

```
(*Foreign*)
DEFINITION MODULE ASCII;
(*
  Module:      Mnemonics for ASCII control characters
  From:        ???
  Version:     $Id: ASCII.def,v 1.3 1994/06/24 12:48:07 ceriel Exp $
*)
```

```
CONST
```

```
  nul = 00C;   soh = 01C;   stx = 02C;   etx = 03C;
  eot = 04C;   enq = 05C;   ack = 06C;   bel = 07C;
  bs  = 08C;   ht  = 11C;   lf  = 12C;   vt  = 13C;
  ff  = 14C;   cr  = 15C;   so  = 16C;   si  = 17C;
  dle = 20C;   dc1 = 21C;   dc2 = 22C;   dc3 = 23C;
  dc4 = 24C;   nak = 25C;   syn = 26C;   etb = 27C;
  can = 30C;   em  = 31C;   sub = 32C;   esc = 33C;
  fs  = 34C;   gs  = 35C;   rs  = 36C;   us  = 37C;
  del = 177C;
```

```
END ASCII.
```

```
DEFINITION MODULE Arguments;
```

```
(*
  Module:      Access to program arguments and environment
  Author:      Ceriel J.H. Jacobs
  Version:     $Id: Arguments.def,v 1.3 1994/06/24 12:48:13 ceriel Exp $
*)
```

```
VAR   Argc: CARDINAL;
```

```
(* Number of program arguments, including the program name, so it is at least 1. *)
```

```
PROCEDURE Argv (argnum : CARDINAL;      VAR argument : ARRAY OF CHAR) : CARDINAL;
```

```
(* Stores the "argnum'th" argument in "argument", and returns its length,
  including a terminating null-byte. If it returns 0, the argument was not
  present, and if it returns a number larger than the size of "argument",
  "argument" was'nt large enough.
  Argument 0 contains the program name.
*)
```

```
PROCEDURE GetEnv (name : ARRAY OF CHAR;  VAR value : ARRAY OF CHAR) : CARDINAL;
```

```
(* Searches the environment list for a string of the form 'name=value'
  and stores the value in "value", if such a string is present.
  It returns the length of the "value" part, including a terminating
  null-byte. If it returns 0, such a string is not present, and
  if it returns a number larger than the size of the "value",
  "value" was'nt large enough.
  The string in "name" must be null_terminated.
*)
```

```
END Arguments.
```

DEFINITION MODULE ArraySort;

(*
Module: Array sorting module
Author: Cerial J.H. Jacobs
Date: \$Id: ArraySort.def,v 1.2 1994/06/24 12:48:16 cerial Exp \$

Interface is like the qsort() interface in C, so that an array of values can be sorted. This does not mean that it has to be an ARRAY, but it does mean that the values must be consecutive in memory, and the order is the "memory" order. The user has to define a comparison procedure of type CompareProc. This routine gets two pointers as parameters. These are pointers to the objects that must be compared. The sorting takes place in ascending order, so that f.i. if the result of the comparison is "less", the first argument comes in front of the second.

*)

FROM SYSTEM IMPORT ADDRESS; (* no generics in Modula-2, sorry *)

TYPE CompareResult = (less, equal, greater);
CompareProc = PROCEDURE (ADDRESS, ADDRESS) : CompareResult;

PROCEDURE Sort (base: ADDRESS; (* address of array *)
nel: CARDINAL; (* number of elements in array *)
size: CARDINAL; (* size of each element *)
compar: CompareProc); (* the comparison procedure *)

END ArraySort.

DEFINITION MODULE Conversions;

(*
Module: Numeric-to-string conversions
Author: Cerial J.H. Jacobs
Version: \$Id: Conversion.def,v 1.3 1994/06/24 12:48:29 cerial Exp \$
*)

PROCEDURE ConvertOctal (num, len: CARDINAL; VAR str: ARRAY OF CHAR);

(* Convert number "num" to right-justified octal representation of "len" positions, and put the result in "str".
If the result does not fit in "str", it is truncated on the right.

*)

PROCEDURE ConvertHex (num, len: CARDINAL; VAR str: ARRAY OF CHAR);

(* Convert a hexadecimal number to a string *)

PROCEDURE ConvertCardinal (num, len: CARDINAL; VAR str: ARRAY OF CHAR);

(* Convert a cardinal number to a string *)

PROCEDURE ConvertInteger (num: INTEGER; len: CARDINAL; VAR str: ARRAY OF CHAR);

(* Convert an integer number to a string *)

END Conversions.

```

DEFINITION MODULE CSP;
(*
  Module:      Communicating Sequential Processes
  From:        "A Modula-2 Implementation of CSP",
               M. Collado, R. Morales, J.J. Moreno,
               SIGPlan Notices, Volume 22, Number 6, June 1987.
  Version:     $Id: CSP.def,v 1.3 1994/06/24 12:48:22 ceriel Exp $

  See this article for an explanation of the use of this module.
*)

FROM SYSTEM IMPORT BYTE;

TYPE Channel;

PROCEDURE COBEGIN;

(* Beginning of a COBEGIN .. COEND structure *)

PROCEDURE COEND;

(* End of a COBEGIN .. COEND structure *)

PROCEDURE StartProcess (P: PROC);

(* Start an anonymous process that executes the procedure P *)

PROCEDURE StopProcess;

(* Terminate a Process (itself) *)

PROCEDURE InitChannel (VAR ch: Channel);

(* Initialize the channel ch *)

PROCEDURE GetChannel (ch: Channel);

(* Assign the channel ch to the process that gets it *)

PROCEDURE Send (data: ARRAY OF BYTE; VAR ch: Channel);

(* Send a message with the data to the cvchannel ch *)

PROCEDURE Receive (VAR ch: Channel; VAR dest: ARRAY OF BYTE);

(* Receive a message from the channel ch into the dest variable *)

PROCEDURE SELECT (n: CARDINAL);

(* Beginning of a SELECT structure with n guards *)

PROCEDURE NEXTGUARD (): CARDINAL;

(* Returns an index to the next guard to be evaluated in a SELECT *)

PROCEDURE GUARD (cond: BOOLEAN; ch: Channel; VAR dest: ARRAY OF BYTE): BOOLEAN;

(* Evaluates a guard, including reception management *)

PROCEDURE ENDSELECT (): BOOLEAN;

(* End of a SELECT structure *)

END CSP.

