LOGLAN'82

USER'S GUIDE

Institute of Informatics
University of Warsaw
January 1988

revised October 1994
LITA
Université de Pau

revised April 2011
UKSW

TABLE of CONTENTS

PREFACE

This document provides information necessary to compile and execute Loglan programs.

This manual assumes basic knowledge of Loglan-82 language, described in "Report on the Loglan Programming Language" (see Bibliography).

1. USING LOGLAN-82 SYSTEM

The following three steps are required to execute a Loglan program:

- Compilation (to intermediate code), Command: loglan
- Generation of the interpreted code (from intermediate code), Command: gen
- Interpretation (i.e. execution of program). Command: int

Compilation is accomplished by invoking Loglan compiler. This step creates two destination files: the intermediate code file(.lcd) and the listing file(.lst). The intermediate code file is the input file for the second step: generation of the code accepted by interpreter. In this step two files containing object code(.pcd & .ccd) are produced. They are the input files for the third step: interpretation. This step is equivalent to execution of a program.

1.1. COMPILATION

To invoke the Loglan compiler without specifying any command line parameters, type:

LOGLAN

Then the prompt appears on your terminal:

File name:

and the compiler waits for file specification.The default extension is LOG.

The compiler will produce (optionally) listing file with the same file name and the extension LST and will produce, if no error occurs, the code file with the extension LCD. Destination files will be stored on the same drive and directory as the source file.

Examples:

$ LOGLAN

File name: PROGRAM <ENTER>

Loglan compiler compiles program from PROGRAM.LOG file and creates PROGRAM.LCD.

$ LOGLAN A:PROGRAM.DAT

In this case the source file is A:PROGRAM.DAT. The file PROGRAM.LCD will be created on drive A.

$ LOGLAN /home/vous/PROGRAM2

If any error occurs, the code file is not produced. At the end of compilation the following message is printed:

1.2. COMPILER SWITCHES

There are two possibilities to specify compiler's options: by compiler switches (i.e. external options) or by comments in the source program (see chapter 2.). You may enter the compiler switches in command line after file name in the following format:

where swi consists of character that designates the name of the option and either '+' or '-'.

Examples:

$ LOGLAN PROGRAM L- T+

$ LOGLAN PROGRAM

In this case the default switches values are assumed.

Scope of the switch is the entire program. All switches ,except H, correspond to options. A switch has greater priority then options: when you specify switch, all corresponding options inside source program will be ignored. Full description of each option is given in chapter 2.2. Switch L has additional significance. When this switch is set off no listing file is produced.

1.3. CODE GENERATION

In this step information from the intermediate code file is read and two destination files containing the code are produced. No switch is permitted for this step. To generate code files, type:

GEN <file name>

You type file name without extension (extension is ignored).

Examples:

$ GEN
FILE_NAME: PROGRAM

Information is read from file PROGRAM.LCD from default drive and directory. Two destination files are produced: PROGRAM.CCD and PROGRAM.PCD and stored in the same directory as the input file.

$ GEN /home/vous/PROGRAM2

Files PROGRAM.CCD and PROGRAM.PCD are stored on drive A. ♥

1.4. PROGRAM EXECUTION

To interprete (execute) the Loglan program you must invoke the interpreter INT or HINT (if tNe switch H+ was specified). File name must be specified in command line. The file extension is ignored. The interpreter reads input files with the given name and extensions CCD and PCD and executes the Loglan program.

The syntax for calling the interpreter is

INT <options> <file name>

The following options are supported:

/m < n >	set memory size for Loglan program (in 16 bit words for small and 32 bit words for huge memory). For concurrent programs it means memory size for every process.
/i	information about garbage collection-compactification is printed.
/r < n >	used to invoke interpreter on nodes different from console (see 3.4.). option parameter is console node number (as defined by D-Link Network).
/d	causes trace to be printed to the file with .TRD extension provided that the option or switch D+ was used during compiling.

At the end of interpretation the following message is printed:

End of LOGLAN-82 program execution

Examples:

```
$   LOGLAN \DAT\EXAMP.SRC, L+
```

The file \DAT\EXAMP.LCD and \DAT\EXAMP.LST are generated.

```
$ GEN  \DAT\EXAMP
```

The files \DAT\EXAMP.CCD and \DAT\EXAMP.PCD are created. ♣

Then the program can be interpreted by:

```
$ INT  \DAT\EXAMP
```

1.5. COMPILE TIME ERRORS

The errors detected during the compilation are printed on the listing file, if this file is created. In the scope of option L- or if the switch L is set off only the incorrect lines and errors messages are printed . When the switch (not option !) L is set off then listing file is not produced and incorrect lines and error messages are printed on the user's terminal.

Error message has the following format:

*** ln ERROR en txt id

where:

ln - index of incorrect line,
en - error's number (see Appendix B),
txt- text that explain type of the error,
id - identifier helpful to situate the error.

Error messages are printed in the source listing after incorrect lines.
For syntax errors (numbered 101-147, 201-212), sign '?' indicates the error's position in the line.
Error may be detected beyond the line containing it.
Identifier helpful to find an error is printed as soon as possible.

For codes 331-338 error message is printed after first line of virtual module declaration.
Errors like "undeclared identifier" are printed in each module once, after first reference to this identifier. Further references are ignored.
The errors related to case instruction may appear before the incorrect line.

1.6. RUN-TIME ERRORS

Loglan run-time errors are detected by Loglan run-time system. When any of these errors occurs, the appropriate system signal is raised and error message is printed if handler is not found. All of these error messages are described in Appendix C. moreover the line number of the last executed statement is printed on the user's terminal.

2. COMPILER OPTIONS

Options, like switches are used to pass some information to the compiler. Options are placed in source program in comments. Scope of options in source program is textual. Option may appear in any place of source program, but it is active from the beginning of the nearest instruction. Listing option L is active from the next line after line containing setting this option on up to the line containing setting this option off. Options overwrite defaults, but are overwritten by switches (external options). Option definition is not allowed before the keyword program.

2.1. OPTION FORMAT

Options may be placed in source program in comments in the following format:

(*$opt1,opt2,...*)

where opti consists of character that designates the option and either '+' or '-' e.g.: (*$L-,T+*). Options in one comment should be separated by commas. Spaces in such comment are not allowed.

2.2. OPTIONS LIST

D - trace
  D+ - causes the line numbers of the executed instruction to be printed,
  D- - default,
L - listing
  L- - default, only incorrect lines are printed on the terminal
  L+ - all lines are printed on the listing file
O - optimization
  O+ - optimization of some arithmetical and logical expressions are included to generated code (default),
  O- - generate code without optimization,
T - type conflict checking
  T+ - default, dynamic checking of type conflict in assignment instructions and in parameter transmissions,
  T- - no dynamic checking

2.3. FRAGMENTATION

It is possible to split one Loglan program into different files. The preprocessor puts together the fragments of a program coming from different files and enables in this way the a compilation and, later, an execution of the entire Loglan source.

In any place of Loglan text where it is possible to put semicolon (;) you can put aside the following compiler directive

#include <*file*>

It means that any two declarations of instructions can be separated by the directive. Do not forget however to end the preceding declaration or instruction by the semicolon.

Grammar

#include "{path>}<file name>" <CR> CR stands for end of line Carriage Return

Remarks

1. It may be a white space between the word 'include' and the character ".
2. You need not to specify the path to the included file if it is stored in the current directory, i.e. the one from which you began the compilation.
3. The ortography of path should correspond to the plateform used (i.e. in DOS you use \ characters, in Unix it will be / character)

Example

```
program pr;
  var x: real;
#include c:\loglan\examples\simulation.log
```

```
   unit c: class;
   end c;
begin
   read(x);
   ...
end
```

The content of the file c:\loglan\examples\simulation.log replaces the line include.

3. CURRENT LOGLAN-82 IMPLEMENTATION SPECIFICATION

3.1. IMPLEMENTED SUBSET OF LOGLAN

The following constructions described in the report of Loglan-82 have not been implemented:
- local attributes,
- separate compilation,

File system is described in 3.3.

3.2. PREDEFINED LANGUAGE ELEMENTS

Predefined constants, procedures and functions are added to the language (see Appendix A). Moreover keywords char (short form of character) and bool (short form of boolean) are added.

The character set, defined in the report of Loglan-82, is extended by lower-case letters and the tabulation character (decimal code 9). It is possible to use operator '<>' which stands for 'not equal'.

3.3. FILE SYSTEM

Loglan contains the predefined reference type file and a set of statements and standard procedures to manipulate files.

Both sequential and direct access files are implemented.

3.4. FILE VARIABLES

Variables of the type file can be declared in the Loglan program and can be used as any variables of a reference type.

Example:

```
var f:file,
    A:arrayof file;

unit p:procedure(f:file); ... end p;
begin
    ......
    f := A(i);
    ......
end;
```

3.4.1. FILE GENERATION

A file object is generated by open statement of the form:

open(f,T,A) for external files

where
f is a file variable,

T = text for text files,
char for binary sequential files of character,
integer integer or
real real values
direct for direct access binary files.

A is an expression of the type arrayof char designating external file name. After execution of open statement the new file object is created and it becomes a value of the file variable f. If the file is opened as an external one, then it references to the file A.

Example:

open(data,text) - new internal text file data is opened
open(num ,integer) - new internal binary file num is opened
(the file components are integer numbers)
open(f,text,unpack("my.dat")) - external text file f is opened;
it references to the file my.dat stored on the default drive and directory.
open(f,direct,A) - an external direct access file with name in array A is opened.

3.5. FILE DEALLOCATION

The file can be closed and deallocated by execution of the statement kill.

3.6. GENERAL FILE OPERATIONS

There are three standard procedures associated with files: RESET, REWRITE and UNLINK.

call RESET(f) rewinds the file f. After execution of RESET on sequential files only read/get operations are available.

call REWRITE(f) creates a new empty file. After execution of REWRITE on sequential files only write/put operations are available.

call UNLINK(f) closes and deletes file f. File object is deallocated and f is set to one.

RESET or REWRITE must be performed on the file opening before the first I/O operation on it.

3.7. TEXT FILES

The following operations are available to text files: read, readln, eoln, write, writeln, eof. The

first parameter of the operation is a file variable. If it is omitted, then a standard input/output file assigned to user's terminal is used.

Example:

```
read(f,a,b);
read(c);
writeln(g," .... ");
if eof(f) then ....
```

For more information see [1].

3.8. BINARY SEQUENTIAL FILES

Any file created with the parameter T = integer, real or char is a binary one. It is a sequence of components of the type T. Only objects of type T can be read from or written to this file.
The following operations are available to binary files:

```
put(f, w1, ..., wn)
get(f, x1, ..., xn)
eof(f)
```

where f is a file opened with the type T, wi is an expression of the type T and xi is a variable of the type T.
The statement put(f, w1, ..., wn) writes the components w1, ...,wn to the file f. The statement get(f, x1, ..., xn) reads the next n components from the file f and assigns them to the variables x1, ..., xn. The statement eof is the same as for text files.

3.9. DIRECT ACCESS BINARY FILES

Direct access files are treated as a sequence of bytes without any interpretation. Operations RESET and REWRITE prepare a file for both reading and writing. RESET is used for existing files,
REWRITE for the new ones.
The following additional operations are available:

call SEEK(f, offset, base) - moves the file pointer to the position designated by offset relative to base.
Offset is a signed integer specifying the number of bytes.
Possible values for base are:
0 - begining of file,
1 - current position of file pointer,
2 - end of the file.

Examples:

call SEEK(f, 0, 0) - rewinds file f,
call SEEK(f, -3, 1) - backspaces file f by 3 bytes,
call SEEK(f, 0, 2) - moves the file pointer to the first byte after end of file

POSITION(f) - returns current position of the file pointer associated with f.

PUTREC(f, A, n) - where A is an array of any primitive type and n is an integer variable. Let k be the number of bytes occupied by elements of array A. This operation writes min(k, n) bytes from A to the file f and advances file pointer by the number of written bytes. The number of bytes written to the file is returned in the variable n.

GETREC(f, A, n) - where A is an existing array of any primitive type and n is an integer variable. Let k be the number of bytes occupied by elements of array A This operation reads min(k,n) bytes (or less, if end of file is encountered) from the file and advances the file pointer by the number of read bytes. The number of bytes read from the file is returned in the variable n.

