Transporte de Programación por Emulación de Bibliotecas

(Software porting by Library Emulation)

Marta Sananes
Rafael Tineo
Instituto de Estadística Aplicada y Computación (IEAC), FACES
Universidad de Los Andes
sananes@faces.ula.ve
tineo@ing.ula.ve

XLVII Convención Anual ASOVAC
Valencia, Noviembre 1997

Abstract

Introducción: El desarrollo multiplataforma de software es conveniente si se desea abarcar un rango amplio de tipos de usuarios que se diferencien ya sea por variedad de necesidades o por disponibilidad de recursos. Para los desarrolladores que adopten este punto de vista, es importante diseñar sus productos de forma tal que se facilite la portabilidad a diversas plataformas. Se adopta como estrategias de portabilidad: (1) Construcción de Módulos de Emulación de las Bibliotecas standard de soporte de aplicaciones en la plataforma de partida sobre Bibliotecas standard disponibles con la capacidad de rendir tareas equivalentes en la de llegada. (2) Para el diseño de programación, uso del criterio de separación en componentes Núcleo-Control- Presentación, bajo el cual los módulos de Emulación pertenecen a la capa de Control. En este trabajo se describen casos de transporte de un sistema desarrollado originalmente para plataforma PC-DOS-BorlandGRAPH a las plataformas SPARC-SunOS-Xlib y PC/WINDOWS. **Objetivos**: (1) Conservar al máximo el código original de la plataforma de partida (2) Proporcionar a los usuarios el mismo estilo visual y operacional en las distintas plataformas (3) Aprovechar con eficiencia los recursos propios de las plataformas destino (4) Producir Bibliotecas de uso general que permitan el fácil transporte de otras aplicaciones en la misma plataforma de origen o similares. **Metodología**: Desarrollo de las siguientes etapas: (1) Estudio de factibilidad con exploración de recursos disponibles en la plataforma destino (2) Estudio detallado de las Bibliotecas de soporte en el destino (3) Preparación de interfaces de enlace hacia las Bibliotecas en el destino (4) Diseño, construcción, prueba y ajuste de cada elemento emulado. (5) Ajustes necesarios en

el código original para adaptación a la plataforma de destino. **Resultados**: Biblioteca conformada por Módulos de Emulación del contenido de las Bibliotecas nativas: comunicación en modo texto, graficación, gestión de archivos, varios. Versión transportada del sistema original haciendo uso de la Biblioteca desarrollada. Manuales de usuario de la Biblioteca de Emulación y del Sistema transportado. Conjunto de pruebas de verificación de consistencia entre plataformas. **Conclusiones**: La estrategia de emulación si bien tiene un costo inicial -su construcción- facilita la futura conversión de otras aplicaciones sobre la misma plataforma original. La conservación del código original facilita el mantenimiento y la generación de nuevas versiones, que pueden liberarse simultáneamente sin retrasos significativos, pues el trabajo de implantación de cambios se homogeiniza. La estrategia es aplicable a diversas plataformas de destino.

Reconocimientos

A la Universidad de Los Andes, Facultad de Ciencias Económicas y Sociales e Instituto de Estadística Aplicada y Computación por el permiso durante el cual se realizó el trabajo de transporte del GLIDER a plataforma UNIX/Solaris.

Al Consejo de Desarrollo Científico y Humanístico de la ULA por el fiananciamiento al Proyecto del cual es parte este trabajo.

A *San Diego State University, Departament of Matemathical Sciences* e *Interdisciplinary Researh Center*, por recibirme como profesor Visitante y por concederme el uso de sus recursos.

Al Prof. Jose Castillo, de SDSU, por facilitar todos los detalles de nuestra visita y por su estímulo en la realización del trabajo.

Introducción

Este trabajo se realizó como parte del cumplimiento del objetivo de construír versiones multiplataforma dentro del proyecto de desarrollo del Lenguaje de Simulación GLIDER [1] y de su ambiente integrado.