```

```

(*$Foreign *)
DEFINITION MODULE EM;

(*
  Module:      Interface to some EM instructions and data
  Author:      Cerieel J.H. Jacobs
  Version:     $Id: EM.def,v 1.4 1994/06/24 12:48:36 ceriel Exp $
*)

FROM SYSTEM IMPORT ADDRESS;

TYPE  TrapHandler = PROCEDURE (INTEGER);

PROCEDURE FIF (arg1, arg2: LONGREAL; VAR intres: LONGREAL) : LONGREAL;

(* multiplies arg1 and arg2, and returns the integer part of the result in "intres" and
   the fraction part as the function result.
*)

PROCEDURE FEF (arg: LONGREAL; VAR exp: INTEGER) : LONGREAL;

(* splits "arg" in mantissa and a base-2 exponent.
   The mantissa is returned, and the exponent is left in "exp".
*)

PROCEDURE TRP (trapno: INTEGER);

(* Generate EM trap number "trapno"      *)

PROCEDURE SIG (t: TrapHandler): TrapHandler;

(* install traphandler t; return previous handler      *)

PROCEDURE FILN (): ADDRESS;

(* return current program file-name. This only works if file-name and line-number
   generation is not disabled during compilation
*)

PROCEDURE LINO (): INTEGER;

(* return current program line-number. This only works if file-name and line-number
   generation is not disabled during compilation
*)

END EM.

```

```

(*$Foreign*)
DEFINITION MODULE Epilogue;
(*
  Module:      install module termination procedures to be called atprogram termination
  Author:      Cerieel J.H. Jacobs
  Version:     $Id: Epilogue.def,v 1.5 1994/06/24 12:48:42 ceriel Exp $

  MODULA-2 offers a facility for the initialization of modules, but there is no
  mechanism to have some code executed when the program finishes. This module is a
  feeble attempt at solving this problem.
*)

PROCEDURE CallAtEnd (p: PROC): BOOLEAN;

(* Add procedure "p" to the list of procedures that must be executed when the
   program finishes.
   When the program finishes, these procedures are executed in the REVERSE
   order in which they were added to the list.
   This procedure returns FALSE when there are too many procedures to be
   called (the list has a fixed size).
*)

END Epilogue.

```

```

DEFINITION MODULE InOut;
(*
  Module:      Wirth's Input/Output module
  From:        "Programming in Modula-2", 3rd, corrected edition, by N. Wirth
  Version:     $Id: InOut.def,v 1.4 1994/06/24 12:48:45 ceriel Exp $
*)

CONST EOL = 12C;

VAR   Done : BOOLEAN;
      termCH : CHAR;

PROCEDURE OpenInput (defext: ARRAY OF CHAR);
(* Request a file name from the standard input stream and open this file for reading.
   If the filename ends with a '.', append the "defext" extension.
   Done := "file was successfully opened".
   If open, subsequent input is read from this file.
*)

PROCEDURE OpenOutput (defext : ARRAY OF CHAR);
(* Request a file name from the standard input stream and open this file for writing.
   If the filename ends with a '.', append the "defext" extension.
   Done := "file was successfully opened".
   If open, subsequent output is written to this file.
   Files left open at program termination are automatically closed.
*)

PROCEDURE OpenInputFile (filename: ARRAY OF CHAR);
(* Like OpenInput, but filename given as parameter.
   This procedure is not in Wirth's InOut.
*)

PROCEDURE OpenOutputFile (filename: ARRAY OF CHAR);
(* Like OpenOutput, but filename given as parameter.
   This procedure is not in Wirth's InOut.
*)

PROCEDURE CloseInput;
(* Close input file. Subsequent input is read from the standard input stream. *)

PROCEDURE CloseOutput;
(* Close output file. Subsequent output is written to the standard output stream. *)

PROCEDURE Read (VAR ch : CHAR);
(* Read a character from the current input stream and leave it in "ch".
   Done := NOT "end of file".
*)

PROCEDURE ReadString (VAR s : ARRAY OF CHAR);
(* Read a string from the current input stream and leave it in "s". A string is any
   sequence of characters not containing blanks or control characters; leading blanks
   are ignored. Input is terminated by any character <= " ".
   This character is assigned to termCH.
   DEL or BACKSPACE is used for backspacing when input from terminal.
*)

PROCEDURE ReadInt (VAR x : INTEGER);
(* Read a string and convert it to INTEGER.
   Syntax: integer = ['+'|'-'] digit {digit}.
   Leading blanks are ignored.
   Done := "integer was read".
*)