3.10. CONCURRENCY

Implemented concurrency mechanisms differ much from those described in the LOGLAN-82 report []. In particular, only distributed processes are implemented, so they cannot communicate through shared variables. For this reason semaphores had to be replaced by an entirely new communication mechanism. Such a mechanism has been designed and it is based on the rendez-vous schema.

3.10.1. INVOKING THE LOGLAN INTERPRETER FOR CONCURRENT PROGRAMS

A concurrent LOGLAN program may run on a single computer with concurrency simulated by time slicing. In this case LOGLAN interpreter is invoked as usual. One must only remember that /m optional parameter (see 1.4.) denotes memory size for each process rather than for the whole program.

To achieve true parallel (multiprocessor) execution, a network of IBM PC computers may be used. For the time being, only D-Link Network Version 3.21 is supported. In order to run a LOGLAN program in the network environment take the following steps:

1) make sure that every node is logged on,

2) select arbitrarily one node as a console,

3) invoke the LOGLAN interpreter on every node except the console, giving it /r option with the console node number (see 1.4.). You must give the same program file to all interpreters. Most conveniently it may be achieved by accessing a file on a disk connected through the network to each node.

4) invoke the interpreter on the console without the /r option (in the usual way). Give it the same program file as above.

After the last step the main program process begins its execution on the console node. Other processes may be created dynamically on any node on which an interpreter is running.

Regardless of the fact whether the network is used or not, more than one process may be

executed on the same computer.

3.10.2. RESTRICTIONS AND DIFFERENCES FROM THE REPORT

All processes (even those executed on the same computer) are implemented as distributed, i.e. without any shared memory. This fact implies some restrictions on how processes may be used. Not all restrictions are enforced by the present compiler, so it is the programmer's responsibility to respect them. This is the list of restrictions:

1) all process units must be declared as global, i.e. directly inside the main program,
2) a process cannot access global variables (except for the main program process),
3) any remote access to a process object other than a procedure call is inhibited
4) each parameter of
   - a process,
   - a procedure called by remote access to a process object,
   - a procedure parameter of a process,

   must be one of the following:
   - a value of the primitive type (Integer, Real, Char, Boolean, String)
   - a procedure declared directly inside a process
   - a procedure which is a formal parameter of a process
   - any reference to a process object.

This restriction implies that references to objKcts other than processes have only local meaning (in a single process) and cannot be passed among the processes.

5) comparisons, IS, IN and QUA operations are not allowed for the references to processes.
6) operations which require dynamic type checking on the references to processes are not allowed.
7) a process may be attached only by a proper coroutine generated by it.
8) the variable MAIN is accesible only in the main program process.

The following concurrent constructs described in the report are not implemented at all:

- semaphores and all operations on them
- the WAIT expression.

Semantics of the NEW generator is slightly modified when applied to the processes. The first parameter of the first process unit in the prefix sequence ***must*** be of type INTEGER. This parameter denotes the node number of the computer on which this process will be created. For a single computer operation this parameter must be equal to 0.

Example:

```
unit A:class(msg:string);
...
end A;
unit P:A process(node:integer, pi:real);
...
end P;
...
var x:P;
...
begin
```

```
...
 (* Create process on node  4.  The  first  parameter  is  the  *)
 (* string required by the prefix A, the second is the node number *)
 x := new P("Hello", 4, 3.141592653);
...
end
```

The following parallel constructs are implemented as defined in the report:

- KILL operation for a process
- RESUME statement
- STOP statement without parameter.

3.10.3. COMMUNICATION MECHANISM

Processes may communicate and synchronize by a mechanism based on rendez-vous. It will be referred to as "alien call" in the following description.

An alien call is either:
- a procedure (or function) call performed by a remote access to a process object, or
- a call of a procedure which is a formal parameter of a process, or
- a call of a procedure which is a formal parameter of an alien-called procedure (this is a recursive definition).

Every process object has an enable mask. It is defined as a subset of all procedures declared directly inside a process unit or any unit from its prefix sequence (i.e. subset of all procedures that may be alien-called).

A procedure is enabled in a process if it belongs to that process' enable mask. A procedure is disabled if it does not belong to the enable mask.

Immediately after generation of a process object its enable mask is empty (all procedures are disabled).

Semantics of the alien call is different from the remote call described in the report. Both the calling process and the process in which the procedure is declared (i.e. the called process) are involved in the alien call. This way the alien call may be used as a synchronization mechanism.

The calling process passes the input parameters and waits for the call to be completed.

The alien-called procedure is executed by the called process. Execution of the procedure will not begin before certain conditions are satisfied. First, the called process must not be suspended in any way. The only exception is that it may be waiting during the ACCEPT statement (see below). Second, the procedure must be enabled in the called process.

When the above two conditions are met the called process is interrupted and forced to execute the alien-called procedure (with parameters passed by the calling process).

Upon entry to the alien-called procedure all procedures become disabled in the called process.

Upon exit the enable mask of the called process is restored to that from before the call (regardless of how it has been changed during the execution of the procedure). The called process is resumed at the point of the interruption. The execution of the ACCEPT statement is ended if the called process was waiting during the ACCEPT (see below).

At last the calling process reads back the output parameters and resumes its execution after the call statement.

The process executing an alien-called procedure can easily be interrupted by another alien call if the enable mask is changed.

There are some new language constructs associated with the alien call mechanism. The following statements change the enable mask of a process:

ENABLE p1, ..., pn

enables the procedures with identifiers p1, ..., pn. If there are any processes waiting for an alien call of one of these procedures, one of them is chosen and its request is processed. The scheduling is done on a FIFO basis, so it is strongly fair. The statement:

DISABLE p1, ..., pn

disables the procedures with identifiers p1, ..., pn.

In addition a special form of the RETURN statement:

RETURN ENABLE p1, ..., pn DISABLE q1, ..., qn

allows to enable the procedures p1, ..., pn and disable the procedures q1,...,qn after the enable mask is restored on exit from the alien-called procedure. It is legal only in the alien-called procedures (the legality is not enforced by the compiler).

A called process may avoid busy waiting for an alien call by means of the ACCEPT statement:

ACCEPT p1, ..., pn

adds the procedures p1, ..., pn to the current mask, and waits for an alien call of one of the currently enabled procedures. After the procedure return the enable mask is restored to that from before the ACCEPT statement.

Note that the ACCEPT statement alone (i.e. without any ENABLE/DISABLE statements or options) provides a sufficient communication mechanism. In this case the called process may execute the alien-called procedure only during the ACCEPT statement (because otherwise all procedures are disabled). It means that the enable mask may be forgotten altogether and the alien call may be used as a pure totally synchronous rendez-vous. Other constructs are introduced to make partially asynchronous communication patterns possible.

3.11. SYSTEM SIGNALS

System signals are connected to runtime errors (see APPENDIX C).

These signals are the following:

ACCERROR - reference to non existing object,
CONERROR - array index outside the range or lower bound is greater than upper bound during array object generation,
LOGERROR - errors related to control transfer,
MEMERROR - memory overflow,
NUMERROR - errors related to arithmentic operations like division by zero, floating point overflow,
TYPERROR - type conflict in assignment statement, during parameter tran smission or headline conflict for actual parameter function and procedure.
SYSERROR - errors related to file system, like reading after writing, too many files etc.

3.12. IMPLEMENTATION RESTRICTIONS

- Text line in source program can't be longer than 80 characters.
- Maximal length of identifier is 20 characters, but entire length of all identifiers and keywords should be less than 3000 characters.
- String constant can't be longer than 260 characters.
- For case instructions:
  - up to 6 levels of nested case instructions are allowed,
  - range of labels can't be greater than 160.
- Number of formal parameters can't be greater than 40, whereas up to 35 output or input parameters are allowed. Total number of formal parameters and variables declared in one module can't be greater than 130.
- Number of array indices (i.e. arrayof) can't be greater than 63,
- Standard type integer has the range (-32767,+32767) for small memory (16 - bit word). For huge memory (32-bit word) the range is (-2147483647,+2147483647), but values of constant expressions in a program must lie within the range (-2767, 32767).
- Real numbers have the range (-8.43E-37, 3.37E+38) with 24-bit mantissa and 8-bit exponenet for small memory , giving about 7 digits of precision. For huge memory the range is (4.19E-307, 1.67E+308) with 53-bit mantissa and 11-bit exponent, giving about 15 digits of precision.Values of constant expression in a program must lie in the range (-8.43E-37, 3.37E+38).

Warning

Compiler computes values of expressions built from constants without range checking. It means, that integer overflow, floating point overflow or underflow cause incorrect result without any message.

4. APPENDIX A : PREDEFINED CONSTANTS

INTSIZE
The size in bytes of integer variables (2 for small memory, 4 for huge memory)

REALSIZE
The size in bytes of real variables (4 for small memory, 8 for huge memory)

5. APPENDIX B : PREDEFINED CLASSES

The present version of Loglan system is distributed together with two predefined classes:
IIUWGRAPH which enables operating in a graphic mode and MOUSE which permits to control the events of mouse.

We are sorry to tell you that there are three different implementations of the classes, they are destined to: PC/AT version, 486/386 version and Unix/Xwindows version of Loglan system. In each version you are obliged to use similar, but not identical procedures. Sorry for this inconvenience.

The early versions of library IIUWGRAPH have been elaborated by
Piotr Carlsson, Miroslawa Milkowska, Janina Jankowska,
Michal Jankowski at Institute of Informatics,
University of Warsaw 1987,
and added to Loglan system by Danuta Szczepanska 1987,

the recent versions were done at LITA, Pau,
by
Pawel Susicki (1991) for Unix,
Sebastien Bernard (1992) for ATARI, see a separate document,
Eric Becourt et Jerôme Larrieu (1993) for Unix and Xwindows, see a separate document on Xiiuwgraf ,
Frederic Pataud (1994) for 386/486 machines

{ the predefined class IIUWGRAPH is included in all versions of interpreter of Loglan, with the *exception* of the present version of interpreter for VAX/VMS.}

DOS 486/386 version

unit IIUWGRAPH: class;

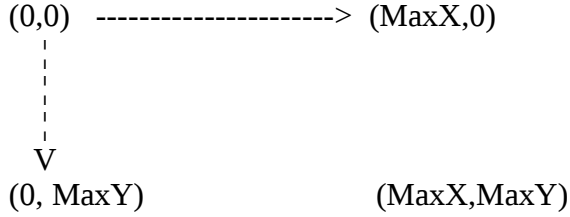
{ this predefined class enables basic graphic operations
for DOS machines based on **486** or 386 processors }

{ this document gives the specification of new version of IIUWGRAPH
class made in October 1994 by ***Frederic Pataud*** ŕ Pau
}

hidden MaxX, MaxY, current_X, current_Y, is_graphic_On,
current_Colour, current_Background_Colour, current_Style,
current_Palette, current_Pattern ;

const MaxX =
MaxY =

{ the screen's coordinates are



}

var currentDriver : integer, { see NOCARD below }
current_X, current_Y: integer { it is the current position }
is_graphic_On: Boolean, { evidently tells whether we are in
graphics mode }
current_Colour : integer, { }
current_Background_Colour : integer,
current_Style : integer, { }
current_Palette : integer,
current_Pattern

unit GRON : **procedure** (i: integer);
{ procedure sets the monitor in graphic mode and clears the buffer
of screen. The parameter determines the resolution and the number of colours.
The user should assure that the resolution chosen should correspond to that which set by means of command
SET go32 drivers {path}<driver.file> <width> <height><noColours>
eg.
set go32 drivers c:\loglan\svga\drivers\vesa.grn gw 1024 gh 480 nc 256
An execution of instruction call gron(i) ***must precede*** any of the graphic commands described below.

case (i)
{
0 : 640x480x16
1 : 640x480x256
2 : 800x600x16
3 : 800x600x256
4 : 1024x768x16
5 : 1024x768x256
6 : 1280x1024x16
7 : 1280x1024x256
8 : 1600x1280x16
9 : 1600x1280x256
}
}

unit GROFF : **procedure**;
{ the procedure sets the monitor in the text mode filling it with spaces.
DO NOT FORGET to set the monitor in the text mode before you terminate your program
}

unit CLS : **procedure**;
{ the screen will be cleared and filled with colour 0 }

