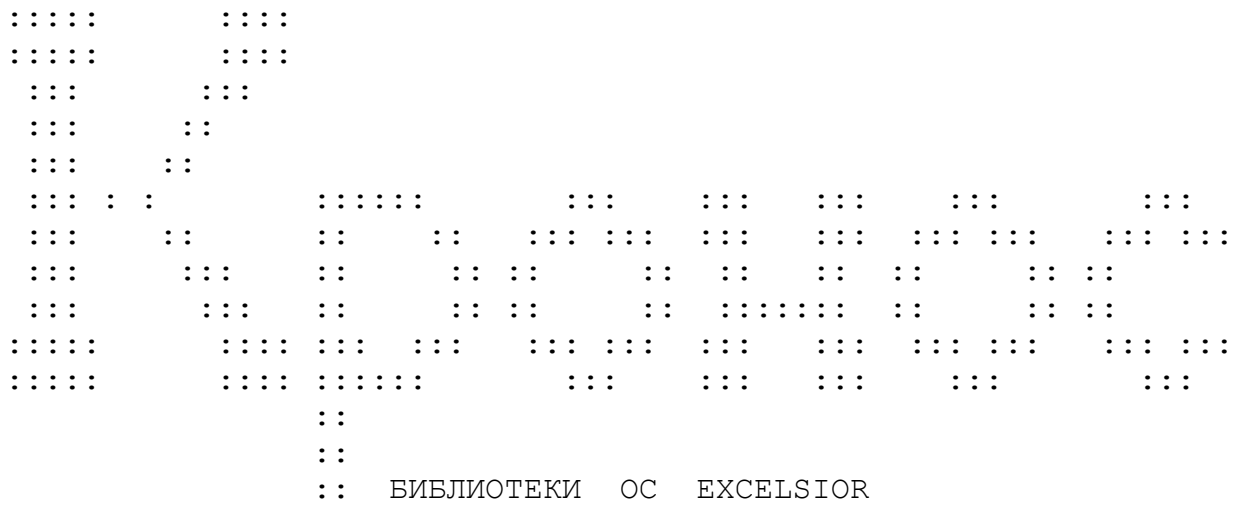
- Сибирское отделение АН СССР -

- Институт систем информатики -



БИБЛИОТЕКИ ОС EXCELSIOR

Новосибирск - 90

Содержание

1. ВВЕДЕНИЕ

Основным документом, определяющим каждую библиотеку, является ее определяющий модуль (DEFINITION MODULE), содержащий интерфейсы всех экспортируемых процедур, описания типов констант и переменных и подробные комментарии (на русском языке) об их возможном использовании.

Данный том представляет собой справочное пособие по библиотекам ОС Excelsior. Определяющие модули библиотек приведены в алфавитном порядке и составляют содержательную часть тома.

1.1. Структура библиотек

Библиотеки делятся на две группы:

1) определяющие внутренний интерфейс системы (иннерфейс);
2) предоставляющие возможность пользоваться системой удобно.

Вторая группа библиотек, в свою очередь, подразделяется на

а) интерфейсные - осуществляющие интерфейс программ пользователя с системой;
б) прикладные - надстройка над интерфейсными;
в) специализированные - библиотеки, на которых базируются специальные или прикладные системы (например, система графики, Модула-компилятор).

1.2. Какие библиотеки следует использовать

Пользовательским программам доступны для импорта все библиотеки. Мы хотим лишь дать некоторые рекомендации в целесообразности использования тех или иных групп библиотек.

Так, иннерфейсные библиотеки (их названия начинаются с "def") предназначены для внутренних нужд системы. Все они "прикрыты" соответствующими интерфейсными библиотеками, из которых и следует импортировать. Тем не менее, программисту полезно ознакомиться с определяющими модулями иннерфейсных библиотек и комментариями к ним.

Советуем также с осторожностью относиться к импорту из специальных библиотек. Эти библиотеки, являясь, по сути, частью специальных систем, могут претерпеть изменения по мере развития этих систем.

1.3. Библиотеки и их назначение

Как правило, название библиотеки отражает ее назначение.

Тем не менее, приведем список всех библиотек с указанием их возможного использования.

1.3.1. Интерфейсные библиотеки:

ASCII - содержит мнемоники некоторых ASCII-символов, их семантику, соответствующие контрольные символы;

BIO - осуществляет интерфейс файловой подсистемы;

Exceptions - обработка исключительных ситуаций;

Formats - универсальный форматный вывод;

Heap - распределение памяти из кучи;

Keyboard - поставляет процедуры работы с клавиатурой;

Lexicon - работа с драйверами сообщений;

lowLevel - поддержка низкоуровневого программирования;

myEditor - используется при написании фильтров. С помощью фильтров можно выполнять различные операции над редактируемым текстом.

Printer - реализует функции, определяемые возможностями конкретного принтера, для утилиты печати.

Signals - поставляет процедуры синхронизации процессов;

Statistics - позволяет получить информацию о некоторых параметрах системы;

StdIO - осуществляет стандартный ввод-вывод;

strEditor - поставляет процедуры редактирования строки; при этом ввод-вывод осуществляется на терминал;

Strings - работа со строками;

Tasks - работа с задачами;

Terminal - работа с терминалом;

Threads - работа с процессами;

Time - осуществляет службу времени в системе;

tskArgs - осуществляет обработку параметров, переданных задаче при запуске;

tskEnv - работа с окружением задачи.

1.3.2. Прикладные библиотеки:

Sorts - методы сортировки;

fsWalk - прогулки по файловому дереву;

realMath - операции над вещественными числами;

regExpr - работа с регулярными выражениями.

1.3.3. Иннерфейсные библиотеки. Рекомендуются для ознакомления. Поставляют константы и определяют типы:

defCode - структура кодофайла;

defCodes - мнемоники системы команд процессоров Кронос;

defErrors - общесистемные номера ошибок;

defKeyboard - определяет константы клавиатуры;

defPrinter - для принтеров;

defRequest - определяет тип запроса для драйверов;

defTasks - для задач;

defTerminal - для терминалов.

1.3.4. Специальные библиотеки для Модула-компилятора (рекомендуются для ознакомления):

coolDefs - определяет ввод-вывод компилятора;

coolIO - реализует ввод-вывод;

coolSym - структура симфайла;

coolSystem - содержит процедуры, необходимые компилятору от системы (чтобы ничего не импортировать из самой системы);

visCode - поддержка визуализации кодофайла;

xRef - структура реффайла.

```
DEFINITION MODULE ASCII; (* Leo & Ned 14-Apr-87. (c) KRONOS *)

CONST
(*****************************************************************
**  Мнемоника      Семантика       Контрольный Дополнительная **
** ASCII-7.68                        символ     мнемоника    **
**                                  CNTRL+char                **
*****************************************************************)

  NUL =  0c; (* NUL NUL -- line break     @  *)
  SOH =  1c; (* Начало заголовка          A  *)
  STX =  2c; (* Начало текста             B  *)
  ETX =  3c; (* Конец текста              C  *)  BREAK = ETX;
  EOT =  4c; (* Конец передачи            D  *)
  ENQ =  5c; (* Запрос                    E  *)
  ACK =  6c; (* Подтверждение             F  *)
  BEL =  7c; (* Звонок                    G  *)
   BS = 10c; (* Шаг назад                 H  *)
   HT = 11c; (* Горизонтальная табуляция  I  *)
   LF = 12c; (* Перевод строки            J  *)
   VT = 13c; (* Вертикальная табуляция    K  *)
   FF = 14c; (* Перевод формата           L  *)
   CR = 15c; (* Возврат каретки           M  *)
   SO = 16c; (* Национальный регистр      N  *)
   SI = 17c; (* Латинский регистр         O  *)

  DLE = 20c; (* Авторегистр 1             P  *)
  DC1 = 21c; (* Управление устройством 1  Q  *)  XON  = DC1;
  DC2 = 22c; (* ---------------------- 2  R  *)
  DC3 = 23c; (* ---------------------- 3  S  *)  XOFF = DC3;
  DC4 = 24c; (* ---------------------- 4  T  *)
  NAK = 25c; (* Отрицание                 U  *)
  SYN = 26c; (* Синхронизация             V  *)
  ETB = 27c; (* Конец блока               W  *)
  CAN = 30c; (* Аннулирование             X  *)
   EM = 31c; (* Конец носителя            Y  *)
  SUB = 32c; (* Замена                    Z  *)
  ESC = 33c; (* Авторегистр 2             [  *)
   FS = 34c; (* Разделитель файлов        \  *)
   GS = 35c; (* Разделитель групп         ]  *)
   RS = 36c; (* Разделитель записей       ^  *)
   US = 37c; (* Разделитель элементов     _  *)

SPACE = 40c; (* Пробел                       *)
  DEL =177c; (* "Забой"                      *)  RUBBOUT = DEL;
-----------------------------------------------------------------
CONST  (* Соглашения ОС  Excelsior о семантике нек. символов: *)

    NL = RS;     (* New Line. Отрабатывается как CR LF для
                    телевизоров  и для других последовательных
                    устройств.  Разделитель  строк в текстовых
                    файлах.
```

```
                *)
    EOF = FS;   (* End Of File. Конец файла. *)
-----------------------------------------------------------------

CONST (* типы символов *)
  control = 00; (* контрольный 0c..37c, 200c..237c       *)
  special = 01; (* специальный, т.е.  ?,/+!"#$%&  и т.д. *)
    digit = 02; (* 0123456789                            *)
    dig16 = 03; (* 0123456789ABCDEF                      *)
    cyril = 04; (* буква кириллицы                       *)
    latin = 05; (* буква латинского алфавита             *)
    small = 06; (* строчная буква                        *)
  capital = 07; (* прописная буква                       *)

PROCEDURE  KIND(ch:  CHAR):  BITSET;
(* Возвращает набор признаков символа *)

PROCEDURE   SMALL(ch: CHAR): CHAR;
(* превращает букву в строчную, не изменяет остальных символов *)

PROCEDURE CAPITAL(ch: CHAR): CHAR;
(* превращает букву в прописную, не изменяет остальных символов
*)

(*****************************************************************

------------------------  ПРИМЕЧАНИЯ  ------------------------
                         --------------

   OS  Excelsior  II  поддерживает работу с набором символов
   КОИ-8.   При   этом   символы   с   кодировками  0c..177c
   соответствуют    стандарту    ASCII-7.68,    в    остальных
   кодировках  располагаются  символы  кириллицы и некоторые
   служебные.

   Все символы делятся на следующие группы:
      - буквы (латиницы или кириллицы),
      - цифры;
      - специальные символы (?/+;!"#$%&'()=-_*:>.|\<,@01...)
      - контрольные (управляющие) символы

*****************************************************************)

END ASCII.
```

```
DEFINITION MODULE BIO; (* Leo 04-Sep-89. (c) KRONOS *)

(* Интерфейс файловой системы. *)

IMPORT SYSTEM;
IMPORT defRequest;

TYPE FILE;
     PATHs;

VAL null: FILE;
      cd: FILE;     (* рабочая директория           *)

    here: PATHs;    (* путь только из "."           *)
   empty: PATHs;    (* пустой путь                  *)
   cmask: BITSET;   (* маска создания файлов        *)
   iolen: INTEGER;  (* число байтов, переданных
                    последней операцией чтения/записи *)

    done: BOOLEAN;
   error: INTEGER;  ename: ARRAY [0..31] OF CHAR;

PROCEDURE open  (VAR file: FILE; path,mode: ARRAY OF CHAR);
PROCEDURE create(VAR file: FILE; path,mode: ARRAY OF CHAR;
                                      size: INTEGER);
PROCEDURE close (VAR file: FILE);
PROCEDURE purge (VAR file: FILE);
PROCEDURE flush (    file: FILE);
PROCEDURE link  (    file: FILE; path,mode: ARRAY OF CHAR);
PROCEDURE unlink(                path: ARRAY OF CHAR);
PROCEDURE mkdir (                path: ARRAY OF CHAR);

PROCEDURE equal(f0,f1: FILE): BOOLEAN;

PROCEDURE chmode(file: FILE; mode: ARRAY OF CHAR);

PROCEDURE fname(file: FILE; VAR name: ARRAY OF CHAR);

PROCEDURE mknode(path,device_name: ARRAY OF CHAR);

PROCEDURE dup(VAR dup: FILE; file: FILE);

PROCEDURE mount  (path,dev,info: ARRAY OF CHAR;
                        VAR lab: ARRAY OF CHAR; ro: BOOLEAN);
PROCEDURE unmount(path: ARRAY OF CHAR;      method: INTEGER);
(* method=0 unmount only if all ok *)
(* method=1 unmount only if all ok, otherwise all bad files
   purged *)
(* method=2 unconditionly unmount *)

PROCEDURE seek(file: FILE; offset,origin: INTEGER);
PROCEDURE pos (file: FILE): INTEGER;
```

```
PROCEDURE eof (file: FILE): INTEGER;

PROCEDURE cut   (file: FILE; size: INTEGER);
PROCEDURE end   (file: FILE; size: INTEGER);
PROCEDURE extend(file: FILE; size: INTEGER);

CONST
   is_file = {0};        is_dir  = {1};
   is_disk = {2};        is_tty  = {3};
   is_spec = {4};        is_sys  = {5};

PROCEDURE kind(file: FILE): BITSET;

PROCEDURE is_hidd(f: FILE): BOOLEAN;

PROCEDURE get_attr(file: FILE; no: INTEGER; VAR val: SYSTEM.WORD);
PROCEDURE set_attr(file: FILE; no: INTEGER;     val: SYSTEM.WORD);
CONST (* any attribute may be read by everyone *)
  a_links = 0;  -- read always, write for superuser only
  a_inode = 1;  -- read always, write for superuser only
  a_ctime = 2;  -- read always, write for owner     only
  a_wtime = 3;  -- read always, write for owner     only
  a_gid   = 4;  -- read only
  a_uid   = 5;  -- read only
  a_pro   = 6;  -- read only

PROCEDURE du    (cd: FILE; VAR free,used: INTEGER);
PROCEDURE fstype(f : FILE; VAR type,blocksize: INTEGER);
(* type 0: Excelsior-II, 1: Excelsior-iV *)

PROCEDURE buffers(f: FILE; no,len: INTEGER);

PROCEDURE check_io(halt_on_error: BOOLEAN);

PROCEDURE read (file: FILE; buf: SYSTEM.ADDRESS; len: INTEGER);
PROCEDURE write(file: FILE; buf: SYSTEM.ADDRESS; len: INTEGER);

PROCEDURE fread (file: FILE; buf: SYSTEM.ADDRESS; pos,len:
INTEGER);
PROCEDURE fwrite(file: FILE; buf: SYSTEM.ADDRESS; pos,len:
INTEGER);

PROCEDURE get(file: FILE; VAR data: ARRAY OF SYSTEM.WORD; len:
INTEGER);
PROCEDURE put(file: FILE;     data: ARRAY OF SYSTEM.WORD; len:
INTEGER);

----------------------- STREAMS -------------------------------

PROCEDURE getch(file: FILE; VAR ch: CHAR);
PROCEDURE putch(file: FILE;     ch: CHAR);
```

```
PROCEDURE getstr(file: FILE; VAR str: ARRAY OF CHAR; pos:
INTEGER);
PROCEDURE putstr(file: FILE;     str: ARRAY OF CHAR; pos:
INTEGER);

PROCEDURE print(file: FILE; fmt: ARRAY OF CHAR; SEQ args:
SYSTEM.WORD);

-----------------------------------------------------------

CONST (* protection bits: *)
  own_exec   =  {0};    gro_exec   =  {4};    oth_exec   =  {8};
  own_write  =  {1};    gro_write  =  {5};    oth_write  =  {9};
  own_read   =  {2};    gro_read   =  {6};    oth_read   = {10};
  own_search = own_exec;   gro_search = gro_exec;
                                        oth_search = oth_exec;

  link_pro  = {12};
  run_priv  = {14};   -- may be seted by superuser only
  run_uid   = {15};

PROCEDURE access(file: FILE): BITSET;
(* {own_exec..own_unlink,run_uid,run_gid} subset for current user
*)

PROCEDURE owner(file: FILE; VAR owner,group: INTEGER);

PROCEDURE chcmask(mask: BITSET);

PROCEDURE chaccess(file: FILE; mask: BITSET);
PROCEDURE chowner (file: FILE; owner,group: INTEGER);

PROCEDURE fopen  (cd: FILE; VAR file: FILE; path,mode: ARRAY OF
CHAR);
PROCEDURE fcreate(cd: FILE; VAR file: FILE;
          path,mode: ARRAY OF CHAR; size: INTEGER);

PROCEDURE flink  (cd: FILE; file: FILE; path,mode: ARRAY OF CHAR);
PROCEDURE funlink(cd: FILE;             path: ARRAY OF CHAR);
PROCEDURE fmkdir (cd: FILE;             path: ARRAY OF CHAR);

PROCEDURE fmknode(cd: FILE;        path,name: ARRAY OF CHAR);

PROCEDURE fmvdir (from,to: FILE; path,name: ARRAY OF CHAR);

PROCEDURE fmount  (cd: FILE; path,dev,info: ARRAY OF CHAR;
                                  VAR  lab: ARRAY OF CHAR; ro:
BOOLEAN);
PROCEDURE funmount(cd: FILE; path    : ARRAY OF CHAR;  method:
INTEGER);

PROCEDURE splitpathname(VAR path: ARRAY OF CHAR; VAR name: ARRAY
OF CHAR);
```

```
PROCEDURE lock(milisec: INTEGER; file: FILE); (* delay *)
PROCEDURE unlock(                file: FILE);

PROCEDURE chdir (path: ARRAY OF CHAR);
PROCEDURE chroot(path: ARRAY OF CHAR);

PROCEDURE open_paths (VAR dirs: PATHs; pathnames: ARRAY OF CHAR);
PROCEDURE close_paths(VAR dirs: PATHs);
PROCEDURE get_paths  (VAR dirs: PATHs;   envname: ARRAY OF CHAR);

PROCEDURE lookup(dirs: PATHs; VAR file: FILE; name,mode: ARRAY OF
CHAR);

CONST
   s_none     = 00;
   s_reverse  = 01;
   s_dirfwd   = 02;
   s_name     = 04;
   s_ext      = 08;
   s_time     = 16;
   s_cre      = 32;
   s_eof      = 64;

PROCEDURE dir_walk(cd: FILE; sort_kind: INTEGER);

PROCEDURE get_entry(cd: FILE; VAR name: ARRAY OF CHAR;
                              VAR mode: BITSET): BOOLEAN;
CONST -- file entry modes:
  e_file   = { };
  e_dir    = {0};
  e_hidden = {1};
  e_esc    = {2};
  e_sys    = {3};

PROCEDURE restart_walk(cd: FILE);

PROCEDURE end_walk (cd: FILE);

PROCEDURE mkfs  (device: FILE;
                 fstype: INTEGER;
                 block : INTEGER;
                 label : ARRAY OF CHAR;
                 bads  : ARRAY OF INTEGER);


-------------  for Xackers' applications only  -----------------
              ---------------------------------

PROCEDURE doio(f: FILE; VAR r: defRequest.REQUEST);


(*************************************************************
```

Ввиду обширности данной библиотеки комментарий к ней здесь не приводится. Библиотека подробно прокомментирована в описании файловой системы (см.).

**********)

END BIO.

```
DEFINITION MODULE BMG; (* nick 02-Mar-90. (c) KRONOS *)

IMPORT  SYSTEM, defScreen, defBMD;

TYPE
  REGION = defScreen.REGION;  BITMAP = defBMD.BITMAP;
  BLOCK  = defScreen.BLOCK;   BMD    = defBMD.BMD;
  FONT   = defScreen.FONT;

CONST
  rep = defScreen.rep;    bitmap  = defScreen.bitmap;
  xor = defScreen.xor;    reverse = defScreen.reverse;
  or  = defScreen.or;     normal  = defScreen.normal;
  bic = defScreen.bic;


PROCEDURE get_BMD(rgn: REGION; VAR ptr_bmd: BITMAP);
PROCEDURE set_BMD(rgn: REGION;     ptr_bmd: BITMAP);
PROCEDURE ini_BMD(bmd: BMD);

---------------------- BitBlock Procedures ---------------------
                       ---------------------

PROCEDURE cross(VAR des: BLOCK; blk0,blk1: BLOCK);

PROCEDURE overlap(des: REGION; VAR dblk: BLOCK;
                  sou: REGION; VAR sblk: BLOCK);

PROCEDURE bblt   (des: REGION; dblk: BLOCK;
                  sou: REGION; sblk: BLOCK);

---------------  Graphic Primitive Procedures  -----------------
                ---------------------------------

PROCEDURE erase  (bmd: REGION);

PROCEDURE fill   (bmd: REGION; w: INTEGER; SEQ pattern:
SYSTEM.WORD);
PROCEDURE pattern(bmd: REGION; w: INTEGER;     pattern: ARRAY OF
SYSTEM.WORD);

PROCEDURE dot  (bmd: REGION; X,Y: INTEGER);
PROCEDURE line (bmd: REGION; X0,Y0,X1,Y1: INTEGER);
PROCEDURE rect (bmd: REGION; X0,Y0,X1,Y1: INTEGER);
PROCEDURE frame(bmd: REGION; X0,Y0,X1,Y1: INTEGER);
PROCEDURE circ (bmd: REGION; X,Y,R: INTEGER);
PROCEDURE arc  (bmd: REGION; X0,Y0,xa,ya,xb,yb,r: INTEGER);
PROCEDURE arc3 (bmd: REGION; x0,y0,x1,y1,x2,y2: INTEGER);

PROCEDURE circf(bmd: REGION; X,Y,R: INTEGER);
PROCEDURE trif (bmd: REGION; x0,y0,x1,y1,x2,y2: INTEGER);
```

```
PROCEDURE contour(bmd: REGION; xy: ARRAY OF INTEGER);
PROCEDURE poligon(bmd: REGION; xy: ARRAY OF INTEGER);

PROCEDURE ring(bmd: REGION; x0,y0,r1,r2: INTEGER);

------------------  Graphic Text Support  --------------------
                   ------------------------

PROCEDURE len(fnt: FONT; fmt: ARRAY OF CHAR; SEQ arg:
SYSTEM.WORD): INTEGER;

PROCEDURE write(bmd: REGION;  x,y: INTEGER;
                fnt: FONT;    str: ARRAY OF CHAR; pos,len:
INTEGER);

PROCEDURE xwrite(bmd: REGION; x,y: INTEGER;
                 fnt: FONT;   str: ARRAY OF CHAR; pos,len:
INTEGER): INTEGER;

PROCEDURE print(bmd: REGION;  x,y: INTEGER;
                fnt: FONT; format: ARRAY OF CHAR; SEQ arg:
SYSTEM.WORD);

PROCEDURE xprint(bmd: REGION;  x,y: INTEGER;
                 fnt: FONT; fmt: ARRAY OF CHAR; SEQ arg:
SYSTEM.WORD): INTEGER;

PROCEDURE writech(bmd: REGION; x,y: INTEGER; fnt: FONT; ch:
CHAR);

END BMG.
```

```
DEFINITION MODULE CCD; (* Leg 21-Mar-90. (c) KRONOS *)

(* Interface to coordinates control devices *)

IMPORT  SYSTEM;

VAL
  done : BOOLEAN;
  error: INTEGER;
  note : ARRAY [0..63] OF CHAR;

  type : INTEGER;
  keys : INTEGER;
  on   : ARRAY [0..31] OF CHAR;
  off  : ARRAY [0..31] OF CHAR;
  rel  : BOOLEAN;
  x_range,y_range: INTEGER;

PROCEDURE set_key(no: INTEGER; on: BOOLEAN; ch: CHAR);

PROCEDURE state(): BITSET;

PROCEDURE read(VAR x,y: INTEGER; VAR ch: CHAR);

PROCEDURE busy_read(VAR x,y: INTEGER; VAR ch: CHAR);

PROCEDURE ready(): INTEGER;

PROCEDURE wait(timeout: INTEGER);

PROCEDURE write(ch: CHAR);

PROCEDURE clear;

PROCEDURE action(no: INTEGER; SEQ args: SYSTEM.WORD);

PROCEDURE attach(inp_dev_name,out_dev_name: ARRAY OF CHAR);

END CCD.
```