```

```
PROCEDURE ReadCard (VAR x : CARDINAL);

(* Read a string and convert it to CARDINAL.
  Syntax: cardinal = digit {digit}.
  Leading blanks are ignored.
  Done := "cardinal was read".
*)

PROCEDURE Write (ch : CHAR);

(* Write character "ch" to the current output stream. *)

PROCEDURE WriteLn;

(* Terminate line. *)

PROCEDURE WriteString (s : ARRAY OF CHAR);

(* Write string "s" to the current output stream *)

PROCEDURE WriteInt (x : INTEGER; n : CARDINAL);

(* Write integer x with (at least) n characters on the current output stream.
  If n is greater than the number of digits needed, blanks are added
  preceding the number.
*)

PROCEDURE WriteCard (x, n : CARDINAL);

(* Write cardinal x with (at least) n characters on the current output stream.
  If n is greater than the number of digits needed, blanks are added
  preceding the number.
*)

PROCEDURE WriteOct (x, n : CARDINAL);

(* Write cardinal x as an octal number with (at least) n characters on the
  current output stream. If n is greater than the number of digits needed,
  blanks are added preceding the number.
*)

PROCEDURE WriteHex(x, n : CARDINAL);

(* Write cardinal x as a hexadecimal number with (at least) n characters on
  the current output stream.
  If n is greater than the number of digits needed, blanks are added
  preceding the number.
*)

END InOut.
```

```
DEFINITION MODULE MathLib0;
```

```
(*  
  Module:      Some mathematical functions  
  From:        "Programming in Modula-2", 3rd, corrected edition, by N. Wirth  
  Version:     $Id: MathLib0.def,v 1.4 1994/06/24 12:48:55 ceriel Exp $
```

```
  Exists for compatibility.
```

```
  A more elaborate math lib can be found in Mathlib.def
```

```
*)
```

```
PROCEDURE sqrt (x : REAL) : REAL;  
PROCEDURE exp (x : REAL) : REAL;  
PROCEDURE ln (x : REAL) : REAL;  
PROCEDURE sin (x : REAL) : REAL;  
PROCEDURE cos (x : REAL) : REAL;  
PROCEDURE arctan (x : REAL) : REAL;  
PROCEDURE real (x : INTEGER) : REAL;  
PROCEDURE entier (x : REAL) : INTEGER;  
END MathLib0.
```

```
DEFINITION MODULE Mathlib;
```

```
(*  
  Module:      Mathematical functions  
  Author:      Ceriel J.H. Jacobs  
  Version:     $Id: Mathlib.def,v 1.5 1994/06/24 12:49:01 ceriel Exp $
```

```
*)
```

```
(* Some mathematical constants: *)
```

```
CONST
```

```
(* From:      Computer Approximations  
              Hart, Cheney, e.a.  
              The SIAM Series in Applied Mathematics  
              John Wiley & Sons, INC. New York London Sydney, 1968
```

```
*)
```

```
pi          = 3.14159265358979323846264338327950288;  
twicepi     = 6.28318530717958647692528676655900576;  
halfpi      = 1.57079632679489661923132169163975144;  
quartpi     = 0.78539816339744830961566084581987572;  
e           = 2.71828182845904523536028747135266250;  
ln2         = 0.69314718055994530941723212145817657;  
ln10        = 2.30258509299404568401799145468436421;  
  
longpi      = 3.14159265358979323846264338327950288D;  
longtwicepi = 6.28318530717958647692528676655900576D;  
longhalfpi  = 1.57079632679489661923132169163975144D;  
longquartpi = 0.78539816339744830961566084581987572D;  
longe       = 2.71828182845904523536028747135266250D;  
longln2     = 0.69314718055994530941723212145817657D;  
longln10    = 2.30258509299404568401799145468436421D;
```

```
(* basic functions *)
```

```
PROCEDURE pow (x: REAL; i: INTEGER): REAL;  
PROCEDURE longpow (x: LONGREAL; i: INTEGER): LONGREAL;  
PROCEDURE sqrt (x: REAL): REAL;
```

```

PROCEDURE longsqrt (x: LONGREAL): LONGREAL;
PROCEDURE exp (x: REAL): REAL;
PROCEDURE longexp (x: LONGREAL): LONGREAL;
PROCEDURE ln (x: REAL): REAL; (* natural log *)
PROCEDURE longln (x: LONGREAL): LONGREAL; (* natural log *)
PROCEDURE log (x: REAL): REAL; (* log with base 10 *)
PROCEDURE longlog (x: LONGREAL): LONGREAL; (* log with base 10 *)
(* trigonometric functions; arguments in radians *)
PROCEDURE sin (x: REAL): REAL;
PROCEDURE longsin (x: LONGREAL): LONGREAL;
PROCEDURE cos (x: REAL): REAL;
PROCEDURE longcos (x: LONGREAL): LONGREAL;
PROCEDURE tan (x: REAL): REAL;
PROCEDURE longtan (x: LONGREAL): LONGREAL;
PROCEDURE arcsin (x: REAL): REAL;
PROCEDURE longarcsin (x: LONGREAL): LONGREAL;
PROCEDURE arccos (x: REAL): REAL;
PROCEDURE longarccos (x: LONGREAL): LONGREAL;
PROCEDURE arctan (x: REAL): REAL;
PROCEDURE longarctan (x: LONGREAL): LONGREAL;
(* hyperbolic functions *)
PROCEDURE sinh (x: REAL): REAL;
PROCEDURE longsinh (x: LONGREAL): LONGREAL;
PROCEDURE cosh (x: REAL): REAL;
PROCEDURE longcosh (x: LONGREAL): LONGREAL;
PROCEDURE tanh (x: REAL): REAL;
PROCEDURE longtanh (x: LONGREAL): LONGREAL;
PROCEDURE arcsinh (x: REAL): REAL;
PROCEDURE longarcsinh (x: LONGREAL): LONGREAL;
PROCEDURE arccosh (x: REAL): REAL;
PROCEDURE longarccosh (x: LONGREAL): LONGREAL;
PROCEDURE arctanh (x: REAL): REAL;
PROCEDURE longarctanh (x: LONGREAL): LONGREAL;
(* conversions *)
PROCEDURE RadianToDegree (x: REAL): REAL;

```

```

PROCEDURE longRadianToDegree (x: LONGREAL): LONGREAL;
PROCEDURE DegreeToRadian (x: REAL): REAL;
PROCEDURE longDegreeToRadian (x: LONGREAL): LONGREAL;
END Mathlib.

```

```

DEFINITION MODULE PascalIO;

```

```

(*)
Module:    Pascal-like Input/Output
Author:    Cerial J.H. Jacobs
Version:   $Id: PascalIO.def,v 1.9 1994/06/24 12:49:09 cerial Exp $

```

```

This module provides for I/O that is essentially equivalent to the I/O provided by
Pascal with "text", or "file of char".
Output buffers are automatically flushed at program termination.
The CloseOutput routine is just there for compatibility with earlier versions of
this module.

```

```

*)

```

```

CONST Eos = 0C;                (* End of string character *)

```

```

TYPE Text;

```

```

VAR Input, Output: Text;

```

```

    (* standard input and standard output available immediately.
       Standard output is not buffered when connected to a terminal. *)

```

```

VAR Notext: Text;             (* Initialize your Text variables with this *)

```

```

PROCEDURE Reset (VAR InputText: Text; Filename: ARRAY OF CHAR);

```

```

    (* When InputText indicates an open textfile, it is first flushed and closed. Then, the
       file indicated by "Filename" is opened for reading.
       If this fails, a runtime error results. Otherwise, InputText is associated with the
       new input file.

```

```

    *)

```

```

PROCEDURE Rewrite (VAR OutputText: Text; Filename: ARRAY OF CHAR);

```

```

    (* When OutputText indicates an open textfile, it is first flushed and closed. Then,
       the file indicated by "Filename" is opened for writing.
       If this fails, a runtime error results. Otherwise, OutputText is associated with the
       new output file.

```

```

    *)

```

```

PROCEDURE CloseOutput ();

```

```

    (* To be called at the end of the program, to flush all output buffers. *)

```

```

    (*****
      Input routines;
      All these routines result in a runtime error when not called with either "Input", or
      a "Text" value obtained by Reset. Also, the routines that actually advance the "read
      pointer", result in a runtime error when end of file is reached prematurely.
    *****)

```

```

PROCEDURE NextChar (InputText: Text): CHAR;