El trabajo de transportar el precompilador del Lenguaje de Simulación GLIDER y su Biblioteca de Simulación a la plataforma UNIX/Solaris/ [2] XWindow [3] fué realizado por Marta Sananes durante el transcurso de un año de permiso

[1] Diseño original: C. Domingo y M. Hernandez. Diseño extendido: C.Domingo, J.G. Silva, G.Tonella. Grupo de Desarrollo GLIDER: C.Domingo, G. Tonella, J.G. Silva, H. Hoeger, T. Jiménez, S. Quiroz, M. Sananes, O. Terán, K. Tucci. (ULA, Venezuela). Consultores: R. del Canto (ULA), C. Zoltan (USB). ©CESIMO-IEAC, Universidad de Los Andes, Mérida, Venezuela

[2] UNIX es marca registrada de UNIX System Laboratories, Inc. Solaris y todos los productos de SunSoft san marcas o marcas registradas de Sun Microsystems, Inc.

[3] El Sistema XWindow es producto y marca registrada del Massachusetts Institute of Technology

de la ULA a M. Sananes en el que estuvo como visitante en San Diego State University, CA.

El Lenguaje de Simulación GLIDER se diseñó tomando el lenguaje de propósito general TurboPascal [4], una extensión popular a la definición Pascal standard, como lenguaje subyacente. La implementación del Compilador y Biblioteca para la versión original en plataforma PC/DOS se programó con el mismo TurboPascal (versión 6.0) para PC/DOS. Un objetivo final del Proyecto GLIDER es tener el lenguaje de simulación implementado y mantenido para UNIX/XWindow además de para PC/DOS y PC/Windows.

Para realizar el trabajo de transporte o conversión a nuevas plataformas se ha adoptado la estrategia de sustituír las Bibliotecas de soporte o Librerías -como se dice comunmente- de la plataforma de partida, MSDOS/BORLAND TURBO Pascal en nuestro caso, por Librerías que las *emulen* en las nuevas plataformas.

La aplicación de esta estrategia al caso de conversión a la plataforma UNIX (Solaris)/XWindow condujo al desarrollo de una Librería de Emulación parcial de Librerías standard de BORLAND TURBO Pascal, a la cual le dimos por nombre **TPEX** (Turbo Pascal Emulation for uniX solaris).

En la continuación del desarrollo del Proyecto ya en la ULA, Rafael Tineo revisó y expandió la Librería TPEX dotándola de capacidades no existentes en la Librería original con el fin de permitirle a los programadores aprovechar los recursos y facilidades del ambiente X para UNIX.

El trabajo de transporte a distintas plataformas se facilita si en la programación se adopta un criterio de separación de componentes o módulos en tres categorías [5]:

- **Núcleo**: independiente de plataforma
- **Presentación**: totalmente dependiente de plataforma
- **Control o Enlace**: interfaz para Presentación

Este enfoque aplicado al desarrollo de nuevos productos de programación, facilita conversión a otras plataformas.

[4] Turbo Pascal y todos los productos de Borland son marcas registradas de Borland International, Inc.

[5] Metodología delineada en solicitud de financiamiento al programa BID/CONICIT por Marta Sananes y Giorgio Tonella