VARIABLES

type : INTEGER;
CCD type identifier.

keys : INTEGER;
number of buttons on CCD.

on : ARRAY [0..31] OF CHAR;
Used only first -keys- symbols.
They denote symbols wich appear in internal buffer

```
    when respective button is being supressed.

  off  : ARRAY [0..31] OF CHAR;
    Used only first -keys- symbols.
    They denote symbols wich appear in internal buffer
    when respective button is being released.

  rel  : BOOLEAN;
    Characterizes features of CCD.
    After calling "read" x & y contains:
      "rel"=TRUE : shift relatively to previous position
      "rel"=FALSE: absolute value.

  x_range,y_range: INTEGER;
    In case of "rel"=FALSE they contain range of changing
    x & y coordinates in areas [0..x_range-1] & [0..y_range-1],
    otherwise contain -1.

-------------------------- PROCEDURES --------------------------
                           ------------

PROCEDURE set_key(No: INTEGER; pr: BOOLEAN; ch: CHAR);
          ~~~~~~~
  Defines symbol appearing in input buffer on
supressing("pr"=TRUE)
  or releasing("pr"=FALSE) of button # No on CCD
  Initially for "pr"=TRUE  & "No"=1.. :  on [No]=001c..
            for "pr"=FALSE & "No"=1.. :  off[No]=201c..


PROCEDURE state(): BITSET;
          ~~~~~
  Returns state of buttons on CCD
  (bit #0 in bitset corresponds to button #0).


PROCEDURE read(VAR x,y: INTEGER; VAR ch: CHAR);
          ~~~~
  After calling x & y contain absolute coordinates ("rel" is
FALSE) or
  shift relatively to previous position ("rel" is TRUE),
  "ch" may contain
     a)   symbol corresponds to buttons state change;
     b)   symbol, wich was placed in internal
          buffer with the help of procedure write;
     otherwise 0c.

  x,y & ch will never be equal to zero simultaniously,
  excluding case of calling "write(0c)".


PROCEDURE busy_read(VAR x,y: INTEGER; VAR ch: CHAR);
          ~~~~~~~~~
```

```
  Read without wait. If internal buffer is empty then
  x,y and ch contain zero.


PROCEDURE ready(): INTEGER;
          ~~~~~
  Returns number of ready items in internal buffer.


PROCEDURE wait(timeout: INTEGER);
          ~~~~
  Waits till internal buffer is not empty
  or "timeout" (in miliseconds) is exhausted.
  If timeout occurs then "done"=FALSE "error"=timeout.


PROCEDURE write(ch: CHAR);
          ~~~~~
  Puts symbol "ch" in internal buffer with
  x=0, y=0 (in case of "rel"=TRUE) OR
  x="last_x", y="last_y" (in case of "rel"=FALSE).


PROCEDURE clear;
          ~~~~~
  Clears internal buffer, adjusting right result of state().


PROCEDURE action(no: INTEGER; SEQ args: SYSTEM.WORD);
          ~~~~~~
  No need to use it.


PROCEDURE attach(inp_dev_name,out_dev_name: ARRAY OF CHAR);
          ~~~~~~
  Connects new CCD with driver name for output "out_dev_name" &
  driver name for input "inp_dev_name".


Before initialization this module task argument string or
environment may contain strings "ccdIN" & "ccdOUT" to define
input & output device names respectively. If they are absent or
failure during opening then variable "done" is FALSE, variable
"error" contains error code and it is necessary to call "attach"
with proper names.
```

```
DEFINITION MODULE coolDefs; (* Ned 06-Jan-90. (c) KRONOS *)

IMPORT  SYSTEM;

CONST -- io kinds
  def    = 0;   -- definition module
  imp    = 1;   -- implementation module
  main   = 2;   -- module
  text   = 3;   --
  sym_in = 4;
  sym_ou = 5;
  ref    = 6;
  code   = 7;

TYPE
  PRINT  = PROCEDURE (ARRAY OF CHAR, SEQ SYSTEM.WORD);
  io_ptr = POINTER TO io_rec;
  io_rec = RECORD
             kind : INTEGER;
             doio : PROCEDURE (io_ptr);
             done : BOOLEAN;
             print: PRINT;          -- for error message
             buf  : STRING;
             len  : INTEGER;        -- of buf
             exts : SYSTEM.ADDRESS;
           END;

TYPE
  INI    = PROCEDURE (VAR io_ptr, ARRAY OF CHAR, INTEGER, PRINT);
  EXI    = PROCEDURE (VAR io_ptr);
  ERROR  = PROCEDURE (
                       INTEGER,           -- line
                       INTEGER,           -- col
                       ARRAY OF CHAR,     -- source line
                       ARRAY OF CHAR,     -- format
                       SEQ SYSTEM.WORD    -- args
                       );
            -- line and col defines position of error in
            -- source line. format and args defines error
message.

END coolDefs.
```

```
DEFINITION MODULE coolIO; (* Ned 04-Mar-90. (c) KRONOS *)

IMPORT  def: coolDefs;

PROCEDURE ini(VAR io: def.io_ptr;
                name: ARRAY OF CHAR;
                unit: INTEGER;
               print: def.PRINT);

PROCEDURE exi(VAR io: def.io_ptr);

PROCEDURE set(text_path,sym_path,out: ARRAY OF CHAR; print:
def.PRINT);

PROCEDURE dispose;

END coolIO.
```

```
DEFINITION MODULE coolSym; (* Ned 25-May-88. (c) KRONOS *)
                           (* Ned 01-Nov-88. (c) KRONOS *)
                           (* Ned 15-Apr-90. (c) KRONOS *)

(* Модуль определяет стандарт симфайла (реффайла) *)

CONST -- Номера стандартных типов
  null      =  0;    (*              *)
  addr      =  1;    (* ADDRESS      *)
  word      =  2;    (* WORD         *)
  int       =  3;    (* INTEGER      *)
  char      =  4;    (* CHARACTER    *)
  real      =  5;    (* REAL         *)
  bool      =  6;    (* BOOLEAN      *)
  bitset    =  7;    (* BITSET       *)
  string    =  8;    (* STRING       == DYNARR OF CHAR *)
  nan       = 31;    (* NAN          *)

CONST -- теги типовых значений

  enumtype  = 20h;     --
  range     = 21h;     -- X(base)  X(min) X(max)
  array     = 22h;     -- X(index) X(base)
  openarr   = 23h;     -- X(base)
  pointer   = 24h;     --                      -- see linkage
  record    = 25h;     -- X(base)  X(size)
  set       = 26h;     -- X(base)
  proctype  = 27h;     --
  functype  = 28h;     -- X(base)
  hidden    = 29h;     -- X(size)
  linkage   = 2Ah;     -- X(pointer_type) X(base)
  dynarr    = 2Bh;     -- X(base) X(dimensions)
  class     = 2Ch;     -- X(base)

CONST -- теги объектов и теги компонент

  const     = 30h;     -- ident X(base) X(val)
  struct    = 31h;     -- ident X(base) X(mod_no) X(addr)
                       --                X(words) { byte }
  sconst    = 32h;     -- ident X(base) X(mod_no) X(addr)
  enum      = 33h;     -- ident X(base) X(mod_no) X(val)

  var       = 34h;     -- ident X(base) X(scope)  X(addr) X(kind)
  proc      = 35h;     -- ident X(base) X(scope)  X(addr)
  type      = 36h;     -- ident X(base) X(mod_no)
  module    = 37h;     -- ident X(mod_no)

  parm      = 40h;     -- ident X(base) X(addr)   X(kind)
  field     = 41h;     -- ident X(base) X(addr)
  method    = 42h;     -- ident X(base) X(addr)

CONST -- биты в виде параметра и переменной
```

```
  varparam = 0;
  seqparam = 1;
  readonly = 2;
--         = 3;
  private  = 4;        -- private generation tags [4..31]
                       -- (may have different meaning for
                       -- different compilers and/or
                       -- architectures)

CONST -- определение внешнего модуля
  import    = 050h;    -- module_ident language_ident
                       -- X(def_time)  - время компиляции
                       --                def модуля
                       -- 1(module_kind)
                       -- X(addr)

CONST -- виды модулей
  def  = 0;     -- definition module
  imp  = 1;     -- implementaion module
  prog = 2;     -- program module

CONST -- дополнительные атомы
  strange   = 051h;    -- X(bytes) { byte }
  atrs      = 052h;    -- 1(number of atrs) { X(attribute) }
  xpos      = 053h;    -- X(proc_no) X(pc) 2(line) 1(col)
  endmodule = 054h;    -- X(mod_no)
  endproc   = 055h;    -- X(proc_no)
  end_CU    = 056h;    -- X(no_proc) X(no_var)
  eosf      = 0FDh;    -- end of symfile

CONST
  MAGIC   = 04D5953FFh; -- первое слово симфайла: "ЪSYM"
  VERSION = 3;          -- текущая версия

TYPE -- заголовок симфайла
  header = RECORD
             magic   : INTEGER;
             vers    : INTEGER;
             offset  : INTEGER;
             def_time: INTEGER;
             imp_time: INTEGER;
           END;

(*************************************************************
```

---------- ПРИМЕЧАНИЕ ----------

Симфайл(реффайл) состоит из заголовка, некоторой дополнительной информации (может отстутствовать) и собствено информации о объектах модуля. Далее под словом симфайл будет пониматься эта информация. Заголовок содержит "волшебное" слово, номер версии, смещение (в байтах) до начала собственно симфайла и

времена компиляции определяющего и реализующего модуля. Между заголовком и симфайлом может рассполагаться дополнительная информация любой длины.

Симфайл - это последовательность областей видимости, соответствующих процедурам и модулям. Область видимости - это последовательность атомов. Атомами являются типовые значения, компоненты, объекты, странные сегменты, элементы соответствия срок коду. Все типовые значения пронумерованы в порядке появления, причем нумерация начинается с 32. Диапазон номером [0..31] используется для стандартных типов. Типовые значения для стандартных типов в симфайле не встречаются. Каждый атом представлен тегом (1 байт) за которым идет (возможно пустая) последовательность атрибутов. Представление атрибута может быть фиксированой длины (например 1 байт), быть упакованным (т.е. занимать разное число байтов в зависимости от значения, или являться последовательностью байтов указанной длины.

Данный модуль определяет константы - номера стандартных типов, теги объектов, типовых значений и компонент, и теги оформления симфайла и упакованных атрибутов. Для каждого тега указан набор обязательных атрибутов. Используются следующие термины:

```
base   - базовый тип (номер типа)
index  - тип индекса (номер типа)
size   - размер      (в словах)
ident  - имя объекта    { byte } 0c
mod_no - номер в списке импорта внешнего модуля, которому
         принадлежит объект. Не путать со scope!!!
         Все объекты у которых этот атрибут отсутствует,
         принадлежат модулю 0 (т.е. модулю, для которого
         построен симфайл).
scope  - информация для генерации.
         если< 0 -- индикант внешнего модуля
         если>=0 -- уровень описания объекта
addr   - информация для генерации.
         индикант (номер, индекс, адрес, смещение, ...)
         переменной, поля, процедуры и т.д.
```

Используются следующие обозначения:

```
1(имя_атрибута)  - атрибут занимает 1 байт
X(имя_атрибута)  -  упакованный  атрибут
```

За каждым атомом может идти последовательность дополнительных атрибутов состоящая из тега, числа атрибутов и упакованных атрибутов.

Метод упаковки атрибутов:

значение	интерпретация	диапазон
[8..255]	значение-128	-120..127
0	значение следующего байта	0..255
1	- значение следующего байта	-255..0
2	значение следующих 2 байтов	0..10000h
3	значение следующих 4 байтов	все
[4..7]	для расширений	

```
  sym_file::
                header version
                { import }
                unit
                eosf
  unit::        { atom [ atrs ] }
  atom::        type_value | component | object | xpos | strange

*****************************************************************)

END coolSym.
```

```
DEFINITION MODULE coolSystem; (* Leo 05-Jun-88. (c) KRONOS *)
                              (* Ned 12-Oct-88. (c) KRONOS *)
                              (* Ned 28-Sep-89. (c) KRONOS *)

IMPORT  sys: SYSTEM;
IMPORT comp: coolDefs;

VAR
  ini  : comp.INI;
  exi  : comp.EXI;
  error: comp.ERROR;
  print: comp.PRINT;

--------------------------  SERVICE  --------------------------
                           -----------

PROCEDURE xprint(format: ARRAY OF CHAR; SEQ args: sys.WORD);

PROCEDURE append(VAR s: ARRAY OF CHAR; f: ARRAY OF CHAR; SEQ
args: sys.WORD);
PROCEDURE sprint(VAR s: ARRAY OF CHAR; f: ARRAY OF CHAR; SEQ
args: sys.WORD);
PROCEDURE app_time(VAR s: ARRAY OF CHAR; time: INTEGER);

PROCEDURE time(): INTEGER;

PROCEDURE final(closure_proc: PROC);

-----------------------  M E M O R Y  -----------------------
                        ---------------

PROCEDURE   ALLOCATE(VAR a: sys.ADDRESS; size: INTEGER);
PROCEDURE DEALLOCATE(VAR a: sys.ADDRESS; size: INTEGER);
PROCEDURE REALLOCATE(VAR a: sys.ADDRESS; VAR high: INTEGER;
                                    len,byte_size: INTEGER);

----------------------  STACKS & QUEUES  ----------------------
                       -------------------

TYPE QUEUE;

PROCEDURE lifo (VAR q: QUEUE); -- инициализация стека
PROCEDURE fifo (VAR q: QUEUE); -- инициализация очереди

PROCEDURE clear(VAR q: QUEUE); -- очистка

PROCEDURE push(q: QUEUE; info: sys.WORD);
PROCEDURE pop (q: QUEUE; VAR info: sys.WORD): BOOLEAN;

END coolSystem.
```

```
DEFINITION MODULE defBMD; (* nick 01-Jun-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE
  BITMAP = POINTER TO BMD;

  BMD    = RECORD
             W,H,WPL: INTEGER;
             BASE   : SYSTEM.ADDRESS;
             PATTERN: BITSET;
             layers : ARRAY [0..31] OF SYSTEM.ADDRESS
           END;

END defBMD.
```

```
DEFINITION MODULE defCCD; (* Leg 21-Mar-90. (c) KRONOS *)

(*            Coordinates Control Devices              *)

TYPE STATE = RECORD
               type   : INTEGER;
               keys   : INTEGER;
               on     : ARRAY [0..31] OF CHAR;
               off    : ARRAY [0..31] OF CHAR;
               rel    : BOOLEAN;
               x_range: INTEGER;
               y_range: INTEGER;
             END;

CONST
      _info    = 01h;      -- (POINTER TO STATE);
      _clear   = 02h;      -- (POINTER TO STATE);
      _code    = 03h;      -- key_no on_off  char
      _state   = 04h;      -- ADR of keys bitset

END defCCD.
```

```
DEFINITION MODULE defCode; (* Ned 01-Dec-89. (c) KRONOS *)

(* Определяет стандарт кода *)

CONST
  info_size  = 16;       (* размер информационной части     *)
  vers_mask  = {0..7};   (* маска номера версии в поле vers *)
  cur_vers   = {1};      (* текущий номер версии            *)
  check_time = {8};      (* признак контроля времени        *)
  cpu_mask   = {24..31}; (* система команд                  *)

TYPE
  code_ptr = POINTER TO code_rec;
  code_rec = RECORD
               vers     : BITSET;  -- версия кода
               def_time : INTEGER;
               imp_time : INTEGER;
               str_size : INTEGER; -- размер строкового пула
               code_size: INTEGER; -- размер кода
               min_stack: INTEGER; -- минимальный размер стека
               add_stack: INTEGER; -- оценка стека
               glo_size : INTEGER; -- размер глобальной области
               no_exts  : INTEGER; -- размер локальной DFT
               no_proc  : INTEGER; -- число процедур
               no_mg    : INTEGER; -- число мультиглобалов
               size     : INTEGER; -- размер кода
               language : ARRAY [0..3] OF CHAR;
               tag      : INTEGER;
               usercode : BITSET;  -- наследуется из файла
               unused__F: INTEGER;
             END;

(*************************************************************
```

------------------------ ПРИМЕЧАНИЯ ------------------------

Структура кода (c: code_ptr):

- информационная часть <c>;
- строковый пул <info_size>;
  начинается с имени модуля;
- код <info_size+c^.str_size>;
  состоит из процедурной таблицы и собственно
  кода процедур;
- таблица мультиглобалов
  <info_size+c^.str_size+c^.code_size>
  состоит из пар слов:
  номер в глобальной области и размер в словах;
- список импорта
  <info_size+c^.str_size+c^.code_size+c^.no_mg*2>
  состоит из времени компиляции

```
          соответствующего определяющего модуля и
          имени модуля, дополненного символами 0c до
          границы слова (один символ 0c всегда есть).
          Число имен в списке импорта = no_exts-1.
```