```

```
(* Returns the next character from the InputText,  C on end of file. Does not advance
the "read pointer", so behaves much like "input^" in Pascal. However, unlike Pascal,
if Eoln(InputText) is TRUE, it returns the newline character, rather than a space.
*)
```

```
PROCEDURE Get (InputText: Text);
```

```
(* Advances the "read pointer" by one character. *)
```

```
PROCEDURE Eoln (InputText: Text): BOOLEAN;
```

```
(* Returns TRUE if the next character from the InputText is a linefeed. Unlike Pascal
however, it does not produce a runtime error when called when Eof(InputText) is
TRUE.
*)
```

```
PROCEDURE Eof (InputText: Text): BOOLEAN;
```

```
(* Returns TRUE if the end of the InputText is reached. *)
```

```
PROCEDURE ReadChar (InputText: Text; VAR Char: CHAR);
```

```
(* Read a character from the InputText, and leave the result in "Char". Unlike Pascal,
if Eoln(InputText) is TRUE, the newline character is put in "Char".
*)
```

```
PROCEDURE ReadLn (InputText: Text);
```

```
(* Skip the rest of the current line from the InputText, including the linefeed. *)
```

```
PROCEDURE ReadInteger (InputText: Text; VAR Integer: INTEGER);
```

```
(* Skip leading blanks, read an optionally signed integer from the InputText, and leave
the result in "Integer". If no integer is read, or when overflow occurs, a runtime
error results. Input stops at the character following the integer.
*)
```

```
PROCEDURE ReadCardinal (InputText: Text; VAR Cardinal: CARDINAL);
```

```
(* Skip leading blanks, read a cardinal from the InputText, and leave the result in
"Cardinal". If no cardinal is read, or when overflow occurs, a runtime error
results. Input stops at the character following the cardinal.
*)
```

```
PROCEDURE ReadReal (InputText: Text; VAR Real: REAL);
```

```
(* Skip leading blanks, read a real from the InputText, and leave the result in "Real".
Syntax:
real --> [(+|-)] digit {digit} [. digit {digit}] [ E [(+|-)] digit {digit} ]
If no real is read, or when overflow/underflow occurs, a runtime error results.
Input stops at the character following the real.
*)
```

```
PROCEDURE ReadLongReal (InputText: Text; VAR Real: LONGREAL);
```

```
(* Like ReadReal, but for LONGREAL *)
```

```
(*****
Output routines;
All these routines result in a runtime error when not called with either
"Output", or a "Text" value obtained by Rewrite.
*****)
```

```

PROCEDURE WriteChar (OutputText: Text; Char: CHAR);
(* Writes the character "Char" to the OutputText. *)

PROCEDURE WriteLn (OutputText: Text);

(* Writes a linefeed to the OutputText. *)

PROCEDURE Page (OutputText: Text);

(* Writes a form-feed to the OutputText *)

PROCEDURE WriteInteger (OutputText: Text; Integer: INTEGER; Width: CARDINAL);

(* Write integer "Integer" to the OutputText, using at least "Width" places, blank-
padding to the left if needed.
*)

PROCEDURE WriteCardinal (OutputText: Text; Cardinal, Width: CARDINAL);

(* Write cardinal "Cardinal" to the OutputText, using at least "Width" places, blank-
padding to the left if needed.
*)

PROCEDURE WriteBoolean (OutputText: Text; Boolean: BOOLEAN; Width: CARDINAL);

(* Write boolean "Boolean" to the OutputText, using at least "Width" places, blank-
padding to the left if needed.
Equivalent to WriteString(OutputText, " TRUE", Width), or
WriteString(OutputText, " FALSE", Width).
*)

PROCEDURE WriteString (OutputText: Text; String: ARRAY OF CHAR; Width: CARDINAL);

(* Write string "String" to the OutputText, using at least "Width" places, blank-
padding to the left if needed. The string is terminated either by the character Eos,
or by the upperbound of the array "String".
*)

PROCEDURE WriteReal (OutputText: Text; Real: REAL; Width, Nfrac: CARDINAL);

(* Write real "Real" to the OutputText. If "Nfrac" = 0, use scientific notation,
otherwise use fixed-point notation with "Nfrac" digits behind the dot.
Always use at least "Width" places, blank-padding to the left if needed.
*)

PROCEDURE WriteLongReal (OutputText: Text; Real: LONGREAL; Width, Nfrac: CARDINAL);

(* Like WriteReal, but for LONGREAL *)

END PascalIO.

```

DEFINITION MODULE Processes;

(*
Module: Processes
From: "Programming in Modula-2", 3rd, corrected edition, by N. Wirth
Version: \$Id: Processes.def,v 1.4 1994/06/24 12:49:15 ceriel Exp \$
*)

(*
As discussed in "Unfair Process Scheduling in Modula-2", by D. Hemmendinger, SIGplan
Notices Volume 23 nr 3, march 1988, the scheduler in this module is unfair, in that
in some circumstances ready-to-run processes never get a turn.
*)

TYPE SIGNAL;

PROCEDURE StartProcess (P: PROC; n: CARDINAL);

(* Start a concurrent process with program "P" and workspace of size "n" *)

PROCEDURE SEND (VAR s: SIGNAL);

(* One process waiting for "s" is resumed *)

PROCEDURE WAIT (VAR s: SIGNAL);

(* Wait for some other process to send "s" *)

PROCEDURE Awaited (s: SIGNAL): BOOLEAN;

(* Return TRUE if at least one process is waiting for signal "s". *)

END Processes.

```
DEFINITION MODULE random;
```

```
PROCEDURE Init (VAR s: SIGNAL);
```

```
(* Compulsory initialization *)
```

```
(*  
  Module:    random numbers  
  Author:    Cerieel J.H. Jacobs  
  Version:   $Id: random.def,v 1.3 1994/06/24 12:51:24 ceriel Exp $  
)
```

```
PROCEDURE Random (): CARDINAL;
```

```
(* Return a random CARDINAL *)
```

```
PROCEDURE Uniform (lwb, upb: CARDINAL): CARDINAL;
```

```
(* Return CARDINALs, uniformly distributed between "lwb" and "upb".  
  "lwb" must be smaller than "upb", or "lwb" is returned.  
)
```

```
PROCEDURE StartSeed (seed: CARDINAL);
```

```
(* Initialize the generator. You don't have to call this procedure, unless you don't  
  want the system to pick a starting value for itself.  
)
```

```
END random.
```

```
DEFINITION MODULE RealConversions;
```

```
(*  
  Module:    string-to-real and real-to-string conversions  
  Author:    Cerieel J.H. Jacobs  
  Version:   $Id: RealConver.def,v 1.6 1994/06/24 12:49:21 ceriel Exp $  
)
```

```
PROCEDURE StringToReal (str: ARRAY OF CHAR; VAR r: REAL; VAR ok: BOOLEAN);
```

```
(* Convert string "str" to a real number "r" according to the syntax:
```

```
  ['+'|'-'] digit {digit} ['.' digit {digit}] ['E' ['+'|'-'] digit {digit}]
```

```
  ok := "conversion succeeded"
```

```
  Leading blanks are skipped; Input terminates with a blank or any control character.
```

```
*)
```

```
PROCEDURE StringToLongReal (str: ARRAY OF CHAR; VAR r: LONGREAL; VAR ok: BOOLEAN);
```

```
PROCEDURE RealToString (r: REAL; width, digits: INTEGER; VAR str: ARRAY OF CHAR;  
                        VAR ok: BOOLEAN);
```

```
(* Convert real number "r" to string "str", either in fixed-point or exponent notation.  
  "digits" is the number digits to the right of the decimal point,  
  "width" is the maximum width of the notation.  
  If digits < 0, exponent notation is used, otherwise fixed-point.  
  If fewer than "width" characters are needed, leading blanks are inserted.  
  If the representation does not fit in "width", then ok is set to FALSE.  
)
```

```
PROCEDURE LongRealToString(r: LONGREAL; width, digits: INTEGER; VAR str: ARRAY OF CHAR;  
                           VAR ok: BOOLEAN);
```

```
END RealConversions.
```

```
DEFINITION MODULE RealInOut;
```

```
(*  
  Module:      InOut for REAL numbers  
  From:        "Programming in Modula-2", 3rd, corrected edition, by N. Wirth  
  Version:     $Id: RealInOut.def,v 1.6 1994/06/24 12:49:28 cerial Exp $  
*)
```

```
VAR Done: BOOLEAN;
```

```
PROCEDURE ReadReal (VAR x: REAL);
```

```
(* Read a real number "x" according to the syntax:
```

```
  ['+'|'-'] digit {digit} ['.' digit {digit}] [('E'|'e') ['+'|'-'] digit {digit}]
```

```
  Done := "a number was read".
```

```
  Input terminates with a blank or any control character.
```

```
  When reading from a terminal, backspacing may be done by either  
  DEL or BACKSPACE, depending on the implementation of ReadString.
```

```
*)
```

```
PROCEDURE ReadLongReal (VAR x: LONGREAL);
```

```
(* Like ReadReal, but for LONGREAL *)
```

```
PROCEDURE WriteReal (x: REAL; n: CARDINAL);
```

```
(* Write x using n characters.
```

```
  If fewer than n characters are needed, leading blanks are inserted.
```

```
*)
```

```
PROCEDURE WriteLongReal (x: LONGREAL; n: CARDINAL);
```

```
(* Like WriteReal, but for LONGREAL *)
```

```
PROCEDURE WriteFixPt (x: REAL; n, k: CARDINAL);
```

```
(* Write x in fixed-point notation using n characters with k digits after the decimal  
  point. If fewer than n characters are needed, leading blanks are inserted.
```

```
*)
```

```
PROCEDURE WriteLongFixPt (x: LONGREAL; n, k: CARDINAL);
```

```
(* Like WriteFixPt, but for LONGREAL *)
```

```
PROCEDURE WriteRealOct (x: REAL);
```

```
(* Write x in octal words. *)
```

```
PROCEDURE WriteLongRealOct (x: LONGREAL);
```

```
(* Like WriteRealOct, but for LONGREAL *)
```

```
END RealInOut.
```

