaFuncion: funcion;
...........
```

```
unaFuncion:=escala2;
..........
```

the following code will have to be placed instead:

```
type funcion = ^function(x: real):real;
function escala1(x: real): real;
  ....
end;
function escala2(x: real): real;
  ....
end;
....
var  unaFuncion: funcion;
..........
unaFuncion:=addr(escala2);
..........
```

The *nil* value may be assigned to procedural variables.

- Untyped var parameters: TP accepts procedure and function parameters preceded by the reserved word *var* without type specification. In this case the compiler do not perform any type checking, it just pass the address of the referred variable. SP also offers the same feature, but you have to precede the complete type declaration with the SP reserved word *univ*. For instance, instead of declaring in TP: `procedure convert(var a; var b);` In SP you could write:

  ```
  procedure convert(var a: univ real; var b: univ char);
  ```

5 TPEX Library Reference

5.1 Types

Defined in file **ptipos.h**

It follows the list in alphabetical order of the SPARCPascal type declarations. Comments indicates whether the type identifier is a TURBOPascal predefined type or a new type added by the TPEX library.

```
     byte      = 0..255;          {*TP type*}
EventStructure = record           {*TPEX type. For handling user events*}
      nwindow    : integer; {*Window in which the event occurred*}
      x          : integer; {*Mouse x Coordinate*}
      y          : integer; {*Mouse y Coordinate*}
      button     : integer; {*Mouse Button pressed*}
      key        : integer; {*Key pressed*}
      visibility : integer; {*Type of visibility*}
      mouseevent : integer; {*Type of Mouse Event*}
                 end;
    File       = file of char;    {*TPEX type*}
    longint    = integer;         {*TP type *}
    OUTPUT     = output;          {*TPEX type. For use in GLIDER programs,
                                              otherwise is not defined *}
    PChar      = ^char;           {*TPEX type*}
    STRING     = string;          {*TPEX type. Synonimum for the predefined
                                               SPARCPascal type
                                               'string=array[1..80] of char'
                                               Must be used in GLIDER programs
                                               to avoid the incompatibility with
                                               the same type identifier that
                                               translates to SP type
                                               'varying[255] of char' *}
    t_formato  = string;          {*TPEX type; type of the C-style output format
                                               in procedure SP str *}
    tpFile     = record           {*TPEX type, for emulating record
                                               oriented file processing in
                                               TURBOPascal *}
                   name: varying[80] of char;
                   pName: ^char;
                 recSize: integer;
                   cFile: univ_ptr;
                   end;

    varstring  = varying[255]
                   of char;       {*TPEX type, equivalent to the predefined
                                               TP type 'string' *}
    word       = 0..65536;        {*TP type *}
```

5.2 Constants

Defined in file **globals.h**

5.2.1 Constants for specifying variable type in procedure val call

```
pINTEGER = 0;
pLONGINT = 1;
pSINGLE  = 2;
pDOUBLE  = 3;
```

5.2.2 Colors

```
BLACK        =  0;
BLUE         =  1;
GREEN        =  2;
CYAN         =  3;
RED          =  4;
MAGENTA      =  5;
BROWN        =  6;
LIGHTGRAY    =  7;
DARKGRAY     =  8;
LIGHTBLUE    =  9;
LIGHTGREEN   = 10;
LIGHTCYAN    = 11;
LIGHTRED     = 12;
LIGHTMAGENTA = 13;
YELLOW       = 14;
WHITE        = 15;
```

5.2.3 Line types for graphics

```
SolidLn    = 0;
DottedLn   = 2;  {*no correspondent in XLib*}
CenterLn   = 2;  {*no correspondent in XLib*}
DashedLn   = 1;
UserBitLn  = 0;  {*no correspondent in XLib*}
NormWidth  = 1;
ThickWidth = 3;
```

5.2.4 Text justification options

```
LeftText   = 0;
CenterText = 1;
RightText  = 2;
BottomText = 0;
TopText    = 1;
```

5.2.5 Text orientation options

```
HorizDir = 0;
VertDir  = 1;
```

5.2.6 Types of fonts

```
DefaultFont   = 0;
TriplexFont   = 1;  {*not defined for SP*}
SmallFont     = 2;  {*not defined for SP*}
SansSerifFont = 3;  {*not defined for SP*}
GothicFont    = 4;  {*not defined for SP*}
```

5.2.7 Event Types

```
VisibilityEvent = 1;
KeyboardEvent   = 2;
MouseEvent      = 3;
```

5.2.8 Visibility Types

```
TotalVisibility   = 0;
PartialVisibility = 1;
NoVisibility      = 2;
```

5.2.9 Mouse Event Types

```
ButtonDown   = 1;
ButtonUp     = 2;
ButtonMotion = 3;
```

5.2.10 Mouse Cursor Types