После имени раздела указывается адрес начала раздела относительно начала кода.

```
c^.vers:
        - содержит в младшем байте номер текущей версии
          кода и в остальных дополнительные признаки.

c^.glo_size:
        - определяет размер глобальной области,
          которая состоит из локальной DFT (размером c^.no_exts)
          и области для глоабльных переменных
          (размером c^.glo_size-c^.no_exts).

c^.no_exts:
        - определяет размер локальной DFT. Он равен числу
          импортируемых модулей плюс 1.

c^.min_stack:
        - определяет минимальный размер стека для модуля.
          При загрузке задачи минимальный размер стека для
          задачи получается как сумма минимальных размеров для
          всех модулей задачи.

c^.add_stack:
        - оценка дополнительного размера стека вычисляемая
          компилятором.
          При загрузке задачи оценка стека задачи есть
          максимум из оценок стеков модулей.

c^.no_mg:
        - число мультиглобалов модуля.
          размер таблицы мультиглобалов равен c^.no_mg*2.

c^.language:
        - мнемоника языка и/или компилятора (не используется
          загрузчиком).

**********************************************************)

END defCode.
```

```
DEFINITION MODULE defCodes; (* Shu 11-Jul-86. (c) KRONOS *)
                            (* Ned 28-Sep-89. (c) KRONOS *)

(* Модуль содержит мнемоники команд процессора. *)

CONST

   li0  = 00h;   llw   =  20h;   lxb   =  40h;   lsw0  =  60h;
   li1  = 01h;   lgw   =  21h;   lxw   =  41h;   lsw1  =  61h;
   li2  = 02h;   lew   =  22h;   lgw2  =  42h;   lsw2  =  62h;
   li3  = 03h;   lsw   =  23h;   lgw3  =  43h;   lsw3  =  63h;
   li4  = 04h;   llw4  =  24h;   lgw4  =  44h;   lsw4  =  64h;
   li5  = 05h;   llw5  =  25h;   lgw5  =  45h;   lsw5  =  65h;
   li6  = 06h;   llw6  =  26h;   lgw6  =  46h;   lsw6  =  66h;
   li7  = 07h;   llw7  =  27h;   lgw7  =  47h;   lsw7  =  67h;

   li8  = 08h;   llw8  =  28h;   lgw8  =  48h;   lsw8  =  68h;
   li9  = 09h;   llw9  =  29h;   lgw9  =  49h;   lsw9  =  69h;
   li0A = 0Ah;   llw0A =  2Ah;   lgw0A =  4Ah;   lsw0A =  6Ah;
   li0B = 0Bh;   llw0B =  2Bh;   lgw0B =  4Bh;   lsw0B =  6Bh;
   li0C = 0Ch;   llw0C =  2Ch;   lgw0C =  4Ch;   lsw0C =  6Ch;
   li0D = 0Dh;   llw0D =  2Dh;   lgw0D =  4Dh;   lsw0D =  6Dh;
   li0E = 0Eh;   llw0E =  2Eh;   lgw0E =  4Eh;   lsw0E =  6Eh;
   li0F = 0Fh;   llw0F =  2Fh;   lgw0F =  4Fh;   lsw0F =  6Fh;

   lib  = 10h;   slw   =  30h;   sxb   =  50h;   ssw0  =  70h;
   lid  = 11h;   sgw   =  31h;   sxw   =  51h;   ssw1  =  71h;
   liw  = 12h;   sew   =  32h;   sgw2  =  52h;   ssw2  =  72h;
   lin  = 13h;   ssw   =  33h;   sgw3  =  53h;   ssw3  =  73h;
   lla  = 14h;   slw4  =  34h;   sgw4  =  54h;   ssw4  =  74h;
   lga  = 15h;   slw5  =  35h;   sgw5  =  55h;   ssw5  =  75h;
   lsa  = 16h;   slw6  =  36h;   sgw6  =  56h;   ssw6  =  76h;
   lea  = 17h;   slw7  =  37h;   sgw7  =  57h;   ssw7  =  77h;

   jflc = 18h;   slw8  =  38h;   sgw8  =  58h;   ssw8  =  78h;
   jfl  = 19h;   slw9  =  39h;   sgw9  =  59h;   ssw9  =  79h;
   jfsc = 1Ah;   slw0A =  3Ah;   sgw0A =  5Ah;   ssw0A =  7Ah;
   jfs  = 1Bh;   slw0B =  3Bh;   sgw0B =  5Bh;   ssw0B =  7Bh;
   jblc = 1Ch;   slw0C =  3Ch;   sgw0C =  5Ch;   ssw0C =  7Ch;
   jbl  = 1Dh;   slw0D =  3Dh;   sgw0D =  5Dh;   ssw0D =  7Dh;
   jbsc = 1Eh;   slw0E =  3Eh;   sgw0E =  5Eh;   ssw0E =  7Eh;
   jbs  = 1Fh;   slw0F =  3Fh;   sgw0F =  5Fh;   ssw0F =  7Fh;



   reset= 80h;   lss   = 0A0h;   move  = 0C0h;   incl  = 0E0h;
   quit = 81h;   leq   = 0A1h;   chknil= 0C1h;   excl  = 0E1h;
   getm = 82h;   gtr   = 0A2h;   lsta  = 0C2h;   inl   = 0E2h;
   setm = 83h;   geq   = 0A3h;   comp  = 0C3h;   quot  = 0E3h;
   trap = 84h;   equ   = 0A4h;   gb    = 0C4h;   inc1  = 0E4h;
   tra  = 85h;   neq   = 0A5h;   gb1   = 0C5h;   dec1  = 0E5h;
   tr   = 86h;   abs   = 0A6h;   chk   = 0C6h;   inc   = 0E6h;
```

```
   idle = 87h;   neg   = 0A7h;   chkz  = 0C7h;   dec   = 0E7h;

   add  = 88h;   or    = 0A8h;   alloc = 0C8h;   stot  = 0E8h;
   sub  = 89h;   and   = 0A9h;   entr  = 0C9h;   lodt  = 0E9h;
   mul  = 8Ah;   xor   = 0AAh;   rtn   = 0CAh;   lxa   = 0EAh;
   div  = 8Bh;   bic   = 0ABh;   nop   = 0CBh;   lpc   = 0EBh;
   shl  = 8Ch;   in    = 0ACh;   cx    = 0CCh;   bbu   = 0ECh;
   shr  = 8Dh;   bit   = 0ADh;   ci    = 0CDh;   bbp   = 0EDh;
   rol  = 8Eh;   not   = 0AEh;   cf    = 0CEh;   bblt  = 0EEh;
   ror  = 8Fh;   mod   = 0AFh;   cl    = 0CFh;   pdx   = 0EFh;

   io0  = 90h;   decs  = 0B0h;   cl0   = 0D0h;   swap  = 0F0h;
   io1  = 91h;   drop  = 0B1h;   cl1   = 0D1h;   lpa   = 0F1h;
   io2  = 92h;   lodfv = 0B2h;   cl2   = 0D2h;   lpw   = 0F2h;
   io3  = 93h;   store = 0B3h;   cl3   = 0D3h;   spw   = 0F3h;
   io4  = 94h;   stofv = 0B4h;   cl4   = 0D4h;   sswu  = 0F4h;
   rcmp = 95h;   copt  = 0B5h;   cl5   = 0D5h;   rchk  = 0F5h;
   wmv  = 96h;   cpcop = 0B6h;   cl6   = 0D6h;   rchkz = 0F6h;
   bmv  = 97h;   pcop  = 0B7h;   cl7   = 0D7h;   cm    = 0F7h;

   fadd = 98h;   for1  = 0B8h;   cl8   = 0D8h;   chkbox= 0F8h;
   fsub = 99h;   for2  = 0B9h;   cl9   = 0D9h;   bmg   = 0F9h;
   fmul = 9Ah;   entc  = 0BAh;   cl0A  = 0DAh;   activ = 0FAh;
   fdiv = 9Bh;   xit   = 0BBh;   cl0B  = 0DBh;   usr   = 0FBh;
   fcmp = 9Ch;   addpc = 0BCh;   cl0C  = 0DCh;   sys   = 0FCh;
   fabs = 9Dh;   jump  = 0BDh;   cl0D  = 0DDh;   nii   = 0FDh;
   fneg = 9Eh;   orjp  = 0BEh;   cl0E  = 0DEh;   dot   = 0FEh;
   ffct = 9Fh;   andjp = 0BFh;   cl0F  = 0DFh;   invld = 0FFh;

-----------------------------------------------------------------
--            Команды ввода/вывода 90h..95h                    --
---------------+-----------------------------------+------------
-- Kronos 2.2  |  Kronos 2.5   |                   |            --
-- Kronos 2.6q |               |                   |            --
---------------+---------------+-------------------+------------

   inp  = io0;
   out  = io1;
                 trb  = io3;

-----------------------------------------------------------------

CONST -- ffct
  ffct_float = 0;
  ffct_trunc = 1;

CONST -- quot
  quot_shr   = 0;
  quot_quo   = 1;
  quot_and   = 2;
  quot_rem   = 3;

CONST -- sys
```

```
  sys_cpu    = 0;
  sys_dot    = 1;
  sys_vers   = 2;

CONST -- bmg
  bmg_inrect = 0;
  bmg_dvl    = 1;
  bmg_bblt   = 2;
  bmg_dch    = 3;
  bmg_clip   = 4;
  bmg_line   = 5;
  bmg_circ   = 6;
  bmg_arc    = 7;
  bmg_ftri   = 8;
  bmg_fcirc  = 9;

END defCodes.
```

```
DEFINITION MODULE defErrors; (* Ned 16-Oct-89. (c) KRONOS *)
                             (* Leo 03-Nov-89. (c) KRONOS *)

(*  Модуль содержит определение кодов ошибок OS Excelsior *)

CONST

  ok = 0;  (* см. Примечания *)

                    --- common errors ---

  no_memory          =  80h;    -- нет памяти
  not_enough         =  81h;    -- недостаток ресурса
  busy               =  82h;    -- ресурс занят
  bad_name           =  83h;    -- плохое имя
  bad_parm           =  84h;    -- плохой параметр
  inv_op             =  85h;    -- неверная (нереализованая)
операции
  ipted_op           =  86h;    -- прерванная операции
  bad_desc           =  87h;    -- испорченный дескриптор
  ill_desc           =  88h;    -- непригодный в данном случае
дескриптор
  sec_vio            =  89h;    -- security violation
  su_only            =  8Ah;    -- for super user (white man)
only
  inconsistency      =  8Bh;    -- несогласованность данных
  ill_vers           =  8Ch;    -- некоректная версия данных
  duplicate          =  8Dh;    -- объект с таким именем уже есть
  unsuitable         =  8Eh;    -- неподходящий объект
  no_entry           =  8Fh;    -- нет такого
  undef              =  90h;    -- [еще|уже] неопределенный
объект
  ill_access         =  91h;    -- неправильный метод доступа к
объекту

                  --- file system errors ---

  is_dir             = 0A0h;
  xcross             = 0A1h;
  not_dir            = 0A2h;
  no_data            = 0A3h;
  bad_fsys           = 0A4h;
  too_large          = 0A5h;
  not_blocked        = 0A6h;
  no_space           = 0A7h;
  fsys_full          = 0A8h;

                     --- i/o errors ---

  io_error           =  80000001h;

    not_ready        =  80000101h;    -- { 8}
```

```
    time_out          =  80000201h;    -- { 9}
    write_pro         =  80000401h;    -- {10}
    seek_err          =  80000801h;    -- {11}
    inv_dma           =  80001001h;    -- {12}
    data_crc          =  80002001h;    -- {13}
    head_crc          =  80004001h;    -- {14}
    miss_sid          =  80008001h;    -- {15} sector id not found
    miss_did          =  80010001h;    -- {16} data   id not found
    hw_fail           =  80020001h;    -- {17} hard_ware failure
    seek_0            =  80040001h;    -- {18} seek track 0
incomplete
    bad_block         =  80080001h;    -- {19} marked bad_block
found
    inv_dad           =  80100001h;    -- {20} invalid device
address
    chk_err           =  80200001h;    -- {21} data compare error
    ecc_err           =  80400001h;    -- {22} non correctable ECC
error
    prog_err          =  80800001h;    -- {23} device programming
error
    unsafe            =  81000001h;    -- {24} unsafe device
detected
    par_err           =  82000001h;    -- {25} parity error
    frm_err           =  84000001h;    -- {26} frame error
    over_run          =  88000001h;    -- {27} data over run
    write_fail        =  90000001h;    -- {28} write_fail

                          --- traps ---

  quit                = 02h;   --  QUIT
  mem_vio             = 03h;   --  memory violation
  power_crash         = 04h;   --  power crash
  inv_instr           = 07h;   --  unimplemented instruction
  call_ipt            = 08h;   --  call interrupt
  rtn_ipt             = 09h;   --  return interrupt
  trace_ipt           = 0Bh;   --  trace  interrupt
  pstk_overflow       = 40h;   --  P-stack overflow
  int_overflow        = 41h;   --  integer overflow
  real_overflow       = 42h;   --  real overflow
  real_underflow      = 43h;   --  real underflow
  addr_underflow      = 44h;   --  address underflow
  case_error          = 45h;   --  CASE without ELSE
  rtn_error           = 46h;   --  return from function without
result
  halt                = 47h;   --  HALT
  assert              = 48h;   --  ASSERT
  instr_0FF           = 49h;   --  invalid instruction
  bounds_check        = 4Ah;   --  bounds check
  HW_assert           = 4Bh;   --  hardware ASSERT
  estk_overflow       = 4Ch;   --  E-stack over/underflow
  abort               = 4Dh;   --  ABORT
  heap_abort          = 4Eh;   --  no memory in heap
  inv_parm            = 4Fh;   --  illegal parameter
```

```
  break              = 50h;   --  BREAK (^C)
  unimp_proc         = 51h;   --  unimplemented procedure
  obsolete_proc      = 52h;   --  obsolete procedure
  no_resource        = 53h;   --  resource exhausted
  undef_exception    = 54h;   --  unexpected exception

(*************************************************************

-------------------  П Р И М Е Ч А Н И Я  --------------------
                    ----------------------
```

Модуль содержит определение кодов ошибок OS Excelsior. Диапазон 01h..7Fh содержит номера аппаратных и программных прерываний. Диапазон 80h..0FFh содержит номера ошибок ОС.

Ошибки ввода/вывода нижнего уровня имеют выставленный 31-й бит в коде ошибки. Биты [8..30] используются для передачи информации о причине ошибки.

ЗАМЕЧАНИЕ:

Не следует порождать ошибки код которых содержит выставленный в 1 31-ой бит, т.к. они могут в дальнейшем быть проинтерпритированны как ошибки в/в.

```
-----------------------------  ОК  -----------------------------
                              ------
                    --------------------------
                    -- И все хорошо, а-а,   --
                    -- И все хорошо, а-а... --
                    --                 ДДТ  --
                    --------------------------

*************************************************************)

END defErrors.
```

```
DEFINITION MODULE defKeyboard; (* Ned 23-Aug-89. (c) KRONOS *)

(* Модуль набор кодировок для функциональных клавиш клавиатуры *)

CONST
  lf    = 012c; (* ^J *)
  cr    = 015c; (* ^M *)
  break = 003c; (* ^C break proc(0) *)
  exit  = 005c; (* ^E *)
  back  = 010c; (* ^H *)
  tab   = 011c; (* ^I *)
  vt    = 013c; (* ^K *)
  rep   = 032c; (* ^Z *)
  nak   = 025c; (* ^U break proc(1) *)
  can   = 030c; (* ^X break proc(2) *)
  nl    = 036c; (* ^^ *)

  press = 001c; (* ^A *)
  relea = 002c; (* ^B *)

  up    = 200c;  dw    = 201c;  right = 202c;  left  = 203c;
  pgup  = 204c;  pgdw  = 205c;  home  = 206c;  end   = 207c;
  del   = 210c;  ins   = 211c;  bcktab= 212c;  newln = 213c;
  alt   = 214c;  ctrl  = 215c;  shft  = 216c;  alkb  = 217c;

  f1    = 220c;  f2    = 221c;  f3    = 222c;  f4    = 223c;
  f5    = 224c;  f6    = 225c;  f7    = 226c;  f8    = 227c;
  f9    = 230c;  f10   = 231c;  f11   = 232c;  f12   = 233c;
  f13   = 234c;  f14   = 235c;  f15   = 236c;  center= 237c;


(* alt,ctrl,shift -- порождается только тогда, когда
   клавиатура допускает scancode (или аналогичный) режим
   Для клавиатур не имеющих PRESS/RELEASE режима
   эти три служебных символа могут эмулироваться с помощью
   трех обычных кнопок
*)

TYPE
  BREAK = PROCEDURE (INTEGER);

  STATE = RECORD
            type : INTEGER;
            fkeys: INTEGER;
            ubrk : BREAK;
            togs : BITSET;
            freq : INTEGER;
            dur  : INTEGER;
          END;

CONST (* togs *)
  breakon = {0};
```

```
  foreign = {1};
  caps    = {2};
  shift   = {3};
  raw     = {4};
  scan    = {5};
  autorep = {6};

CONST        --- изменение состояния драйвера ---


  _info    = 00h;      -- (POINTER TO STATE);
  _reset   = 01h;      -- (POINTER TO STATE);
  _restore = 02h;      -- (POINTER TO STATE);

  _ubreak  = 03h;      -- (proc: BREAK)
  _break   = 04h;      -- (ON_OFF: INTEGER)
                       -- TRUE : разрешает стандартную BREAK
реакцию
                       -- FALSE: запрещает реакцию на BREAK

  _foreign = 05h;      -- (ON_OFF: BOOLEAN)

  _bell_fd = 06h;      -- (frequncy,duration: INTEGER)
  _bell    = 07h;      -- generate r.len bells

  _raw     = 08h;      -- (ON_OFF: BOOLEAN) - raw input mode
  _shift   = 09h;      -- (ON_OFF: BOOLEAN)
  _caps    = 0Ah;      -- (ON_OFF: BOOLEAN)
  _autorep = 0Bh;      -- (ON_OFF: BOOLEAN)
  _set_ct  = 0Ch;      -- (POINTER TO ARRAY CHAR OF CHAR) set
coder table
  _get_ct  = 0Dh;      -- (POINTER TO ARRAY CHAR OF CHAR) get
coder table

END defKeyboard.
```

```
DEFINITION MODULE defPrinter; (* Leg 18-Jan-90. (c) KRONOS *)

TYPE
  BARS  = ARRAY [0..2],[0..2] OF CHAR;
  WIDTH = POINTER TO ARRAY CHAR OF INTEGER;
  STATE = RECORD
            type     : INTEGER;
            fonts    : INTEGER;
            font     : INTEGER;
            hchar    : INTEGER;  (* ground box *)
            wchar    : WIDTH  ;  (* in  pixels *)
            something: INTEGER;
            underline: INTEGER;
            Wx2      : INTEGER;
            Hx2      : INTEGER;
            reverse  : INTEGER;
            raw      : INTEGER;
            awp      : INTEGER;
            idle     : INTEGER;
            grounds  : INTEGER;
            ground   : INTEGER;
            hbar     : CHAR   ;
            vbar     : CHAR   ;
            bars     : BARS   ;
            densities: INTEGER;  -- dot per inch
            density  : INTEGER;  -- number of density
          END;

CONST  (* M.S.B. of REQUEST.op when L.S.B. of it = CONTROL *)

  _info      = 10h;  -- (POINTER TO STATE)
  _reset     = 11h;
  _restore   = 12h;  -- (STATE)
  _something = 20h;  -- (on_off: INTEGER);
  _underline = 21h;  -- (on_off: INTEGER);
  _Wx2       = 22h;  -- (on_off: INTEGER);
  _Hx2       = 23h;  -- (on_off: INTEGER);
  _autowrap  = 24h;  -- (on_off: INTEGER);
  _raw       = 25h;  -- (on_off: INTEGER);

  _reverse   = 36h;  -- (on_off: INTEGER);
  _ground    = 37h;  -- (no    : INTEGER);
  _density   = 38h;  -- (no    : INTEGER);
  _font      = 39h;  -- (no    : INTEGER);

                     --- opreations ---

  _back      = 50h;  -- (dots: INTEGER)
  _fwd       = 51h;  -- (dots: INTEGER)
  _left      = 52h;  -- (dots: INTEGER)
  _right     = 53h;  -- (dots: INTEGER)
  _bflf      = 54h;  -- (count: INTEGER)
```

```
  _bhlf      = 55h;  -- (count: INTEGER)
  _fhlf      = 56h;  -- (count: INTEGER)
  _fflf      = 57h;  -- (count: INTEGER)
  _write_ln  = 58h;
  _eject     = 59h;
  _repeat    = 5Ah;  -- (ch: CHAR; count: INTEGER);
  _paint     = 5Bh;  -- (buf: ADDRESS; sz,w,h,dx,dy: INTEGER);
  _load_font = 5Ch;  -- (no,from,to: INTEGER; font: ADDRESS; sz:
INTEGER)
  _set_attr  = 5Dh;
  _get_attr  = 5Eh;

END defPrinter.
```

```
DEFINITION MODULE defRequest; (* Ned 19-Sep-89. (c) KRONOS *)

(* Определяет стандарт запроса к драйверу. *)

IMPORT  sys: SYSTEM;

TYPE
  REQUEST = RECORD (* 28 bytes *)
              op : INTEGER;     -- код операции
              drn: INTEGER;     -- номер под-устройства
              ofs: INTEGER;     -- адрес на устройстве
              buf: sys.ADDRESS; -- адрес буфера в памяти
              pos: INTEGER;     -- смещение в буфере
              len: INTEGER;     -- длина
              res: INTEGER;     -- результат
            END;

CONST  -- operations (* L.S.B. (Low Significant Byte *)

  NOP        =  0;

  LOCK       =  1; (* LOCK   not used yet *)
  UNLOCK     =  2; (* UNLOCK not used yet *)

  READ       =  3; (* чтение данных с устройства           *)
  WRITE      =  4; (* запись данных на устройство          *)

  BUSYREAD   =  5; (* чтение данных без ожидания           *)
  WAIT       =  6; (* ожидание данных                      *)
  READY      =  7; (* запрос о числе байтов в буфере ввода *)
  CONTROL    =  8; (* управляющая операция для клавиатур,  *)
                   (* экранов и т.д. Старшие байты содержат *)
                   (* код операции                         *)

  GET_SPEC   =  9; (* запрос спецификации устройства       *)
  SET_SPEC   = 10; (* определение спецификации устройства  *)
  POWER_OFF  = 11; (* завершение работы                    *)
  FORMAT     = 12; (* форматирование устройства            *)
  SEEK       = 13; (* позиционирование                     *)
  MOUNT      = 14; (* монтирование носителя                *)
  UNMOUNT    = 15; (* размонтирование носителя             *)


(*************************************************************

------------------------  ПРИМЕЧАНИЕ  ------------------------
                          --------------
```

Следущее описание определяет параметры каждой операции и поля через которые эти параметры передаются. Все операции возвращают результат операции в поле res. Для всех операций поле drn содержит номер (под-)устройства.

```
READ, WRITE (disk):
-------------------
        IN
            drn - номер (под-)устройства
            buf - буфер в/в
            ofs - номер сектора на диске
            len - число секторов
        OUT
            res - результат операции

READ, WRITE, BUSYREAD (serial):
-------------------------------
        IN
            drn - номер (под-)устройства
            buf - буфер в/в
            pos - смещение в буфере
            len - число байтов
        OUT
            len - число непрочитанных байтов
            res - результат операции

WAIT (serial):
--------------
        IN
            drn - номер (под-)устройства
        OUT
            res - результат операции


READY (input serial):
---------------------
        IN
            drn - номер (под-)устройства
        OUT
            len - число байтов в буфере ввода
            res - результат операции

CONTROL:
--------
        IN
            op  - старшие байты содержат код
                  управляющей операциии
            drn - номер (под-)устройства
        OUT
            res - результат операции

GET_SPEC
SET_SPEC
POWER_OFF

FORMAT
SEEK
```

```
MOUNT
UNMOUNT

**************************************************************)

END defRequest.
```

```
DEFINITION MODULE defScreen; (* nick 05-May-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE
  BLOCK  = RECORD x,y,w,h: INTEGER END;

  REGION = POINTER TO
             RECORD
               magic: INTEGER;
               mode : INTEGER;
               mask : BITSET;
               fore : BITSET;
               back : BITSET;
               clip : BLOCK;
               kind : INTEGER;
               ext  : SYSTEM.ADDRESS
             END;

  PALETTE = DYNARR OF RECORD r,g,b: INTEGER END;

  SCREEN = POINTER TO
             RECORD
               magic: INTEGER;
               type : INTEGER;
               bpp  : INTEGER;
               rgn  : REGION;
               ldcx : INTEGER;
               ldcy : INTEGER;
               dx   : INTEGER;
               dy   : INTEGER;
               xpix : INTEGER;
               ypix : INTEGER;
               W,H  : INTEGER;
               pal  : PALETTE;
               RGB  : INTEGER
             END;

  FONT = POINTER TO FNTD;
  FNTD = RECORD
           W,H  : INTEGER;
           BASE : INTEGER;
           magic: INTEGER;
           state: BITSET;
           bline: INTEGER;
           uline: INTEGER;
           descW: POINTER TO ARRAY CHAR OF CHAR;
           descH: POINTER TO ARRAY CHAR OF ARRAY [0..2] OF CHAR;
           bases: POINTER TO ARRAY CHAR OF SYSTEM.ADDRESS;
           size : INTEGER;
           rfe12: SYSTEM.WORD;
           rfe13: SYSTEM.WORD;
```

```
            rfe14: SYSTEM.WORD;
            rfe15: SYSTEM.WORD;
            rfe16: SYSTEM.WORD;
          END;

CONST
  chH = 0;  constW = { 0};  ital = {4};
  chY = 1;  constH = { 1};  spec = {5};
             dchar  = { 3};
             noimg  = {31};

CONST (* kinds *)
   bitmap = 444D4224h; (* $BMD *)
   pelmap = 44504124h; (* $APD *)
   other  = 48544F24h; (* $OTH *)

CONST
   rep = 0;  or = 1;  xor = 2;  bic = 3;

   reverse = 4;    normal = 0;

CONST
   CONTROL  = 01h;
  _init     = 02h;
  _set_ldcx = 03h;
  _set_ldcy = 04h;
  _set_rgb  = 05h;
  _get_rgb  = 06h;

END defScreen.
```

```
DEFINITION MODULE defTasks; (* Ned 20-Mar-90. (c) KRONOS *)

CONST  -- состояния задачи

  new     = 1;
  loaded  = 2;
  loaderr = 3;
  ready   = 4;
  running = 5;
  stopped = 6;
  killed  = 7;

CONST  -- номера сигналов задачи

  start   = 0;
  stop    = 1;
  kill    = 2;
  ipr     = 3;
  suspend = 4;
  resume  = 5;

END defTasks.
```

```
DEFINITION MODULE defTerminal; (* Ned 23-Aug-89. (c) KRONOS *)

TYPE
  BARS  = ARRAY [0..2],[0..2] OF CHAR;
  STATE = RECORD (* 72 bytes *)
            type     : INTEGER;          hbar : CHAR;
            lines    : INTEGER;          vbar : CHAR;
            columns  : INTEGER;          bars : BARS;
            min_color: INTEGER;          back : INTEGER;
            max_color: INTEGER;          color: INTEGER;
            fonts    : INTEGER;          font : INTEGER;
            screens  : INTEGER;          scr  : INTEGER;
            avail    : BITSET;           togs : BITSET;
          END;

CONST (* togs *)
  cursor   ={0};        awp   ={5};
  something={1};        raw   ={6};
  reverse  ={2};        smooth={7};
  blinking ={3};        cinter={8};
  underline={4};


CONST  (* M.S.B. of REQUEST.op when L.S.B. of it = CONTROL *)

             --- изменение состояния драйвера ---

  _info         = 00h;      -- (POINTER TO STATE)
  _reset        = 01h;      -- (POINTER TO STATE)
  _restore      = 02h;      -- (POINTER TO STATE)

  _raw          = 08h;      -- (ON_OFF: BOOLEAN) - raw output
mode
  _autowrap     = 09h;      -- (ON_OFF: BOOLEAN)
  _smooth_scroll= 0Ah;      -- (ON_OFF: BOOLEAN)
  _screen       = 0Bh;      -- (no: INTEGER)

                --- операции над экраном ---

  _up           = 11h;      -- count
  _down         = 12h;      -- count
  _left         = 13h;      -- count
  _right        = 14h;      -- count
  _home         = 15h;      --
  _bottom       = 16h;      --
  _erase        = 17h;      -- [0..2] 0 - к концу, 1 - к началу,
2 - все
  _erase_line   = 18h;      -- [0..2] 0 - к концу, 1 - к началу,
2 - все
  _erase_chars  = 19h;      -- count
  _repeat       = 1Ah;      -- char, count
  _set_pos      = 1Bh;      -- (line,col: INTEGER)
```

```
  _roll_up      = 1Ch;      -- count
  _roll_down    = 1Dh;      -- count
  _scroll_up    = 1Eh;      -- count
  _scroll_down  = 1Fh;      -- count
  _ins_char     = 20h;      -- count
  _del_char     = 21h;      -- count
  _ins_line     = 22h;      -- count
  _del_line     = 23h;      -- count
  _cursor       = 24h;      -- (on_off: BOOLEAN);
  _reverse      = 25h;      -- (on_off: BOOLEAN);
  _underline    = 26h;      -- (on_off: BOOLEAN);
  _blinking     = 27h;      -- (on_off: INTEGER);
  _something    = 28h;      -- (on_off: INTEGER);
  _cinter       = 29h;      -- (on_off: INTEGER);
  _color        = 2Ah;      -- (no    : INTEGER);
  _background   = 2Bh;      -- (no    : INTEGER);
                            -- <0   - пониженная интенсивность
                            -- =0   - нормальная интенсивность
                            -- >0   - повышенная интенсивность
  _font         = 2Ch;      -- no
  _load_font    = 2Dh;      -- no,from,to,ADR(font),SIZE(font)

  _set_attr     = 30h;      -- (no: INTEGER;     val: INTEGER);
  _get_attr     = 31h;      -- (no: INTEGER; VAR val: INTEGER);

END defTerminal.
```

```
DEFINITION MODULE Exceptions; (* Ned 20-Apr-90. (c) KRONOS *)

PROCEDURE exceptions(VAR n: INTEGER): BOOLEAN;
PROCEDURE exception(n: INTEGER): BOOLEAN;

PROCEDURE raise(n: INTEGER);

PROCEDURE traps(VAR n: INTEGER): BOOLEAN;
PROCEDURE trap(n: INTEGER): BOOLEAN;

(************************************************************
```

Модуль работы с исключительными ситуациями.

```
PROCEDURE exceptions(VAR n: INTEGER): BOOLEAN;
PROCEDURE traps     (VAR n: INTEGER): BOOLEAN;
---------------------------------------------
```

Процедуры определяют начало ловушки исключительных ситуаций. Ловушка будет снята в момент выхода из процедуры, в которой она была поставлена, или после перехвата исключительной ситуации. Возвращает FALSE при постановке ловушки, TRUE - при возникновении исключительной ситуации (при этом "n" равно индиканту ситуации).