DEFINITION MODULE Semaphores;

```
(*  
  Module:    Processes with semaphores  
  Author:    Ceriel J.H. Jacobs  
  Version:   $Id: Semaphores.def,v 1.3 1994/06/24 12:49:38 ceriel Exp $
```

On systems using quasi-concurrency, the only opportunities for process-switches are calls to Down and Up.

```
*)
```

TYPE Sema;

PROCEDURE Level (s: Sema) : CARDINAL;

```
(* Returns current value of semaphore s *)
```

PROCEDURE NewSema (n: CARDINAL) : Sema;

```
(* Creates a new semaphore with initial level "n" *)
```

PROCEDURE Down (VAR s: Sema);

```
(* If the value of "s" is > 0, then just decrement "s". Else, suspend the current  
  process until the semaphore becomes positive again.
```

```
*)
```

PROCEDURE Up (VAR s: Sema);

```
(* Increment the semaphore "s". *)
```

PROCEDURE StartProcess (P: PROC; n: CARDINAL);

```
(* Create a new process with procedure P and workspace of size "n". Also transfer  
  control to it.
```

```
*)
```

END Semaphores.

DEFINITION MODULE Storage;

```
(*  
  Module:    Dynamic storage allocation  
  From:      "Programming in Modula-2", 3rd, corrected edition, by N. Wirth  
  Version:   $Id: Storage.def,v 1.5 1994/06/24 12:49:44 ceriel Exp $
```

```
*)
```

```
(*  
  Wirth's 3rd edition certainly is confusing: mostly it uses Allocate, but the module  
  at the end of the book defines ALLOCATE. To avoid problems, I included them both.
```

```
*)
```

FROM SYSTEM IMPORT ADDRESS;

PROCEDURE ALLOCATE (VAR a : ADDRESS; size : CARDINAL);

```
(* Allocate an area of the given size and return the address in "a". If no space is  
  available, the calling program is killed.
```

```
*)
```