{ PROCEDURES CONTROLLING THE COLOURS }

unit COLOR : **procedure**(co : integer);
{ sets current color to co
for monochrome displays, 0 means black, non-0 - white
for color displays, 0 means background

see PALLET
}

unit STYLE : **procedure**(styl : integer);
{ sets style of lines and fill shades to a combination
of current color and background color (for mono -
white and black, respectively) according to 5 predefined
patterns:

0	....
1	****
2	***.
3	**..
4	*.*.
5	*...

where '*' means current color, '.' background colour
When drawing the segments the subsequent pixels will have colour determined by cyclic application of style pattern. The first and the last pixels of a segment will have always current colour.
When filling contours the given style will be applied to horizontal lines with even coordinate. The style for odd lines is determined automatically.
The same applies for perpendicular lines.
}

unit BORDER : **procedure** (background_Colour: integer);

{ sets actual background color to i (i = 0,1,...,15) }

~~**unit** PALLET : **procedure** (nr : integer);~~
~~{~~

~~the codes of colors are as follows~~

~~0~~	~~black~~
~~1~~	~~blue dark~~
~~2~~	~~green dark~~
~~3~~	~~turquoise dark~~
~~4~~	~~red dark~~
~~5~~	~~violet~~
~~6~~	~~brown~~
~~7~~	~~grey light~~
~~8~~	~~grey dark~~
~~9~~	~~blue~~
~~10~~	~~green~~
~~11~~	~~turquoise~~

~~12 red light~~
~~13 rose~~
~~14 yellow~~
~~15 white~~
~~}~~

{ PROCEDURES CONTROLLING POSITION }

unit MOVE : **procedure** (x,y :integer);
{ procedure MOVE sets the current position on the screen on the pixel with coordinates
x - column,
y - line }
{ precondition of MOVE:
0≤x≤MaxX & 0≤y≤MaxY
}

unit INXPOS : **function**: integer;
{ function INXPOS returns the x coordinate of the current position }

unit INYPOS : **function** : integer;
{ function INYPOS returns the y coordinate of the current position }

unit PUSHXY : **procedure**;
{ pushes current position, color & style onto the stack.
The stack is kept internally, max depth is 16
}

unit POPXY: **procedure**;

{ restores position, color & style from internal stack }

```
{ Example
unit DIAGONAL : procedure;
   var ix, iy : integer;
begin
        call PUSHXY;
        ix := INXPOS;
        iy := INYPOS;
        call DRAW(ix+10, iy+10);
        call POPXY
end DIAGONAL;
```

}

{ PROCEDURES SERVING POINTS & LINES}

unit POINT : **procedure**(x,y: integer);
{ moves current position to pixel (x,y) and sets it to the current color
}

unit INPIX : **function** (x,y : integer) : integer;
{
moves to pixel (x,y) and returns its color setting;
}

unit DRAW : **procedure**(x,y : integer);
{
draws a line from current screen position to (*x,y*);
sets current position to (*x,y*);
line is drawn in current color, with both terminal pixels
always turned white (non-background) for non-black
(non-background) line color.
}

unit intens: **procedure**(Size :integer; xCoord,yCoord:arrayof integer,
Colour,Filled :integer);
/* draw a polygon*/
{ draw a simple, closed polygon of Size points, the edges of the polygon go from (*xCoord[i], yCoord[i]*) to (*xCoord[i+1], yCoord[i+1]*) for i = 1, ..., Size-1
The colour used will be *Colour*. The polygon will be filled iff *Filled*<>0.
}

unit CIRB : **procedure** (xi, yi, rx,ry : integer, alfa, beta : real,
cbord, fill : integer);

{
draws a circle (or ellipse, depending on aspect value, see below),
optionally filling its interior;
does not preserve position;
(*xi,yi*) - are center coordinates,
rx - radius in pixels (horizontally),
ry - radius in pixels (perpendicularly),
alfa, beta - starting & ending angles; if alfa=beta a full
circle is drawn; values should be given in radians;
cbord - border color,
fill - if fill <>0, interior is filled in current style&color

}

unit hfill: **procedure**(x : integer);
{ draw an horizontal line between the current position and
(x,currentY) with the current color, after it change the current
position to (x, currentY)
}

unit vfill: **procedure**(y : integer);
{ draw a vertical line between the current position and
(currentX,y) with the current color, after it change the current
position to (currentX,y)
}

unit patern: **procedure**(x1,y1,x2,y2,c,b : integer);
{ draw a ***rectangle*** between the points (*x1*,*y1*) and (*x2*,*y2*) with the
color *c* (the current color is not change). if *b*=0 then the box is
empty else it is filled.
}

{ Procedures operating on bitmaps }

unit GETMAP : **function** (x,y : integer) : **arrayof** integer;
{saves rectangular area between current position as
top left corner and (ix,iy) as bottom right corner,
including border lines;
position remains unchanged.
array of integer should have
$4+(rows \cdot \lceil columns/8 \rceil \cdot coeff)$
bytes. The coefficient coeff is 1 for Hercules, 2 for CGA, 4 for EGA
card.
ATTENTION: in DOS 286 environment a bigger size of the array may necessitate the use of *loglan* with *the option H+*, see also memavail
}

unit PUTMAP : **procedure** (a: **arrayof** integer);
{sets rectangular area of screen pixels to that saved
by "getmap" in "iarray";
same size is restored, with top left corner in current
position;
position remains unchanged.
}

unit ORMAP : **procedure** (a : **arrayof** integer);
{same as putmap, but saved bitmap is or'ed into screen
rather than just set.
}

unit XORMAP : **procedure** (a: **arrayof** integer);
{same as putmap, but saved bitmap is xor'ed into screen
rather than just set.
}

{Procedures operating on characters and strings}

unit outstring: **procedure**(x,y: integer, s: string, back_col, front_col: integer);
{ *x*, *y* are the coordinates where to put the string,
s is the string to be shown, in *front_col* colour letters on the *back_col* colour background
}

unit track: **procedure**(x,y,c,valeur : integer);

{ write an integer value *valeur* at the position (*x*,*y*) with the color *c*.
It does not change the current position nor the current color
}

unit inkey : **function** : integer;

{ returns next character from keyboard buffer;
0 is returned if buffer is empty;
special keys are returned as negative numbers;
ALT-NUM method may be used for entering character codes
above 127 (this makes entering special keys 128-132
impossible);
if a character is returned, it is also removed
from the buffer, so MS-DOS will not see it (CTRL-C!);
typeahead is allowed, echo is suppressed.
}

unit HASCII : **procedure**(c: integer);
{'xor's the character = chr(c) in a ~~8*8~~ box with top left corner
in the current position;
moves current position by ~~(8,0)~~;
call hascii(0)- sets complete box to black (=background),
with no change in position.
}

unit hfont: **function**(x,y,lg,min,max,default,col_f,col_e,col_c : integer):
integer;

{ arrange a small 1 line window for ***reading*** an integer value from this window,
the position of the window corner is (*x*, *y*),
the length of the window is *lg* characters,
the value v should be greater than *min* and smaller than *max*,
the default value read is *default*,
the colour of the window is *col_f*,
the colour of the digits is *col_e*,
the colour of cursor is *col_c*

reads in graphic mode an integer in the window which begins at the (x,y) position, window is lg caracteres long. the maximum length of the
integer that is read is 10. there is a default value, a minimum value and a maximum value. the window is drawn with the col_f color, the cursor is in the col_c color and the integer is writing in the col_e
color. you can use 0..9,+,-,backspace,escape and return keys. }

unit HPAGE : **procedure**(x,y,long: integer, A: arrayof char, back, front: integer);
{ this procedure arranges a 1-line high window in position *x*,*y* of length *long* in which a portion of text *A* is shown in colour *front* on the background colour *back*.
Making use of keys controlling the cursor {left, right, PgUp, PgDn}
the user can scroll the text (horizontally) in the window. Pressing the Enter key terminates the procedure}

end IIUWGRAPH;

unit MOUSE: **class**;

{ *init* -lors de l'initialisation de la souris, on peut définir les événements qui vont faire réagir la fonction getpress; le premier et le deuxičme paramčtre représentent respectivement la souris et le clavier, si une valeur non nulle est donnée comme paramčtre alors getpress réagira ŕ l'événement.

Une paire (1,1) va permettre de prendre en compte ŕ la fois les événements de la souris et ceux du clavier; une paire (1,0) quand ŕ elle ne prendra en compte que la souris. Pour une plus grande souplesse d'utilisation, il est possible lors du programme, aprčs l'initalisation, de changer cette prise en compte, cela se fera par l'appel de la procedure *getmovement*, procédure ayant les męmes paramčtres (avec le męme ordre) que la fonction init.

Pour detecter les événements, on utilisa la fonction *getpress*, qui retourne un booléen indiquant la présence ou l'absence d'événement (respectivement les valeurs true et false). Il est bon de noter qu'ainsi définie la fonction getpress n'est pas bloquante. Les paramčtres en retour sont soit nuls (pas d'événement) soit correspondent:

bool:=getpress(v,p,h,l,r,c : integer);
v = position en y de la souris
p = keyboard status (Touche control_left,control_right, alt, alt_gr, shift_left, shift_right)
h = position en x de la souris
l = touche clavier
r = flags
c = boutons de la souris (0=aucun, 1=gauche, 2=droite, 3=gauche et droite)
Nb: le bouton central n'est pas géré.

NOTEZ BIEN! Lorsque les événements du clavier sont pris en compte dans le gestionnaire, **il ne faut pas** utiliser les fonctions d'entrées clavier readl, readln, hfont, hfont8, hpage, inkey,...) *sous peine de plantage de l'ordinateur*.
}

unit init: **procedure**(checkMouse, checkKeyboard: integer);
{ initializes the Mouse driver.
tells which events will be checked:
if checkMouse <>0 then the events of Mouse will be reported to getpress, see below otherwise ignored;
if checkKeyboard <>0 then the events of Keyboard will be reported to getpress, otherwise ignored
Attention please! While the events of the keyboard are taken under control by *init* or *getmovement*
do not use the functions or procedures: read, readln, hfont, hfont8, hpage, inkey that read keys
You risk to hang your system!
}

end init

unit getmovement: **procedure**(checkMouse, checkKeyboard: integer);
tells which events will be checked:
if checkMouse <>0 then the events of Mouse will be reported to getpress, see below otherwise ignored;
if checkKeyboard <>0 then the events of Keyboard will be reported to getpress, otherwise ignored
Attention please! While the events of the keyboard are taken under control by *init* or *getmovement*
do not use the functions or procedures: read, readln, hfont, hfont8, hpage, inkey that read keys
You risk to hang your system!
end getmovement;

unit getpress: **function**(v,p,h,l,r,c : integer): Boolean;
{ *v* = y coordinate of the cursor,
h = x coordinate of the cursor,
p = keybord status control_left,control_right, alt, alt_gr, shift_left, shift_right
l = code of key pressed
r = flags
c = buttons pressed (0=aucun, 1=gauche, 2=droite, 3=gauche et droite)
Nb: the middle button is not taken into account.
end getpress

unit showcursor: **procedure;**
{the cursor becomes visible and follows the movements of the mouse}
end showcursor;

unit hidecursor: **procedure;**
{the cursor becomes invisible}
end hidecursor;

end MOUSE;