```
TargetCursor   = 128;
TcrossCursor   = 130;
Left_ptrCursor = 132;
WatchCursor    = 150;
XtermCursor    = 152;
```

5.2.11 Numerical constants

```
maxreal    = 1.7E+38;
maxinteger = 32767;
maxlongint = 2147483647;
maxword    = 65535;
pi         = 3.14159265358979;
```

6 Global variables

Defined in file **globals.h**

```
IO_ERROR    : IOerror_codes;
IOResult    : integer16;    {*For compatibility with TURBOPascal*}
tpIOResult  : integer;      {*For tpFile processing*}
wherex,                     {*See gotoxy procedure in the next section*}
wherey      : integer16:=0;
reversevideo: boolean:=false;
```

7 Functions and Procedures

This section documents the functions and procedures currently implemented in the TPEX Library. The documentation style follows that of Borland and in many cases the function descriptions are identical to those of the Turbo Pascal Library Reference [2]. However, functions and procedures that do not have correspondent in the TP Run-Time Library or those that have syntactical or semantical differences have appropriated descriptions.

This documentations also contains descriptions for the C functions whose prototypes are included in some of the TPEX Library header files. The C function description are partially taken from the book: C A Reference Manual. [3]

append procedure

pturbo.h

Function Opens a file for appending.

Declaration `procedure append(var f: univ File; nom: string);`

Remarks Note the difference with the TP version, that requires the file already opened, ie., *assign*ned.

See also *assign.*

Arc procedure

xgraph.h

Function Draws a circular arc from start angle to end angle, using (x,y) as the center point and radius given.

Declaration

```
procedure Arc(x, y:integer; StAngle, EndAngle, radius: word);
external c;
```

Remarks *StAngle* and *EndAngle* are the values of start and end angles respectively. The circular arc is drawn in the current drawing color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Circle, SetColor.*

assign procedure

pturbo.h

Function Assigns the name of an external file to a file variable.

Declaration `procedure assign(var f: univ File; nom: string);`

Remarks *f* is a file variable of any type, and *nom* is a SP string type variable, ie., fixed string of 80 characters. It is equivalent to the SP procedure: `open(f,nom,'unknown')`

See also *append.*

Bar procedure

xgraph.h

Function Draws a bar using the current fill style and color.

Declaration `procedure Bar(x1, y1, x2, y2: integer); external c;`

Remarks

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Bar3D, GraphResult, SetFillStyle, SetLineStyle.*

Bar3D procedure

xgraph.h

Function Draws a 3-D bar using the current fill style and color.

Declaration `procedure Bar3D(x1, y1, x2, y2: integer; Depth: word; Top: boolean); external c;`

Remarks

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Bar, GraphResult, SetFillStyle, SetLineStyle.*

blockread procedure

pturbo.h

Function Reads one or more bytes into a variable.

Declaration `procedure blockread(var f: univ File; var b: univ string; len: integer);`

Remarks Note the differences with the TP version: instead of specifying the number of records to be read, the total number of bytes have to be specified. The value of the *len* parameter has to be the number of records to be read times the size of the type associated to the file variable.

See also *blockwrite.*

blockwrite procedure

pturbo.h

Function Writes one or more bytes from a variable.

Declaration `procedure blockwrite(var f: univ File; var b: univ string; len: integer);`

Remarks Note the differences with the TP version: instead of specifying the number of records to be written, the total number of bytes have to be specified. The value of the *len* parameter has to be the number of records to be written times the size of the type associated to the file variable.

See also *blockread.*

Circle procedure

xgraph.h

Function Draws a circle using (x,y) as the center point with the given radius.

Declaration `procedure Circle(x, y: integer; radius: word); external c;`

Remarks The circle is drawn in the current drawing color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Arc, SetColor, SetLineStyle.*

ClearDevice procedure

xgraph.h

Function Clears the graphic window.using the background color.

Declaration `procedure ClearDevice; external c;`

Remarks Clears the window using the background color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before. than a window werw opened.

See also *CloseGraph, CloseWindow, InitGraph, InitWindow, SelectWindow, SetBkColor.*

CloseGraph procedure

xgraph.h

Function Closes the graphic mode.

Declaration `procedure CloseGraph; external c;`

Remarks

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *InitGraph.*

CloseWindow procedure

xgraph.h

Function Closes a graphic window.

Declaration `procedure CloseWindow(aWin: integer); external c;`

Remarks

aWin is the identification number of the Window to be closed.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before and InitWindow if more than one window werw opened.

See also *InitGraph, InitWindow.*

clrscr procedure

cturbo.h

Function Clears the terminal window or screen.

Declaration `procedure clrscr; external c;`

copy function

ptipos.h

Function Returns a substring of a string.

Declaration