PROCEDURE Allocate (VAR a : ADDRESS; size : CARDINAL);

```
(* Identical to ALLOCATE *)
```

```
PROCEDURE DEALLOCATE (VAR a : ADDRESS; size : CARDINAL);
```

```
(* Free the area at address "a" with the given size. The area must have been allocated  
by "ALLOCATE", with the same size.  
)
```

```
PROCEDURE Deallocate (VAR a : ADDRESS; size : CARDINAL);
```

```
(* Identical to DEALLOCATE *)
```

```
PROCEDURE Available (size : CARDINAL) : BOOLEAN;
```

```
(* Return TRUE if a contiguous area with the given size could be allocated.  
Notice that this only indicates if an ALLOCATE of this size would succeed, and that  
it gives no indication of the total available memory.  
)
```

```
END Storage.
```

```
DEFINITION MODULE Streams;
```

```
(*  
Module: Stream Input/Output  
Author: Cerial J.H. Jacobs  
Version: $Id: Streams.def,v 1.4 1994/06/24 12:49:54 cerial Exp $
```

```
* This module provides sequential IO through streams.  
* A stream is either a text stream or a binary stream, and is either in reading,  
writing or appending mode.  
* By default, there are three open text streams, connected to standard input,  
standard output, and standard error respectively.  
* These are text streams.  
* When connected to a terminal, the standard output and standard error streams are  
linebuffered.  
* The user can create more streams with the OpenStream call, and delete streams with  
the CloseStream call.  
* Streams are automatically closed at program termination.  
)
```

```
FROM SYSTEM IMPORT BYTE;
```

```
TYPE StreamKind = (text, binary, none);  
StreamMode = (reading, writing, appending);  
StreamResult = (succeeded, illegaloperation,  
nomemory, openfailed, nostream, endoffile);  
StreamBuffering = (unbuffered, linebuffered, blockbuffered);  
TYPE Stream;
```

```
VAR InputStream, OutputStream, ErrorStream: Stream;
```

```
PROCEDURE OpenStream (VAR stream: Stream; filename: ARRAY OF CHAR; kind: StreamKind;  
mode: StreamMode; VAR result: StreamResult);
```

```
(* Associates a stream with the file named filename.  
If kind = none, result is set to illegaloperation.  
mode has one of the following values:  
reading: the file is opened for reading  
writing: the file is truncated or created for writing  
appending: the file is opened for writing at end of file, or created for  
writing.  
On failure, result is set to openfailed.  
On success, result is set to succeeded.  
)
```

```
PROCEDURE SetStreamBuffering (stream: Stream; b: StreamBuffering;  
                             VAR result: StreamResult);
```

```
(* This procedure is only allowed for output streams.  
The three types of buffering available are unbuffered, linebuffered, and  
blockbuffered. When an output stream is unbuffered, the output appears as soon as  
written; when it is blockbuffered, output is saved up and written as a block; When  
it is linebuffered (only possible for text output streams), output is saved up  
until a newline is encountered or input is read from standard input.  
)
```

```
PROCEDURE CloseStream (VAR stream: Stream; VAR result: StreamResult);
```

```
(* Closes the stream.  
result is set to nostream if "stream" was not associated with a stream.  
)
```

```
PROCEDURE FlushStream (stream: Stream; VAR result: StreamResult);
```

```
(* Flushes the stream.  
result is set to nostream if "stream" was not associated with a stream.  
It is set to illegaloperation if "stream" is not an output or appending stream.  
)
```

```
PROCEDURE EndOfStream (stream: Stream; VAR result: StreamResult): BOOLEAN;
```

```
(* Returns true if the stream is an input stream, and the end of the file has been  
reached.  
result is set to nostream if "stream" was not associated with a stream.  
It is set to illegaloperation if the stream is not an input stream.  
)
```

```
PROCEDURE Read (stream: Stream; VAR ch: CHAR; VAR result: StreamResult);
```

```
(* reads a character from the stream. Certain character translations may occur, such as  
the mapping of the end-of-line sequence to the character 12C.  
result is set to nostream if "stream" was not associated with a stream.  
It is set to endoffile if EndOfStream would have returned TRUE before the call to  
Read. In this case, "ch" is set to 0C.  
It is set to illegaloperation if the stream is not a text input stream.  
)
```

```
PROCEDURE ReadByte (stream: Stream; VAR byte: BYTE; VAR result: StreamResult);
```

```
(* reads a byte from the stream. No character translations occur.  
result is set to nostream if "stream" was not associated with a stream.  
It is set to endoffile if EndOfStream would have returned TRUE before the call to  
ReadByte. In this case, "byte" is set to 0C.  
It is set to illegaloperation if the stream is not a binary input stream.  
)
```

```
PROCEDURE ReadBytes (stream: Stream; VAR bytes: ARRAY OF BYTE;  
                   VAR result: StreamResult);
```

```
(* reads bytes from the stream. No character translations occur. The number of bytes is  
determined by the size of the parameter.  
result is set to nostream if "stream" was not associated with a stream.  
It is set to endoffile if there are not enough bytes left on the stream. In this  
case, the rest of the bytes are set to 0C.  
It is set to illegaloperation if the stream is not a binary input stream.  
)
```

```
PROCEDURE Write (stream: Stream; ch: CHAR; VAR result: StreamResult);
```

```
(* writes a character to the stream. Certain character translations may occur, such as
the mapping of a line-feed or carriage return (12C or 15C) to the end-of-line
sequence of the system.
result is set to nostream if "stream" was not associated with a stream.
It is set to illegaloperation if the stream is not a text output stream.
*)
```

```
PROCEDURE WriteByte (stream: Stream; byte: BYTE; VAR result: StreamResult);
```

```
(* writes a byte to the stream. No character translations occur.
result is set to nostream if "stream" was not associated with a stream.
It is set to illegaloperation if the stream is not a binary output stream.
*)
```

```
PROCEDURE WriteBytes (stream: Stream; bytes: ARRAY OF BYTE; VAR result: StreamResult);
```

```
(* writes bytes to the stream. No character translations occur.
The number of bytes written is equal to the size of the parameter.
result is set to nostream if "stream" was not associated with a stream.
It is set to illegaloperation if the stream is not a binary output stream.
*)
```

```
PROCEDURE GetPosition (stream: Stream; VAR position: LONGINT;
                      VAR result: StreamResult);
```

```
(* gives the actual read/write position in "position".
"result" is set to illegaloperation if "stream" is not a stream.
*)
```

```
PROCEDURE SetPosition (stream: Stream; position: LONGINT; VAR result: StreamResult);
```

```
(* sets the actual read/write position to "position".
"result" is set to nostream if "stream" is not a stream.
It is set to illegaloperation if the stream was opened for appending and the
position is in front of the current position, or it failed for some other reason
(f.i. when the stream is connected to a terminal).
*)
```

```
PROCEDURE isatty (stream: Stream; VAR result: StreamResult): BOOLEAN;
```

```
(* return TRUE if the stream is connected to a terminal.
"result" is set to nostream if "stream" is not a stream, and
set to illegaloperation if the operation failed for some other reason.
*)
```

```
END Streams.
```

```
DEFINITION MODULE Strings;
```

```
(*
Module:      String manipulations
Author:      Cerial J.H. Jacobs
Version:     $Id: Strings.def,v 1.3 1994/06/24 12:50:00 cerial Exp $
*)
```

```
(* Note: truncation of strings may occur if the user does not provide large enough
variables to contain the result of the operation.
*)
```

```
(* Strings are of type ARRAY OF CHAR, and their length is the size of the array, unless
a 0-byte occurs in the string to indicate the end of the string.
*)
```

```
PROCEDURE Assign (source: ARRAY OF CHAR; VAR dest: ARRAY OF CHAR);
```

```
(* Assign string source to dest *)
```

```

PROCEDURE Insert (substr: ARRAY OF CHAR; VAR str: ARRAY OF CHAR; inx: CARDINAL);

(* Insert the string substr into str, starting at str[inx].
   If inx is equal to or greater than Length(str) then substr is appended to the end of
   str.
*)

PROCEDURE Delete (VAR str: ARRAY OF CHAR; inx, len: CARDINAL);

(* Delete len characters from str, starting at str[inx].
   If inx >= Length(str) then nothing happens.
   If there are not len characters to delete, characters to the end of the string are
   deleted.
*)

PROCEDURE Pos (substr, str: ARRAY OF CHAR): CARDINAL;

(* Return the index into str of the first occurrence of substr.
   Pos returns a value greater than HIGH(str) if no occurrence is found.
*)

PROCEDURE Copy (str: ARRAY OF CHAR; inx, len: CARDINAL; VAR result: ARRAY OF CHAR);

(* Copy at most len characters from str into result, starting at str[inx]. *)

PROCEDURE Concat (s1, s2: ARRAY OF CHAR; VAR result: ARRAY OF CHAR);

(* Concatenate two strings. *)

PROCEDURE Length (str: ARRAY OF CHAR): CARDINAL;

(* Return number of characters in str. *)

PROCEDURE CompareStr (s1, s2: ARRAY OF CHAR): INTEGER;

(* Compare two strings, return -1 if s1 < s2, 0 if s1 = s2, and 1 if s1 > s2. *)

END Strings.

(*$Foreign language module *)
DEFINITION MODULE StripUnix;

(*
   Module:    interface to some Unix systemcalls
   Author:    Cerieel J.H. Jacobs
   Version:   $Id: StripUnix.def,v 1.2 1994/06/24 12:50:06 ceriel Exp $

   This is a stripped down version of Unix.def, needed to compile some of the modules on
   small machines
*)

FROM SYSTEM IMPORT ADDRESS;

CONST ILLBREAK = ADDRESS (NIL-1);

VAR   errno: INTEGER;

PROCEDURE sbrk (incr: INTEGER) : ADDRESS;

PROCEDURE close (fildes: INTEGER) : INTEGER;