Enclosed you find a sample program

```
Program SystemeGraph;
    (* by Frederic Pataud, October 1994 *)
Begin
Pref iiuwgraph block     (* inherit the graphic functions *)
 Begin
 Pref mouse block       (* inherit the mouse functions *)
```

```
  (*************************************************************************
)
  (*              P r o g r a m  m e  P r i n c i p a l              *)
  (*************************************************************************
)
    var v,p,h,i : integer,
      l,r,c : integer,
      rep : arrayof char,
      d : boolean,
      xx,yy : arrayof integer,
      status,code,x,y,flags,button : integer;

  Begin

    call gron(0);           (* enter the graphic mode *)
    call init(1,0);          (* initialize the mouse, disregard the keyboard events, check for mouse
events *)

    call showcursor;              (* show cursor *)
    call patern(5,5,635,475,2,0);        (* make a frame around the screen *)
    call outstring(10,10,"x=",2,0);
    call outstring(100,10,"y=",2,0);
    call outstring(10,30,"status = ",2,0);
    call outstring(10,50,"code   = ",2,0);
    call outstring(10,70,"flags  = ",2,0);
    call outstring(10,90,"button = ",2,0);
    call patern(100,210,300,320,3,1);        (* make a rectangle filled in colour 3 *)

    array xx dim (1:6);
    array yy dim (1:6);
    xx(1):=410; yy(1):=10;
    xx(2):=450; yy(2):=30;
    xx(3):=460; yy(3):=50;
    xx(4):=430; yy(4):=80;
    xx(5):=420; yy(5):=40;
    xx(6):=480; yy(6):=30;
    call intens(6,xx,yy,8,1);             (* show a polygon filled*)
    for i:=1 to 6
     do
      yy(i):=yy(i)+100;
     od;
    call intens(6,xx,yy,15,0);            (* show another polygon empty *)

    call cirb(500,300,50,40,100,3500,10,0);   (* draw an empty pie or camembert *)
    call cirb(400,400,40,40,600,4000,11,1);     (* draw a filled pie *)
```

```
    i:=hfont(100,350,6,-9999999,9999999,500,9,0,15);      (* read integer from a window *)
    call hpage(100,400,10,unpack("Il fait beau dans ma verte campagne"),9,0);   (* show text *)
    rep:=hfont8(100,430,10,80,unpack("tototutu"),9,0,15);              (* read text *)

    call getmovement(1,1);          (* take into consideration both key events and mouse events *)

    do
     d:=getpress(v,p,h,l,r,c);              (* ask about an event *)
     if (d)
     then call outstring(10,400,"Event",2,0);
          call patern(80,25,130,100,0,1);
          call track(40,10,v,0,4);                    (* print integer *)
          call track(140,10,p,0,4);
          call track(80,30,h,0,4);
          call track(80,50,l,0,4);
          call track(80,70,r,0,4);
          call track(80,90,c,0,4);
          if((h=164 and l=27) or (c=3))               (* exit if either two buttons were pressed c=3 or
Ctrl+Esc key *)
          then exit;
          fi;
     fi;
    od;
    call groff;                             (* leave the graphic mode and return to the text mode *)
    writeln("i=",i);
    for i:=lower(rep) to upper(rep)
    do
      write(rep(i));
    od;
    writeln;
   End
 End
End.
```

DOS PC/AT

unit IIUWGRAPH: class;

{ Each interpreter is equipped with one version of graphic library which corresponds to one of the following possibilities:
- EGA card, use egaint *also if you have a VGA card*

- Hercules mono card, use hgcint

- IBM CGA card use cgxxint
several variants are offered
CGA colour,
CGA mono 320 x 200
CGA mono 640 x 200
all above versions were tested in DOS 3.3 environment
}

```
hidden    MaxX, MaxY,  current_X, current_Y, is_graphic_On,
          current_Colour, current_Background_Colour,  current_Style,
          current_Palette,  current_Pattern ;


                  {   Hercules          EGA/VGA             CGA
                                                              }

const  MaxX =          719  ;        {    639                 319   }
       MaxY =          347 ;         {    349                 199   }
```

{ the screen's coordinates are

```
     (0,0)  ----------------------> (MaxX,0)
        |
        |
        |
        |
        V
     (0, MaxY)                     (MaxX,MaxY)
```

}

```
var  currentDriver : integer,                 { see NOCARD below }
     current_X, current_Y:  integer          { it is the current position }
     is_graphic_On:  Boolean,               { evidently tells whether we are in
                 graphics mode }
     current_Colour : integer,             { }
     current_Background_Colour : integer,
```

current_Style : integer, { }
current_Palette : integer,
current_Pattern

unit GRON : procedure (i: integer);
{ procedure sets the monitor in graphic mode and clears the buffer of screen. The parameter is *meaningless*, the only exception is made for the IBM CGA card in this case if you have chosen the mode 320x200 pixels the the value 1 of the parameter means colours, the value 0 means a mono screen is connected to the card
An instruction call gron(i) *must precede* any of the graphic commands described below.
}

unit GROFF : procedure;
{ the procedure sets the monitor in the text mode filling it with spaces.
DO NOT FORGET to set the monitor in the text mode before you terminate your program
}

unit NOCARD : function : integer;
{ the value given by this function determines the type of the currently used monitor and it is equal to
1 for Hercules mono card,
2 for IBM CGA color
3 for IBM CGA mono 320 x 200
4 for IBM CGA mono 640 x 200
5 for EGA/VGA card
6 for ATARI STE
7 for Unix versions equipped with XWindows
You can not call the function nocard before GRON sets the graphic mode
}

unit CLS : procedure;
{ the screen will be cleared and filled with colour 0 }

unit HPAGE : procedure(nr, : integer, clear : boolean);
{ the procedure is applicable to the cards EGA/VGA and Hercules only! only hgcint and egaint support it
it selects a page of video memory with the number = nr,
clears its contents if clear is set <>0,
and sets the mode
mode = 0 the content of the page is shown as text,
mode = 1 the content of the page is shown graphically,
mode = -1 a worktime buffer is associated with the page.
Mode -1 does not change the number not the way it is shown. Mode 0 links the buffer with the selected page. For card Hercules only call HPAGE(0, 1, 1) is equivalent to call

GRON(99) and call HPAGE(0, 0, 1) is equivalent to call GROFF.

Example of an animating loop

```
var nr: integer;
begin
  call GRON(0);
  nr := 1;
  (* draw first image *)
  call DRAW(...)
  ...
  while more
  do
      call HPAGE(1-nr, 1,0);      (* displaying *)
      call HPAGE(nr, -1,1);      (* buffering *)
      (* draw modified image *)
      call DRAW( ...)
       ...
      nr := 1-nr
   od
end example
```

```
unit VIDEO : procedure( A: array of integer);
      { this procedure can not be applied with egaint = EGA/VGA card }
         { the worktime buffer will be associated with the array A.
            A call of VIDEO does not change the contents of the buffer.
            All subsequent calls of the procedures modifying the screen will
            concern the array A. The screen does not change.
            A ready image can be moved to the screen with the help of
GETMAP/PUTMAP procedures or it can be stored on disk.
            The array should have 16 kBytes for IBM CGA card or
           32 kBytes for Hercules card.}
```

{ PROCEDURES CONTROLLING THE COLOURS }

```
unit COLOR : procedure(co : integer);
{         sets current color to co
     for monochrome displays, 0 means black, non-0 - white
         for color displays, 0 means background
    see PALLET
}

unit STYLE : procedure(styl : integer);
{         sets style of lines and fill shades to a combination
          of current color and background color (for mono -
          white and black, respectively) according to 5 predefined
```

patterns:

0	....
1	****
2	***.
3	**..
4	*.*.
5	*...

where '*' means current color, '.' background colour
When drawing the segments the subsequent pixels will have colour determined by cyclic application of style pattern. The first and the last pixels of a segment will have always current colour.
When filling contours the given style will be applied to horizontal lines with even coordinate. The style for odd lines is determined automatically.
The same applies for perpendicular lines.
There are other possibilities of mixing colours, cf procedure PATERN.
}

unit PATERN : procedure (par, par1, par2, par3 : integer);
{ sets style of lines and fill shades to an explicitly specified
combination of colours. When drawing lines the only parameter of importance will be par. When filling the parameters par and par2 concern the horiwontal (resp. perpendicular) lines with the coordinate x (resp: y) even. lines
combination of colors : "iv" for even scan lines, "io" for odd.
Color encoding is decimal, allowing 4 pixels.
Lines are drawn always according to "iv".

Examples:

call patern(1100,0011)
is equivalent to
call color(1), call style(3)

call patern(1212,2121)
produces a shade that cannot be otherwise achieved
(a dotted line consisting of pixels in colors 1 and 2)

unit BORDER : procedure (background_Colour: integer);
[IBM color mode only i.e. cgaxxint]

sets actual background color to i (i = 0,1,...,15)

unit PALLET : procedure (nr : integer);
{

the codes of colors are as follows

0	black
1	blue dark
2	green dark
3	turquoise dark
4	red dark
5	violet
6	brown
7	grey light
8	grey dark
9	blue
10	green
11	turquoise
12	red light
13	rose
14	yellow
15	white

the procedure does not applies for Hercules card}

unit INTENS : procedure (i : integer);

{ changes current intensity, 1 means more intensity, 0 less;
default intensity is 1
Applies to IBM CGA only
}

{ PROCEDURES CONTROLLING POSITION }

unit MOVE : procedure (x,y :integer);
{ procedure MOVE sets the current position on the screen on the pixel with coordinates
x - column,
y - line }
{ precondition of MOVE:
0≤x≤MaxX & 0≤y≤MaxY
}

unit INXPOS : function: integer;
{ function INXPOS returns the x coordinate of the current position }

unit INYPOS : function : integer;

```
    { function INYPOS returns the y coordinate of the current      position }


unit PUSHXY : procedure;
{       pushes current position, color & style onto the stack.
        The stack is kept internally, max depth is 16
}


unit POPXY: procedure;

{       restores position, color & style from internal stack   }

{ Example
unit DIAGONAL : procedure;
   var ix, iy : integer;
begin
        call PUSHXY;
        ix := INXPOS;
        iy := INYPOS;
        call DRAW(ix+10, iy+10);
        call POPXY
end DIAGONAL;
}


unit TRACK : procedure (x,y : integer);
{       displays a small (8*8) arrow-shaped cursor which can be     moved around with cursor
keys; a single keystroke moves
        it by 5 pixels, in NUM mode step size is 1 pixel;
        "home" key returns the cursor to the initial (x,y);
        "end" removes cursor from screen, and returns - the new current
        position can be read with "INXPOS" and "INYPOS" above.

ATTENTION: if you have a mouse then read on the predefined class Mouse, which permits to
control the mouse
}

{ PROCEDURES SERVING POINTS }

unit POINT : procedure(x,y: integer);
{               moves current position to pixel (x,y) and sets it to the current                color
 }

unit INPIX : function (x,y : integer) : integer;
        {
                moves to pixel (x,y) and returns its color setting;
```

}

unit DRAW : procedure(x,y : integer);
{
draws a line from current screen position to (x,y);
sets current position to (x,y);
line is drawn in current color, with both terminal pixels
always turned white (non-background) for non-black
(non-background) line color.
Bresenham's algorithm is used, pixels belonging to the segment change their state depending on current colour and style.
}

unit CIRB : procedure (xi, yi, ri : integer, alfa, beta : real,
cbord, fill, p, q : integer);

{
draws a circle (or ellipse, depending on aspect value, see below),
optionally filling its interior;
does not preserve position;
(xi,yi) - are center coordinates
ri - radius in pixels (horizontally)
alfa, beta - starting & ending angles; if alfa=beta a full
circle is drawn; values should be given in radians;
cbord - border color,
fill - if fill <>0, interior is filled in current style&color
p,q - aspect ratio; if p/q=1, a perfect circle is drawn,
if p/q<1, the horizontal axis is longer,
if p/q>1 - the vertical axis is longer;
}

unit HFILL : procedure(x: integer);

{ fills current row (horizontally) from current position
(INXPOS, INYPOS) up to (x,INYPOS) with bit pattern depending
on current color, style and/or pattern and position on
the screen in such a way that adjacent "hfill"ed" rows
will produce a shade simulating color;

ATTENTION *hfill does not change current position*;

It is advised to use hfill when filling contours since it works faster then DRAW. Procedure hfill is capable to similate additional colours.
}

unit VFILL : procedure(y: integer);

{ fills current column (vertically) from current
position (INXPOS, INYPOS) up to (INXPOS, y) in a similiar way
that "hfill" does;
rectangular area "vfill'ed" is not distinguishable
on the screen from same shape "hfill'ed", except that
it will take much longer to fill.