```
copy(X: varstring;Y: integer;Z:integer):varstring;
```

Remarks

It is a *macro* substitution for the original `substr()` function in SP Pascal.

See also *delete, insert, pos*

delay procedure

cturbo.h

Function Delays a specified number of seconds.

Declaration `procedure delay(x: double); external c;`

Remarks Note the difference with TP version: The elapsed time is specified in seconds, not in milliseconds.

delete procedure

pturbo.h

Function Deletes a substring from a string.

Declaration

```
procedure delete(var s: univ varstring; desde, cuantos: byte);
```

Remarks Note that the procedure applies only to varying types in SP Pascal, not to SP string and alfa types. *s* is a varstring type of any maximum length. *desde* is the position in the string of the first character to be deleted and *cuantos* is the total number of characters to be deleted.

See also *copy, insert, pos*

fclose function

pturbo.h

Function Closes a file opened with the *fopen* C function.

Declaration `function fclose(f: univ_ptr): integer; external c;`

Remarks *fclose* belongs to the standard C Library. Note that *f* has to be a C pointer file value. See *fopen* Remarks before using.

See also *fopen, fread, fwrite, fseek*

fopen function

pturbo.h

Function Opens a stream file for IO operation with the C file stream functions.

Declaration

```
function fopen(pname, pmodo: univ_ptr): univ_ptr; external c;
```

Remarks *fopen* belongs to the standard C Library. Note that *f* has to be a C pointer file value. The SP function *getfile* performs the conversion from a Pascal file type value to the equivalent C value. Before calling *fopen*, a call to the SP Pascal *open* procedure have to be done in order to define value to a previously declared Pascal file type variable, if it is the case. Then a call to the Pascal conversion function *getfile* must follow in order to define the value for the C compatible file pointer.

pname and *pmode* are pointers that point to the first character of null terminated strings -SP alfa, string or varying. The SP constant **minchar** is

the null value that must be appended at the end of the string to conform with C style strings. *pname* has to point to the external name of the file and *pmode* has to point to a string having one of the following values:

'r' Open an existing file for Input

'w' Create a new file, or truncate an existing one, for Output

'a' Create a new file, or append to an existing one, for output

'r+' Open an existing file for update, starting at the beginning of the file

'w+' Create a new file, or truncate an existing one, for update

'a+' Create a new file, or append to an existing one, for update

See also *fclose, fread, fwrite, fseek*

fread function

pturbo.h

Function Reads a file in binary mode.

Declaration

```
function  fread(var b: univ string; esize: integer; len: integer; f: univ_ptr)
: integer; external c;
```

Remarks *fread* belongs to the standard C Library. Note that *f* has to be a C pointer file value. See *fopen* Remarks before using. *fread* reads up to *len* elements of the *esize* element size into the address pointed by *b*. The actual number of items read is the value returned as function, 0 in case of error.

See also *fopen, fclose, fwrite, fseek*

freemem procedure

cturbo.h

Function Disposes a dynamic variable of a given size.

Declaration