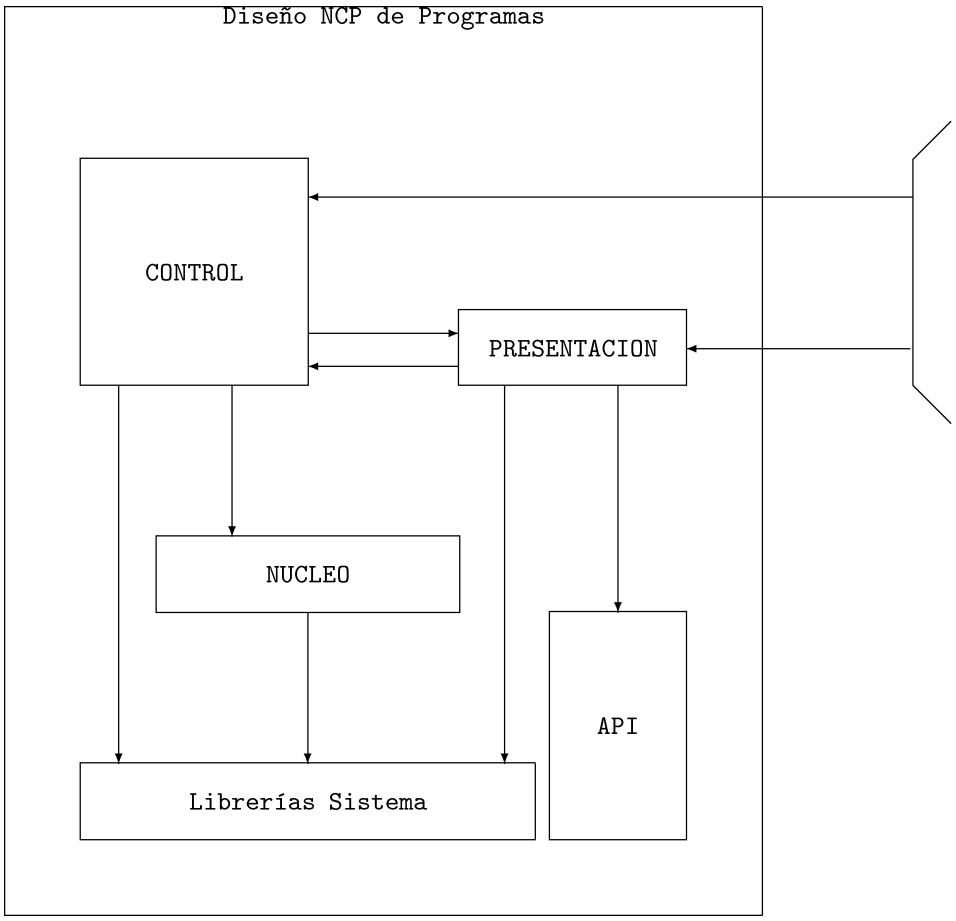

Figure 1: Estructuración de Programas NCP

Objetivos

Para la realización de trabajos de conversión o transporte hacia distintas plataformas, nos fijamos los siguientes criterios:

1. Conservar al máximo posible el código original de la plataforma de partida, para reducir al mínimo la necesidad de reprogramar, ahorrar tiempo y evitar introducir errores.
2. Proporcionar a los usuarios similar estilo visual y operacional en las distintas plataformas
3. Aprovechar con eficiencia los recursos propios de las plataformas destino
4. Producir Librerías de uso general que permitan el fácil transporte de otras aplicaciones en la misma plataforma de origen o similares.

Los objetivos generales del trabajo de conversión fueron:

1. Preparar una versión para UNIX a partir de la versión original existente PC/DOS del Lenguaje de Simulación GLIDER, que comprende el Compilador GLIDER y la Biblioteca GLIDER de Procedimientos de Interfaz y de Simulación.
2. Preparar versión en la máquina **zeus** del Laboratorio de Investigación de FACES que está instalada como plataforma SunOS/SUN Pascal/C. Hubo que hacer algunos ajustes a la versión Solaris/SPARC para salvar las diferencias entre el Sistema Operativo y los compiladores Pascal y C con respecto a los de SDSU.
3. Preparar una versión para PC/WINDOWS a partir también de la versión existente PC/DOS.

Metodología

Para cada conversión se pasa por las siguientes etapas:

1. Estudio de factibilidad
2. Estudio detallado de las Librerías de soporte y lenguajes disponibles en el destino
3. Preparación de interfaces de enlace hacia las Librerías en el destino
4. Diseño, construcción, prueba y ajuste de cada elemento emulado.
5. Ajustes necesarios en el código original para adaptación a la plataforma de destino, cuando -como es frecuente- no hay compatibilidad total entre los lenguajes de desarrollo

Estudio de factibilidad

Se inició con exploración de los recursos disponibles en las plataformas de destino.

Destino: UNIX/Solaris

La diversidad de extensiones existentes a Pascal y las limitaciones de la standard, hacen que en principio no sea recomendable el uso de Pascal para desarrollo multiplataforma de software, siendo preferible usar los lenguajes C o C++. Sin embargo, la versión UNIX Pascal para estaciones de trabajo y servidores Sun, SPARC Pascal para Solaris 1.x y 2.x, además de poseer característica extendidas -varios de ellos similares conceptualmente a algunas de TurboPascal- provee la facilidad de vincular módulos programados en SPARC Pascal con módulos programados en otros lenguajes, como C, C++ o FORTRAN. También SPARC Pascal acepta el uso directo de las Librerías completas de C o C++ y de Xlib y XView [6], componentes del sistema XWindow para construir Interfaces Gráficas de Usuario. Estas características facilitan la conversión de programas desde TurboPascal a la familia de computadoras de Sun -o a cualquier otra que admita el mismo sistema operativo, Solaris o compatible- en dos sentidos:

- Programando módulos o Librerías en C o C++ que implementen las extensiones de TurboPascal. En algunos casos puede hacerse directamente en el mismo SPARC Pascal. Esto permite conservar al máximo el código original y se evita la fastidiosa y riesgosa tarea de reprogramación.
- Si se opta por una conversión total a C o C++, permite hacerlo en forma incremental, lo cual minimiza o por lo menos reduce el riesgo de cometer errores de conversión.