Процедура exceptions ставит ловушку на пользовательские ситуации, возбуждаемые процедурой raise. Процедура traps ставит ловушку на ситуации, возбуждаемые аппаратурой или командой trap.

```
PROCEDURE exception(n: INTEGER): BOOLEAN;
PROCEDURE trap     (n: INTEGER): BOOLEAN;
---------------------------------------------
```

Процедуры exception и trap ставят ловушку на конкретную исключительную ситуацию и не реагируют на дугие ситуации. Остальное аналогично процедурам exceptions и traps.

```
PROCEDURE raise(n: INTEGER);
----------------------------
```

Возбуждает исключительную ситуацию. Заметим, что именно значение n вернет процедура exception.

ОШИБКИ:

TRAP(54h) -- нет ловушки.

Способ использования:

```
PROCEDURE p;
  VAR e: INTEGER;
```

```
BEGIN
  IF exceptions(e) THEN
(*1*)
    print('exception(%d)\n',e); RETURN
  END;
-- procedure body
END p;
```

Если при выполнении тела процедуры будет возбуждена исключительная ситуация, то выполнение тела будет прекращено и управление попадет в точку (*1*).

```
**********************************************************)

END Exceptions.
```

```
DEFINITION MODULE Fonts;  (* nick 07-May-90. (c) KRONOS *)

IMPORT  SYSTEM;
IMPORT  def: defScreen;

CONST
  chH = def.chH; constW = def.constW; ital = def.ital; noimg =
def.noimg;
  chY = def.chY; constH = def.constH; spec = def.spec; dchar =
def.dchar;

TYPE
  FONT = def.FONT;

VAR main: FONT;
    done: BOOLEAN;
   error: INTEGER;

PROCEDURE new    (VAR fnt: FONT; w,h: INTEGER; state: BITSET);
PROCEDURE dispose(VAR fnt: FONT);

PROCEDURE read (VAR fnt: FONT; file_name: ARRAY OF CHAR);
PROCEDURE write(    fnt: FONT; file_name: ARRAY OF CHAR);

END Fonts.
```

```
DEFINITION MODULE Formats; (* Sem  07-Aug-88. (c) KRONOS *)
                           (* Leo  03-Nov-88. (c) KRONOS *)
                           (* Hady 17-Aug-89. (c) KRONOS *)

(* Универсальный форматный вывод. *)

IMPORT  SYSTEM;

TYPE write_proc = PROCEDURE (
          SYSTEM.WORD,   -- определяет направление вывода
          ARRAY OF CHAR, -- выводимая строка
          INTEGER,       -- позиция в выводимой строке
          INTEGER        -- длина значимой части строки в байтах
                          );

PROCEDURE format(parameter: SYSTEM.WORD;   -- для write_proc
                 write    : write_proc;
                 format   : ARRAY OF CHAR;
             SEQ arguments: SYSTEM.WORD);
(* Выводит процедурой write аргументы в соответствии с форматом
*)

(*************************************************************

  format     ::= { текст | "%" формат | "%%" | "\n"
                   | "\r" | "\l" | "\\" } 0c.
  текст      ::= "любая последовательность символов ASCII-8,
                  кроме '%','\' и 0c ".
                  Если требуется символ '%', предварите его '%':
                  '%%' преобразуется в просто '%'.
                  Если требуется символ '\', предварите его '\':
                  '\\' преобразуется в просто '\'.
  формат     ::= { модификатор } база.
  база       ::= ("d"|"h"|"b"|"s"|"{}"|"i").
  ширина     ::= цифра { цифра } | "*".
  точность   ::= "." цифра { цифра } | ".*" .
  модификатор::= ( space | "$" | "-" | "+" | "#" | "|" | ширина
                   | точность | нач.позиция ) .
  d   (Decimal)       - десятичное;
  h,H (Hexidecimal)   - шестнадцатеричное
                        (h,x -- "A".."F" прописные)
  x,X (Hexidecimal)   - эквивалентно 'h'
                        (H,X -- "a".."f"  строчные)
  b,B (Octal)         - восьмеричное;
  o,O (Octal)         - эквивалентно 'b'
  s,S (String)        - строка;
  c,C (Char  )        - одиночный символ
  {}  (set)           - битсет;
  i,I (bIn)           - двоичное;
  f,F (Float)         - вещественное число в формате:
                          [+|-]dddddddd.ddddd
                            |__  n1  __|_ t _|
```

Число цифр до запятой n1 - минимально необходимое для представления числа. Число цифр после запятой t - задается точностью (по умолчанию 6).

e,E (Exponent) - вещественное число в формате:
[+|-]d.dddddE(+|-)dd или
[+|-]D.DDDDDDe(+|-)DD
Число цифр до запятой 1
Число цифр после запятой t - задается точностью (по умолчанию 6)
(формат 'e' -- 'E' в результате)
(формат 'E' -- 'e' в результате)

g,G (General) - вещественное число в формате:
если FLOAT(TRUNC(число))=число,
то в формате ddddd;
иначе в формате 'f' или 'e'
- какой короче.

\# - показывает число с указанием основания (например: image(s,"%$10#h",12abcdefH); эквивалентно image(s,"012ABCDEFh"));

\- - значение пишется слева в поле установленной ширины;

| - значение пишется в центре поля установленной ширины;

\+ - показывает число со знаком, независимо от знака числа;

$ (zero) - дополнить до нужного количества разрядов ведущими нулями;

space - выставляется знак, если число отрицательное, иначе пробел;

точность - задает число значащих цифр после(!) запятой в форматах 'f' , 'e', 'g', число символов строки в формате 's' или число повторений символа в формате "c";

ширина - задает общую(!) ширину поля (с учетом возможных символов основания, знака и т.п.); если указанной ширины недостаточно для представления выводимого значения, происходит автоматическое расширение поля до минимально необходимого; если вместо спецификации ширины и/или точности указаны '*', то значения ширины и точности берутся из соответствующих аргументов. Внимание! Значение точности и ширины должно быть из диапазона [0..255]; в противном случае оно принимается равным значению по умолчанию.

ПРИМЕЧАНИЯ:

У базы "s" может быть третий числовой модификатор, обозначающий номер позиции в строке-аргументе, с которой

эта строка читается. Этот модификатор отделяется от предыдущих символом "." и может быть как прямо указан в строке формата, так и передаваться как аргумент; в последнем случае вместо числа в модификаторе следует использовать символ "*". Первые числовые модификаторы могут быть опущены, но разделитель "." должен оставаться. Пример: %..10s означает вывод с 10 байта.

Модификаторы '$' и '-', естественно, не совместимы.
Модификаторы '+' и space, естественно, не совместимы.
Модификатор точность может быть использован только с базами 's','f','e','g'.
Модификаторы '+' и space могут быть использованы только с базами 'i','f','e','g'.
Модификаторы '$', '+' и space НЕ могут быть использованы с базами 's','{}','c'.
\n обозначает CR+LF
\r обозначает CR
\l обозначает LF
\e обозначает ESC
\g обозначает BEL
\\ обозначает \
ВНИМАНИЕ! \n, \r,... обрабатываются ТОЛЬКО в формате, но не в строках-аргументах!!!

**********)

END Formats.

```
DEFINITION MODULE fsWalk; (* Leo 13-Oct-89. (c) KRONOS *)

(* Прогуливается по файловому дереву. *)

IMPORT  BIO;

TYPE TREE;

VAL null: TREE;
     pos: INTEGER; (* position where error 'bad_name' occured *)
    done: BOOLEAN;
   error: INTEGER;

(* all procedures if it is'nt specialy declared      *)
(* set 'done=TRUE' if operation done OK, else        *)
(* they set 'done=FALSE' and 'error=cause of error' *)

PROCEDURE walk(VAR fs: TREE; patt: ARRAY OF CHAR; only_files:
BOOLEAN);
(* start of tree search *)

PROCEDURE next_dir(fs: TREE): BOOLEAN;
(* returns FALSE when tree iteration finished *)

PROCEDURE dir(fs: TREE): BIO.FILE; (* may return BIO.NULL *)
(* returns directory in tree after 'next_dir' *)

PROCEDURE dpath(fs: TREE; VAR path: ARRAY OF CHAR); (* not side
effect to *)
PROCEDURE dname(fs: TREE; VAR name: ARRAY OF CHAR); (* "done" and
"error" *)

PROCEDURE next_entry(fs: TREE;
               VAR name: ARRAY OF CHAR; VAR mode: BITSET):
BOOLEAN;
(* returns FALSE when next entry not found in the directory *)

PROCEDURE substitute(fs: TREE; model: ARRAY OF CHAR;
                           VAR dest: ARRAY OF CHAR);

PROCEDURE restart_dir(fs: TREE);
(* restart search at directory *)

PROCEDURE fpath(fs: TREE; VAR path: ARRAY OF CHAR); (* not side
effect to *)
PROCEDURE fname(fs: TREE; VAR name: ARRAY OF CHAR); (*
*)
PROCEDURE fmode(fs: TREE; VAR mode: BITSET);        (* "done" and
"error" *)

PROCEDURE restart_walk(fs: TREE);
(* global restart of tree *)
```

```
PROCEDURE dispose(VAR fs: TREE);
(* terminate tree search and dispose all memory occupied by the
TREE *)

(*************************************************************

PROCEDURE walk(VAR fs: TREE; pattern: ARRAY OF CHAR; only_files:
BOOLEAN);
--------------

 pattern      = root_prefix path_pattern node_pattern .
 root_prefix  = [ "/" ] .
 path_pattern = { RE "/" } tree_postfix .
 tree_postfix = [ "/" ] .
 RE           = 'any regular expresion without "/" '

 if syntax error in pattern occured then
 'pos' pointed at error position

 file_pattern determine pattern for
 all files at tree nodes iterated by 'next_entry'.
 if 'fileonly'=TRUE then names of subdirectories
 not generatedv by 'next_entry' call.

 ".." NEVER generated by 'next_entry' call.

 ".." may be used as a 'pathpattern' BUT without $<digit> and

 parenthis:
    /util/*/(..)$1/*/*.@  are legal but has empty result
    /util/*/../*/*.@      are legal and has good  result

 Before (and only before!) file_pattern "//" determines
 sub-tree iteration.

 Examples:

   .//*.?          all files *.? at current and subdirectorities
   //my_best.file  all "my_best.file" files in file tree

   /util/*/(?*.[md@])$1

 ILLEGAL:

   /usr//tmp/../..//leo/*
                   theoretical possible but very difficult in
                   implementation, so, not realized yet.
                   (may be later...)

*************************************************************)

END fsWalk.
```

```
DEFINITION MODULE Heap; (* Andy & Leo 20-Dec-89. (c) KRONOS *)

IMPORT  SYSTEM;

VAL done: BOOLEAN;
   error: INTEGER;
   limit: INTEGER;
  credit: INTEGER;

(* Процедуры, возбуждающие TRAP 4Eh при нехватке памяти: *)

PROCEDURE   ALLOCATE(VAR a: SYSTEM.ADDRESS;    size: INTEGER);
PROCEDURE DEALLOCATE(VAR a: SYSTEM.ADDRESS;    size: INTEGER);
PROCEDURE REALLOCATE(VAR a: SYSTEM.ADDRESS; VAR high: INTEGER;
                                  len,el_byte_size: INTEGER);

(* Процедуры, при нехватке памяти возвращающие NIL: *)

PROCEDURE   allocate(VAR a: SYSTEM.ADDRESS;    size: INTEGER);
PROCEDURE deallocate(VAR a: SYSTEM.ADDRESS;    size: INTEGER);
PROCEDURE reallocate(VAR a: SYSTEM.ADDRESS; VAR high: INTEGER;
                                  len,el_byte_size: INTEGER);

PROCEDURE set_credit(size: INTEGER); (* words *)

PROCEDURE set_limit(total: INTEGER); (* words *)

TYPE
  GARBAGE_COLLECTOR = PROCEDURE (): BOOLEAN;
  (* TRUE, если удалось что-то вернуть, иначе FALSE *)

PROCEDURE install_gc(gc: GARBAGE_COLLECTOR);

PROCEDURE  remove_gc(gc: GARBAGE_COLLECTOR);

PROCEDURE statistics(VAR  from_os: INTEGER;
                     VAR     free: INTEGER;
                     VAR user_sum: INTEGER);

(************************************************************
```

Библиотека поставляет процедуры работы с динамически распределяемой областью памяти - "кучей".

Длина запрашиваемого и возвращаемого куска всегда неотрицательна.

Запрос на 0 слов памяти считается корректным - возвращается NIL (в том числе в процедурах, возбуждающих прерывания по нехватке памяти). Можно вернуть куче кусок памяти с адресом начала NIL и произвольной длиной.

```
PROCEDURE  allocate(VAR a: SYSTEM.ADDRESS;   size: INTEGER);
```

Пытается выделить память размером size слов. Если в динамической области задачи нет куска такого размера, то запрашивает кусок размером max(size, credit) у ОС. Если и в этом случае нет памяти, то a=NIL.

```
PROCEDURE deallocate(VAR a: SYSTEM.ADDRESS;   size: INTEGER);
```

Возвращает кусок памяти, начинающийся с адреса a размером size слов. Если при этом освободился целиком кусок памяти, взятой у ОС, то он будет возвращен.

```
PROCEDURE reallocate(VAR a: SYSTEM.ADDRESS; VAR high: INTEGER;
                                    len,el_byte_size: INTEGER);
```

Пытается переразместить массив, начинающийся по адресу "a", под "len" элементов, длиной "el_byte_size" БАЙТОВ каждый.

В случае успеха копирует содержимое массива и перевычисляет "a" и "high=len-1". В случае расширения массива значения новых элементов не определены.

В случае неудачи "a" и "high" не изменяются.

```
PROCEDURE  ALLOCATE(VAR a: SYSTEM.ADDRESS;   size: INTEGER);
PROCEDURE REALLOCATE(VAR a: SYSTEM.ADDRESS; VAR high: INTEGER;
                                    len,el_byte_size: INTEGER);
```

Отличаются от соответствующих процедур выше тем, что при нехватке памяти возбуждается прерывание 4Eh.

```
PROCEDURE DEALLOCATE(VAR a: SYSTEM.ADDRESS;   size: INTEGER);
```

Ничем не отличается от процедуры deallocate.

```
PROCEDURE statistics(VAR  from_os: INTEGER;
                     VAR      free: INTEGER;
                     VAR user_sum: INTEGER);
```

Возвращается общий размер памяти, взятой у OS (from_os), общий размер памяти в списке свободных сегментов (free) и сумма запросов пользователя на память (user_sum) (при этом from_os >= free + user_sum).

```
PROCEDURE set_limit(total: INTEGER);
```

Установление верхнего предела на общий размер памяти,

который может быть запрошен у OS. При total<0 верхний предел отменяется.

```
PROCEDURE set_credit(size: INTEGER);
```

Установление размера запроса памяти у OS.

Начальное значение отлично от 0 и устанавливается при инициализации данного модуля. Библиотеки, использующие set_credit(0) в своей инициализации, обязаны вернуть значение credit в начальное.

```
PROCEDURE install_gc(gc: GARBAGE_COLLECTOR);
```

Процедура "gc" будет вызвана при нехватке памяти, и если она вернет TRUE, попытка запросить память будет повторена. Если после вторичной попытки памяти все-таки не оказалось, "gc" повторно НЕ вызывается.

```
PROCEDURE  remove_gc(gc: GARBAGE_COLLECTOR);
```

Удаляет "gc" из множества зарегистрированных "мусорщиков".

****

Замечания по реализации Heap 1.1

Планировщик не оптимизирует работу с маленькими кусками памяти.

Минимальный полезный размер сегмента памяти - 2 слова.

Накладные расходы на каждый сегмент памяти - 2 слова.

Работа ведется на динамической области памяти задачи.

**********)

```
END Heap.
```

```
DEFINITION MODULE Keyboard; (* Ned 21-Sep-89. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE BREAK = PROCEDURE (INTEGER);

VAL
  done : BOOLEAN;       error: INTEGER;
  type : INTEGER;
  fkeys: INTEGER;       caps : INTEGER;
  ubrk : BREAK;         shift: INTEGER;      autorep: INTEGER;
  freq : INTEGER;       raw  : INTEGER;      breakon: INTEGER;
  dur  : INTEGER;       scan : INTEGER;      foreign: INTEGER;

PROCEDURE read(VAR ch: CHAR);

PROCEDURE busy_read(VAR ch: CHAR);

PROCEDURE pressed(): INTEGER;

PROCEDURE wait(timeout: INTEGER);
(* until pressed or timeout (milisecs) *)

PROCEDURE bell(n: INTEGER);

PROCEDURE set_bell(freq,dura: INTEGER);

PROCEDURE set_raw(ON_OFF: INTEGER);

PROCEDURE user_break(new: BREAK);

PROCEDURE set_break  (ON_OFF: INTEGER);
PROCEDURE set_shift  (ON_OFF: INTEGER);
PROCEDURE set_caps   (ON_OFF: INTEGER);
PROCEDURE set_foreign(ON_OFF: INTEGER);
PROCEDURE set_autorep(ON_OFF: INTEGER);

PROCEDURE action(no: INTEGER; SEQ args: SYSTEM.WORD);

PROCEDURE attach(dev_name: ARRAY OF CHAR);

PROCEDURE reset; (* to initial state *)

CONST
  lf    = 012c; (* ^J *)
  cr    = 015c; (* ^M *)
  break = 003c; (* ^C break proc(0) *)
  exit  = 005c; (* ^E *)
  back  = 010c; (* ^H *)
  tab   = 011c; (* ^I *)
  vt    = 013c; (* ^K *)
  rep   = 032c; (* ^Z *)
```

```
  nak   = 025c; (* ^U break proc(1) *)
  can   = 030c; (* ^X break proc(2) *)
  nl    = 036c; (* ^^ *)

  press = 001c; (* ^A *)
  relea = 002c; (* ^B *)

  up    = 200c;  dw    = 201c;  right = 202c;  left  = 203c;
  pgup  = 204c;  pgdw  = 205c;  home  = 206c;  end   = 207c;
  del   = 210c;  ins   = 211c;  bcktab= 212c;  newln = 213c;
  alt   = 214c;  ctrl  = 215c;  shft  = 216c;  alkb  = 217c;

  f1    = 220c;  f2    = 221c;  f3    = 222c;  f4    = 223c;
  f5    = 224c;  f6    = 225c;  f7    = 226c;  f8    = 227c;
  f9    = 230c;  f10   = 231c;  f11   = 232c;  f12   = 233c;
  f13   = 234c;  f14   = 235c;  f15   = 236c;  center= 237c;



(*************************************************************

type -- уникальный идентификатор клавиатуры
        1-Фрящик
        2-Labtam XT
        4-Labtam HE
        7-Elorg

scan = TRUE,
       если драйвер клавиатуры выдает коды на нажатие и отжатие
ALT,
       CTRL, SHIFT.
       Более детальные сканкоды можно получить в прозрачном
режиме.

f_range -- число ключей f1,f2,...; минимум - 10, максимум - 15.
```

Частоты звукового сигнала в звукогенераторах для первой октавы:

C	7643
C#	7214
D	6809
D#	6427
E	6066
F	5726
F#	5404
G	5101
G#	4815
A	4545
A#	4289
B	4049

```
            C1        3822

       Удваивается или делится пополам для остальных октав.

        Bell duration in microseconds (for some
        implementation rounded upto multiple of 20)


user_break
=========

TYPE BREAK (n):

   n=0 -- break active      (etx 03c ^C)
   n=1 -- break sequence    (nak 25c ^U) (exa.: shell command
file)
   n=2 -- break underground (can 30c ^X)

   new: new break reaction

   host reaction at any keyboard stored
   and restored after task finish or when keyboard reattached.

attach
======

  action(_init) executed:
          type,scan,fkeys -- changed

  if next keyboard attached and 'set_break' was executed for
  previos attached keyboard, 'old' break reaction proc for
previos
  keyboard restored. So you can't redefine break action at
  more then one keyboard at a time!
  State of device (raw mode, break on/off e.t.c) saved
  at init or atach time and restored before halt or reattach

wait
====

  when timeout occured 'done=FALSE' 'error=time_out'

**********************************************************)

END Keyboard.
```

```
DEFINITION MODULE Lexicon; (* Leo 18-Jan-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE LEX;

VAL null: LEX;
    done: BOOLEAN;
   error: INTEGER;
  sysmsg: LEX;

PROCEDURE open(VAR lex: LEX; lexicon_device_name: ARRAY OF CHAR);

PROCEDURE get(lex: LEX; code: INTEGER; VAR data: STRING);

PROCEDURE dispose(VAR data: STRING);

PROCEDURE sprint(VAR str: ARRAY OF CHAR;
                      lex: LEX;
                     code: INTEGER;
                   format: ARRAY OF CHAR;
                 SEQ args: SYSTEM.WORD);

PROCEDURE close(VAR lex: LEX);

PROCEDURE perror(VAR str: ARRAY OF CHAR;
                     code: INTEGER;
                   format: ARRAY OF CHAR;
                 SEQ args: SYSTEM.WORD);

(************************************************************

   Lexicon: Multiple lexicons interface

   ATTENTION! Library is ussing memory allocated in Heap.

     Внимание!  Библиотека  использует  память,  отведенную
средствами библиотеки Heap.

PROCEDURE open(lex: LEX; special_device_name: ARRAY OF CHAR);
------------------------------------------------------------

PROCEDURE close(lex: LEX);
--------------------------

    'lex' - lexicon file (driver for special device with
  appropriate name must exist and be running in
  the system).
     At instalation time text file read at 'lex' device
  and this library provides access to readen text.

     "lex"   -   файл   лексикона   (драйвер   для   специального
```

устройства с соответствующим именем должен существовать и быть запущенным в системе).
Во время инсталяции
??
и эта библиотека обеспечивает доступ к прочитанному тексту.