```
procedure freemem(var p: univ_ptr; nbytes: integer16);
external c;
```

Remarks *p* is a pointer variable of any pointer type that had to be previously assigned with a call to the procedure *getmem*.

See also *getmem*.

fseek function

pturbo.h

Function Prepares a file to perform the next read or write from or to a specified position.

Declaration `function fseek(f: univ_ptr; nr_offset: integer; seek_modo:integer): integer; external c;`

Remarks *fseek* belongs to the standard C Library. Note that *f* has to be a C pointer file value. See *fopen* Remarks before using. *fseek* allows random access within the file. *nr_offset* specifies the number of bytes to be skipped. *seek_mode* is a seek code that indicates from what point the *nr_offset* should be measured. The valid options for this code are:

0: from the beginning of the file

1: from the current position of the file

2: from the end of the file (negative values of *nr_offset* specifies positions before the end; positive values extend the file with unspecified content)

See also *fopen, fclose, fwrite, fread*

fwrite function

pturbo.h

Function Writes a file in binary mode.

Declaration `function fwrite(var b: univ string; esize: integer; len: integer; f: univ_ptr) : integer; external c;`

Remarks *fwrite* belongs to the standard C Library. Note that *f* has to be a C pointer file value. See *fopen* Remarks before using. *fwrite* writes *len* elements of the *esize* element size from the address pointed by *b*. The actual number of items written is the value returned as function, 0 in case of error.

See also *fopen, fclose, fread, fseek*

GetBkColor function

xgraph.h

Function Returns the number of the current background color.

Declaration `function GetBkColor: word; external c;`

Remarks See subsection **Colors** for the list of color numbers.

See also *GetColor, InitGraph, SetBkColor, SetColor*

getc function

pturbo.h

Function Takes the next character from the input stream *f*

Declaration `function getc(f: univ_ptr): integer; external c;`

Remarks *getc* belongs to the standard C Library. Note that *f* has to be a C pointer file value. See *fopen* Remarks before using.

See also *fopen, fclose, fread, fwrite, fseek*

getchar function

pturbo.h

Function Takes the next character from the standard input stream.

Declaration `function getchar: char; external c;`

Remarks *getc* belongs to the standard C Library.

GetColor function

xgraph.h

Function Returns the number of the current color value.

Declaration `function GetColor: word; external c;`

Remarks See subsection **Colors** for the list of color numbers.

See also *GetBkColor, InitGraph, SetBkColor, SetColor*

getdate procedure

pturbo.h

Function Returns the current date set in the operating system.

Declaration `procedure getdate(var a,m,d,ds: word);`

Remarks *a*: year; *m*: month; *d*: day; *ds*: day of week (0 correspond to Sunday)

See also *GetTime*

GetMaxColor function

xgraph.h

Function Returns the highest color number that can be passed to the *SetColor* procedure.

Declaration `function GetMaxColor: word; external c;`

Remarks

See also *GetColor, InitGraph, SetBkColor, SetColor*

GetMaxX function

xgraph.h

Function Returns the maximum X value for the current graphic window.

Declaration `function GetMaxX: word; external c;`

Remarks

See also *InitGraph, GetMaxY, MoveTo*

GetMaxY function

xgraph.h

Function Returns the maximum Y value for the current graphic window.

Declaration `function GetMaxY: word; external c;`

Remarks

See also *InitGraph, GetMaxX, MoveTo*

getmem procedure

cturbo.h

Function Creates a new dynamic variable of the specified size, and puts the address of the block in a pointer variable.

Declaration `procedure getmem(var p: univ_ptr; nbytes: integer16); external c;`

Remarks *p* is a pointer variable of any pointer type. *nbytes* specifies the size in bytes of the dynamic variable to allocate.

See also *freemem*

gettime procedure

pturbo.h

Function Returns the current time set in the operating system.

Declaration `procedure gettime(var h,m,s,d: word);`

Remarks Ranges of the values returned are *h*: 0..23, *m*: 0..59, *s*: 0..59, *d*: (hundredth of seconds) 0..99.

See also *getdate*

gotoxy procedure

cturbo.h

Function Positions the cursor in the terminal screen.

Declaration `procedure gotoxy(x, y: byte); external c;`

Remarks The coordinates are adjusted to the size of the window. The last position of the cursor in the screen or terminal window are stored in the global variables *wherex, wherey*. Note the difference with the original TP version. In this version *wherex, wherey* are global variables, not functions. They are not updated after read o write operations.

See also Global variables *wherex, wherey* in the Global variables section.

InitGraph procedure

xgraph.h

Function Sets the X Window graphic mode and open the first graphic window..

Declaration `procedure InitGraph(x, y: integer; WinNamePtr: univ_ptr); external c;`

Remarks

x, y are the width and height of the first window that is created after initialization of the X graph mode. *WinNamePtr* must be a pointer or Pchar reference to a C-type string that contains a caption or title to that window.

See also *CloseGraph, CloseWindow, InitWindow*

InitWindow function

xgraph.h

Function Opens a new window for graphic output.

Declaration

`function InitWindow(x,y: integer; WinNamePtr: univ_ptr): integer; external c;`

Remarks

The returned value is an integer that identifies the created Window. The user must use this value for selecting the Window as active window. After execution the created window is automatically selected as the active window. *x, y* are the width and height of the window that is created. *WinNamePtr* must a pointer or Pchar reference to a C-type string that contains a caption or title to that window.

See also *CloseWindow, CloseGraph, SelectWindow*

insert procedure

pturbo.h

Function Inserts a substring into a string.

Declaration

```
procedure insert(si: varstring; var s: univ varstring; en: byte);
```

Remarks *si* is the substring to be inserted into the string *s* at position *en*. Note that *s* has to be of varstring type.

See also *copy, delete, pos*

IOerr_f function

pturbo.h

Function Implements the IO error recovery capability.

Declaration