Se exploraró la posibilidad de usar un ambiente de alto nivel basado en XWindow, como es el XView. Se concluyó que un ambiente de alto nivel es excelente para el desarrollo de aplicaciones nuevas diseñadas desde el principio dentro del modelo de procesamiento por eventos o para una reingeniería total, pero no pareció conveniente aplicarlo a la conversión de una aplicación diseñada, como era el caso, para procesamiento secuencial. Por lo tanto, se prefirió usar el nivel bajo del sistema XWindow, la Librería **Xlib**.

El estudio detallado de las Librerías de soporte y lenguajes disponibles en el destino comprendió el estudio de manuales y pruebas con:

- SPARC Pascal
- SPARC C
- X11/Xlib

[6] Sistema original de Sun Microsystems, ahora parte del estandard XWindow

Concepto	**TURBO Pascal**	**SPARC Pascal**
Modularidad	Sí: Unit 's	Sí: module 's
Interfaz de módulo y referencia	Constructos del lenguaje	Archivo separado y macrolenguaje
Mezcla con otros lenguajes	Sólo PC-Macroassembler	Acceso a bibliotecas de C, C++, FORTRAN
Vinculación con módulos compilados de otros lenguajes	PC-Macroassembler; C: limitado	C, C++, FORTRAN
Programación de Objetos	Sí	No
Uso de Mayúsculas/ minúsculas	Indiferente	Diferencia
TAD cadena	Tipo `string:` variable hasta 255 cars.	Tipos `alfa, string:` longitud fija `varying`[$<$ *longitud* $>$] `of char`: hasta 65535 cars.
Inicialización de variables	Definidas como constantes pertenecientes a *tipos*	Se les asigna valor en su declaración
Intercepción de Errores de E/S	Directivas al compilador; Variable de estado `IOResult`	Sustitución de procesamiento standard
Tipos de funciones y procedimientos variables	Se define: $<$ *prototipo* $>$	Se define: $\uparrow<$ *prototipo* $>$
Parámetros sin tipo	var $<$ *parametro* $>$;	$<$ *parametro* $>$: univ $<$ *cualquier* $-$ *tipo* $>$
Tipo pointer genérico	`pointer`	`univ-ptr`

Figure 2: Cuadro Comparativo de TURBO Pascal con SPARC-Pascal

El manejo de módulos del SPARC Pascal es enteramente parecido al del C: ambos admiten la inclusión de instrucciones en macro lenguaje. El macro lenguaje, que es interpretado por un Precompilador, permite coordinar la organización multimodular. Por lo tanto, la estructura modular de la versión SPARC Pascal puede tomarse casi idénticamente para una versión en C o C++.

El siguiente cuadro resume caraterísticas y compara las facilidades disponibles en las versiones de compiladores Pascal, el TP en el origen y el SP, en el destino del transporte.

La conversión al ambiente del sistema operativo UNIX puede hacerse a dos niveles: al UNIX simple o al UNIX extendido con capacidad de presentación gráfica en ventanas. El ambiente gráfico de ventanas estandard para UNIX es XWindow. Las Librerías de X son directamente accesibles para programar en lenguajes C o C++. Durante la primera mitad del período de visita en SDSU se hizo la conversión al UNIX simple y en la segunda mitad, se completó una versión con producción de salida gráfico basada en la Librería **Xlib** de X.

Para el estudio de la Librería XLib fué muy útil el texto de Yang y Ali, [8], algunos de cuyos ejemplos usamos como punto de partida para desarrollar la emulación de funciones y procedimientos de la Librería GRAPH.

Destino: WINDOWS

Se adoptó como lenguaje de desarrollo para la plataforma PC/WINDOWS el ObjectPascal [7] [6] , incluído dentro del ambiente de Desarrollo Rápido de Aplicaciones -RAD- DELPHI, producto de BORLAND introducido al mercado en el año 1995. Este ObjectPascal es una extensión Orientada a Objetos de Pascal en cuyo diseño original participó el mismo Wirth y que ha sido revisada y extendida por BORLAND. Es altamente compatible con TURBO Pascal. El ambiente DELPHI proporciona herramientas de programación visual y de desarrollo con SMDB en modelo Cliente/Servidor. Están disponibles nuevas versiones para WINDOWS 95 y NT.