PROCEDURE get(lex: LEX; code: INTEGER; VAR data: STRING);

Take string of bytes from device 'lex',
ALLOCATE 'data' in Heap, and copy taken string
to 'data' memory.

Берет строку байтов с устройства "lex", аллоцирует "data" в области динамической памяти и копирует взятые строки в выделенную память "data".

Array 'data' is always 1 byte longer then
string from 'lex' and value of this addition
byte is 000c.

Массив "data" всегда длиннее на один байт, чем строка из "lex", и значение этого дополнительного байта - 000c.

Text file data base with special format:
<file> = <line> { 036c <line> } .
<line> = <num> ["h"] " " <text> .
must be read at instalation time at 'lex' device.

База данных текстового файла имеет специальный формат:
<файл> = <строка> { 036c <строка> } .
<строка> = <число> ["h"] " " <текст> .

и должна быть прочитана во время инсталяции на устройстве "lex".

If format of file was wrong 'inconsistency' error is returning.

Если формат файла неправильный, возвращается ошибка "inconsistency".

If line with such number absent in file 'no_entry'
error is returning.

Если строка с указанным номером в файле отсутствует, возвращается ошибка "no_entry".

If 'lex' device absent, string: "%s.%04hh",lex,code
allocated and returned, error="open device error".

Если устройство "lex" отсутствует, строка "%s.%04hh",lex,code

и возвращается ошибка "open device error".

NB:
You must 'dispose' 'data' when you have no need to use it. If you have memory allocated for 'data' before 'get' or 'read' call this memory will be lost.

Замечание. Вы должны освободить память "data", если не собираетесь использовать ее. Если у вас была память, отведенная под "data", перед вызовом процедуры "get" или "read", эта память будет утрачена.

```
PROCEDURE dispose(VAR data: STRING);
------------------------------------
```

Dispose 'data' array allocated by 'get' or 'read' call.
NB: You must always call 'dispose' after 'get',
even if you have bad result at 'get' operation.

Освобождает память, отведенную под "data" вызовом процедур "get" или "read".

```
PROCEDURE print(VAR str: ARRAY OF CHAR;
                    lex: LEX;
                   code: INTEGER;
                 format: ARRAY OF CHAR;
               SEQ args: SYSTEM.WORD);
----------------------------------------
```

String read by 'get' procedure will be inserted at position %%s of 'format'.

Строка, прочитанная процедурой "get", будет вставлена в позицию %%s указанного формата "format".

If 'lex' device absent, string: "%s.%04hh",lex,code inserted at %%s position.

Если устройство "lex" отсутствует, строка "%s.%04hh",lex,code вставляется в позицию %%s указанного формата "format".

```
PROCEDURE perror(VAR str: ARRAY OF CHAR;
                    code: INTEGER;
                  format: ARRAY OF CHAR;
                SEQ args: SYSTEM.WORD);
----------------------------------------
```

Prints error message in string using 'MSG' tskEnv parameter

Печатает сообщение об ошибке в строке, используя параметр "MSG" из окружения задачи.

**********)

END Lexicon.

```
DEFINITION MODULE lowLevel; (* Leo 29-Nov-89. (c) KRONOS *)
IMPORT  SYSTEM;

VAL
  cpu,
  cpu_model: INTEGER;

PROCEDURE move(des,sou: SYSTEM.ADDRESS; size: INTEGER);

PROCEDURE cmove(des: SYSTEM.ADDRESS; des_ofs: INTEGER;
                sou: SYSTEM.ADDRESS; sou_ofs: INTEGER; bytes:
INTEGER);

PROCEDURE _zero(adr: SYSTEM.ADDRESS; size: INTEGER);
PROCEDURE _fill(adr: SYSTEM.ADDRESS; size: INTEGER; val:
SYSTEM.WORD);

PROCEDURE zero(VAR area: ARRAY OF SYSTEM.WORD);
PROCEDURE fill(VAR area: ARRAY OF SYSTEM.WORD; val: SYSTEM.WORD);

PROCEDURE QUIT;

END lowLevel.
```

```
.PAGE

DEFINITION MODULE myEditor; (* Leo 26-Jan-89. (c) KRONOS *)
IMPORT SYSTEM;


VAR

  f_name : PROCEDURE (VAR ARRAY OF CHAR);

  last   : PROCEDURE (): INTEGER;   (* last line in text *)
  jump   : PROCEDURE (INTEGER);     (* jumps to line     *)

  crs_pos: PROCEDURE (VAR INTEGER, VAR INTEGER);

  get    : PROCEDURE (VAR ARRAY OF CHAR, VAR INTEGER);
  put    : PROCEDURE (    ARRAY OF CHAR,     INTEGER);
  app    : PROCEDURE (    ARRAY OF CHAR,     INTEGER): BOOLEAN;
  del    : PROCEDURE (INTEGER); (* delete N lines *)
  ins    : PROCEDURE (INTEGER); (* insert N lines *)
  adr    : PROCEDURE (): SYSTEM.ADDRESS;
  size   : PROCEDURE (): INTEGER;

  refresh: PROCEDURE;
  goto   : PROCEDURE (INTEGER,INTEGER); (* moves center of screen
to pos *)

           --             line0        column0      line1
column1
  frame  : PROCEDURE (VAR INTEGER, VAR INTEGER, VAR INTEGER, VAR
INTEGER);

           --           wait     format            args
  message: PROCEDURE (BOOLEAN, ARRAY OF CHAR, SEQ SYSTEM.WORD);

  mark   : PROCEDURE (INTEGER, INTEGER, ARRAY OF CHAR, SEQ
SYSTEM.WORD);
           --           fname
  onread : PROCEDURE (ARRAY OF CHAR): BOOLEAN;          -- TRUE
if possible
           --                          exit
  onwrite: PROCEDURE (ARRAY OF CHAR, BOOLEAN): BOOLEAN; -- TRUE
if possible
  onbreak: PROCEDURE (): BOOLEAN;                       -- TRUE
if possible

  start  : PROC;  (* this procedure called at the beginning of
editor work *)
                  (* but after reading source file
*)


END myEditor.
```

```
DEFINITION MODULE myShell; (* Leo 16-Jan-89. (c) KRONOS *) IMPORT
SYSTEM;

TYPE print_proc = PROCEDURE (ARRAY OF CHAR, SEQ SYSTEM.WORD);


(*
PROCEDURE system(string: ARRAY OF CHAR; print: print_proc);

PROCEDURE get_prompt(VAR prompt: ARRAY OF CHAR);
*)

VAR

     system: PROCEDURE ((*string:*) ARRAY OF CHAR, (*print:*)
print_proc);

 get_prompt: PROCEDURE (VAR (*prompt:*) ARRAY OF CHAR);

     result: INTEGER;
             (* result of last runned task *)

    history: ARRAY [0..1023] OF CHAR;
             (* history of of last unsuccessfully terminated task
*)

END myShell.
```

```
DEFINITION MODULE Printer; (* Leg 10-Dec-89. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE BARS   = ARRAY [0..2] OF ARRAY [0..2] OF CHAR;
     WIDTH  = POINTER TO ARRAY CHAR OF INTEGER;
     STATUS =
          RECORD
            type     : INTEGER;
            fonts    : INTEGER;
            font     : INTEGER;
            hchar    : INTEGER;  (* ground box *)
            wchar    : WIDTH  ;  (* in  pixels *)
            something: INTEGER;
            underline: INTEGER;
            Wx2      : INTEGER;
            Hx2      : INTEGER;
            reverse  : INTEGER;
            raw      : INTEGER;
            awp      : INTEGER;
            idle     : INTEGER;
            grounds  : INTEGER;
            ground   : INTEGER;
            hbar     : CHAR   ;
            vbar     : CHAR   ;
            bars     : BARS   ;
            densities: INTEGER;  -- number of density
            density  : INTEGER;  -- dot per inch
          END;
     STATE = POINTER TO STATUS;

VAL    done: BOOLEAN;
      error: INTEGER;
      state: STATE;

-------------------------  STANDARD  -------------------------
                           ------------
PROCEDURE Write(ch: CHAR);

PROCEDURE WriteString(s: ARRAY OF CHAR);

PROCEDURE WriteLn;

-------------------------  EXTEND  --------------------------
                           ----------

PROCEDURE write(buf: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE repeat(ch: CHAR; no: INTEGER);

---------------  CHARACTER TYPEFACE CONTROL  -----------------
                 ------------------------------
```

```
PROCEDURE set_font(no: INTEGER);

PROCEDURE set_something(on: INTEGER);

PROCEDURE set_reverse(on: INTEGER);

PROCEDURE set_underline(on: INTEGER);

PROCEDURE set_Wx2(on: INTEGER);

PROCEDURE set_Hx2(on: INTEGER);

PROCEDURE set_ground(c: INTEGER);


--------------------------- MOVEMENT ---------------------------
                             ----------
PROCEDURE fflf(n: INTEGER); (* Forward Full Line Feed *)
PROCEDURE fhlf(n: INTEGER); (* Forward Half Line Feed *)
PROCEDURE bflf(n: INTEGER); (* Back    Full Line Feed *)
PROCEDURE bhlf(n: INTEGER); (* Back    Half Line Feed *)

PROCEDURE fwd (pixels: INTEGER);
PROCEDURE back(pixels: INTEGER);

PROCEDURE right(pixels: INTEGER); (* move carrige left  *)
PROCEDURE left (pixels: INTEGER); (* move carrige right *)

PROCEDURE eject;
(* eject sheet or (for roll paper) form feed *)

--------------------------   GRAPHIC  --------------------------
                            ------------

PROCEDURE set_density(no: INTEGER);

PROCEDURE paint(map: ARRAY OF SYSTEM.WORD;
                  w,h: INTEGER;
                dx,dy: INTEGER);
(* dx is horizontal margin, dy - vertical one.
   After successfull painting carrige will be lower
   by h+dy dots and at begining of line position *)


----------------------------- MISC -----------------------------
                               ------

PROCEDURE load_font(no: INTEGER; from,to: CHAR;
                    font: ARRAY OF SYSTEM.WORD);

PROCEDURE set_awp(on: INTEGER);
```

```
PROCEDURE set_raw(on: INTEGER);

PROCEDURE set_attr(no,val: INTEGER);

PROCEDURE get_attr(no: INTEGER): INTEGER;

PROCEDURE restore(status: STATUS);

PROCEDURE reset;

PROCEDURE nop;

PROCEDURE attach(name: ARRAY OF CHAR);

PROCEDURE ioctl(op: INTEGER; SEQ args: SYSTEM.WORD);

END Printer.
```

N O T E

This module does not attach driver at initialization. Attaching is peformed when first calling of driver occure. Variable -state- points to blank record before attaching. Hence, if you want to use variable -state- before any calling to driver, you have to call procedure -nop- to fulfil attaching.

```
DEFINITION MODULE realMath; (* Dima 27-Mar-88. (c) KRONOS *)

CONST
  pi  = 3.1415926;
  e   = 2.7182818;
  eps = 0.000001;

PROCEDURE sqrt(x: REAL): REAL;

PROCEDURE sin(x: REAL): REAL;
PROCEDURE cos(x: REAL): REAL;
PROCEDURE tan(x: REAL): REAL;

PROCEDURE arctan(x: REAL): REAL;
PROCEDURE arccos(x: REAL): REAL;
PROCEDURE arcsin(x: REAL): REAL;

PROCEDURE exp(x: REAL): REAL;
PROCEDURE ln (x: REAL): REAL;
PROCEDURE lg (x: REAL): REAL;

PROCEDURE log(x,y: REAL): REAL; -- log_x_(y)

PROCEDURE power(x,y: REAL): REAL; -- x**y

END realMath.
```

```
DEFINITION MODULE regExpr; (* Leo 07-Oct-89. (c) KRONOS *)

(* This module manage strings comparisions with regular
expressions *)

TYPE
  EXPR;

VAL null: EXPR;
    done: BOOLEAN;    epos: INTEGER; (* position where error
occured *)
   error: INTEGER;

PROCEDURE compile(expr: ARRAY OF CHAR; VAR reg: EXPR);
(* compile regular expression             *)
(* error will be not changed if done      *)
(* errors: no_memory, bad_parm            *)

PROCEDURE const(reg: EXPR): BOOLEAN;
(* returns TRUE iff 'reg' is a constant expression *)

PROCEDURE match(re: EXPR; string: ARRAY OF CHAR; pos: INTEGER):
BOOLEAN;
(* matching 'string' with regular expression 're' *)

PROCEDURE len(re: EXPR; n: INTEGER): INTEGER;
PROCEDURE pos(re: EXPR; n: INTEGER): INTEGER;
(* returns length and position start of substring *)
(* matched to $n at last call of 'match' proc     *)

PROCEDURE substitute(re      : EXPR;
                     string  : ARRAY OF CHAR;
                     model   : ARRAY OF CHAR;
                     VAR dstr: ARRAY OF CHAR);

PROCEDURE dispose(VAR reg: EXPR); (* not changed "done", "error"
*)
(* makes memory occupied by regular expression free *)


END regExpr.


(* This module manage strings comparisions with regular  *)
(* expressions with following syntax:                    *)
(*                                                       *)
(*   RE     = term { "|" term } .                        *)
(*   term   = factor { "&" factor } .                    *)
(*   factor = ["^"] factor | "(" RE ")" pfix | simple .  *)
(*   simple = re { re } .                                *)
(*   pfix   = [ "$" dig1_9 ] .                           *)
(*   re     = str | "[" ["^"] set "]" pfix               *)
```

```
(*                | "{" ["^"] set "}" pfix              *)
(*                | "*" pfix                            *)
(*                | "?" pfix .                          *)
(*   str    = char1 { char1 } .                         *)
(*   char1  = ' any character except                    *)
(*              "[","*","?","\","(","{",")", 0c '       *)
(*          | "\" od od od | "\*" | "\?" | "\[" | "\&"  *)
(*          | "\|" |  "\^" | "\(" | "\{" | "\)"        *)
(*          | "\n" |  "\r" | "\l" | "\$" | "\\" .      *)
(*   set    = { char2 | char2 "-" char2 } .             *)
(*   char2  = ' any character except "]","-",0c '       *)
(*          | "\" od od od | "\-" | "\]" | "\^"        *)
(*          | "\n" |  "\r" | "\l" | "\\" .             *)
(*   od     = "0" | "1" | "2" | "3" |                   *)
(*   dig1_9 = "1" | "2" | "3" | "4" | "5" |             *)
(*            "6" | "7" | "8" | "9" .                   *)

(* NOTE: implementation uses Heap!                     *)
```

```
DEFINITION MODULE Screen; (* Leo & nick 26-Mar-90. (c) KRONOS *)

IMPORT  SYSTEM;
IMPORT  defScreen;

TYPE
  BLOCK  = defScreen .BLOCK;       SCREEN  = defScreen .SCREEN;
  REGION = defScreen .REGION;      PALETTE = defScreen .PALETTE;

VAR   cd: SCREEN;  (* current display *)

VAL done: BOOLEAN;
   error: INTEGER;

PROCEDURE open (VAR scr: SCREEN; name: ARRAY OF CHAR);
PROCEDURE close(VAR scr: SCREEN);

PROCEDURE set_ldcx(scr: SCREEN; x: INTEGER);
PROCEDURE set_ldcy(scr: SCREEN; y: INTEGER);

PROCEDURE set_palette(scr: SCREEN;       p: PALETTE; from,len:
INTEGER);

PROCEDURE refresh      (scr: SCREEN; x,y,l: INTEGER; dsp:
REGION);
PROCEDURE refresh_block(scr: SCREEN; x,y,l: INTEGER; dsp: REGION;
blk: BLOCK);

PROCEDURE op(scr: SCREEN; cmd: INTEGER; SEQ args: SYSTEM.WORD);

PROCEDURE initREGION(rgn: REGION);

END Screen.
```

```
DEFINITION MODULE Signals;(* Ned 23-Jan-90. (c) KRONOS *)

TYPE
  SIGNAL;
  MUTEX;

VAL null: SIGNAL;

PROCEDURE new_signal(VAR s: SIGNAL; no: INTEGER; VAR done:
BOOLEAN);
PROCEDURE rem_signal(VAR s: SIGNAL);

PROCEDURE send(s: SIGNAL);
PROCEDURE wait(s: SIGNAL);

PROCEDURE clear(s: SIGNAL);

PROCEDURE broadcast(s: SIGNAL);
PROCEDURE signal   (s: SIGNAL; n: INTEGER);

PROCEDURE delay_wait(VAR cause: INTEGER; milisec: INTEGER; s:
SIGNAL);
-- cause=-1, if timeout, else 0

PROCEDURE alt (VAR s: SIGNAL;  milisec: INTEGER; SEQ ss: SIGNAL);
-- s=null, if timeout, else signal

PROCEDURE alts(VAR n: INTEGER; milisec: INTEGER;     ss: ARRAY OF
SIGNAL);
-- n=-1,   if timeout, else index of signal

PROCEDURE awaited (s: SIGNAL): BOOLEAN;
PROCEDURE sendings(s: SIGNAL): INTEGER;

-----------------------------------------------------------------

PROCEDURE new_mutex(VAR m: MUTEX; VAR done: BOOLEAN);
PROCEDURE rem_mutex(VAR m: MUTEX);

PROCEDURE acquire(m: MUTEX);
PROCEDURE release(m: MUTEX);

(*****************************************************************

     Модуль определяет механизмы синхронизации двух видов:

        - сигнальная синхронизация;
        - критические интервалы.

Модуль использует память из области задачи.

--------------------------  СИГНАЛЫ  --------------------------
```

Сигнал представляет собой пару:

- счетчик посланных сигналов;
- очередь процессов, ждущих сигнал.

Над сигналом определены две базовые операции: посылка и ожидание.

Ожидание сигнала: если счетчик посланных больше 0, то уменьшает счетчик, иначе процесс задерживается и становится в конец очереди задержанных процессов.

Посылка сигнала: есди очередь задержанных сигналов пуста, то увеличивает счетчик сигналов, иначе забирает из очереди задержанных первый процесс и делает его готовым. Операция посылки не задерживает посылающий процесс.

Все операции возбуждают прерывание 4Ah (check bounds), если сигнал не проинициалицирован процедурой new_signal.

```
PROCEDURE new_signal(
---------------------
                    VAR s: SIGNAL;
                       no: INTEGER;
                  VAR done: BOOLEAN);
```

Операция создания нового сигнала. Созданый сигнал будет послан no раз. Параметр done=FALSE, если нет памяти.

ОШИБКИ:
- ASSERT(illegal parameter), если no<0
- done=FALSE, если нет памяти

```
PROCEDURE rem_signal(VAR s: SIGNAL);
------------------------------------
```

Освобождение памяти занятой сигналом. Операция корректна только если никто не ждет сигнала. Проверьте, что сигнал не используется после освобождения.

ОШИБКИ:
- ASSERT(illegal parameter), если сигнала кто-то ждет.

```
PROCEDURE send(s: SIGNAL);
--------------------------
```

Посылка сигнала.

```
PROCEDURE wait(s: SIGNAL);
--------------------------
```

Ожидание сигнала.

```
PROCEDURE clear(s: SIGNAL);
---------------------------
```

Очистка счетчика сигналов. Пустое действие, если счетчик сигналов равен нулю.

```
PROCEDURE broadcast(s: SIGNAL);
-------------------------------
```

Посылает сигнал всем процессам, стоящим в очереди к сигналу. Пустое действие, если очередь задержанных пуста.

```
PROCEDURE signal(s: SIGNAL; n: INTEGER);
----------------------------------------
```

Посылает сигнал n раз.

```
PROCEDURE delay_wait(
---------------------
                     VAR cause: INTEGER;
                         delay: INTEGER;
                             s: SIGNAL);
```

Ожидание сигнала в течении времени. Процесс будет продолжен или при получении сигнала, или по истечению времени. Время измеряется в системных квантах (тиках). Для пересчета из нормальных единиц в тики используйте библиотеку Time. Если delay<0, то эквивалентна процедуре wait(s) (время задержки бесконечно велико). Параметр cause содержит причину продолжения процесса:

```
        cause = -1      - кончилось время задержки;
        cause =  0      - получен сигнал.
```

```
PROCEDURE alt(VAR s: SIGNAL; delay: INTEGER; SEQ ss: SIGNAL);
-------------------------------------------------------------
```

Альтернативное ожидание нескольких сигналов в течении времени. Процесс будет продолжен по истечении времени задержки, или при получении одного из сигналов

```
        { ss[i] | i=0..HIGH(ss) & ss[i]#null }
```

Если время задержки меньше 0, то задержка бесконечно велика. Если сигналов нет (HIGH(ss)<0), или все сигналы равны null, то процесс будет ждать окончания времени задержки. Параметр s определяет причину продолжения процесса: s = null - кончилось время задержки, иначе получен сигнал s.

```
PROCEDURE alts(VAR n: INTEGER; delay: INTEGER; ss: ARRAY OF
SIGNAL);
```

Аналогично процедуре alt. Параметр n определяет причину

продолжения процесса: n = -1 - кончилось время задержки, иначе n равен индексу в массиве ss полученного сигнала.

```
PROCEDURE awaited(s: SIGNAL): BOOLEAN;
--------------------------------------
```

Возвращает TRUE, если очередь задержанных процессов не пуста.

```
PROCEDURE sendings(s: SIGNAL): INTEGER;
---------------------------------------
```

Возвращает значение счетчика посланных сигналов.

---------- КРИТИЧЕСКИЕ ИНТЕРВАЛЫ ----------

Критический интервал (mutual exclusion) позволяет защитить общие данные от одновременного обращения к ним нескольких процессов. Процесс может захватить КИ и до того, как он освободит этот КИ никакой другой процесс не может его захватить. Все процесса пытающиеся захватить захваченный КИ, становятся в очередь к нему. При освобождении КИ будет продолжен процесс, стоящий первым в очереди к нему. Процесс может захватить несколько КИ и должен освобождать их в обратном (стековом) порядке. Процесс также может несколько раз захватить один и тот же КИ и должен освободить его столько раз, сколько захватил.

При завершении процесса, все захваченные им КИ будут освобождены.

```
PROCEDURE new_mutex(VAR m: MUTEX; VAR done: BOOLEAN);
-----------------------------------------------------
```

Операция создания нового критического интервала. Параметр done=FALSE, если не хватило памяти для создания его.

```
PROCEDURE rem_mutex(VAR m: MUTEX);
----------------------------------
```

Удаление КИ. КИ должен быть свободен.

```
PROCEDURE acquire(m: MUTEX);
----------------------------
```

Захват критического интервала.

```
PROCEDURE release(m: MUTEX);
----------------------------
```

Освобождение критического интервала.

```
*************************************************************)

END Signals.
```

```
DEFINITION MODULE Sorts; (* Ned 03-Mar-90. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE
  COMP  = PROCEDURE (SYSTEM.WORD,INTEGER,INTEGER): INTEGER;
  COMPW = PROCEDURE (SYSTEM.WORD,SYSTEM.WORD): INTEGER;
  SWAP  = PROCEDURE (SYSTEM.WORD,INTEGER,INTEGER);

PROCEDURE quick(x: SYSTEM.WORD; len: INTEGER; comp: COMP; swap:
SWAP);
PROCEDURE heap (x: SYSTEM.WORD; len: INTEGER; comp: COMP; swap:
SWAP);

PROCEDURE quickw(VAR x: ARRAY OF SYSTEM.WORD; len: INTEGER; comp:
COMPW);
PROCEDURE heapw (VAR x: ARRAY OF SYSTEM.WORD; len: INTEGER; comp:
COMPW);

PROCEDURE str_comp(s1,s2: ARRAY OF CHAR): INTEGER;
PROCEDURE abc_comp(s1,s2: ARRAY OF CHAR): INTEGER;
PROCEDURE ABC_comp(s1,s2: ARRAY OF CHAR): INTEGER;

(************************************************************
```

Модуль предоставляет операции сортировки произвольных структур двумя методами: быстрая сортировка и пираминдальная сортировка (см. Н.Вирт, Алгоритмы+Структуры Данных = Программы, стр.89-99). Для каждого из этих методов реализованы по две процедуры:

- сортировка массивов слов;
- сортировка произвольных структур.