```
function  IOerr_f(in errcod:IOerror_codes; fp:univ_ptr):boolean;
```

Remarks This function is exclusive to the TPEX Library. The user must never call this function directly, instead the user may just enable or disable its use, as follows.

The function implements the IO error recovery capability returning always *true* to its caller, the system IOerror handling routine. In this way, if the function has been enabled, after an IO error occurrence the execution resumes at the point following the file routine invocation that caused the error and the predefined global variable **IOResult** contains the system error code value, that is an integer corresponding to the relative position -order- of the error in the SP type **IOerror_codes**, which list may be found in the SP Reference Manual. A zero value means that no error occurred.

For this mechanism to work properly, the user must reset the global variable **IOResul** to zero before the IO statement that could cause error.

For enabling the IO error recovery capability, prior to the execution of the instruction that could raise an IO error, the SP `set_ioerr_handler` procedure had to called with the expression **addr(IOerr_f)** as its only parameter. To disable the error trapping mechanism, call again the `set_ioerr_handler` procedure with the **nil** value as argument. After that, the program will crash in case of any IO error.

Line procedure

xgraph.h

Function Draws a line from the point (*x1,y1*) to (*x2,y2*)

Declaration `procedure Line(x1, y1, x2, y2: integer); external c;`

Remarks Draws the line in the thickness and style defined by *SetLineStyle* and the current Color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *LineRel, LineTo, MoveTo, Rectangle, SetColor, SetLIneStyle*

LineRel procedure

xgraph.h

Function Draws a line from the last point addressed to a point at offsets (*dx,dy*) from its x and y coordinates.

Declaration `procedure LineRel(dx, dy: integer); external c;`

Remarks Draws the line in the thickness and style defined by *SetLineStyle* and the current Color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Line, LineTo, MoveTo, Rectangle, SetColor, SetLIneStyle*

LineTo procedure

xgraph.h

Function Draws a line from the last point addressed to the point (*x,y*)

Declaration `procedure LineTo(x, y: integer); external c;`

Remarks Draws the line in the thickness and style defined by *SetLineStyle* and the current Color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Line, LineRel, MoveTo, Rectangle, SetColor, SetLIneStyle*

mkeformat function

pturbo.h

Function Returns a *t_formato* type string to use as the *f* format parameter in calls to the *str* procedure. The string returned contains a c-style string format for outputting floating point variables with exponent notation. *t* is the minimum length and *d* the number of digits to the right of the decimal point.

Declaration `function mkeformat(t,d: integer): t_formato;`

Remarks It is exclusive to the TPEX Library.

See also *mkfformat*

mkfformat function

pturbo.h

Function Returns a *t_formato* type string to use as the *f* format parameter in calls to the *str* procedure. The string returned contains a c-style string format for outputting floating point variables with fixed notation. *t* is the minimum length and *d* the number of digits to the right of the decimal point.

Declaration `function mkfformat(t,d: integer): t_formato;`

Remarks It is exclusive to the TPEX Library.

See also *mkeformat*

MoveRel procedure

xgraph.h

Function Addresses a new point from the last point addressed with offsets *dx, dy* to the x, y coordinates.

Declaration `procedure MoveRel(dx, dy: integer); external c;`

Remarks

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *LineRel, LineTo, MoveTo.*

MoveTo procedure

xgraph.h

Function Addresses the new point (x,y)

Declaration `procedure MoveTo(x, y: integer); external c;`

Remarks

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *LineRel, LineTo, MoveRel.*

NextEvent function

xgraph.h

Function Waits for a user event.

Declaration

```
function NextEvent(EventType: integer; var EventData: EventStructure):
integer; external c;
```

Remarks

Waits until a user event of the specified *EventType* occurs. The data associated to the *EventType* is saved in the record *EventData*. See the section **Types** for the definition of the type *EventStructure* and section **Constants** for the valid values and meanings of *EventType*.

Restrictions

See also *ThisEvent*

GetActiveWindow function

xgraph.h

Function Returns the identification number of the active window.

Declaration `function GetActiveWindow: integer; external c;`

Remarks

Restrictions

See also *GetLastWindow, CloseWindow, InitWindow, SelectWindow*

GetImage procedure

xgraph.h

Function Gets the image contained inside a rectangular area.

Declaration

```
procedure GetImage(x1, y1, x2, y2 : integer; var BitMap : univ_ptr );
external c;
```

Remarks

x1, y1, x2, y2 are the coordinates of the left top and the right bottom corners of the rectangle. The image is stored in an internal area. The last parameter is not used in the TPEX version. The image stored may be copied to another location using the procedure *PutImage*

Restrictions

See also *PutImage*

GetLastWindow function

xgraph.h

Function Returns the identification number of the last window opened.

Declaration `function GetLastWindow: integer; external c;`

Remarks

Restrictions

See also *GetActiveWindow, CloseWindow, InitWindow, SelectWindow*

OutText procedure

xgraph.h

Function Output the c-style string pointed by *Texto* at the last addressed point.

Declaration `procedure OutText(Texto: univ_ptr); external c;`

Remarks Note the difference with the TP original version. *Texto* is a pointer that must point to the first character of a null terminated string. The SP predefined constant **minchar** has the value of the null character. It may be used to append the null character to a pascal-style string.

The text is written using the font and output options set by *SetTextStyle*.

After *OutText* execution, the last addressed point remains the unchanged.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *OutTextXY, SetTextStyle.*

OutTextXY procedure

xgraph.h

Function Output the c-style string pointed by *Texto* at the new addressed point (x,y).

Declaration

`procedure OutTextXY(x, y: integer; Texto: univ_ptr); external c;`

Remarks Note the difference with the TP original version. *Texto* is a pointer that must point to the first character of a null terminated string. The SP predefined constant **minchar** has the value of the null character. It may be used to append the null character to a pascal-style string.

The text is written using the font and output options set by *SetTextStyle*.

After *OutTextXY* execution, the last addressed point is the point (x,y).

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *OutText, SetTextStyle.*

paramcount function

ptipos.h

Function Returns the number of parameters passed to the program in the command line.

Declaration