ATTENTION *hfill does not change current position*;

{ Procedures operating on bitmaps }

unit GETMAP : function (x,y : integer) : arrayof integer;
{saves rectangular area between current position as
top left corner and (ix,iy) as bottom right corner,
including border lines;
position remains unchanged.
array of integer should have
$4+(rows \cdot \lceil columns/8 \rceil \cdot coeff)$
bytes. The coefficient coeff is 1 for Hercules, 2 for CGA, 4 for EGA card.
ATTENTION: in DOS environment a bigger size of the array may necessitate the use of *loglan* with *the option H+*, see also memavail
}

unit PUTMAP : procedure (a: array of integer);
{sets rectangular area of screen pixels to that saved
by "getmap" in "iarray";
same size is restored, with top left corner in current
position;
position remains unchanged.
}
unit ORMAP : procedure (a : arrayof integer);
{same as putmap, but saved bitmap is or'ed into screen
rather than just set.
}

unit XORMAP : procedure (a: arrayof integer);
{same as putmap, but saved bitmap is xor'ed into screen
rather than just set.
}

{Procedures operating on characters and strings}

unit INKEY : function : integer;

{returns next character from keyboard buffer;

```
        0 is returned if buffer is empty;
        special keys are returned as negative numbers;
        ALT-NUM method may be used for entering character codes
        above 127 (this makes entering special keys 128-132
        impossible);
        if a character is returned, it is also removed
        from the buffer, so MS-DOS will not see it (CTRL-C!);
        typeahead is allowed, echo is suppressed.
        }

unit HASCII : procedure
        {'xor's the character in a 8*8 box with top left corner
        in the current position;
        moves current position by (8,0);
 call hascii(0)-character code 0 sets complete box to black ( background ),
        with no change in position.
        BIOS ROM font for IBM color card is used. If the font
        table is not at F000:FA6E, the character will probably
        be unrecognizable, and most certainly wrong.
        For codes >127, table pointed to by interrupt vector 31
        is used. }

unit HFONT
        {sets 8*8 horizontal font table address to iseg:ioffs.}


unit HFONT8
      {includes a copy of IBM ROM 8*8 font and returns address suitable for passing to "hfont";
      Use of "hfont8" makes program larger but quarantees BIOS ROM independence.}


unit INHLINE : procedure (a: arrayof char; output n : integer);
      { reads a line of at most "n" characters from the keyboard, storing them in the "a" array;
      characters are echoed at current position with "hascii"  as they are typed in; a blinking cursor
      prompts for the next character;
      BACKSPACE works as expected, RETURN completes the reading;
      typing "n"-th character also completes the line;
      "l" is blank filled up to "n" bytes;
      on return "n" is the total number of characters read. }


unit OUTHLINE : procedure (a : arrayof char; n : integer);
      { calls "hascii"  "n" times with subsequent bytes from "a" array as arguments; before each
      character is written, "hascii(0)" is called. }


end IIUWGRAPH;
```

5.1. MOUSE

{Applies only to DOS/AT versions}
{For UNIX and 386 versions see the corresponding documents}

A predefined class MOUSE provides basic support for mouse. An external resident Microsoft compatible mouse driver (such as MOUSE.SYS) must be installed to use this class. MOUSE contains following procedures and functions:

unit MOUSE: class;

init:function(output b:integer):boolean
{Initializes mouse driver. Number of mouse buttons is returned in b. Returns true iff mouse hardware and software are installed.}

showcursor:procedure
{This procedure increments the internal cursor counter. If the counter is 0 it displays the cursor on the screen. The cursor tracks the motion of the mouse, changing position as the mouse changes position.}

hidecursor:procedure
{This procedure removes the cursor from the screen and decrements the internal cursor counter. Although the cursor is hidden it still tracks the motion of the mouse, changing position as the mouse changes position.}

status:procedure(output h, v:integer, l, r, c:boolean)
{This procedure reports the status of the buttons and cursor. l, r, c are true iff respectively left, right and center (if it exists) buttons are down when the procedure is called. Also position of cursor is returned in h and v. Position is expressed in Color Graphics Adapter pixels (with resolution 640x200).}

setposition:procedure(h, v:integer)
{This procedure sets the cursor to the specified horizontal and vertical positions on the screen. The new values must be within the specified ranges of the virtual screen. The values are rounded to the nearest values permitted by the screen for horizontal and vertical positions.}

getpress:procedure(b:integer; output h, v, p:integer, l, r, c:boolean)
{This procedure gives a count of selected button presses (on p) since the last call to it and the position of the cursor (on h and v) the last time the button was pressed. Parameter b selects button to be checked: 0 - left, 1 - right, 2 - center. In addition current button status is returned in l, r and c (see procedure status).}

getrelease:procedure(b:integer; output h, v, p:integer, l, r, c:boolean)

{This procedure gives a count of selected button releases (on p) since the last call to it and the position of the cursor (on h and v) the last time the button was released. Parameter b selects button to be checked: 0 - left, 1 - right, 2 - center. In addition current button status is returned in l, r and c (see procedure status).}

setwindow:procedure(l, r, t, b:integer)

{Restricts the cursor movement to window described by l, r, t, b. L and r are minimum and maximum horizontal cursor position, t and b are minimum and maximum vertical cursor position (in pixels)}

defcursor:procedure(s, x, y:integer)

{Selects text mode cursor characteristics. When s is 0, software cursor is selected and x, y define masks to be used when modifying character-attribute word in screen memory associated with position under cursor. This word is logicaly ANDed with x and the result is XORed with y. When s is 1, a hardware cursor is selected and x, y define first and last scan lines of the cursor box within character box. X must be not greater than y and both must be in range 0-7 for Color Graphics Adapter or 0-13 for Monochrome Display Adapter, Hercules Graphics Card and Enhanced Graphics Adapter.

Examples:

call defcursor(0, -1, 30464)

- selects standard (reverse video) software cursor

call defcursor(1, 11, 12)

- selects standard hardware cursor for HGC}

getmovement:procedure(output h, v:integer)

{Returns relative mouse movement since last call (in 1/200 inches).}

setspeed:procedure(h, v:integer)

{H and v specify horizontal and vertical cursor speed relative to mouse speed. It is expressed in mouse steps (1/200 inch) corresponding to 8 pixels on screen. Default is 8 horizontaly and 16 verticaly.

Examples:

call setspeed(1, 1)

- set maximum cursor speed

call setspeed(16, 32)

- set cursor speed two times slower than default}

setthreshold:procedure(s:integer)

{sets threshold speed for double speed feature. If the mouse moves faster than the threshold, the cursor speed on the screen is doubled. Default threshold is 64 mouse steps/second.

Example:
call setthreshold(10000)
- efectively disable double speed feature.}

end MOUSE;

UNIX/XWINDOWS

unit IIUWGRAF: class;
{ This is a description of XIIUWGRAF co-process which cooperates with the unix process executing your Loglan programs.
The implementation was done by Eric Becourt and Jerôme Larrieu, Pau, LITA 1993.
The description of the predefined class enabling to control the Xwindows windows and mouse follows.

Table de matičres

4 : Description des commandes de gestion de la souris

1 : The essential differences between XIIUWGRAPH and IIUWGRAPH

Ce paragraphe a pour objet de donner certaines particularités de XIIUWGRAF, ceci afin de comprendre son fonctionnement général.

Tout d'abord il est important de signaler qu'ŕ la différence de IIUWGRAPH, XIIUWGRAF est un programme ŕ part entičre (plus exactement un processus crée par l'interpręteur LOGLAN).C'est pour cela qu'il est déconseillé (sauf cas de force majeure) de faire CONTROL-C pour terminer un programme : en effet, ceci a pour effet de terminer l'exécution de l'interpręteur sans terminer XIIUWGRAF (création d'un processus zombie). Pour terminer une session graphique, il faudra donc automatiquement taper dans le programme en LOGLAN la commande GROFF car elle va terminer l'exécution de XIIUWGRAF.

Certaines commandes de IIUWGRAPH n'ont pas été implémentées (par exemple HIDECURSOR, SHOWCURSOR, PALLET, ...), soit parce qu'elles seraient d'un intéręt trčs faible dans la gestion de XWindows, soit parce qu'elles seraient difficilement réalisables, soit parce que les programmeurs ont été atteints de fainéantise chronique.

Enfin, dans vos programmes il faudra impérativement que vos unités gérant le graphisme héritent de la classe IIUWGRAPH sous peine d'erreurs ŕ la compilation .

2 : To open and to close a window using XIIUWGRAF

XIIUWGRAF permits the user yo open up to sixteen windows on the screen. The choice of the active window is done by means of Gron command described below.

2.1 : La procédure HPAGE

unit HPAGE: procedure(numerofenetre,x,y: INTEGER);

This procedure is used in order to define the position of new window on the screen and the size of the window. It is necessary to call the procedure **twice** in order to create **one** window.

HPAGE receives three parameters reçoit trois paramętres : le premier est le numéro de la fenętre (un entier compris entre 0 et 15), les deux suivants sont soit les coordonnées de la fenętre ŕ l'écran, soit la taille de cette fenętre. Un troisičme appel de HPAGE avec l'un des deux derniers paramętres nuls aura pour effet de l'effacer.

Exemple : CALL HPAGE(0, posx, posy);
CALL HPAGE(0, longueur, hauteur);

Le coin en haut ŕ gauche de la fenętre 0 sera aux coordonnées (posx,posy) et la fenętre aura une taille de longueur X hauteur.

```
        CALL HPAGE(0, 0, valeur)
ou      CALL HPAGE(0, valeur, 0)
ou      CALL HPAGE(0, 0, 0)
```

La fenętre 0 est effacée.

2.2 : La procédure GRON

unit GRON: procedure(numerofenetre: INTEGER);

La procédure GRON affiche la fenętre de numéro numerofenetre ŕ l'écran. Ensuite pour sélectionner la fenętre dans laquelle on veut travailler, on refait un deuxičme appel de cette commande.

Exemple :

```
CALL HPAGE(0,0,0);
CALL HPAGE(1,150,0);
CALL HPAGE(0,100,100);
CALL HPAGE(1,200,150);
CALL GRON(0);      (* Affichage de la fenętre 0 *)
CALL GRON(1);      (* Affichage de la fenętre 1 *)
...
CALL GRON(0);      (* Sélection de la fenętre 1 *)
...
```

2.3 : La procédure GROFF

unit GROFF: procedure;

L'appel ŕ cette commande a pour conséquence l'effaçage de toutes les fenętres et la fin d'exécution du processus XIIUWGRAF.

3 : Description des différentes commandes graphiques

3.1 : Procédure COLOR

unit COLOR: procedure(couleur: INTEGER);

Permet de déterminer la couleur d'avant plan (0 pour noir et une valeur supérieure ou égale ŕ 1 pour blanc). Cette commande a une action locale ŕ la fenętre sélectionnée par GRON.

3.2 : Procédure BORDER

unit BORDER: procedure(couleur: INTEGER);

Commande qui sélectionne la couleur de fond.

3.3 : Procédure MOVE

unit MOVE: procedure(posx, posy: INTEGER);

posx et posy deviennent les coordonnées courantes dans la fenętre. Comme COLOR, MOVE n'agit que sur la fenętre sélectionnée.

3.4 : Procédure CLS

unit CLS: procedure;

Efface la fenętre en blanc par défaut ou de la couleur spécifiée par la commande BORDER.

3.5 : Procédure POINT

unit POINT: procedure(x,y: INTEGER);
Affiche un point aux coordonnées (x,y) de la couleur spécifiée par la commande COLOR ou noir par défaut. La position courante dans la fenętre devient (x,y).

3.6 : Procédure DRAW

unit DRAW: procedure(x,y: INTEGER);

Affiche une ligne qui part de la position courante dans la fenętre vers la position (x,y). La position courante dans la fenętre devient (x,y). Elle est affichée avec la couleur courante (sélectionnée avec COLOR) et avec le style de tracé courant (sélectionnée par la commande STYLE décrite plus loin);

3.7 : Procédure CIRB

unit CIRB: procedure(posx,posy,rayon:INTEGER,alpha,beta:REAL,cbord,style,p,q:INTEGER);

Si style a pour valeur 0, CIRB affiche un arc de centre (posx,posy), de rayon rayon. alpha et beta sont les angles de départ et d'arrivée de l'arc en question. Si alpha=beta alors un cercle (ou une ellipse) est dessiné. Si p=q alors on obtient un cercle, si p>q une ellipse allongée dans le sens vertical est obtenue, sinon si p>q on a pour résultat une ellipse allongée dans le sens horizontal. Cet affichage est fait avec la couleur d'avant plan courante et le style de tracé courant.

Si style vaut 1, CIRB affiche un arc rempli ressemblant ŕ une portion de camenbert avec la

couleur d'avant plan courante.

Si style vaut 2, l'intérieur de l'arc délimité par sa courbure et la corde joignant ses deux extręmités est rempli avec la couleur d'avant plan courante.