Сравнение скорости работы процедур (на случайных данных):

quickw < quick < heapw < heap.

Пираминдальная сортировка хороша тем, что скорость ее работы практически не зависит от данных. Быстрая сортировка может работать довольно медленно на некоторых данных.

Процедуры сравнения:

```
  COMP  = PROCEDURE (SYSTEM.WORD,INTEGER,INTEGER): INTEGER;
  COMPW = PROCEDURE (SYSTEM.WORD,SYSTEM.WORD): INTEGER;
```

Процедуры сравнения должны возвращать

```
        <0, если первый аргумент < второго
         0, если первый аргумент = второму
        >0, если первый аргумент > второго
```

```
PROCEDURE quick & heap
----------------------
     (x: SYSTEM.WORD; len: INTEGER; comp: COMP; swap: SWAP);
```

Сортировка произвольных структур. Первый параметр не используется собственно процедурами сортировки и просто передается процедурам сравнения и обмена. Процедуры сортируют индексы в диапазоне [0..len-1].

```
PROCEDURE quickw & heapw
------------------------
     (VAR x: ARRAY OF SYSTEM.WORD; len: INTEGER; comp: COMPW);
```

Сортировка массива слов в диапазоне [0..len-1]. Процедуре сравнения передаются элементы массива.

Операции сравнения строк. Строки ДОЛЖНЫ завершаться символом 0c.

```
PROCEDURE str_comp(s1,s2: ARRAY OF CHAR): INTEGER;
--------------------------------------------------
```

Сравнение строк. Порядок на литерах определяется стандартом принятым в системе (ДКОИ-8).

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

Сравнение строк. Буквы как русского, так и латинского алфавита распологаются в алфавитном порядке. Порядок остальных символов определяется стандартом.

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

Сравнение строк. Буквы как русского, так и латинского алфавита распологаются в алфавитном порядке. Порядок остальных символов определяется стандартом. Большие и маленькие буквы не различаются.

```
**********************************************************)

END Sorts.
```

```
DEFINITION MODULE Statistics; (* Leo 28-Feb-90. (c) KRONOS *)

IMPORT  SYSTEM;

CONST
  os_vers    = 101h;    (* system version no      *)
  os_runtime = 102h;    (* runtime in seconds     *)

  mem_top    = 201h;    (* main memory top        *)
  mem_core   = 202h;    (* occupied by system     *)
  mem_total  = 203h;    (* total memory           *)
  mem_free   = 204h;    (* total free             *)

  fs_chsize  = 301h;    (* file cash in words     *)
  fs_dkwrite = 302h;    (* total writen sectors   *)
  fs_dkread  = 303h;    (* sectors read from disk *)
  fs_chread  = 304h;    (* sectors read from cash *)

PROCEDURE get(attr: INTEGER; VAR val: SYSTEM.WORD);

END Statistics.
```

```
DEFINITION MODULE strEditor;  (* Andy 09-Jul-89. (c) KRONOS *)
                              (* Ned  05-Oct-89. (c) KRONOS *)
                              (* Leo  08-Nov-89. (c) KRONOS *)
                              (* Andy 06-Jan-90. (c) KRONOS *)

(* Модуль поставляет процедуры редактирования строки *)

TYPE
  LINE_BUFFER;

  VALID = PROCEDURE (VAR CHAR): BOOLEAN;
   READ = PROCEDURE (VAR CHAR);
   BELL = PROCEDURE (INTEGER);

  descriptor = POINTER TO desc_rec;
    desc_rec = RECORD
                   buf : LINE_BUFFER;
                   last: CHAR;     -- последний символ
                  valid: VALID;    -- фильтр вводимых символов
                   ins : BOOLEAN;  -- режим вставки
                   bel : BOOLEAN;  -- режим звукового сигнала при
ошибках
                   how : BITSET;   -- режим показа строки
                   read: READ;     -- процедура чтения символа
                   bell: BELL;     -- процедура подачи сигнала
                                   -- при ошибке редактирования
               END;

CONST -- display flags
      _empty   = 0;
      _bol     = 1;
      _refresh = 2;

CONST -- how:
      empty   = {_empty,_refresh};
      show    =        {_refresh,_bol};
      confirm =        {_refresh};

PROCEDURE read_str(    prompt: ARRAY OF CHAR;
                   VAR string: ARRAY OF CHAR;
                           dsc: descriptor;
              SEQ terminators: CHAR);

PROCEDURE edit_str(    prompt: ARRAY OF CHAR;
                   VAR string: ARRAY OF CHAR;
                line,col1,col2: INTEGER;
                           dsc: descriptor;
              SEQ terminators: CHAR);

PROCEDURE new    (VAR desc: descriptor; no_lines: INTEGER);
PROCEDURE dispose(VAR desc: descriptor);
```

```
PROCEDURE set_prefix(desc: descriptor; prefix: ARRAY OF CHAR;
                                       VAR done: BOOLEAN);

(************************************************************

       Модуль поставляет процедуры редактирования строки.
       ==================================================
         Ввод   осуществляется  процедурой read, содержащейся
                в дескрипторе редактора.
         Вывод  осуществляется  через  модуль Terminal.

         Каждый введенный символ проверяется на допустимость
         и, возможно, изменяется фильтром типа VALID,
         содержащимся в дескрипторе редактора.
         При valid(symbol)=TRUE возвращенный фильтром символ
         интерпретируется редактором, иначе он игнорируется.

         Флаги, управляющие показом строки
         в начале редактирования:

  _empty   - строка считается пустой
  _refresh - перед началом редактирования
             строка перерисовывается
  _bol     - курсор ставится в начало, а не в конец строки

         Режимы показа строки в начале редактирования

  empty    - строка считается пустой и показывается
  show     - показывает строку, курсор в начале строки
  confirm  - показывает строку, курсор в конце строки

PROCEDURE new(VAR desc: descriptor; n: INTEGER);
-------------
Генерация нового дескриптора с буфером на n строк
Все строки буфера инициализируются пустой строкой
ins=TRUE, bel=TRUE, how=empty, valid=std_valid,
read=Keyboard.read, bell=Keyboard.bell
При нехватке памяти возвращает NIL

  Стандартный фильтр std_valid допускает символ, если он
  является либо функциональным символом редактора (см. ниже),
  либо не контрольным символом.

PROCEDURE dispose(VAR desc: descriptor);
-----------------

PROCEDURE set_prefix(desc: descriptor; prefix: ARRAY OF CHAR;
-------------------                    VAR done: BOOLEAN);
 Строка -prefix- привязывается к дескриптору -desc-.
При следующем вызове редактора с этим дескриптором
строка будет проинтерпретирована, как если бы она
была прочитана символ за символом процедурой read.
 После окончания сеанса редактирования строка уничтожается.
```

```
 done = TRUE, если строку удалось привязать;
      = FALSE при нехватке памяти.

-------------------------------------------------------------
 Набор функциональных символов редактора:

 Keyboard.cr,
 ASCII.NL
 и любой символ
 из последовательности
 -terminators-      : конец редактирования;
                      полученная строка помещается в буфер,
                      если она не пуста и не равна
                      последней помещенной в буфер строке;
                      символ-терминатор записывается
                      в дескриптор редактора
 Keyboard.lf        : аналогично Keyboard.cr,
                      но предварительно уничтожается хвост
                      строки справа от курсора
 Keyboard.f1        : если следующий прочитанный символ
                         допускается фильтром редактора
                         и равен одному из символов:
                         Keyboard.back или Keyboard.del,
                         то выполняется такое же действие,
                         что и в текстовом редакторе "ex",
                      иначе символ игнорируется
 Keyboard.f2        : если следующий прочитанный символ
                         допускается фильтром редактора
                         и равен одному из символов:
                         Keyboard.left, Keyboard.right,
                         Keyboard.back или Keyboard.del,
                         то выполняется такое же действие,
                         что и в текстовом редакторе "ex",
                      иначе символ игнорируется
 Keyboard.f4        : уничтожение всей строки
 Keyboard.f8        : уничтожение хвоста строки
                      справа от курсора
 Keyboard.tab       : табуляция вправо
 Keyboard.bcktab    : табуляция влево
 ^W                 : начало первого слова справа от курсора
                      подтягивается в текущую позицию
 ^D                 : начало первого слова строки
                      подтягивается в текущую позицию
 Keyboard.left      : курсор влево  на один символ
 Keyboard.right     : курсор вправо на один символ
 Keyboard.back      : уничтожение символа слева от курсора
 Keyboard.ins       : вставка пробела в текущую позицию
 Keyboard.del       : уничтожение символа в текущей позиции
 Keyboard.home      : курсор на начало строки
 Keyboard.end       : курсор в конец строки
 Keyboard.rep       : смена режима вставки/замены символа
 ASCII.BEL (^G)     : смена режима звукового сигнала
                      в случае ошибки
```

```
 Keyboard.up        : текущая позиция в буфере строк
                      увеличивается на 1, если это возможно;
                      в строку копируется текущий
                      элемент буфера
 Keyboard.dw        : текущая позиция в буфере строк
                      уменьшается на 1, если это возможно;
                      в строку копируется текущий
                      элемент буфера
 Keyboard.f7        : в хвост строки справа от курсора
                      копируется соответствующий хвост
                      (текущего+1) элемента буфера

 специальные возможности для ввода символов:
 ASCII.DC2 (^R)     : в строку помещается байт, значение
                      которого вводится в виде
                      ровно трех восьмеричных цифр;
                      байт фильтром не обрабатывается
 ASCII.DC4 (^T)     : в строку помещается байт, прочитанный
                      напрямую, без обработки фильтром редактора
-------------------------------------------------------------

PROCEDURE read_str(    prompt: ARRAY OF CHAR;
------------------ VAR string: ARRAY OF CHAR;
                           dsc: descriptor;
              SEQ terminators: CHAR);
Если dsc=NIL, то используется некоторый дескриптор по умолчанию

Редактирование производится в однострочной области
от позиции курсора в начале редактирования (после вывода
строки -prompt-) до конца экрана.
Для корректной работы длина редактируемой строки не должна
превышать размера области.

PROCEDURE edit_str(    prompt: ARRAY OF CHAR;
------------------ VAR string: ARRAY OF CHAR;
                line,col1,col2: INTEGER;
                           dsc: descriptor;
              SEQ terminators: CHAR);
Если dsc=NIL, то используется некоторый дескриптор по умолчанию

Редактирование производится в однострочной области экрана:
строка -line-, колонки с -col1- включительно по -col2-
включительно. Строка -prompt- выводится в этой же области.

*************************************************************)

END strEditor.
```

```
DEFINITION MODULE StdIO; (* Andy 17-Oct-89. (c) KRONOS *)
                         (* Leo  27-Jun-90. (c) KRONOS *)

IMPORT  SYSTEM;
IMPORT  BIO;

             (* Стандартный потоковый ввод/вывод *)

VAL done: BOOLEAN;
   error: INTEGER;
   iolen: INTEGER;

VAL EOF: CHAR;

VAR
  in : BIO.FILE;
  out: BIO.FILE;

PROCEDURE Write(ch: CHAR);
PROCEDURE Read(VAR ch: CHAR);
PROCEDURE WriteString(s: ARRAY OF CHAR);
PROCEDURE WriteLn;
PROCEDURE ReadString(VAR s: ARRAY OF CHAR);

PROCEDURE is_tty(s: BIO.FILE): BOOLEAN;

PROCEDURE read    (VAR  ch: CHAR);
PROCEDURE readstr (VAR str: ARRAY OF CHAR);
PROCEDURE write   (     ch: CHAR);
PROCEDURE writestr(    str: ARRAY OF CHAR);
PROCEDURE writeln;

PROCEDURE print (format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);
PROCEDURE perror(errcod: INTEGER;
                 format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);


(*************************************************************

VAL done: BOOLEAN; -- Результат выполнения последней операции
   error: INTEGER;
   iolen: INTEGER;

VAL EOF: CHAR;

VAR
  in : BIO.FILE;
  out: BIO.FILE;

PROCEDURE  is_tty(f: BIO.FILE): BOOLEAN;

PROCEDURE Write(ch: CHAR);
```

```
PROCEDURE write(ch: CHAR);
(* Выводит один символ в файл стандартного вывода. *)

PROCEDURE Read(VAR ch: CHAR);
PROCEDURE read(VAR ch: CHAR);
(* Читает один символ из файла стандартного ввода;   *)
(* возвращает EOF при попытке чтения за концом файла *)

PROCEDURE WriteString(s: ARRAY OF CHAR);
PROCEDURE writestr   (s: ARRAY OF CHAR);
(* Выдает  в файл стандартного вывода содержимое строки символ
   за  символом. Концом строки считается либо физический конец
   предоставленного  массива, либо символ 0c, в зависимости от
   того, что встретится раньше.
*)

PROCEDURE WriteLn;
PROCEDURE writeln;

PROCEDURE ReadString(VAR s: ARRAY OF CHAR);
PROCEDURE readstr   (VAR s: ARRAY OF CHAR);

PROCEDURE print (format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);
PROCEDURE perror(errcod: INTEGER;
                 format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);

**********************************************************)
END StdIO.
```

```
DEFINITION MODULE Strings; (* Ned 20-Jun-89. (c) KRONOS *)

IMPORT  SYSTEM;

PROCEDURE len (str: ARRAY OF CHAR): INTEGER;

PROCEDURE app (VAR dst: ARRAY OF CHAR; str: ARRAY OF CHAR);

PROCEDURE copy(VAR dst: ARRAY OF CHAR; str: ARRAY OF CHAR);


PROCEDURE print (VAR str: ARRAY OF CHAR;
                     fmt: ARRAY OF CHAR;
                 SEQ arg: SYSTEM.WORD);

PROCEDURE append(VAR str: ARRAY OF CHAR;
                     fmt: ARRAY OF CHAR;
                 SEQ arg: SYSTEM.WORD);

PROCEDURE image (VAR str: ARRAY OF CHAR;
                 VAR pos: INTEGER;
                     fmt: ARRAY OF CHAR;
                 SEQ arg: SYSTEM.WORD);


PROCEDURE delete (VAR str: ARRAY OF CHAR; pos,len: INTEGER);
PROCEDURE insert (VAR str: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE sub_str(VAR dst: ARRAY OF CHAR;
                      str: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE sub_arr(VAR dst: ARRAY OF CHAR;
                      str: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE replace(VAR dst: ARRAY OF CHAR;
                      str: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE skip  (str: ARRAY OF CHAR; VAR pos: INTEGER; ch: CHAR);
PROCEDURE search(str: ARRAY OF CHAR; VAR pos: INTEGER; ch: CHAR);

PROCEDURE scan(str : ARRAY OF CHAR;
           VAR pos : INTEGER;
               patt: ARRAY OF CHAR;
           VAR done: BOOLEAN);

PROCEDURE iscan(VAR num : SYSTEM.WORD;
                    str : ARRAY OF CHAR;
                VAR pos : INTEGER;
                VAR done: BOOLEAN);

PROCEDURE rscan(VAR real: REAL;
                    str : ARRAY OF CHAR;
```

```
                VAR pos : INTEGER;
                VAR done: BOOLEAN);

PROCEDURE move(VAR to: ARRAY OF CHAR; t_ofs: INTEGER;
                 from: ARRAY OF CHAR; f_ofs: INTEGER; len:
INTEGER);

(*************************************************************
```

Строка - массив литер любой длины. Все процедуры (кроме sub_arr) работает с частью строки от начала строки до символа "конец строки" (000c) или до конца массива (HIGH) в котором хранится строка. Во всех процедурах возбуждается ошибка "неверный аргумент", если позиция или длина <0.

ВНИМАНИЕ:

Все процедуры "добавляющие" в строку игнорируют не влезающие символы. И гарантируют наличия символа "конец строки" (000c) в строке-результате, если массив для сохранения результата имеет не нулевой размер (HIGH>=0).

ОБЫЧНЫЕ ОПЕРАЦИИ

PROCEDURE len

Подсчитывает длину строки до 0c или HIGH.

PROCEDURE app

Дописывает строку str в строку dst с позиции len(dst) до 0c или HIGH(dst) или HIGH(str) (что встретится раньше).

PROCEDURE copy

Копирует строку str в строку dst до 0c или HIGH(dst) или HIGH(str) (что встретится раньше).

ФОРМАТНЫЙ ВЫВОД В СТРОКИ

PROCEDURE print

Формирует строку, определяемую парой (format,args).

PROCEDURE append

Добавляет строку, определяемую парой (format,args) к строке str, начиная с позиций в которой стоит символ 0c.

PROCEDURE image

Добавляет строку, определяемую парой (format,args) к строке

str, начиная с позиций pos. После вызова pos указывает на символ 0c, или за конец (HIGH) строки.

РАБОТА С ПОДСТРОКАМИ

PROCEDURE delete

Удаляет часть строки начиная с pos длиной len

PROCEDURE insert

Вставляет в строку len пробелов начиная с позиции pos

PROCEDURE sub_str

Копирует в dst подстроку строки str начиная с pos длиной не больше чем len. Если pos меньше чем длинна строки str подсчитаная функцией len(str), результатом будет пустая строка.

PROCEDURE sub_arr

Аналогична sub_str но не пользуется функцией len(str), что позволяет "добывать" из строки "str" произвольный под-массивы.

PROCEDURE replace

Заменяет в dst символы начиная с позиции pos на символы из строки str (начиная с начала). Замена прекращается, если кончилась строка str или строка dst или в строке str встретился символ 0c. НЕ завершает замененные символы символом 0c.

PROCEDURE skip

Пропускает символы равные ch, начиная с позиции pos, увеличивая pos. Завершается если кончилась строка (pos>HIGH(str)) или текущий символ не равен ch (str[pos]#ch).

PROCEDURE search

Поиск символа ch, начиная с позиции pos, Завершается если кончилась строка (pos>HIGH(str)) или текущий символ равен ch (str[pos]=ch).

PROCEDURE scan(

```
              str : ARRAY OF CHAR;
          VAR pos : INTEGER;
              patt: ARRAY OF CHAR;
```

```
            VAR done: BOOLEAN);
```

Пропускает символы в строке, начиная с позиции pos, пока они совпадают с образцом patt. Если образец совпал с подстрокой, то done=TRUE.

```
PROCEDURE iscan(
---------------
                VAR num : SYSTEM.WORD;
                    str : ARRAY OF CHAR;
                VAR pos : INTEGER;
                VAR done: BOOLEAN);
```

Считывает из строки str число, начиная с позиции pos. Пропускает пробелы. После вызова:

```
        done  -- = TRUE, если удалось считать число;
        pos   -- индекс следующего за числом символа;
        num   -- число, если done.
```

Число может быть представлено в любом виде, допустимом в Модуле-2:

```
        123456789
       -1
        0ABCDEFh
        177b
       -177b
        377c
```

```
PROCEDURE rscan(
---------------
                VAR real: REAL;
                    str : ARRAY OF CHAR;
                VAR pos : INTEGER;
                VAR done: BOOLEAN);
```

Аналогично iscan, только считывает вещественное число.

```
*************************************************************)

END Strings.
```

```
DEFINITION MODULE Tasks; (* Ned 19-Nov-89. (c) KRONOS *)

IMPORT       SYSTEM;
IMPORT  def: defTasks;
IMPORT  syn: Signals;

TYPE TASK;

TYPE WORDs = DYNARR OF SYSTEM.WORD;

VAL done: BOOLEAN;
   error: INTEGER;
    note: ARRAY [0..79] OF CHAR;

(* "note" setted when error in "create","chpaths" only *)

VAL
  null: TASK;
  self: TASK;

PROCEDURE chpaths;

PROCEDURE create(VAR task: TASK;
                     papa: TASK;
                     name: ARRAY OF CHAR;
                    alias: ARRAY OF CHAR;
                    stack: INTEGER;
                     parm: ARRAY OF CHAR;
                    );

PROCEDURE run(task: TASK);

PROCEDURE open (VAR task: TASK; papa: TASK; id: INTEGER);
PROCEDURE close(VAR task: TASK);

PROCEDURE caller(VAR id: INTEGER);

------------------------------------------------------------------

CONST
  stop = def.stop;
  kill = def.kill;
  ipr  = def.ipr;

PROCEDURE signal(T: TASK; no: INTEGER);

PROCEDURE  get_signal(VAR s: syn.SIGNAL; T: TASK; no: INTEGER);
PROCEDURE free_signal(VAR s: syn.SIGNAL; T: TASK; no: INTEGER);

------------------------------------------------------------------

PROCEDURE son    (task: TASK; VAR id: INTEGER);
```

```
PROCEDURE brother(task: TASK; VAR id: INTEGER);
PROCEDURE papa   (task: TASK; VAR id: INTEGER);

PROCEDURE get_attr(T: TASK; no: INTEGER; VAR val: SYSTEM.WORD);
PROCEDURE set_attr(T: TASK; no: INTEGER;     val: SYSTEM.WORD);

CONST a_status = 0;     -- task status  (read only)
      a_mem    = 1;     -- task memory  (read only)
      a_user   = 2;     -- user         (read/write)
      a_id     = 3;     -- task ident   (read only)
      a_ipr    = 4;     -- independent? (read/write)
      a_res    = 5;     -- task result  (read only)

PROCEDURE history(task: TASK; VAR cause: INTEGER;
                              VAR his  : ARRAY OF CHAR);

PROCEDURE lookup_module(task: TASK; name: ARRAY OF CHAR);

PROCEDURE find(task: TASK; name: ARRAY OF CHAR);
(* try lookup_module otherwise try find through BIN paths *)


------------------------  ENVIRONMENT  -------------------------
                         ---------------

PROCEDURE put_env(task: TASK;
                  name: ARRAY OF CHAR;
                  data: ARRAY OF SYSTEM.WORD;
                  priv: BOOLEAN;
                  );

PROCEDURE put_str(task: TASK;
                  name: ARRAY OF CHAR;
                  data: ARRAY OF CHAR;
                  priv: BOOLEAN
                  );

PROCEDURE get_str(task: TASK; name: ARRAY OF CHAR;
                            VAR str : STRING);
PROCEDURE get_env(task: TASK; name: ARRAY OF CHAR;
                            VAR data: WORDs);

PROCEDURE del_env(task: TASK; name: ARRAY OF CHAR);

-------------------------------------------------------------------

PROCEDURE xole(task: TASK; VAR x: SYSTEM.WORD);

(*****************************************************************

-------------------------  COMMENTs  -------------------------
                          ------------
```

Модуль реализует операции создания, запуска и общения с задачами, работу с окружением задачи.