```
paramcount:integer;
```

Remarks It is a *macro* substitution for the original `argc()` function in SPPascal.

See also *paramstr.*

paramstr function

pturbo.h

Function Returns a specified command line parameter.

Declaration `function paramstr(index: word): string;`

Remarks Note the difference with the original TP version. The returned value is a SP *string* type value (fixed string of 80 characters).

See also *paramcount.*

pos function

ptipos.h

Function Searches for a substring in a string.

Declaration `pos(X,Y)`

Remarks *Pos* searches for *X* within *Y*. It is a *macro* substitution for the original `index(Y,X)` function in SPPascal.

See also *copy, insert, delete.*

PutImage procedure

xgraph.h

Function Copies a rectangular saved image in a new location.

Declaration

```
procedure PutImage(x1, y1 : integer; var BitMap : univ_ptr; BitBlt : word);
external c;
```

Remarks

The rectangular image saved using the procedure *GetImage* is copied to the location at coordinates *x1, y1*. The other parameters are not used in the TPEX version.

It is a *macro*

See also *GetImage.*

PutPixel procedure

xgraph.h

Function Plots a pixel at point (*x*,*y*).

Declaration `procedure PutPixel(x, y: integer; Pixel: word); external c;`

Remarks (*x*,*y*) is set as the last addressed point. Plots the point in the current color. The last parameter is not used in the TPEX version.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *SetColor*

RANDOM function

pturbo.h

Function Returns a *real* random number between 0 and 1.

Declaration `function RANDOM: real;`

Remarks Note the difference with the original TP version. This version has no parameters. The random number generator may be initialized by assigning a value to the predefined global variable *randomize*.

See also *randomize*.

readkey function

pturbo.h

Function Reads a character from the keyboard.

Declaration `function readkey: char;`

Remarks Unlike the original TP version, this version echoes to the screen. In the TPEX Library there is no implementation of the *KeyPressed* function that often is used in conjunction with *ReadKey*.

Rectangle procedure

xgraph.h

Function Draws a rectangle using the current line style and color.

Declaration `procedure Rectangle(x1, y1, x2, y2: integer); external c;`

Remarks Draws the rectangle in the thickness and style defined by *SetLineStyle* and the current Color.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Bar, Bar3D, SetColor, SetLineStyle*

seek procedure

pturbo.h

Function Moves the current position of a file to a specified component.

Declaration

```
procedure seek(var f: univ File; r_size: integer; nr: integer);
```

Remarks Note the difference with the original TP version that has two parameters because the component size is implicit to the file type. *f* is any file type variable except text, *n* must be the size of the component type declared for the file variable and *nr* is the component number to be positioned. The first component is the number 0.

SelectWindow procedure

xgraph.h

Function Sets the active window.

Declaration

```
procedure SelectWindow(aWindow: integer);
external c;
```

Remarks

aWindow is the identification number of the window selected to be the new active window.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *CloseWindow, GetActiveWindow, GetLastWindow, InitWindow*

SetBkColor procedure

xgraph.h

Function Sets the number of the next current background color.

Declaration

```
procedure SetBkColor(ColorNum: word); external c;
```

Remarks Color numbers are defined in the file for inclusion *globals.h*. See **Colors** in the subsection **Constants** in this TPEX Library Reference.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *GetBkColor, GetColor, SetColor*

SetColor procedure

xgraph.h

Function Sets the next current drawing color.

Declaration `procedure SetColor(Color: word); external c;`

Remarks Color numbers are defined in the file for inclusion *globals.h*. See **Colors** in the subsection **Constants** in this TPEX Library Reference.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *GetBkColor, GetColor, SetBkColor*

SetFillStyle procedure

xgraph.h

Function Sets the fill color.

Declaration `procedure SetFillStyle(Pattern, Color: word); external c;`

Remarks Note the difference with the original TP version. In this version the selection of fill patterns is not yet implemented. Color numbers are defined in the file for inclusion *globals.h*. See **Colors** in the subsection **Constants** in this TPEX Library Reference.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Bar, Bar3D*

SetLineStyle procedure

xgraph.h

Function Sets the next current line style and thickness.

Declaration