```

```
PROCEDURE creat (path: ADDRESS; mode: INTEGER) : INTEGER;
(* Sys5 *) PROCEDURE fcntl (fildes, request, arg: INTEGER) : INTEGER;
PROCEDURE getpid () : INTEGER;
PROCEDURE ioctl (fildes, request: INTEGER; arg: ADDRESS) : INTEGER;
PROCEDURE lseek (fildes: INTEGER; offset: LONGINT; whence: INTEGER) : LONGINT;
PROCEDURE open (path: ADDRESS; oflag: INTEGER) : INTEGER;
PROCEDURE read (fildes: INTEGER; buf: ADDRESS; nbyte: CARDINAL) : INTEGER;
PROCEDURE time (tloc: ADDRESS) : LONGINT;
PROCEDURE write (fildes: INTEGER; buf: ADDRESS; nbyte: CARDINAL) : INTEGER;
END StripUnix.
```

```
(*  
  (c) copyright 1988 by the Vrije Universiteit, Amsterdam, The Netherlands.  
  See the copyright notice in the ACK home directory, in the file "Copyright".  
*)
```

```
(*  
  Module:      Interface to termcap database  
  From:        Unix manual chapter 3  
  Version:     $Id: Termcap.def,v 1.4 1994/06/24 12:50:10 ceriel Exp $  
*)
```

```
(*  
  Use this like the C-version. In this Modula-2 version, some of the buffers, that are  
  explicit in the C-version, are hidden. These buffers are initialized by a call to  
  Tgetent. The "ARRAY OF CHAR" parameters must be null-terminated.  
  You can call them with a constant string argument, as these are always null-  
  terminated in our Modula-2 implementation. Unlike the C version, this version takes  
  care of UP, BC, PC, and ospeed automatically.  
  If Tgetent is not called by the user, it is called by the module itself, using the  
  TERM environment variable, or "dumb" if TERM does not exist.  
*)
```

```
DEFINITION MODULE Termcap;
```

```
TYPE  STRCAP;  
      PUTPROC = PROCEDURE (CHAR);
```

```
PROCEDURE Tgetent (name: ARRAY OF CHAR) : INTEGER;
```

```
PROCEDURE Tgetnum (id: ARRAY OF CHAR): INTEGER;
```

```
PROCEDURE Tgetflag (id: ARRAY OF CHAR): BOOLEAN;
```

```
PROCEDURE Tgoto (cm: STRCAP; col, line: INTEGER): STRCAP;
```

```
(* Result exists until next call to Tgoto *)
```

```
PROCEDURE Tgetstr (id: ARRAY OF CHAR; VAR res: STRCAP) : BOOLEAN;
```

```
(* Returns FALSE if capability does not exist;  
  Result exists until next call to Tgetent.
```

```
*)
```

```
PROCEDURE Tputs (cp: STRCAP; affcnt: INTEGER; p: PUTPROC);
```

```
END Termcap.
```

```

DEFINITION MODULE Terminal;
(*
  Module:      Input/Output to/from terminals
  From:        "Programming in Modula-2", 3rd, corrected edition, by N. Wirth
  Version:     $Id: Terminal.def,v 1.3 1994/06/24 12:50:16 ceriel Exp $
*)

PROCEDURE Read (VAR ch : CHAR);

(* Read a character from the terminal and leave it in ch *)

PROCEDURE BusyRead(VAR ch : CHAR);

(* Read a character from the terminal and leave it in ch. This is a non-blocking call.
   It returns 0C in ch if no character was typed.
*)

PROCEDURE ReadAgain;

(* Causes the last character read to be returned again upon the next call of Read. *)

PROCEDURE Write (ch : CHAR);
(* Write character ch to the terminal. *)

PROCEDURE WriteLn;

(* Terminate line. *)

PROCEDURE WriteString (s : ARRAY OF CHAR);

(* Write string s to the terminal. *)

END Terminal.

DEFINITION MODULE Traps;
(*
  Module:      Facility for handling traps
  Author:      Ceriel J.H. Jacobs
  Version:     $Id: Traps.def,v 1.9 1994/06/24 12:50:22 ceriel Exp $
*)

IMPORT EM;

CONST  ERRTOOLARGE = 64;    (* stack size of process too large *)
      ERRTOOMANY   = 65;    (* too many nested traps + handlers *)
      ERRNORESULT  = 66;    (* no RETURN from function procedure *)
      ERRCARDOVFL  = 67;    (* CARDINAL overflow *)
      ERRFORLOOP   = 68;    (* value of FOR-loop control variable changed
                             in loop
                             *)
      ERRCARDUVFL  = 69;    (* CARDINAL underflow *)
      ERRINTERNAL  = 70;    (* Internal error; should not happen *)
      ERRUNIXSIG   = 71;    (* received unix signal *)

TYPE   TrapHandler = EM.TrapHandler;

PROCEDURE InstallTrapHandler(t: TrapHandler): TrapHandler;

(* Install a new trap handler, and return the previous one.
   Parameter of trap handler is the trap number.
   When a trap occurs, the default trap handler is re-installed before calling the new
   handler.
*)

```

```

PROCEDURE Message(str: ARRAY OF CHAR);

(* Write message "str" on standard error, preceded by filename and linenumber if
   possible
*)

PROCEDURE Trap(n: INTEGER);

(* cause trap number "n" to occur *)

END Traps.

(*$Foreign language module *)
DEFINITION MODULE Unix;
(*
   Module:      interface to some Unix systemcalls
   Author:      Cerieel J.H. Jacobs
   Version:     $Id: Unix.def,v 1.7 1994/06/24 12:50:29 ceriel Exp $
*)
   FROM SYSTEM IMPORT WORD, ADDRESS;

(* Type needed for Signal *)

TYPE   SignalPrc = PROCEDURE (INTEGER);

CONST  SIGDFL = SignalPrc (NIL);
        SIGIGN = SignalPrc (NIL+1);
        ILLBREAK = ADDRESS (NIL-1);

VAR  errno: INTEGER;

(* Possible values of errno: *)
CONST  EPERM = 1;   (* Not owner *)
        ENOENT = 2; (* No such file or directory *)
        ESRCH = 3;  (* No such process *)
        EINTR = 4;  (* Interrupted system call *)
        EIO = 5;    (* I/O error *)
        ENXIO = 6;  (* No such device or address *)
        E2BIG = 7;  (* Arg list too long *)
        ENOEXEC = 8; (* Exec format error *)
        EBADF = 9;  (* Bad file number *)
        ECHILD = 10; (* No child processes *)
        EAGAIN = 11; (* No more processes *)
        ENOMEM = 12; (* Not enough space *)
        EACCES = 13; (* Permission denied *)
        EFAULT = 14; (* Bad address *)
        ENOTBLK = 15; (* Block device required *)
        EBUSY = 16;  (* Mount device busy *)
        EEXIST = 17; (* File exists *)
        EXDEV = 18;  (* Cross-device link *)
        ENODEV = 19; (* No such device *)
        ENOTDIR = 20; (* Not a directory *)
        EISDIR = 21; (* Is a directory *)
        EINVAL = 22; (* Invalid argument *)
        ENFILE = 23; (* File table overflow *)
        EMFILE = 24; (* Too many open files *)
        ENOTTY = 25; (* Not a typewriter *)
        ETXTBSY = 26; (* Text file busy *)
        EFBIG = 27;  (* File too large *)
        ENOSPC = 28; (* No space left on device *)
        EPIPE = 29;  (* Illegal seek *)
        EROFS = 30;  (* Read-only file system *)
        EMLINK = 31; (* Too many links *)
        EPIPE = 32;  (* Broken pipe *)