Индикант задачи можно получить двумя способами: создать задачу и открыть созданную задачу. Все остальные операции работают с индикатном открытой задачи. Каждая задача имеет свой номер, который и является уникальным именем задачи.

Общение с задачей происходит с помощью сигналов. Сигнал идентифицируется номером сигнала. Задаче можно послать сигнал с данным номером и у задачи можно получить сигнал, которого можно потом ждать. На уровне ядра системы фиксируются два номера сигналов. Это сигналы stop и kill.

Дескриптор задачи существует пока задача не уничтожена или задача открыта. При создании задачи она открывается. Все открытые задачи будут автоматически закрыты при завершении открывшей задачи.

```
PROCEDURE create(
-----------------
                VAR task: TASK;
                    papa: TASK;
                    name: ARRAY OF CHAR;
                   alias: ARRAY OF CHAR);
```

Создает новую задачу на базе задачи papa. Если papa=null, то создается независимая задача. Создание на базе означает что:

- при загрузке задачи кодофайлы будут искаться по списку (ветви дерева) прямых предков;
- если кодофайл, найденный у какого-либо предка уникален, то задача будет разделять (т.е. совместно использовать) глобалы этого модуля;
- при уничтожении задачи, все задачи созданные на ее базе (ее потомки) будут уничтожены.

Все задачи используют коды ядра системы (и глобалы уникальных модулей), а независимые задачи (papa=null) используют только их.

Параметр name определяет имя головного модуля задачи, alias - определяет задание для загрузчика (разделитель - пробел).

```
alias:: { выключение | уникализация | переопределение }

выключение     :: "-"имя_модуля
уникализация   :: "+"имя_модуля
переопределение:: "="имя_модуля имя_файла
```

Выключение: даже если код модуля с таким именем будет

найден среди предков задачи или в ядре системы, он будет прочитан из файла. ОШИБКА: busy, если модуль уникален.

Уникализация: модуль с этим именем становиться уникальным для данной задачи (не оказывает влияния на другие параллельно создаваемые задачи). ОШИБКА: no_entry, если модуль не найден у предков.

Переопределение: код модуля с таким именем будет прочитан из файла с именем имя_файла. Подразумевает выключение этого модуля. ОШИБКА: busy, если модуль уникален.

Действия "+","-" и "+","=" для одного модуля несовместимы. Запрещается также повторное переопределение. ОШИБКА: duplicate.

При загрузке файла с кодом в нем может быть несколько кодофайлов. При этом у всех кодофайлов должно быть выставлено поле size (defCode.code_ptr ^.size) равное размеру кодофайла в словах. Все эти кодофайлы, в том числе те, которые не используются в задаче, будут находиться в памяти до уничтожения задачи.

В случае ошибки переменная note содержит некоторую дополнительную информацию об ошибке.

ОШИБКИ:
```
        файловые ошибки при чтении кодофайла;
        no_memory
        inv_vers, inconsistency - некоректность кодофайлов;
```

```
PROCEDURE run(
--------------
              task: TASK;
        stack_size: INTEGER;
              parm: ARRAY OF CHAR;
               ipr: BOOLEAN);
```

Запусткает задачу на стеке размером stack_size, причем этот размер задает прибавку к вычисленному компилятором и загрузчиком минимальному размеру стека. Копирует окружение задачи из окружения запускающей задачи (может не совпадать с папой).

```
PROCEDURE open(VAR task: TASK; papa: TASK; id: INTEGER);
--------------------------------------------------------
```

Открывает задачу по ее номеру. Если papa=null, то поиск задачи будет выполняться по всему дереву задач, иначе только по потомкам задачи papa.

```
PROCEDURE close(VAR task: TASK);
--------------------------------
```

Закрывает задачу. ОШИБКА: busy, если не все сигналы задачи освобождены (см. free_signal).

```
PROCEDURE signal(T: TASK; no: INTEGER);
---------------------------------------
```

Посылает задаче сигнал с номером no.

```
PROCEDURE get_signal(VAR s: syn.SIGNAL; T: TASK; no: INTEGER);
--------------------------------------------------------------
```

Выдает сигнал задачи с номером no. Этот сигнал можно только ждать. Сигнал может быть послан задачей при изменении ее состояния. После использования сигнал должен быть освобожден.

```
PROCEDURE free_signal(VAR s: syn.SIGNAL; T: TASK; no: INTEGER);
```

Освобождает взятый сигнал.

```
PROCEDURE install(T: TASK; no: INTEGER; proc: PROC);
```

Определяет реакию на сигнал. Процедура proc будет вызвана при получении задачей сигнала с номером no.

```
PROCEDURE papa   (task: TASK; VAR id: INTEGER);
PROCEDURE son    (task: TASK; VAR id: INTEGER);
PROCEDURE brother(task: TASK; VAR id: INTEGER);
-----------------------------------------------
```

Возвращает номер задачи-папы/сына/брата, если таковой известен, иначе id=-1.

```
PROCEDURE get_attr(T: TASK; no: INTEGER; VAR val: SYSTEM.WORD);
--------------------------------------------------------------
```

Возвращает атрибут задачи.

```
PROCEDURE set_user(T: TASK; user: INTEGER);
-------------------------------------------
```

Устанавливает пользователя - хозяина задачи. Только для суперюзера!

```
PROCEDURE history(task: TASK; VAR his: ARRAY OF CHAR);
-----------------------------------------------------
```

История головного процесса задачи.

```
------------------------  ENVIRONMENT  ------------------------
                          ---------------

PROCEDURE put_env(task: TASK;
                  name: ARRAY OF CHAR;
                  data: ARRAY OF SYSTEM.WORD;
                  priv: BOOLEAN;
                  );

PROCEDURE put_str(task: TASK;
                  name: ARRAY OF CHAR;
                  data: ARRAY OF CHAR;
                  priv: BOOLEAN
                  );

PROCEDURE get_str(task: TASK; name: ARRAY OF CHAR; VAR str :
STRING);
PROCEDURE get_env(task: TASK; name: ARRAY OF CHAR; VAR data:
WORDs);

PROCEDURE del_env(task: TASK; name: ARRAY OF CHAR);

-------------------------------------------------------------

PROCEDURE xole(task: TASK; VAR x: SYSTEM.WORD);

*************************************************************)

END Tasks.

4D in main == exit
```

```
DEFINITION MODULE Terminal; (* Leo 18-Oct-85. (c) KRONOS *)
IMPORT  SYSTEM;
                            (* Leo 24-Apr-86. (c) KRONOS *)
                            (* Ned 21-Aug-89. (c) KRONOS *)
                            (* Ned 19-Sep-89. (c) KRONOS *)

(* Определяет операции над текущим терминалом задачи. *)

----------------------  TERMINAL STATE  ----------------------
                        ------------------
TYPE
  BARS = ARRAY [0..2],[0..2] OF CHAR;

VAL
   done: BOOLEAN;
  error: INTEGER;
  iolen: INTEGER;

  type     : INTEGER;          hbar  : CHAR;
  lines    : INTEGER;          vbar  : CHAR;
  columns  : INTEGER;          bars  : BARS;
  min_color: INTEGER;          back  : INTEGER;
  max_color: INTEGER;          color : INTEGER;
  fonts    : INTEGER;          font  : INTEGER;
  screens  : INTEGER;          scr   : INTEGER;

  cursor   : INTEGER;          awp   : INTEGER;
  something: INTEGER;          raw   : INTEGER;
  reverse  : INTEGER;          smooth: INTEGER;
  blinking : INTEGER;          cinter: INTEGER;
  underline: INTEGER;

--------------------------  STANDARD  --------------------------
                           ------------
PROCEDURE Write(ch: CHAR);

PROCEDURE WriteString(s: ARRAY OF CHAR);

PROCEDURE WriteLn;

PROCEDURE Show(str: ARRAY OF CHAR);

--------------------------  EXTENDED  --------------------------
                           ------------

PROCEDURE print(format: ARRAY OF CHAR; SEQ args: SYSTEM.WORD);

PROCEDURE write(str: ARRAY OF CHAR; pos,len: INTEGER);

PROCEDURE perror(errcode: INTEGER;
                  format: ARRAY OF CHAR;
                SEQ args: SYSTEM.WORD);
```

SCREEN

```
PROCEDURE set_pos(line,col: INTEGER);
PROCEDURE home;
PROCEDURE bottom;

PROCEDURE repeat(ch: CHAR; times: INTEGER);

PROCEDURE erase     (how: INTEGER);
PROCEDURE erase_line(how: INTEGER);
PROCEDURE erase_chars(no: INTEGER);

PROCEDURE roll_down(n: INTEGER);
PROCEDURE roll_up  (n: INTEGER);

PROCEDURE scroll_down(n: INTEGER);
PROCEDURE scroll_up  (n: INTEGER);

PROCEDURE up   (n: INTEGER);            PROCEDURE left (n:
INTEGER);
PROCEDURE down (n: INTEGER);            PROCEDURE right(n:
INTEGER);

PROCEDURE ins_char(n: INTEGER);         PROCEDURE ins_line(n:
INTEGER);
PROCEDURE del_char(n: INTEGER);         PROCEDURE del_line(n:
INTEGER);

PROCEDURE set_cursor   (ON_OFF: INTEGER);
PROCEDURE set_reverse  (ON_OFF: INTEGER);
PROCEDURE set_underline(ON_OFF: INTEGER);
PROCEDURE set_blinking (ON_OFF: INTEGER);
PROCEDURE set_something(ON_OFF: INTEGER);
PROCEDURE set_cinter   (ON_OFF: INTEGER);
PROCEDURE set_font     (no: INTEGER);
PROCEDURE load_font    (no: INTEGER; from,to: CHAR;
                        font: ARRAY OF SYSTEM.WORD);

PROCEDURE set_attr     (no: INTEGER; val: INTEGER);
PROCEDURE get_attr     (no: INTEGER): INTEGER;

PROCEDURE set_color(color: INTEGER);
PROCEDURE set_back (color: INTEGER);

PROCEDURE reset; (* to initial state *)
```

```
PROCEDURE set_raw   (ON_OFF: INTEGER);
PROCEDURE set_awp   (ON_OFF: INTEGER);
PROCEDURE set_smooth(ON_OFF: INTEGER);
```

```
PROCEDURE set_scr   (no: INTEGER);

PROCEDURE action(no: INTEGER; SEQ args: SYSTEM.WORD);

PROCEDURE attach(dev_name: ARRAY OF CHAR);

END Terminal.
```

NOTES

```
PROCEDURE lock;
  Terminal automaticaly locked after any operation.
  All other processes will wait until 'unlock', that
  automaticaly performed after HALT, succesfull attach
  of other terminal or direct 'unlock' operation.

PROCEDURE unlock;
  Allow one of other tasks waiting for terminal to 'lock' it
  To prevent deadlocks for tasks needed Keyboard and Terminal
  together, we recomended (and follow this recomendation in
  (c) KRONOS software) to locks those devices in next order
      Keyboard.unlock; Terminal.unlock;

        (* here other tasks may lock Keyboard and (after first)
Terminal *)

      Keyboard.lock;   Terminal.lock;


(* NOW WE KNOW NEXT TYPES OF TERMINALS:

    1    Friashik "Electronika-EA-15-000-013"
   92    Labtam-3000 (something less then VT100)
  100    VT-100      (I never see it! Leo 10-Nov-89)
  197    Facit-2000A (something less then VT220)
  200    VT-200
  220    VT-220


PROCEDURE erase     (how: INTEGER);  (* Очищает экран *)
PROCEDURE erase_line(how: INTEGER);  (* Очищает строку *)
(* how:
        0 - очистить от текущей позиции до конца экрана/строки
        1 - очистить от начала экрана/строки до текущей позиции
        2 - очистить целиком экран/строку
*)

PROCEDURE home;   (* Перемещает курсор в верхний левый угол
экрана *)
PROCEDURE bottom; (* Перемещает курсор в верхний нижний угол
экрана *)
```

```
PROCEDURE roll_down(n: INTEGER);      (* Сдвиг экрана вниз *)
PROCEDURE roll_up  (n: INTEGER);      (* Сдвиг экрана вверх *)

PROCEDURE scroll_down(n: INTEGER);      (* Сдвиг экрана вниз *)
PROCEDURE scroll_up  (n: INTEGER);      (* Сдвиг экрана вверх *)

PROCEDURE set_pos(line: INTEGER; col: INTEGER); (*Установить
позицию курсора*)

PROCEDURE set_background(color: INTEGER);
(*
   FOR type=92:
     The BACKGROUND color is the color the screen is set to
     when any ERASURE operation is performed. But character
     background always BLACK!!!
*)


PROCEDURE attach(dev_name: ARRAY OF CHAR);
(* Смена терминала на терминал определенный драйвером
   с именем dev_name.
*)
```

В ДРАЙВЕРАХ ЖЕЛАТЕЛЬНО ВЫБИРАТЬ

something выделитель в таком порядке
(по отсутсвию возможностей)

REVERSE
COLOR
UNDERLINE
HIGH INTENSITY
LOW INTENSITY
FONT (ITALIC, GOTIC, FRANFURT)
BLINKING

```
DEFINITION MODULE Threads; (* Ned 20-Oct-89. (c) KRONOS *)

IMPORT  SYSTEM;
IMPORT  Signals;

TYPE THREAD;

VAL null: THREAD;        -- NIL for THREADs

-------------------------  THREADs  --------------------------
                           -----------

TYPE Forkee = PROCEDURE (SEQ SYSTEM.WORD);

PROCEDURE fork(VAR thread: THREAD;
                     proc: Forkee;
                work_size: INTEGER;
                 VAR done: BOOLEAN;
                 SEQ args: SYSTEM.WORD
                     );

PROCEDURE xfork(VAR thread: THREAD;
                      proc: Forkee;
                work_size: INTEGER;
                      halt: Signals.SIGNAL;
                 VAR done: BOOLEAN;
                 SEQ args: SYSTEM.WORD
                     );

PROCEDURE rem_thread(VAR thread: THREAD; VAR done: BOOLEAN);


-----------------------------------------------------------------

PROCEDURE abort(thread: THREAD; how: INTEGER);

PROCEDURE status(thread: THREAD; VAR n: INTEGER);
CONST
  invalid   = 0;
  ready     = 1;
  blocked   = 2;
  suspended = 3;
  aborted   = 4;

PROCEDURE history(thread: THREAD; VAR packed: ARRAY OF CHAR);

PROCEDURE cause(thread: THREAD; VAR n: INTEGER);

-----------------------------------------------------------------

PROCEDURE self(): THREAD;
```

```
PROCEDURE get_prio(thread: THREAD; VAR prio: INTEGER);
PROCEDURE set_prio(thread: THREAD;     prio: INTEGER);
```

DELAYs

```
PROCEDURE delay(milisec: INTEGER);

PROCEDURE suspend(thread: THREAD; milisec: INTEGER);
PROCEDURE resume (thread: THREAD);

(************************************************************


PROCEDURE fork(
               VAR thread: THREAD;
                     proc: Forkee;
                work_size: INTEGER;
                 VAR done: BOOLEAN;
                 SEQ args: SYSTEM.WORD
                     );
```

Создает и запускает новый процесс с параметрами args. done=FALSE, если не хватило памяти под процесс. При завершении процесса будет выполнена стандартная процедура завершения задачи.

```
PROCEDURE xfork(VAR thread: THREAD;
                      proc: Forkee;
                work_size: INTEGER;
                     halt: Signals.SIGNAL;
                 VAR done: BOOLEAN;
                 SEQ args: SYSTEM.WORD
                     );
```

Создает и запускает новый процесс с параметрами args. done=FALSE, если не хватило памяти под процесс. При завершении процесса будет послан сигнал halt.

```
PROCEDURE rem_thread(VAR thread: THREAD; VAR done: BOOLEAN);
```

Освобождения памяти, занимаемой процессом.

```
PROCEDURE abort(thread: THREAD; how: INTEGER);
```

Останавливает процесс(ы).

how:

0 - будет завершен только данный процесс;

1 - будет завершен данный процесс и его потомки;

```
PROCEDURE status(thread: THREAD; VAR n: INTEGER);
-------------------------------------------------
```

Выдает состояние процесса:

```
  invalid   -- что-то не в порядке;
  ready     -- активный или готовый к исполнению процесс;
  blocked   -- процессс ждет сигнала или блокирован на воротах;
  suspended -- задержанный процесс;
  aborted   -- завершенный процесс.
```

```
PROCEDURE history(thread: THREAD; VAR packed: ARRAY OF CHAR);
------------------------------------------------------------
```

Выдает историю завершенного процесса.

```
PROCEDURE cause(thread: THREAD; VAR n: INTEGER);
------------------------------------------------
```

Выдает причину завершения процесса (код ошибки).

```
PROCEDURE self(): THREAD;
-------------------------
```

Текущий процесс.

```
PROCEDURE get_prio(thread: THREAD; VAR prio: INTEGER);
PROCEDURE set_prio(thread: THREAD;     prio: INTEGER);
-----------------------------------------------------
```

Установка и определение приоритета процесса. Игнорируются в текущей реализации.

```
PROCEDURE delay(milisec: INTEGER);
----------------------------------
```

Приостановка текущего процесса на время. Пустое действие, если время задержки <=0.

```
PROCEDURE suspend(thread: THREAD; milisec: INTEGER);
---------------------------------------------------
```

Задерживает процесс на указанное время, а если время задержки <0, то процесс может быть продолжен только явным вызовом процедуры resume.

```
PROCEDURE resume(thread: THREAD);
---------------------------------
```

Восстановление задержанного процесса. Пустое действие, если процесс не задержан.

```
**********************************************************)

END Threads.
```

```
DEFINITION MODULE Time; (* Leo & Ned 27-Feb-90. (c) KRONOS *)

(* Модуль  осуществляет  службу времени в системе. *)

PROCEDURE time(): INTEGER;
PROCEDURE set_time(time: INTEGER);

PROCEDURE zone(): INTEGER;
PROCEDURE set_zone(zone: INTEGER);

TYPE UNIT = (tick,microsec,milisec,sec,minute,hour);

PROCEDURE sys_time(unit: UNIT): INTEGER;

PROCEDURE eval(from: UNIT; value: INTEGER; to: UNIT): INTEGER;

PROCEDURE delay(time_value: INTEGER; time_unit: UNIT);

PROCEDURE pack  (y,m,d,ho,mn,sc: INTEGER): INTEGER;
PROCEDURE unpack(t: INTEGER; VAR y,m,d,ho,mn,sc: INTEGER);

PROCEDURE day(time: INTEGER): INTEGER;

PROCEDURE scan_date(VAR time: INTEGER; str: ARRAY OF CHAR;
                        night: BOOLEAN);

END Time.

(************************************************************

   Модуль  осуществляет  службу времени в системе.
 Здесь всюду INTEGER  выражает  время  в  секундах, начиная
 с 00:00.00 01/01/1986 года.
   Отрицательное время свидетельствует об ошибке. Последнее
 допустимое время - 23:59.59 31/12/2017 года (1986+31).

PROCEDURE time(): INTEGER;
--------------------------
Выдает (зональное) время в секундах от начального.
Время по Гринвичу вычисляется как: time()-zone()*3600

PROCEDURE set_time(time: INTEGER);
----------------------------------
Устанавливает время, for SUPRUSER only

PROCEDURE zone(): INTEGER;
--------------------------
Выдает номер временно'й зоны (-11..+12)

PROCEDURE set_zone(zone: INTEGER);
----------------------------------
Устанавливает номер временно'й зоны (-11..+12).
```

```
Изначально до первого "set_zone" zone()=0 (Grinvich).

TYPE UNIT = (tick,microsec,milisec,sec,minute,hour);

PROCEDURE sys_time(unit: UNIT): INTEGER;
---------------------------------------
Выдает время, прошедшее с момента загрузки системы.

PROCEDURE eval(from: UNIT; value: INTEGER; to: UNIT): INTEGER;
-------------------------------------------------------------
Переводит время из одних единиц в другие

PROCEDURE delay(time_value: INTEGER; time_unit: UNIT);
-----------------------------------------------------
Задерживает обратившийся процесс

PROCEDURE pack(y,m,d,ho,mn,sc: INTEGER): INTEGER;
------------------------------------------------
По  году "y", месяцу "m", дате "d", часу "ho", минутам "mn"
и секундам "sc" выдает время.

PROCEDURE unpack(t: INTEGER; VAR y,m,d,ho,mn,sc: INTEGER);
---------------------------------------------------------
По  времени  "t"  присваивает значения переменным году "y",
месяцу  "m",  дню  "d", часу "ho", минутам "mn" и секундам
"sc".

PROCEDURE day(time: INTEGER): INTEGER;
-------------------------------------
По времени выдает номер дня недели (1 Monday ... 7 Sunday)

PROCEDURE scan_date(VAR time: INTEGER; str: ARRAY OF CHAR;
------------------    night: BOOLEAN);
str ::=   [DD/MN/[19]YY,][HH:MM[.SS]]
        | [[19]YY#DD#MM,][HH:MM[.SS]]
if date ommited if suggested TODAY
   night = FALSE: if HH:MM.SS ommited if suggested 00:00.00
   night = TRUE:  if HH:MM.SS ommited if suggested 23:59.59

   scan_date(t,"01/01/86,00:00.00",FALSE) => t=00000000h

*************************************************************)
.PAGE
```

```
DEFINITION MODULE tskArgs; (* Ned   27-Sep-89. (c) KRONOS *)
                           (* Hady. 14-Oct-89. (c) KRONOS *)

(* Стандартный разбор командной строки *)

VAL words: DYNARR OF STRING;

PROCEDURE del_word(i: INTEGER);

PROCEDURE pack_words(from,to: INTEGER);

PROCEDURE flag(prefix,f: CHAR): BOOLEAN;

PROCEDURE string(name: ARRAY OF CHAR; VAR s: STRING): BOOLEAN;

PROCEDURE number(name: ARRAY OF CHAR; VAR n: INTEGER): BOOLEAN;

PROCEDURE dispose;

PROCEDURE scan_string(str: ARRAY OF CHAR;
                      equ: BOOLEAN;
                      wds: BOOLEAN;
                      esc: BOOLEAN;
                      pre: ARRAY OF CHAR;
                      sep: ARRAY OF CHAR);

(*************************************************************

----------------------  ПРИМЕЧАНИЯ  -------------------------
                         --------------

Формат командной строки "shell" состоит из трех частей:
  [ информация для "shell" о способе запуска задачи ]
  [ имя запускаемой задачи ]
  [ параметры, передаваемые задаче при запуске ]

Данный модуль обеспечивает предварительный разбор третьей части
командной строки - параметров задачи.

Параметры задачи бывают трех видов: СЛОВА, УРАВНЕНИЯ и ФЛАГИ.

СЛОВО - непустая последовательность символов, отличных от
        пробела.
УРАВНЕНИЕ - пара: (имя + значение), где имя - СЛОВО, а значение -
        произвольная строка.
ФЛАГ -  пара: (префикс флага + ассоциированный символ).
        Разрешенный набор префиксов передается параметром
        процедуре, разбирающей строку параметров задачи (см.
ниже).
        Слово, начинающееся с одного из префиксов флагов,
        задает множество флагов с данным префиксом и
ассоциированных
```

с символами, находящимися в слове после символа-префикса.