Por la compatibilidad con el lenguaje de desarrollo original y la relativa facilidad del ambiente para generar aplicaciones para WINDOWS, la tarea de conversión ha sido casi inmediata. Sin embargo, también fué necesario *emular* algunos comportamientos y en particular, emular la Librería GRAPH, ya que la API de WINDOWS que DELPHI envuelve en la subclase `Canvas` es de tipo más bien elemental.

WDgraph es el nombre de la Unit de emulación. Puede ser convertida fácilmente a Librería propiamente dicha para WINDOWS, cambiándole la interfaz a componente de tipo DDL.

Resultados

Biblioteca conformada por Módulos de Emulación del contenido de las Bibliotecas nativas: comunicación en modo texto, graficación, gestión de archivos (para SPARC Pascal), varios. Para el destino UNIX/Solaris: Versión transportada del sistema original haciendo uso de la Biblioteca desarrollada. Manuales de usuario de la Biblioteca de Emulación y del Sistema transportado. Conjunto de pruebas de verificación de consistencia entre plataformas.

[7] Originalmente diseñado y hecho por Apple Co. para su serie Macintosh

Librería TPEX

La Librería TPEX *TURBOPascal Emulation Library for UNIX* tiene la siguiente composición de archivos:

- `ptipos.h`: Archivo para incluír en módulos y programas SPARC Pascal. Contiene definiciones de tipos propios de TURBO Pascal. También contiene algunos casos de emulación de rutinas que se implementaron por simple macro-susitución.

- `globals.h`: Archivo para incluír en módulos y programas SPARC Pascal. Contiene declaraciones de constantes y variables propias del ambiente TURBO Pascal.

- `pturbo.h`: Archivo para incluír en módulos y programas SPARC Pascal. Contiene los *prototipos* o encabezados *headers* de las funciones y procedimientos incluídos en el componente `pturbo` de la Librería. El componente `pturbo` contiene el subconjunto de rutinas emuladas para SPARC Pascal construídas en el mismo SPARC Pascal. También contiene prototipos de un subconjunto de funciones y procedimientos pertenecientes a la Librería standard del lenguaje C. [3]

- `cturbo.h`: Archivo para incluír en módulos y programas SPARC Pascal. Contiene los *prototipos* o encabezados *headers* de las funciones y procedimientos incluídos en el componente `cturbo` de la Librería. El componente `cturbo` contiene el subconjunto de rutinas emuladas para SPARC Pascal construídas en lenguaje C.

- `pxgraph.h`: Archivo para incluír en módulos y programas SPARC Pascal. Contiene los *prototipos* o encabezados *headers* de las funciones y procedimientos incluídos en el componente `xgraph` de la Librería. El componente `xgraph` contiene el subconjunto de rutinas emuladas para SPARC Pascal de la Librería `GRAPH` de TURBO Pascal. Fué construída con lenguaje C usando el nivel `XLib` de la Librería `X11` para presentación gráfica XWindow.

- `xgraph.h`: Archivo para incluír en archivos fuentes en lenguaje C. Contiene los *prototipos* o encabezados *headers* de las funciones y procedimientos incluídos en el componente `xgraph` de la Librería.

- `ccturbo.h`: Archivo para incluír en archivos fuentes en lenguaje C. Contiene los *prototipos* o encabezados *headers* de las funciones y procedimientos incluídos en el componente `cturbo` de la Librería.

- `pturbo.o`: Archivo tipo *Object* -compilado- del componente `pturbo` de la Librería.

- `cturbo.o`: Archivo tipo *Object* -compilado- del componente `cturbo` de la Librería. Puede ser usado en progrmas SPARC Pascal o en programas C incluyendo los archivos `cturbo.h` y `ccturbo.h`, respectivamente.
- `xgraph.o`: Archivo tipo *Object* -compilado- del componente `xgraph` de la Librería. Puede ser usado en progrmas SPARC Pascal o en programas C incluyendo los archivos `pxgraph.h` y `xgraph.h`, respectivamente.