```

```
EDOM = 33; (* Math argument *)
ERANGE = 34; (* Result too large *)
```

```
PROCEDURE access (path: ADDRESS; amode : INTEGER) : INTEGER;
PROCEDURE acct (path: ADDRESS) : INTEGER;
PROCEDURE alarm (sec: CARDINAL) : CARDINAL;
PROCEDURE brk (endds: ADDRESS) : INTEGER;
PROCEDURE sbrk (incr: INTEGER) : ADDRESS;
PROCEDURE chdir (path: ADDRESS) : INTEGER;
PROCEDURE chmod (path: ADDRESS; mode: INTEGER) : INTEGER;
PROCEDURE chown (path: ADDRESS; owner, group: INTEGER) : INTEGER;
PROCEDURE chroot (path: ADDRESS) : INTEGER;
PROCEDURE close (fildes: INTEGER) : INTEGER;
PROCEDURE creat (path: ADDRESS; mode: INTEGER) : INTEGER;
PROCEDURE dup (fildes: INTEGER) : INTEGER;
PROCEDURE execve (path: ADDRESS; argv: ADDRESS; envp: ADDRESS) : INTEGER;
PROCEDURE exit (status: INTEGER);
```

```
(* Sys5 *) PROCEDURE fcntl (fildes, request, arg: INTEGER) : INTEGER;
```

```
PROCEDURE ftime (bufp:ADDRESS) : INTEGER;
PROCEDURE fork () : INTEGER;
PROCEDURE getpid () : INTEGER;
PROCEDURE getppid () : INTEGER;
PROCEDURE getuid () : INTEGER;
PROCEDURE geteuid () : INTEGER;
PROCEDURE getgid () : INTEGER;
PROCEDURE getegid () : INTEGER;
PROCEDURE ioctl (fildes, request: INTEGER; arg: ADDRESS) : INTEGER;
PROCEDURE stty (fildes: INTEGER; buf: ADDRESS) : INTEGER;
PROCEDURE gtty (fildes: INTEGER; buf: ADDRESS) : INTEGER;
PROCEDURE kill (pid, sig: INTEGER) : INTEGER;
PROCEDURE link (path1, path2: ADDRESS) : INTEGER;
PROCEDURE lseek (fildes: INTEGER; offset: LONGINT; whence: INTEGER) : LONGINT;
PROCEDURE mknod (path: ADDRESS; mode, dev: INTEGER) : INTEGER;
PROCEDURE mount (spec, dir: ADDRESS; rwflag: INTEGER) : INTEGER;
PROCEDURE nice (incr: INTEGER) : INTEGER;
PROCEDURE open (path: ADDRESS; oflag: INTEGER) : INTEGER;
PROCEDURE pause ();
PROCEDURE pipe (fildes: ADDRESS) : INTEGER;
PROCEDURE profil (buff: ADDRESS; bufsiz, offset, scale: INTEGER);
PROCEDURE ptrace (request, pid, addr, data: WORD) : INTEGER;
PROCEDURE read (fildes: INTEGER; buf: ADDRESS; nbytes: CARDINAL) : INTEGER;
PROCEDURE setuid (uid: INTEGER) : INTEGER;
PROCEDURE setgid (gid: INTEGER) : INTEGER;
PROCEDURE signal (sig: INTEGER; func: SignalPrc) : SignalPrc;
PROCEDURE sigtrp (trapno, signo: INTEGER) : INTEGER;
```

```
(* Let Unix signal signo cause EM trap trapno to occur *)
```

```
PROCEDURE stat (path: ADDRESS; statbuf: ADDRESS) : INTEGER;
PROCEDURE fstat (fildes: INTEGER; statbuf: ADDRESS) : INTEGER;
PROCEDURE stime (t: LONGINT) : INTEGER;
PROCEDURE sync ();
PROCEDURE time (tloc: ADDRESS) : LONGINT;
PROCEDURE times (buffer: ADDRESS) : LONGINT;
PROCEDURE umask (cmask: INTEGER) : INTEGER;
PROCEDURE umount (spec: ADDRESS) : INTEGER;
PROCEDURE unlink (path: ADDRESS) : INTEGER;
PROCEDURE utime (path: ADDRESS; times: ADDRESS) : INTEGER;
PROCEDURE wait (VAR statloc: INTEGER): INTEGER;
PROCEDURE write (fildes: INTEGER; buf: ADDRESS; nbytes: CARDINAL) : INTEGER;
END Unix.
```

```
(*
(c) copyright 1988 by the Vrije Universiteit, Amsterdam, The Netherlands.
See the copyright notice in the ACK home directory, in the file "Copyright".
*)
```

```
(*
Module:      Interface to termcap database
From:        Unix manual chapter 3
Version:     $Id: XXTermcap.def,v 1.2 1994/06/24 12:50:32 ceriel Exp $
*)
```

```
(*Foreign*)
```

```
DEFINITION MODULE XXTermcap;
```

```
(* See the Unix termcap manual to see what this does. Interfaces directly to C
routines. Not pretty. Use Termcap.def instead for a nicer interface.
*)
```

```
FROM SYSTEM IMPORT ADDRESS;
```

```
TYPE  PUTPROC = PROCEDURE(CHAR);
```

```
VAR   PC           : CHAR;
      UP, BC       : ADDRESS;
      ospeed       : INTEGER [0..32767];
```

```
PROCEDURE tgetent (bp, name: ADDRESS): INTEGER;
```

```
(* name must be null-terminated *)
```

```
PROCEDURE tgetnum (id: ADDRESS): INTEGER;
```

```
(* id must be null-terminated *)
```

```
PROCEDURE tgetflag (id: ADDRESS): INTEGER;
```

```
(* id must be null-terminated *)
```

```
PROCEDURE tgetstr (id: ADDRESS; area: ADDRESS): ADDRESS;
```

```
(* id must be null-terminated *)
```

```
PROCEDURE tgoto (cm: ADDRESS; col, line: INTEGER) : ADDRESS;
```

```
(* cm must be null-terminated *)
```

```
PROCEDURE tputs (cp: ADDRESS; affcnt: INTEGER; p: PUTPROC);
```

```
(* cp must be null-terminated *)
```

```
END XXTermcap.
```