При инициализации модуля строка параметров задачи разбирается стандартным образом вызовом процедуры scan_string (см.ниже):

```
    scan_string(parm_string,
                    equ=TRUE,
                    wds=TRUE,
                    esc=TRUE,
                    pre="+-",
                    sep='""'"'"');
```

В дальнейшем собранные множества могут быть пересобраны явным вызовом данной прцедуры с параметрами, определенными пользователем, а также с ними возможны следующие операции:

ФЛАГИ:

```
  PROCEDURE flag(prefix,f: CHAR): BOOLEAN;
  ----------------------------------------
```

Выдает значение TRUE, если в множестве флагов существует множество флагов, заданных символом prefix и в нем есть флаг, ассоциированный с символом "f".

СЛОВА:

```
  VAL words: DYNARR OF STRING;
  ----------------------------
```

Массив содержит все выделенные при разборе строки слова.

```
  PROCEDURE del_word(i: INTEGER);
  -------------------------------
```

Удаляет из массива words слово с индексом i.

```
  PROCEDURE pack_words(from,to: INTEGER);
  ---------------------------------------
```

Если "from" и "to" находятся в пределах {0..HIGH(words)}, то собирает все words[i] для i=from..to в одну строку, разделяя их пробелами и удаляя из массива "words", и помещает результат в "words[from]".

УРАВНЕНИЯ:

```
  PROCEDURE string(name: ARRAY OF CHAR; VAR s: STRING): BOOLEAN;
  --------------------------------------------------------------
```

Ищет уравнение с именем "name". Если находит, то копирует его значение в "s" и возвращают TRUE. Уравнение ищется сначала среди уравнений, выбранных при разборе командной строки, если не находится, то делается попытка найти его в текстовом окружении задачи (см. Environment.d)

```
  PROCEDURE number(name: ARRAY OF CHAR; VAR n: INTEGER): BOOLEAN;
  ---------------------------------------------------------------
```

Аналогично string, кроме того преобразует

```
  строку в число (если можно).

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ:

  PROCEDURE release;
  ------------------
  Освобождение всей занятой памяти


  PROCEDURE scan_string(str: ARRAY OF CHAR;-- разбираемая строка
  ------------------------------------------------------------
                        equ: BOOLEAN;      -- обработка равенств
                        wds: BOOLEAN;      -- обработка слов
                        esc: BOOLEAN       -- вкл/выкл "\"
                        pre: ARRAY OF CHAR;-- префиксы флагов
                        sep: ARRAY OF CHAR -- пары разделителей
                       );
  Очищает текущее состояние всех множеств и разбирает
  строку "s" в соответствии с остальными параметрами.

  СИНТАКСИС:
    str = { {" "} term {" "} } .

    term = word | equation | flags .
    word = charF { charN } .
    equation = name "=" body .
    flags = prefix { char } .

    name = charF { charN } .
    body = charB { char } | sep0 { charS } sep1 .
    prefix = один из символов "pre".

    IF esc=TRUE THEN
      char  = anychar | "\ " | "\\" .
      charF = anychar кроме prefix | "\ " | "\\" | "\"prefix .
      charN = anychar кроме "=" | "\ " | "\\" | "\=" .
      charB = anychar кроме sep0 | "\ " | "\\" | "\"sep0 .
      charS = anychar кроме СООТВЕТСТВУЮЩЕГО sep1 | "\\" |
"\"sep1 .
    ELSE
      char  = anychar .
      charF = anychar кроме prefix .
      charN = anychar кроме "=" .
      charB = anychar кроме sep0 .
      charS = anychar кроме СООТВЕТСТВУЮЩЕГО sep1 .
    END

    sep0    = символ sep[i*2]   для некоторго i IN {0..HIGH(sep)
DIV 2}
    sep1    = символ sep[i*2+1] для того же   i, что и в sep0.
    anychar = любой символ, кроме " " или 0c.

  ПРИМЕЧАНИЯ:
```

Параметр "sep" состоит из пар (sep0,sep1) и ставит в соответствие открывающему разделителю закрывающий. Данная строка может быть пустой. Если количество символов в ней до 0c или HIGH нечетно, то sep0=sep1=последнему символу.

Разбор строки состоит в последовательно выполняемых операциях ОБРАБОТКИ ФЛАГОВ, ОБРАБОТКИ УРАВНЕНИЙ, ОБРАБОТКИ СЛОВ.

ОБРАБОТКА ФЛАГОВ (pre#"").

1. Выделяет и УДАЛЯЕТ(!) из строки все слова, начинающиеся с символов, заданных строкой "pre" и формирует множество флагов для каждого символа из "pre".
   Например при "pre" = "+-" :
   "+abc" - флаги { (+,a), (+,b), (+,c) } ;
   "-abc" - флаги { (-,a), (-,b), (-,c) } ;
   "-+-" - флаги { (-,+), (-,-) } ;
   "++-" - флаги { (+,+), (+,-) } ;

   Если "esc"=TRUE, и слово начинается с символа "\" то данное слово не считается набором флагов даже в том случае если символ "\" находится в строке "pre". (см ОГРАНИЧЕНИЯ)

   В этом режиме комбинация "\ " заменяется на " " и пробел в этом случае не считается признаком конца набора флагов. Комбинация "\\" заменяется на "\".
   Например :
   "+a\ +\\" дает флаги { (+,a), (+," "), (+,+), (+,\) }
   "\+ab\ +" не является набором флагов и превращается в слово "+ab +".

ОБРАБОТКА УРАВНЕНИЙ (equ=TRUE).

2. Если задана обработка уравнений (equ=TRUE), то после сбора флагов собирает и УДАЛЯЕТ(!) из строки все комбинации вида name=body, где name - слово, (то есть последовательность любых символов кроме пробела), а body - это слово, или ограниченная разделителями последовательность любых символов. Строка sep содержит пары (левый,правый) допустимых разделителей.
   Например при sep = "[]"
   "abc=def" дает равенство ("abc","def") ;
   "abc=[000 111] дает равенство ("abc","000 111") ;
   при sep = ""
   "abc=def" дает равенство ("abc","def") ;
   "abc=[000 111] дает равенство ("abc","[000") и слово "111]";

```
      Если esc=TRUE то комбинация "\=" во всех позициях
      слова кроме первой обозначает символ "=", и не
      является признаком уравнения. "\ " обозначает " "
      и не является признаком завершения имени или значения
      уравнения, "\\" обозначает "\".
      Пример:
       "abc\=de\\f"  дает слово      "abc=de\f".
       "ab\ c=de\ f" дает уравнение ("ab c","de f").
      При использовании пар разделителей (sep0,sep1)
      комбинация "\"sep0 сразу за символом "=", обозначает
      sep0 и значение уравнения считается не строкой, а
      словом. Если же значение начинается с символа sep0,
      то в значении комбинация "\"sep1 обозначает sep1
      и не является признаком окончания значения.
      Пример (sep="[]", esc=TRUE):
       "abc=\[def"          дает уравнение ("abc","[def") ;
       "abc=[def [\] ijk]" дает уравнение ("abc","def [] ijk") ;

  СЛОВА (wds=TRUE).
      Выделяет все слова, и формирует массив слов. Иначе
      все, что осталось в строке параметров после извлечения
      равенств и флагов заносится в строку words[0].
      Если esc=TRUE то комбинация "\=" во всех позициях
      слова кроме первой обозначает символ "=", и не
      является признаком уравнения. "\ " обозначает " "
      и не является признаком завершения слова. Комбинация
      "\\" обозначает "\".

  ОГРАНИЧЕНИЯ:
      Если "esc"=TRUE, то символ "\" в строках pre, sep
      игнорируется.
      Если "equ"=FALSE, то символ "=" теряет свое специальное
      значение ("abc=def" обрабатывается как СЛОВО).


**************************************************************)


END tskArgs.
```

```
DEFINITION MODULE tskEnv; (* Ned 28-Sep-89. (c) KRONOS *)

IMPORT  SYSTEM;

TYPE WORDs = DYNARR OF SYSTEM.WORD;

VAL  id: INTEGER;       -- номер задачи
   done: BOOLEAN;
  error: INTEGER;

-----------------------------------------------------------------

CONST -- имена стандартных атрибутов в окружении задачи

  args    = 'ARGS';       -- строка параметров задачи
  cd      = 'CD';         -- имя текущей директории
  cmask   = 'CMASK';      -- маска защиты создаваемых файлов
  name    = 'NAME';       -- имя задачи
  etc     = 'ETC';        -- путь поиска всяких файлов
  bin     = 'BIN';        -- путь поиска кодофайлов
  sym     = 'SYM';        -- путь поиска симфайлов
  ref     = 'REF';        -- путь поиска реффайлов
  key     = 'KEY';        -- имя драйвера клавиатуры
  tty     = 'TTY';        -- имя драйвера терминала (вывод)
  screen  = 'SCR';        -- имя драйвера экрана
  info    = 'INFO';       -- информационная строка задачи
  msg     = 'MSG';        -- сообщения об ошибках
  stk     = 'STK';        -- размер стека задачи
  prompt  = 'PROMPT';     -- приглашение командной строки
  chain   = 'CHAIN';      --
  lp      = 'LP';         -- имя драйвера принтера
  ccdIN   = 'CCDIN';      -- имя драйвера для Control Coordinate
Device'a
  ccdOUT  = 'CCDOUT';     -- имя драйвера для Control Coordinate
Device'a
  plot    = 'PLOT';       -- имя драйвера для Plotter'a
  wndman  = 'WNDMAN';     -- имя Window Manager'a

PROCEDURE get_str(name: ARRAY OF CHAR; VAR s: STRING);
PROCEDURE get_env(name: ARRAY OF CHAR; VAR w: WORDs );

PROCEDURE put_str(name: ARRAY OF CHAR;
                  data: ARRAY OF CHAR;
                  priv: BOOLEAN);
(* Только значащая часть строки (до 0c) *)

PROCEDURE put_env(name: ARRAY OF CHAR;
                  data: ARRAY OF SYSTEM.WORD;
                  priv: BOOLEAN);

-----------------------------------------------------------------
```

```
PROCEDURE final(p: PROC);

PROCEDURE ipr(): BOOLEAN;

PROCEDURE become_ipr;

PROCEDURE exit(res: INTEGER);

-------------------------------------------------------------

PROCEDURE   allocate(VAR a: SYSTEM.ADDRESS;     size: INTEGER);
PROCEDURE deallocate(VAR a: SYSTEM.ADDRESS;     size: INTEGER);
PROCEDURE reallocate(VAR a: SYSTEM.ADDRESS; VAR high: INTEGER;
                                    len,el_byte_size: INTEGER);

END tskEnv.
```

```
DEFINITION MODULE visCode; (* Ned 28-Sep-89. (c) KRONOS *)

(* Визуализация кода и системы команд *)

IMPORT  SYSTEM;

VAL done: BOOLEAN;
   error: INTEGER;

TYPE
  INTs     = DYNARR OF INTEGER;
  GLOBAL   = RECORD ofs,size: INTEGER END;
  GLOBALs  = DYNARR OF GLOBAL;
  ext_ptr  = POINTER TO
               RECORD
                 time: INTEGER;
                 name: ARRAY [0..31] OF CHAR;
               END;
  EXTs     = DYNARR OF ext_ptr;

  code_ptr = POINTER TO code_rec;
  code_rec = RECORD
               vers: INTEGER;   -- версия кода
           compiler: STRING;    -- название компилятора
           def_time: INTEGER;   -- время компиляции определения
           imp_time: INTEGER;   -- время компиляции реализации
             no_glo: INTEGER;   -- число глобалов
          add_stack: INTEGER;
          min_stack: INTEGER;
                tag: INTEGER;

               name: STRING;
            strings: STRING;    -- строковый и структурный пул
            structs: INTs;      -- строковый и структурный пул
           proc_tab: INTs;      -- процедурная таблица
               code: STRING;    -- команды
          multi_glo: GLOBALs;   -- мульти-глобалы
               exts: EXTs;      -- внешние модули
             END;


PROCEDURE connect(VAR code: code_ptr;
                  VAR base: ARRAY OF SYSTEM.WORD);
(* Распаковка атрибутов и разделов кода *)

PROCEDURE disconnect(VAR code: code_ptr);
(* Освобождает память, занятую процедурой connect *)

------------------------  COMMANDs  --------------------------
                         ------------

VAL cmd_len: ARRAY [0..255] OF CHAR; -- длины команд
```

```
PROCEDURE vis_command(n: INTEGER; VAR s: ARRAY OF CHAR);
(* Выдает мнемонику команды *)

(*************************************************************

PROCEDURE connect(
-----------------
                  VAR code: code_ptr;
                  VAR base: ARRAY OF SYSTEM.WORD);
```

base определяет местонахождение и размер кодофайла в памяти. Процедура распаковывает атрибуты кода и устанавливает указатели (дин. массивы) на части кода.

ВНИМАНИЕ:

Память занятую кодом нельзя освобождать до конца работы с кодом.

ОШИБКИ:

```
        no_memory - не хватило памяти;
        ill_vers  - некорректная версия кода.

PROCEDURE disconnect(VAR code: code_ptr);
------------------
```

Освобождает не только дескриптор кода, но и некоторую дополнительную память, которую пришлось занять при распаковке.

```
PROCEDURE vis_command(n: INTEGER; VAR s: ARRAY OF CHAR);
----------------------
```

Выдает 6-символьную мнемонику команды.

```
*************************************************************)

END visCode.
```

```
DEFINITION MODULE xRef; (* Ned 29-Apr-89. (c) KRONOS *)

IMPORT  sym: coolSym;

VAL done: BOOLEAN;
   error: INTEGER;
   note : ARRAY [0..63] OF CHAR;

-----------------------------------------------------------------

TYPE
  PROJECT;
  cu_ptr      = POINTER TO cu_rec;        -- compilation unit
  context_ptr = POINTER TO context_rec;
  obj_ptr     = POINTER TO obj_rec;
  type_ptr    = POINTER TO type_rec;


TYPE
  TYPEs       = DYNARR OF type_ptr;
  OBJs        = DYNARR OF obj_ptr;
  EXTs        = DYNARR OF cu_ptr;

--------------------------  TYPEs  --------------------------
                           ---------

TYPE
  enum_ptr  = POINTER TO enum_rec;
  enum_rec  = RECORD
                id  : INTEGER;
                val : INTEGER;
                next: enum_ptr;
              END;

  parm_ptr  = POINTER TO parm_rec;
  parm_rec  = RECORD
                type: type_ptr;
                tags: BITSET;
                ofs : INTEGER;
                var : obj_ptr;  -- соответствующий локал
                next: parm_ptr;
              END;

  type_rec  = RECORD
                id   : INTEGER;
                modno: INTEGER;
                base : type_ptr;
                CASE mode: INTEGER OF
                  | sym.range   :  min,max: INTEGER;
                  | sym.array   :  inx    : type_ptr;
                  | sym.record  :  fields : obj_ptr;
                                   size   : INTEGER;
```

```
                  | sym.dynarr  :  dim    : INTEGER;
                  | sym.enumtype:  consts : enum_ptr;
                  | sym.proctype:  parms  : parm_ptr;
                END;
              END;

-------------------------  OBJECTs  -------------------------
                           ----------

TYPE
  xpos_ptr    = POINTER TO xpos_rec;
  xpos_rec    = RECORD
                  pc  : INTEGER;
                  line: INTEGER;
                  col : INTEGER;
                  next: xpos_ptr;
                END;

TYPE
  obj_rec     = RECORD
                  id   : INTEGER;
                  tags : BITSET;
                  type : type_ptr;
                  scope: INTEGER;
                  ofs  : INTEGER;
                  next : obj_ptr;
                  CASE :BOOLEAN OF
                    |FALSE: locs : context_ptr;
                                   (* for proc's & modules *)
                    |TRUE : parm : parm_ptr;
                                   (* for var *)
                  END;
                END;

  context_rec = RECORD
                  procs: obj_ptr;
                  mods : obj_ptr;
                  vars : obj_ptr;
                  cons : obj_ptr;
                  xpos : xpos_ptr;    -- only for proc
                END;

  cu_rec      = RECORD
                  complete: BOOLEAN;
                  unit    : INTEGER;  -- kind of compilation unit
                  def_time: INTEGER;  -- def time
                  imp_time: INTEGER;  -- imp time
                  exts    : EXTs;
                  types   : TYPEs;
                  proc_tab: OBJs;     -- procno -> obj_ptr
                  locs    : context_ptr;
                  names   : STRING;   -- names[0..] - cu name
                  language: INTEGER;  -- index of language
```

```
                                        -- name in names
            END;

-------------------------------------------------------------

VAR main: PROJECT;   -- список всех модулей

PROCEDURE new    (VAR pro: PROJECT);
PROCEDURE release(VAR pro: PROJECT);
(* Удаляет все модули, освобождает всю память. *)

PROCEDURE read_cu(VAR pro: PROJECT;
                  VAR  cu: cu_ptr;
                     name: ARRAY OF CHAR;
                      ref: BOOLEAN);
(* Чтение и разбор файла для модуля с именем name *)

PROCEDURE enter_cu(VAR pro: PROJECT;
                   VAR  cu: cu_ptr;
                      name: ARRAY OF CHAR;
                      text: ARRAY OF CHAR);
(* Разбор симфайла из text для модуля с именем name *)

PROCEDURE exit_cu(cu: cu_ptr);
(* Удаляет информацию о модуле. *)

PROCEDURE text_pos(xpos: xpos_ptr; pc: INTEGER;
                         VAR line,col: INTEGER);
(* Выдает текстовую позицию *)

PROCEDURE id_str(names: ARRAY OF CHAR;
                   id: INTEGER;
              VAR name: ARRAY OF CHAR);
(* Выдает имя по иденту *)

(*************************************************************

PROCEDURE read_cu(
-----------------
                 VAR pro: PROJECT;
                 VAR  cu: cu_ptr;
                    name: ARRAY OF CHAR;
                     ref: BOOLEAN);
```

Ищет модуль с именем name в проекте. Если не находит то ищет

Ищет модуль с именем name в проекте. Если не находит то ищет
файл с именем name.sym или name.ref (если ref=TRUE). то читает реффайл, иначе симфайл. Поиск файла идет соответственно по путям поиска реффайлов или симфайлов.

```
ОШИБКИ:
        файловые ошибки;
        ill_vers        - некорректная версия файла;
```

```
                        note - содержит сообщение.
        inconsistency   - контроль времени компиляции;
                          при этой ошибке построение структуры
                          единицы компиляции не прекращается.
                          note - содержит сообщение о первом
                          конфликте при разборе этого модуля.


PROCEDURE enter_cu(
------------------
                  VAR pro: PROJECT;
                  VAR  cu: cu_ptr;
                     name: ARRAY OF CHAR;
                     text: ARRAY OF CHAR);
```

Ищет модуль с именем name в проекте. Если не находит то разбирает симфайл из text для модуля с именем name.

```
ОШИБКИ:
        ill_vers        - некорректная версия файла;
                          note - содержит сообщение.
        inconsistency   - контроль времени компиляции;
                          при этой ошибке построение структуры
                          единицы компиляции не прекращается.
                          note - содержит сообщение о первом
                          конфликте при разборе этого модуля.

PROCEDURE exit_cu(cu: cu_ptr);
-----------------
```

Удаляет информацию о модуле. Дескриптор cu остается (NOT cu^.complete).

```
PROCEDURE new(VAR pro: PROJECT);
```

Создает новый проект. Проект main создается при инициализации модуля.

```
PROCEDURE release;
-----------------
```

Удаляет все модули из проекта, освобождает всю память.

```
PROCEDURE text_pos(
------------------
                   xpos: xpos_ptr; pc: INTEGER;
                         VAR line,col: INTEGER);
```

Выдает текстовую позицию по pc и нужному списку позиций (xpos).

```
usage:
       text_pos(cu^.ptab[proc_no]^.locs^.xpos,pc,line,col);
```

```
PROCEDURE id_str(
-----------------
                  names: ARRAY OF CHAR;
                     id: INTEGER;
               VAR name: ARRAY OF CHAR);
```

Выдает имя объекта по иденту.

```
usage:
       id_str(cu^.names^,id,name);
```

Стандарт симфайла в системе определяет модуль coolSym. Этот модуль содержит описание структуры симфайла. Данный модуль использует константы определенные в модуле coolSym. А именно:

- номера типов (type_rec.mode);
- номера битов в множестве признаков (obj_rec.tags);
- виды единиц компиляции (cu_rec.unit).

Основным понятием в образе реф-файла является единица компиляции (cu_ptr). Единица компиляции (ЕК) может не содержать информации о модуле (если NOT complete) и в этом случае единственной информацией о ней является имя модуля, в массиве names. Получить информацю о ЕК можно с помощью процедур read_cu и enter_cu. После прочтение ЕК содержит:

1. образ импорта - массив ссылок на внешние модули (exts). По традиции в нулевом элементе массива лежит ссылка на себя.
2. массив типов модуля (types). В этом массиве собраны все типовые значения объявленные в модуле (без стандартных типов), как именнованые, так и анонимные. У именнованых типов выставлены поля id>=0 и mod_no - номер модуля в списке внешних, в котором впервые объявлен этот тип.
3. процедурная таблица (proc_tab). Содержит указатели на все процедуры в ЕК (в том числе на нулевую).
4. массив имен (names). Содержит все имена объектов ЕК, в том числе и имена импортированных внешних объектов, кроме имен самих импортируемых ЕК.
5. указатель на контекст (locs). Все объекты реф-файла попадают в контекст области действия, в которой они были описаны. Заметим, что объектами не являются типы и константы (кроме структурных).
6. Вид единицы компиляции (def,imp,prog).
7. Время компиляции определяющего модуля (для программного модуля - совпадает с временем компиляции модуля) и время компиляции реализации (для определяющего модуля = -1).

ЗАМЕЧАНИЕ:
После чтения ЕК все массивы содержат только значимые элементы, т.е. они обрезаны.

Для объектов:

Пара (scope,ofs) определяет размещение переменной, процедуры или константы. Scope - номер уровня вложенности (если >=0) и номер внешнего модуля, если <=0 (т.е. номер в массиве exts). Ofs - номер процедуры, номер переменной (локальной или глобальной) и смещение структурной константы в строковом пуле.

ПРИМЕР:

```
MODULE main;

IMPORT slave;

MODULE module;

PROCEDURE proc;
  VAR local: INTEGER;
END proc;

END module;

VAR global: INTEGER;

END main.
```

```
cu:
   exts : длина 2, содержит ссылку на себя и
          ссылку на cu_ptr для slave.
   types: длина 1, содержит ссылку на анонимный
          процедурный тип (для proc).
   names: содержит: main Modula-X module proc local global
          в не знаю каком порядке, что неважно. Поле id
          для объектов содержит индекс начала имени
          в массиве names.
   ptab : длина 2, содержит процедуры 0 и 1 (proc).
   locs :
          vars : global
          procs: NIL
          mods : module ^.locs:
          cons : NIL        vars : NIL;
                            procs: proc   ^.locs:
                            mods : NIL        vars : local
                            cons : NIL        procs: NIL
                                                 mods : NIL
                                                 cons : NIL
```

Примечание:

Так как модуль достаточно универсален, то в его использование надо быть достаточно осторожным. Например, вместо реф-файла можно читать сим-файл. При этом естественно все контексты пусты (o^.locs=NIL).

```
Вместо
  x:=ptab[n]^.locs^.xpos;
необходимо использовать:
  p:=ptab[n];
  IF (p=NIL) OR (p^.locs=NIL)  THEN .....; RETURN END;
  x:=p^.locs^.xpos;
```

Компилируйте модули использующие данный с ключом -N (опцией $N+), которая включает контроль указателей на NIL.

```
*****************************************************************)

END xRef.
```