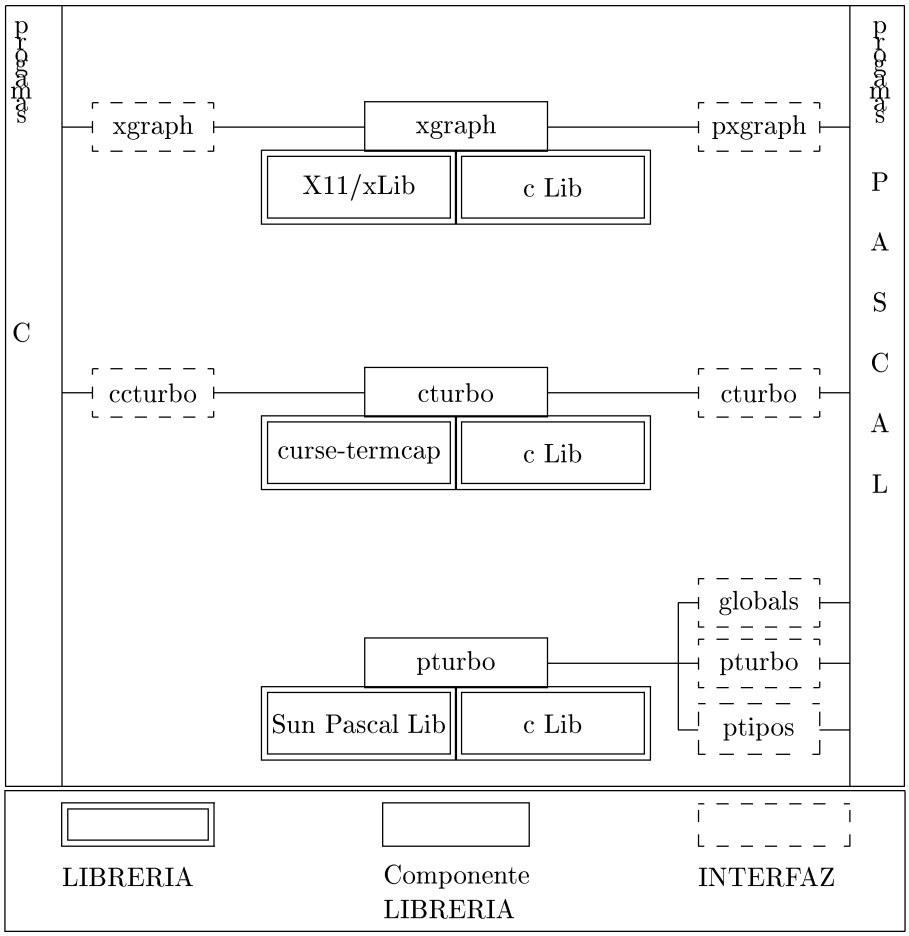

Figure 3: Librería TPEX en ambiente UNIX/Solaris

Conclusiones

La estrategia de emulación tiene el costo inicial del proceso de su construcción pero se gana en cuanto facilita la futura conversión de otras aplicaciones sobre la misma plataforma original.

La conservación del código original facilita el mantenimiento y la generación de nuevas versiones, que pueden liberarse casi simultáneamente sin retrasos significativos, ya que el trabajo de implantación de cambios se homogeiniza.

La estrategia es aplicable a diversas plataformas de destino. Es recomendable la adopción de criterios de diseño de aplicaciones con el concepto NCP o similar, que permite aislar los módulos con dependencias de plataforma.

Anexos

TPEX: Manual de Referencia

A continuación se incluyen algunas páginas de muestra del Manual de Referencia de la Librería.

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]

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.*

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.*

clrscr procedure

---

cturbo.h

Function Clears the terminal window or screen.

Declaration `procedure clrscr; external c;`

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.

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*.

InitGraph procedure

---

xgraph.h

Function Opens a X window for graphic output.

Declaration

```
procedure InitGraph(var gd, gm: integer; var ptdr: univ string);
external c;
```

Remarks In this version the parameters are not used, they are declared just for compatibility with the original TP. In future versions more than one window could be opened and the parameters will be used for specifying size and caption for the new window.

See also *CloseGraph*

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.

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*

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*

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.

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*

[8] Yang, Cui-Qing and Ali, Mahir S. *Xlib by Example.* Academic Press Professional, 1994.

[9] Cutler, E.; Gilly, D; O'Reilly. *The X Window System in a Nutshell.* O'Reilly & Associates, Inc., 1992.