```
procedure SetLineStyle(LineStyle, pattern, Thickness: word);
external c;
```

Remarks Note the difference with the original TP version. In this version the selection of line patterns is not yet implemented. Line style and Thickness numbers are defined in the file for inclusion *globals.h*. See **Line types for graphics** in the subsection **Constants** in this TPEX Library Reference.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *Arc, Bar, Bar3D, Circle, Line, LineRel, LineTo, Rectangle*

SetMouseCursor procedure

xgraph.h

Function Sets the Mouse cursor style.

Declaration `procedure SetMouseCursor(CursorType: word); external c;`

Remarks

This function is exclusive to the TPEX Library.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

set_normalvideo procedure

cturbo.h

Function Sets the screen or terminal window to the normal background and foreground colors after setting reverse video for character writing.

Declaration `procedure set_normalvideo; external c;`

Remarks Note that the TP routines: *NormVideo, HighVideo, LowVideo, TextBackground, TextColor* have not been implemented in the TPEX Library. This procedure is exclusive to the TPEX Library.

See also *set_reversevideo*

set_reversevideo procedure

cturbo.h

Function Sets the screen or terminal window in reverse video mode, i.e., background and foreground colors for character writing are swapped.

Declaration `procedure set_reversevideo; external c;`

Remarks Note that the TP routines: *NormVideo, HighVideo, LowVideo, TextBackground, TextColor* have not been implemented in the TPEX Library. This procedure is exclusive to the TPEX Library.

See also *set_normalvideo*

SetTextStyle procedure

xgraph.h

Function Sets the writing direction for *OutText, OutTextXY.*

Declaration

```
procedure SetTextStyle(tipoFont, Direction, CharSize: word);
external c;
```

Remarks Note the difference with the original TP version. In this version the selection of Fonts and size are not yet implemented. Color numbers are defined in the file for inclusion *globals.h.* See **Colors** in the subsection **Constants** in this TPEX Library Reference.

Restrictions Must be in graphic mode: the InitGraph procedure had to be successfully invoked before.

See also *OutText, OutTextXY*

str procedure

pturbo.h

Function Converts a numerical value to a string representation. According with format specification, x is interpreted as an integer, single or double variable.

Declaration

```
procedure str(var x: univ single; f: t_formato; var s: univ varstring);
```

Remarks Note the differences with the original TP version. In this version x must be an integer, single or double type variable. NOT a real type variable. Instead of two parameters, this version has three parameters. The second parameter, f is a type SP string, i.e. fixed string of up to 80 characters, may be a sting constant. It must contain a *C* style format that tells the type of the variable, the minimum total length of the converted string and the number of decimal places. The *C* style format must start with a % character optionally followed by the length and decimal places specification, and ended with the type specification. The length and decimal places specification is formed by the number of total length followed by the . character followed by the number of decimal places. The type specification must be one of the following alphabetical characters:

d for integer

f for single, fixed format

e for single, exponent format

l for double

The TPEX Library includes the functions *mkeformat* and *mkfformat* to construct a format specification to be used as the f parameters in calls to *str*.

The third parameter has the meaning of the second parameter in the original version and must be any SP *varying* declared variable, NOT of SP *alfa* or *string* types.

See also *val, mkeformat, mkfformat*

ThisEvent function

xgraph.h

Function Checks if a user event of the specified type occurred.

Declaration

```
function ThisEvent(EventType: integer; var EventData: EventStructure): integer;
external c;
```

Remarks

The function returns the identification number of the window in which the event occurred or -1 if none. The data associated to the *EventType* is saved in the record *EventData*. See the section **Types** for the definition of the type *EventStructure* and section **Constants** for the valid values and meanings of *EventType*.

See also *NextEvent*

tpAppend procedure

pturbo.h

Function Opens an existing file for appending.

Declaration `procedure tpAppend(var tpF: tpFile);`

Remarks *tpAppends* belongs to the TPEX implementation of the *tpFile* type. See *tpFile* declaration in the subsection **Types**. *tpF* is a **tpFile** variable that must have been associated with an external file using *tpAssign*

See also *tpAssign, tpClose, tpUpdate, tpRewrite*

tpAssign procedure

pturbo.h

Function Assigns the name of an external file to a *tpFile* variable.

Declaration