Si l'on choisi pour style une valeur <0 ou >3, la valeur 0 est prise.

3.8 : Procédure HFILL

unit HFILL: procedure(y: INTEGER);

Trace une ligne horizontale de la position courante (posx,posy) vers les coordonnées (posx,y) avec la couleur d'avant plan courante et le style de tracé courant. La position courante dans la fenętre devient (posx,y).

3.9 : Procédure VFILL

unit VFILL: procedure(y: INTEGER);

Trace une ligne verticale de la position courante (posx,posy) vers les coordonnées (x,posy) avec la couleur d'avant plan courante et le style de tracé courant. La position courante dans la fenętre devient (x,posy).

3.10 : Fonction INXPOS

unit INXPOS: function: INTEGER;

Retourne la position courante sur l'axe des abscisses de la fenętre courante.

3.11 : Fonction INYPOS

unit INYPOS: function: INTEGER;

Retourne la position courante sur l'axe des ordonnées de la fenętre courante.

3.12 : Commandes de saisie et de restitution d'une partie d'une fenętre.

3.12.1 : Fonction GETMAP

unit GETMAP: function(x,y: INTEGER): arrayof INTEGER;

Sauve dans le tableau tab une partie rectanguraire de la fenętre courante, le coin en haut ŕ gauche étant la position courante dans la fenętre et le coin en bas ŕ droite étant la position (x,y).
Le tableau devrait avoir une taille minimum de: 4 + (nbrelignes * (3 + nbrecol div 8))

octets

En sachant qu'en LOGLAN un entier tient sur 4 octets(en UNIX seulement), il ne vous reste plus qu' ŕ faire votre cuisine.

3.12.2 : Procédure PUTMAP

unit PUTMAP: procedure(tab: arrayof INTEGER);

Affiche la portion d'image sauvée dans tab ŕ la position courante dans la fenętre. Ce qu'il y avait ŕ cette męme position avant l'affichage est totalement effaçé.

3.12.3 : Procédure ORMAP

unit ORMAP: procedure(tab: arrayof INTEGER);

Lors de l'affichage, une opération OR est faite avec la portion d'image sauvée dans tab et celle ŕ la position courante dans la fenčtre: l'image est donc affichée en "transparence".

3.12.4 : Procédure XORMAP

unit XORMAP: procedure(tab: arrayof INTEGER);

Męme chose qu' avec ORMAP ŕ la différence qu'une opération XOR est faite avec l'image sauvée dans tab et celle ŕ la position courante dans la fenętre.

3.14 : Procédure STYLE

unit STYLE: procedure(styl: INTEGER);

Définit le style de tracé dans la fenętre courante.

Si style vaut 0, le tracé sera fait avec la couleur de fond.
Si style vaut 1, le tracé sera fait avec la couleur d'avant plan.
Si style vaut 2,3,4 ou 5, le tracé sera fait avec les motif suivant :

2 : ******...******...******
3 : ****.....****.....****
4 : **...**...**...**
5 : **.....**.....** où * : couleur d'avant plan
. : couleur de fond

3.13 : Fonction INPIX

unit INPIX: function(x,y: INTEGER);

Cette fonction met la postion courante dans la fenętre ŕ (x,y) et renvoie la couleur du point de la fenętre ŕ cette position(0 pour noir et 1 pour blanc).

3.15 : Commandes de saisie et d'affichage de caractčres

3.15.1 : Fonction INKEY

unit INKEY: function: INTEGER;

Retourne le code ascii de la touche tapée au clavier ou la valeur 0 sinon. L'appui sur les touches spéciales (comme SHIFT, les touche F1, F2, ..., CONTROL, ...) renvoient des valeurs négatives. Vous verrez bien par vous-męme quelles sont ces valeurs en faisant un petit programme test.

3.15.2 : Procédure HASCII

unit HASCII: procedure(code_char: INTEGER);

Affiche le caractčre de code ascii code_char avec le coin en haut ŕ gauche du caractčre ŕ la position courante (posx,posy) dans la fenętre. La position courante devient (posx+largeur,posy).

Si code_char=0, une partie rectangulaire de largeur*hauteur est affichée avec la couleur de fond de la fenętre courante et position courante dans la fenętre reste inchangée.

En général la fonte par défaut qui est utilisée sous XWindows a une hauteur de dix points et une largeur de six points.

3.15.3 : Procédure OUTSTRING

unit OUTSTRING: procedure(tab: arrayof CHAR);

Affiche la chaine de caractčre tab ŕ la position courante (posx,posy) de la fenętre. La position courante devient (posx+largeur*longueur_chaine,posy) oů largeur est la largeur de la fonte utilisée.

3.16 : Procédure PUSHXY

unit PUSHXY: procedure;

Sauvegarde le contexte graphique dans une pile, c'est ŕ dire la position courante dans la fenętre,les couleurs de fond et d'avant plan et le style de tracé sélectionné pour cette fenętre.

Chaque Fenętre est dotée de sa pile de sauvegarde qui lui est propre et chaque pile a une profondeur maximale de 16.

3.17 : Procédure POPXY

unit POPXY: procedure;

Restore dans la fenętre courante le contexte graphique situé en haut de la pile de sauvegarde et ce contexte est enlevé de la pile.

4 : Description des commandes de gestion de la souris

4.1 : Procédure STATUS

unit STATUS: procedure(h, v: INTEGER, l, r, c: BOOLEAN);

Cette procédure renvoie la position courante (h,v) du pointeur de la souris ainsi que l'état des boutons de la souris. l,r,c sont respectivement les boutons gauche, droit et du centre de la souris.

Ces valeurs booléennes ont la valeur TRUE si le bouton correspondant est appuyé.

4.2 : Procédure GETPRESS

unit GETPRESS(b: INTEGER; OUTPUT h,v,p : INTEGER, l,r,c : BOOLEAN);

Cette procédure renvoie le nombre p de fois où le bouton sélectionné a été appuyé depuis le dernier appel ŕ cette commande, ainsi que la position (h,v) du curseur la dernič re fois que le bouton considéré a été appuyé.

Le paramętre b permet de sélectionner le bouton ŕ tester :

- 0 : bouton gauche
- 1 : bouton droit
- 2 : bouton du milieu

En sus, la procédure renvoie l'état courant des trois boutons l,r,c.

4.3 : Procédure GETRELEASE

unit GETRELEASE: procedure(b: INTEGER; OUTPUT h,v,p : INTEGER, l,r,c : BOOLEAN);

Cette procédure a la męme fonction que GETPRESS ŕ la différence qu'elle teste le nombre de relâchementss du bouton sélectionné et non l'appui.

4.4 : Procédure GETMOVEMENT

unit GETMOVEMENT: procedure(h,v: INTEGER);

Cette procédure renvoie le mouvement relatif (h,v) du curseur de la souris depuis son dernier appel.

ŕ Pau, le 25 Octobre 1993

Zatsolfolcs...

6. APPENDIX C : PREDEFINED PROCEDURES AND FUNCTIONS

ENDRUN:procedure;
Terminates program execution (ABORT).

RANSET:procedure(x:real);
Initializes random generator (for RANDOM function)

RANDOM:function:real;
Generates uniformly distributed pseudo-random numbers in the interval (0,1).

SQRT:function(x:real):real;
Computes square root of parameter x.

SIN:function(x:real):real;
Computes sinus of parameter x.

COS:function(x:real):real;
Computes cosinus of parameter x.

TAN:function(x:real):real;
Computes tangens of parameter x.

EXP:function(x:real):real;
Computes e**x.

LN:function(x:real):real;
Computes natural logarithmus of parameter x.

ATAN:function(x:real):real;
Computes arcus tangens of parameter x.

ENTIER:function(x:real):integer;
Computes entier part of parameter x.

ROUND:function(x:real):integer;
Computes rounded value of parameter x: ROUND(x)=ENTIER(x+0.5).

IMIN:function(x, y:integer):integer;
Computes minimum of two parameters.

IMAX:function(x, y:integer):integer;
Computes maximum of two parameters.

IMIN3:function(x, y, z:integer):integer;
Returns the minimum of three parameters.

IMAX3:function(x, y, z:integer):integer;
Returns maximum of three parameters.

ISHFT:function(x, k:integer):integer;
Logically shifts x by k bits: left, when k is positive, right otherwise.

IAND:function(n, k:integer):integer;
Returns logical product of parameters (on all bits).

IOR:function(n, k:integer):integer;
Returns logical sum of parameters (on all bits).

XOR:function(n, k:integer):integer;
Returns exlusive sum of parameters (on all bits).

INOT:function(n:integer):integer;
Returns logical complement of parameters (on all bits).

ORD:function(c:char):integer;
Returns number that represents character c (see APPENDIX F).
The following equations are satisfied: CHR(ORD(c)) = c &ORD(CHR(n)) = n

CHR:function(n:integer):char;
Returns character represented by parameter n (see APPENDIX F).

UNPACK:function(s:string):arrayof char;
Returns address of new array object containing characters of the string s.

MEMAVAIL:function:integer;
Returns the size of available memory in the current process (in words).

EXEC:function(cmd:arrayof char):integer;
Calls secondary command processor with cmd as a command string. Exit code is returned as a value of EXEC.

TIME:function: integer;
Returns an integer value indicating the amount of central processor time in seconds used by current process.

RESET:procedure(f:file);
Positions file f at the first component and readies it to reading.

REWRITE:procedure(f:file);
Positions file f at the first component and readies it for output.
The file f becomes empty (eof(f) = true).

UNLINK:procedure(f:file);
Closes and deletes file f (see 3.3.4)

SEEK:procedure(f:file; offset, base:integer);
Positiones file pointer (see 3.3.7)

POSITION:function(f:file):real;
Reads position of file pointer (see 3.3.7)

7. APPENDIX D : ERROR CODES

0 - ***declaration part overloaded
Overflow of compiler data structure of declaration part. Possible reasons: too complicated program structure (too many classes, protection lists, parameter lists,...), too complicated function expressions e.g. f(g(h(...))). It is possible that removing some errors e.g. "unvisible identifier" causes shortening of the program.

10 - ***too many errors
Overflow of error diagnostic table. 1024 first detected errors are printed, but global number of error is equal to number of all detected errors.

41 - ***declaration part overloaded
Comments as for 0.

101 - ':=' expected

102 - ';' expected

103 - 'then' expected

104 - 'fi'/'else' expected

105 - 'od' expected

106 - '(' expected

107 - ')' expected

108 - 'do' expected

109 - identifier expected

110 - too many exits found
Length of sequence exit exit ...exit exceeds level of loop nesting +1.

111 - illegal character

112 - wrong structure of 'if'-statement

113 - 'end' missing

114 - '.' expected

115 - illegal constant in expression
Character constant or text appears in logical or arithmetical expression.

116 - '=' expected

117 - constant expected

118 - ':' expected

119 - unit kind specification expected
Keywords: class, procedure, function, coroutine or process missing in module headline.

120 - 'hidden' or 'close' occurred twice

121 - 'hidden' or 'close' out of a class

122 - 'block' expected

123 - object expression is not a generator
Object expression appearing as instruction is not a generator e.g. new (a).b

124 - 'dim' expected

125 - 'to'/'downto' expected

126 - illegal arithmetic operator

127 - declaration part expected

128 - incorrect identifier at 'end'
Module name after end does not correspond to name in module headline.

129 - wrong structure of 'case'-statement

130 - wrong structure of 'do'-statement

131 - illegal use of 'main'
Name main may be used only as an argument of attach operator; in other cases it is illegal.

132 - 'when' expected

133 - too many branches in 'case'-statement
Number of branches in case instruction is greater than 160.

134 - 'begin' missed

135 - bad option

136 - is it really a loglan program???
There is no Loglan keyword found in source program like: begin, block, unit, class,...

137 - 'block' missed - parsing began
There is no keyword block or program at the beginning of the Loglan program. This message indicates the source line, that is the first compiled line.

138 - 'repeat' out of a loop
The length of sequence: (exit)*repeat exceeds nested depth of the loop.

139 - there is no path to this statement

140 - 'andif'/'orif' mixed

141 - array of 'semaphore' is illegal

142 - wrong handler end
Handler declaration is not ended by instruction end or end handlers.

143 - lastwill inside a structured statement

144 - repeated lastwill
Label LASTWILL appears more than once in the same module.

145 - no parameter specification

146 - wrong register specification

147 - "," expected

191 - ***null program
There is no source program on the input file or there is no module declaration. Causes termination of program compilation.

196 - ***too many identifiers
Entire length of all identifiers and keywords is greater than 3000 characters. This overflow terminates program compilation.

197 - ***too many formal parameters
The length of formal parameter list and declared local variables (in actual module) is greater than 130. This error terminates program compilation.

198 - ***parsing stack overloaded
Too complicated (nested) program structure. This error terminates program compilation.

199 - ***too many prototypes
Too many declarations in program caused overflow of the compiler data structure. This error terminates program compilation.

201 - wrong real constant

202 - wrong comment

203 - wrong character constant

204 - wrong integer constant

205 - integer overflow
Integer constant out of range.

206 - real overflow
Real constant out of range.

211 - identifier too long
Length of identifier is greater than 20 characters.

212 - string too long
Length of string constant is greater than 260 characters.

301 - prefix is not a class id
Prefix name ID is not a class name. It may appear when identifier ID is used earlier (declared more than once).

303 - coroutine/process illegal here as prefix id
Procedure, function or block can't be prefixed by coroutine or process.

304 - hidden identifier cannot be taken id
Identifier ID placed on taken list is on hidden list in the prefixing module.

305 - undeclared identifier id

306 - undeclared type identifier id

307 - type identifier expected id
Identifier ID used in variable or function declaration as a type name, is not declared earlier as a type (but name has been used earlier).

308 - undeclared prefix identifier id

309 - declared more than once id

310 - taken list in unprefixed unit

316 - formal type specification after use id
Formal type ID appears in the parameter list after using this identifier as parameter type e.g. (... x: ID; type ID, ...).

317 - hidden type identifier id
Type name ID is on hidden list in a prefix of one of the modules from SL chain of actual module and it is a nearest declaration of this identifier.

318 - type identifier not taken id
Type name ID is not on taken list in a prefix of one of the modules from SL chain of actual module.

319 - hidden identifier in the list id
Identifier ID from hidden or close list is on hidden list in one of the prefixing modules.

320 - identifier in the list not taken id
Identifer ID from hidden or close list is not placed on taken list in none of the prefixing modules.

321 - identifier cannot be taken id
Identifer ID from taken list is placed on taken list in none of the prefixes.

322 - hidden prefix identifier id
Analogical to 317 error.

323 - prefix identifier not taken id
Analogical to 318 error.

329 - only procedure and function may be virtual
virtual specification appears with class specification.

330 - virtual in unprefixed block/procedure/function

331 - incompatible kinds of virtuals id
Kind of virtual module ID is different from kind of replaced module (e.g. one of them is a function, the other one is a procedure).

332 - incompatible types of virtuals id
Type of virtual function ID is different from type of replaced function.

333 - different lengths of form.param.lists in virtuals id
Virtual module ID and replaced module have different number of formal parameters.

334 - conflict kinds of the 1st level parameters id
In the headline of virtual module ID kind of formal parameter differs from corresponding formal parameter in the headline of replaced module (e.g. type and variable, input and output parameters,.).

335 - incompatible types of the 1st level parameters id
There are formal parameters of different types (function, procedure) in the headline of virtual module ID and in the headline of replaced module on the same position.

336 - different lengths of the 2nd level params lists id
There are formal procedures/functions with different numbers of parameters in the headline of virtual module ID and in the headline of replaced module on the same position.

337 - incompatible kinds of the 2nd level parameters id
There are parameters of different kinds on the same position in the corresponding procedure or function parameters in the headline of virtual module ID and in the headline of replaced module.

338 - incompatible types of the 2nd level parameters id
There are parameters of different types on the same position in the corresponding procedure or function in the headline of virtual module ID and in the headline of replaced module.

341 - ***declaration part overloaded
Analogical to error 0.

342 - ***too many classes declared

343 - ***too many prototypes
Too many modules declared on the same level.

350 - undeclared signal identifier id

351 - hidden signal identifier id
Analogical to error 317.

352 - signal identifier not taken id
Analogical to error 318.

353 - signal identifier expected id
Identifier ID placed in handler declaration as a signal name has not been declared as a signal.

354 - different types of parameters id
In the headlines of signals, that have common handler, parameters of the different types appear on the same position. ID is one of these parameters.

355 - incompatible kinds of parameters id
In the headlines of signals that have common handler, parameters of different kinds appear on the same position. ID is one of these parameters.

356 - different identifiers of parameters id
In the headlines of signals that have common handler parameters of different names appear on the same position. ID is one of these parameters.

357 - incompatible kinds of the 2nd level parameters id
Analogous to error 355 for 2-nd level paramKters.

358 - different types of the 2nd level parameters id
Analogous to error 354 for the 2-nd level parameters.

359 - different lengths of the 2nd level params lists id
There are formal procedures or formal functions with different number of parameters on the same position in the headlines of signals this have common handler. ID is one of these formal parameters/functions.

360 - different lengths of form. param. lists in signals id
There are different number of formal parameters in the signals that have common handler. ID is one of these signals.

361 - non-local formal type cannot be used id
Formal parameter ID of signal is of non local formal type.

362 - repeated handler for signal id
There are more than one handler for signal ID in the same module.

370 - only 'input' is legal here
Formal parameter output or inout is illegal in process.

398 - class prefixed by itself id
Construction unit ID: ID class is not allowed.

399 - cycle in prefix sequence id
ID is a class identifier used in cyclic prefixing i.e. ID prefixes a, a prefixes b, ... , z prefixes ID. This construction is not allowed.

401 - wrong label in 'case' id
Label in case instruction is not a constant.

402 - 'case' statement nested too deeply
Nesting level in case instruction is greater than 6.

403 - too long span of 'case' labels
Range of branches in case instruction is greater than 160.

404 - repeated label in 'case'-statement id
Label ID appears more than once in case instruction.

405 - illegal type of 'case' expression id
Control expression in case statement is not of integer or char type.

406 - different types of labels and 'case' expression

407 - non-logical expression after 'if'/'while' id

408 - real constant out of integer range
Error during conversion of real constant to integer constant.

410 - simple variable expected id
Control variable in for loop is not a simple variable.

411 - non-integer control variable id
Control variable ID in for loop is not of integer type.

412 - non-integer expression id
Expression placed as array index or bound limit in array generation or as step in for loop or as format in write statement should be reducable to integer type.

413 - file expression expected id

414 - string expression expected id

415 - reference expression expected id
Expression placed before dot (remote access), before qua or as a argument of kill or copy statement is not of class type.

416 - array expression expected id

417 - boolean expression expected id

418 - semaphore variable expected

419 - illegal type in 'open'
The type name placed in open is different than TEXT, REAL, INTEGER, CHAR and DIRECT.

420 - variable expected id
Expression placed on the left side of assignment statement or as an argument of read instruction or in array instruction is not a variable.

421 - class identifier after 'new' expected id
Identifier ID placed after new is not a class identifier.

422 - procedure identifier after 'call' expected id

423 - 'new' missing id
Keyword new doesn't appear before class identifier for object generation.

424 - 'call' missing id
Keyword call doesn't appear before procedure identifier for procedure call.

425 - 'inner' out of a class

426 - 'inner' occurred more than once

427 - 'wind'/'terminate' out of a handler

428 - 'inner' inside lastwill

429 - definition cannot be reduced to constant id
Identifier ID placed in constant definition is not a constant.

430 - undefined constant in the definition id

431 - wrong number of indices id
Number of indices in referencing to array element is different from declared number of indices.

432 - index out of range id

433 - upper bound less than lower bound id

434 - too many subscripts id
Dimension of static array ID is greater than 7.

435 - variable is not array id

440 - type identifier expected after 'arrayof' id
Identifier ID placed after arrayof in actual parameter list, corresponding to type parameter is not a type name.

441 - incorrect format in 'write'
There is format for expression of char type or there is double format for expression of type integer or string.

442 - illegal expression in 'write'
Argument of write statement is not of type char, string, integer or real.

443 - illegal type of variable in 'read' id
Argument of read statement is not of type char, integer or real.

444 - no data for i/o transfer
There is only file identifier in I/O instruction.

445 - illegal expression in 'put'

446 - illegal expression in 'get'

448 - 'raise' missing id
There is signal identifier without keyword raise in the context of signal raising.

449 - signal identifier expected id
Identifer ID after keyword raise is not a signal identifier.

450 - illegal procedure occurrence id
Procedure name ID appears in illegal context.

451 - illegal class occurrence id
Class name ID appears in illegal context.

452 - illegal type occurrence id
Type name ID appears in illegal context.

453 - illegal signal occurrence id
Signal name ID appears in illegal context.

454 - illegal operator occurence

455 - wrong number of operands

460 - divided by zero

470 - illegal input parameter id
Actual parameter associated with input parameter is not expression that may have any value: it is e.g. procedure name

471 - illegal output parameter id
Actual parameter corredponded to output parameter is not a variable.

472 - illegal type parameter id
Actual parameter ID associated with type parameter is not a type name.

473 - illegal procedure parameter id
Actual parameter ID associated with procedure parameter is not a procedure name.

474 - illegal function parameter id
Actual parameter ID associated with function parameter is not a function name.

475 - illegal left side of 'is'/'in' id
Left side argument ID of is/in is not a reference expression.

476 - illegal right side od 'is'/'in' id
Right side argument ID of is / in is not a class name.

477 - illegal parameter of 'attach' id
Parameter ID of attach statement is not a reference variable of class object.

478 - illegal type of expression

479 - negative step value

550 - ***stack overloaded
This error may be removed by dividing expressions into subexpressions, making simpler nested callings of arrays, functions, classes and for loops. This error terminates compilation of current module, but other modules will be compiled.

551 - ***too many auxiliary variables needed
Too complicated expressions. This error may be removed by declaration of additional variables and using them as auxiliary variables in expressions.

552 - ***too many auxiliary reference variable needed
Analogical to error 551.

553 - ***statement sequence too long or too complicated
This error may be removed by adding 'goto' statement into sequence of instructions e.g. if false then exit fi, inner, ... or by dividing complicated expression into subexpressions.

554 - ***real constants dictionary overflow
Too many real constant, maybe because of evaluation of expressions built from real constants.

600 - undeclared identifier id

601 - illegal type before '.' id
Expression placed before dot (remote access) is not of class type.

602 - close identifier after '.' id
Identifier ID placed after dot is on close list in the class or its prefix that construct expression before dot.

603 - undeclared identifier after '.' id
Identifier ID placed after dot is not attribute of expression placed before dot. It may be caused by missing declaration or using bad prefix for class constructing expression before dot.

604 - illegal operand type id
One of the arguments in arithmetical expression or in relation is not of arithmetical type.

605 - illegal type in 'div/'mod' term id
Expression identified by ID used as argument of div or mode operation is not of integer type.

606 - incompatible types in comparison id
ID is an identifier of left argument of relation.

607 - unrelated class types in comparison id
ID is an identifier of left argument of relation. Both arguments are of class type and none of these classes prefixes the other one.

608 - string cannot be compared id
ID identifies a string.

609 - incompatible types in assignment/transmission id
ID is an identifier of left side of assignment statement or an identifier of actual parameter in object generation. Types of both sides of instruction or type of formal parameter and type of actual parameter are incompatible.

610 - unrelated class types in assignment/transmission id
Analogical to errors 609 and 607.

611 - constant after '.' id
An attempt to remote access to constant.

612 - this class does not occur in sl-chain id
Class ID appeared in expression this ID, but ID dosn't prefix any module in SL chain of actual module. It may be a cycle.

613,614 - class identifier expected id
For error 613: identifier ID used in expression this ID is not of class type. For error 614: identifier ID used in expression this ID is not name of any type.

615 - illegal type before 'qua' id

Object expression before qua should be of one of the types: class, coroutine, process or simple (not array) formal type.

616,617 - illegal type after 'qua' id

For error 616: identifier ID used after qua is not of any type.
For error 617: identifier ID used after qua is not of class type.

618 - unrelated types in 'qua'-expression id

Identifier ID is a name of class type used after qua. This class type and class type used before qua doesn't prefix each other.

619 - hidden identifier id

Identifier ID used in construction qua ID or this ID is on hidden list in the prefix of one of the module from SL chain of actual module.