```
procedure tpAssign(var tpF: tpFile; fName: string; rSize: integer);
```

Remarks *tpAssign* belongs to the TPEX implementation of *tpFile* type. See *tpFile* declaration in the subsection **Types**. *tpAssign* associates the name of the external file *fName* with *tpF* and the value of *rSize* with the size of the components of the file.

See also *tpAppend, tpClose, tpUpdate, tpRewrite*

tpClose procedure

pturbo.h

Function Closes the file associated to the *tpFile* variable. close el file

Declaration `procedure tpClose(var tpF: tpFile);`

Remarks *tpClose* belongs to the TPEX implementation of *tpFile* type. See *tpFile* declaration in the subsection **Types**.

See also *tpAppend, tpAssign, tpUpdate, tpRewrite*

tpEof function

pturbo.h

Function Returns the end-of-file status of a *tpFile* variable.

Declaration `function tpEof(var tpF: tpFile): boolean;`

Remarks *tpEof* belongs to the TPEX implementation of *tpFile* type. See *tpFile* declaration in the subsection **Types**.

tpRead procedure

pturbo.h

Function Reads a file component of a external file associated to *tpF* into a variable.

Declaration `procedure tpRead(var tpF: tpFile; var p: univ char);`

Remarks *tpRead* belongs to the TPEX implementation of *tpFile* type. See *tpFile* declaration in the subsection **Types**. *tpF* is a *tpFile* variable that must have been associated with an external file using *tpAssign* The current component of the file is transfered to the location in memory of the variable *p*. The component size in bytes is set by *tpAssign*.

See also *tpAssign, tpUpdate, tpWrite, tpSeek*

tpRewrite procedure

pturbo.h

Function Creates and opens a new *tpFile* file.

Declaration `procedure tpRewrite(var tpF: tpFile);`

Remarks *tpWrite* belongs to the TPEX implementation of TP file management. See *tpFile* declaration in the subsection **Types**. *tpF* is a *tpFile* variable that must have been associated with an external file using *tpAssign*

See also *tpAssign, tpUpdate, tpAppend*

tpSeek procedure

pturbo.h

Function Moves the current position of a TP file to a specified component.

Declaration `procedure tpSeek(var tpF: tpFile; nr: integer);`

Remarks *tpSeek* belongs to the TPEX implementation of TP file management. See *tpFile* declaration in the subsection **Types**. *tpF* is a *tpFile* variable that must have been associated with an external file using *tpAssign* The current file position is moved to component number *nr*. Components of a file are numbered starting from 0. The component size in bytes is set by *tpAssign*.

See also *tpAssign, tpUpdate*

tpUpdate procedure

pturbo.h

Function Opens an existing file.

Declaration `procedure tpUpdate(var tpF: tpFile);`

Remarks *tpUpdate* belongs to the TPEX implementation of TP file management. See *tpFile* declaration in the subsection **Types**. *tpF* is a *tpFile* variable that must have been associated with an external file using *tpAssign*

See also *tpAssign, tpRewrite, tpClose*

tpWrite procedure

pturbo.h

Function Writes a variable into a file component.

Declaration `procedure tpWrite(var tpF: tpFile; var p: univ char);`

Remarks *tpWrite* belongs to the TPEX implementation of *tpFile* type. See *tpFile* declaration in the subsection **Types**. *tpF* is a *tpFile* variable that must have been associated with an external file using *tpAssign* The number of bytes specified as component size for the file are transfered from the location in memory of the variable *p* to the current position of the file adding or updating a component. The component size in bytes is set by *tpAssign*.

See also *tpAssign, tpUpdate, tpRead, tpSeek*

TTYFileName procedure

pturbo.h

Function Gets from the Operating System the name as file of the keyboard IO stream.

Declaration `procedure TTYFileName(var TTYName: string);`

Remarks The name is returned in the variable referenced by *TTYName*. This procedure is exclusive to the TPEX Library. The name returned may be used to assign as value to a string variable used in a *rewrite* procedure call. Using the string variable it is possible to switch a text output between the standard output and an user specified file.

upcase function

ptipos.h

Function Returns the corresponding uppercase character.

Declaration `upcase(X)`

Remarks It is a macrosubstitution for the equivalent *toupper* C function.

val procedure

pturbo.h

Function Converts a string value to its numeric representation.

Declaration `procedure val(var s: univ varstring; var x: univ double; tipo: integer; var codErr: integer);`

Remarks *s* is a varstring type variable. *x* is a numeric variable of type in concordance with the values of the integer parameter *tipo*. *s* must contain a valid character representation of a number of the type specified by *tipo*. If not, the procedure returns in *codErr* the value 1; otherwise it returns 0. The values for *tipo* as defined en the **Constants** section are:

pINTEGER = 0;

pLONGINT = 1;

pSINGLE = 2;

pDOUBLE = 3;

Note the differences with the original TP version. In this SP version the user has to indicate explicitly the type of the variable, giving a proper value to the parameter *tipo*. In the TP version this parameter isn't needed. That's why the SP version has four parameters instead of tree. The last parameter in both versions is an error code but in the SP version the returned values can be only 0 or 1. In the TP version in case of error, it returns the position of the offending character in the input string.

References

[1] Borland International, Inc. *Turbo Pascal version 6.0 Reference Manual*

[2] Borland International, Inc. *Turbo Pascal version 6.0 Library Reference*

[3] Harbison, S. and Steele, G.: *C: A Reference Manual* Prentice Hall Software Series.

[4] Jensen, K. and Wirth, N.: *Pascal User Manual and Report* Springer Verlag.

[5] Kernighan, B. and Ritchie, D.: *The C Programming Language* Prentice Hall Software Series.

[6] Shumucker, Kurt J. *Object-Oriented Programming for the Macintosh* Hayden Books, 1986.

[7] Sun Microsystems, Inc *SPARC Pascal Compiler 3.0.3, Reference Manual*