620 - not taken identifier id

Identifier ID used in construction qua ID or this ID is not on taken list in any prefix of any module of actual module.

621 - invisible identifier after '.' id

Identifier ID placed after dot is on hidden list or is not on taken list in prefix.

622 - formal parameter list is shorter id

Identifier ID identifies generated object: class, procedure or function. Formal parameters list of this object is shorter than actual parameters list.

623 - formal parameter list is longer id

Analogical to error 622.

624 - actual parameter is not a reference type id

Actual parameter identified by ID in generated object can't be of primitive type: integer, real, boolean or string.

625 - actual parameter is not a type id

Actual parameter identified by ID is not a type, so it can't replace formal type parameter.

626 - procedure-function conflict between parameters id

Actual parameter, identified by ID, that replaced formal parameter in generated object is function whereas formal parameter is a procedure or vice versa.

627 - unmatched heads-wrong kinds of parameters id

ID identifies actual module that replaced formal module. There are parameters of different kinds on the same position in the headlines of these modules. For input - output conflict the agreement of parameter types is checked also.

628 - unmatched heads-incompatible types in lists id

ID identifies actual module that replaced formal module. There are input /output parameters of different types on the same position in the headlines of actual and formal module.

629 - unmatched heads-unrelated class types in lists id
ID identifies actual module that replaced formal module. There are input/output parameters specifying classes of disjointed prefix sequences
in the headlines of actual and formal module.

630 - unmatched heads-different numbers of parameters id
There are different lengths of headlines in actual module identified by ID and formal module.

631 - incompatible types of function parameters id
There are different types of actual function specified by identifier ID and formal function in generated object.

632 - function/procedure expected id
Actual parameter identified by identifier ID is not function/procedure, whereas corresponding formal parameter is function/procedure.

633 - actual function type defined weaker than formal id
Type of actual function identified by ID is weaker defined than formal function type e.g. formal function type is statically defined, whereas actual function type is formal (external) or formal function is class, whereas actual function type is coroutine or process.

634 - unmatched heads-too weak type in actual list id
There are input/output parameters on the same position in the headlines of actual module identified by identifier ID and formal module, but ID is weaker defined than corresponding formal module parameter (see error 633).

635 - standard function/procedure cannot be actual par. id
ID identifies standard procedure/function used as actual parameter.

636 - illegal use of semaphore id

637 - 'semaphore' cannot be used id

8. APPENDIX E : LOGLAN RUNTIME ERRORS

In the following list system signal name, raised after detection of runtime error, is placed in brackets.

ARRAY INDEX ERROR (CONERROR)
: Index outside range during reference to array variable.

NEGATIVE STEP VALUE (CONERROR)

SL CHAIN CUT OFF (LOGERROR)
: Control transfer to object that has SL link cut off earlier in the consequence of kill operation.

ILLEGAL ATTACH (LOGERROR)
: The value of parameter of attach instruction is none or object differs from coroutine.

ILLEGAL DETACH (LOGERROR)
: An attempt to return by detach to coroutine that has been dealocated (by kill).

ILLEGAL RESUME (LOGERROR)
: An attempt to resume an object which is not a process or a process which is running.

TOO MANY PROCESSES ON ONE MACHINE (SYSERROR)
: Number of processes existing on one computer is greater than 64.

INVALID NODE NUMBER (SYSERROR)
: An attempt to create a process on a computer which is not connected to network.

IMPROPER QUA (LOGERROR)
: Error during computing expression of the form: ...x qua a, when 'x' references to none or 'a' doesn't prefix dynamic type object, which is value of 'x'.

ILLEGAL ASSIGNMENT (TYPERROR)
: Type conflict between left and right side of assignment instruction.

FORMAL TYPE MISSING (LOGERROR)
: Formal type is not accessible because of SL cut off.

ILLEGAL KILL (LOGERROR)
: An attempt to deallocate object in SL chain of active object.

ILLEGAL COPY (LOGERROR)
: An attempt to copy non terminated object (i.e. class before execution of return statement, coroutine before execution of end statement...).

REFERENCE TO NONE (ACCERROR)
: An attempt to remote access (by dot) to attributes of non existing object: dealocated or not generated.

MEMORY OVERFLOW (MEMERROR)

INCOMPATIBLE HEADERS (TYPERROR)

Actual parameter list of generated object (procedure, function or class) is incompatible with formal parameter list from module declaration or formal function type is incompatible with actual function type.

INCORRECT ARRAY BOUNDS (CONERROR)
An attempt to generate dynamic array object, when lower bound of index range is greater than upper bound.

DIVISION BY ZERO (NUMERROR)

COROUTINE TERMINATED (LOGERROR)
An attempt to transfer control to a terminated coroutine.

COROUTINE ACTIVE (LOGERROR)
An attempt to transfer control to an active coroutine.

HANDLER NOT FOUND (LOGERROR)
There is no handler for signal declared by user.

ILLEGAL RETURN (LOGERROR)
An attempt to execute return instruction in handler serving system signal.

UNIMPLEMENTED STANDARD PRC. (LOGERROR)
Standard procedure or function is not implemented.

FORMAL LIST TOO LONG (MEMERROR)
Formal parameter list is greater than 40.

ILLEGAL I/O OPERATION (SYSERROR)
Reading after writing, the type of the read/write parameter does not match the type of the file etc.

I/O ERROR (SYSERROR)
System error during I/O.

CANNOT OPEN FILE (SYSERROR)

INPUT DATA FORMAT BAD (SYSERROR)

SYSTEM ERROR (SYSERROR)
Should not occur.

UNRECOGNIZED ERROR

9. APPENDIX F : CHARACTER SET

At the top of the table are hexadecimal digits (0 to 7), and to the left of the table are hexadecimal digits (0 to F). Hexadecimal code of ASCII character is constructed by contatenation of column label and row label. For example, the value of character representing the plus sign is 2B.

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z

	!	!	!	!	!	!	!	!	!
B	! VT	! ESC	! +	! ;	! K	! [	! k	! {	!
	!	!	!	!	!	!	!	!	!
C	! FF	! FS	! ,	! <	! L	! \	! l	! |	!
	!	!	!	!	!	!	!	!	!
D	! CR	! GS	! -	! =	! M	!]	! m	! }	!
	!	!	!	!	!	!	!	!	!
E	! SO	! RS	! .	! >	! N	! ^	! n	! ~	!
	!	!	!	!	!	!	!	!	!
F	! SI	! US	! /	! ?	! O	! _	! o	! DEL	!

where:

NUL Null	DLE Data Link Escape
SOH Start of Heading	DC1 Device Control 1
STX Start of Text	DC2 Device Control 2
ETX End of Text	DC3 Device Control 3
EOT End of Transmission	DC4 Device Control 4
ENQ Enquiry	NAK Negative Acknowledge
ACK Acknowledge	SYN Synchronous Idle
BEL Bell	ETB End of Transmission Block
BS Backspace	CAN Cancel
HT Horizontal Tabulation	EM End of Medium
LF Line Feed	SUB Substitute
VF Vertical Tab	ESC Escape
FF Form Feed	FS File Separator
CR Carriage Return	GS Group Separator
SO Shift Out	RS Record Separator
SI Shift In	US Unit Separator
SP Space	DEL Delete

BIBLIOGRAPHY

Last update: kwiecień 5, 2011

Should you like to read on Loglan and its companion Algorithmic Logic, here it is, a short list of more important papers.

LOGLAN'82

Bartol,W.M., et al.
Report on the Loglan 82 programming Language,
Warszawa-Lodz, PWN, 1984

A.Kreczmar
A micro-manual of the programming language LOGLAN-82,
Institute of Informatics, University of Warsaw, 1984
(there exists a french translation of the above manual)
(both texts are distributed together with this package)

A.Kreczmar, A.Salwicki, M. Warpechowski,
Loglan'88 - Report on the Programming Language,
Lecture Notes on Computer Science vol. 414, Springer Vlg, 1990,
ISBN 3-540-52325-1

/* do you read polish? there exists a good manual of Loglan! */
A.Szalas, J.Warpechowska,
LOGLAN,
Wydawnictwa Naukowo-Techniczne, Warszawa, 1991 ISBN 82-204-1295-1

Some papers devoted to the problems and challenges of Loglan.

Bartol,W.M., Kreczmar, A., Litwiniuk, A., Oktaba, H.,
Semantic and Implementation of Prefixing at Many Levels,
in Lecture Notes in Computer Science vol.148, Springer Verlag, Berlin,
1983, pp.45-80

Krause,M., Kreczmar, A., Langmaack, H., Salwicki,A.,
Specification and Implementation Problems of Programming Languaages Proper for Hierarchical Data Types,
Report 8410 of Institut fuer Informatik und Praktische Mathematik
Christian-Albrechts-Universitaet Kiel, 1984, pp.1-68

Kreczmar,A., Salwicki,A.,
Concatenable Type Declarations, Their Application and Implementation
in: Programming Languages and System Design, in Programming, Languages and System Design
Proc. IFIP TC2 Conference (J.Bormann ed.) Dresden, 1983
North Holland, Amsterdam, 1983, pp.29-41

Cioni, G., Kreczmar, A.,
Modules in high level programming languages
in: Advanced Programming Methodologies (G.Cioni, A.Salwicki eds.)
Academic Press, London, 1989, 247-340

Kreczmar, A.,
On inheritance Rule in Object Oriented Programming
in: Advanced Programming Methodologies
Academic Press, London, 1989, pp. 141-164

Cioni,G., Kreczmar,A., Vitale, R.,
Storage Management
in: Advanced Programming Methodologies
Academic Press, London, 1989, pp.341-366

Cioni, G., Kreczmar, A.,
Programmed deallocation without Dangling References,
IPL, vol. 18 1984, pp. 179-185

Krause, M., Kreczmar, A., Langmaack, H., Warpechowski, M.,
Concatenation of program modules, an Algebraic Approach to the Semantic and Implementation Problems,
in: Proc. Computation Theory, LNCS 208, Springer Vlg, Berlin, 1986, pp. 134-156
full text in: Report 8701 of Institut fuer Informatik und Praktische Mathematik
Christian-Albrechts-Universitaet Kiel, 1987, pp.1-48

Krause, M.,
Die Korrektheit einer Implementation der Modulpraefigerung mit reiner Static Scope Semantik,
Report 8616 of Institut fuer Informatik und Praktische Mathematik
Christian-Albrechts-Universitaet Kiel, 1986, pp.1-139

Langmaack, H.,
On static Semantic of Prefixing (=inheritance),
Talk delivered during the Summer School on Loglan'82, Zaborow, September 1983

Ph.D. thesis (in polish!) related somehow to Loglan project.

Szalas, A.,

On parallel processes, 1984

Gburzynski, P.,
GPR - theorem prover 1982

Petermann, U.,
On file system and signalling exceptions between processes 1987

Oktaba, H.
On Formalisation of the Notion of Reference and its Applications in Theory of Data Structures, 1982

Bartol, W.M.,
Application of Static Structure of Type Declarations and the System of Dynamic Configurations in a Definition of Semantics of a Universal Programming Language 1981

Szczepanska-Wasersztrum, D.,
A logical system for reasoning about exceptions,1990

Litwiniuk, A.I.,
Several algorithms for optimisation of code in presence of nesting, 1988

Jankowska-Puchalka B.
A code genarator generator for an object oriented language, 1992

Algorithmic Logic

There is a monograph:

G.Mirkowska, A.Salwicki,
Algorithmic Logic,
D.Reidel & Polish Scientific Publ., Dordrecht & Warszawa, 1987, ISBN 83-01-06859-0
the book contains a chapter devoted to certain problems of Loglan.

A new book on AL appeared in polish
G.Mirkowska, A.Salwicki,
Logika algorytmiczna dla programistow,
Wydawnictwa Naukowo-Techniczne, Warszawa, 1993 (ISBN 83-204-1296-X).
An english version in preparation.

There are many papers discussing the applications of AL in programming.

Salwicki, A.,
Development of Software from Algorithmic Specifications
in: Advanced Programming Methodologies
Academic Press, London, 1989, pp.1-40

Salwicki, A.,
On algorithmic theory of Stacks,
in Proc. MFCS'78 (J.Winnkowski ed.), LNCS 63, Springer Berlin 1978, pp.

Salwicki, A.,
On algorithmic theory of dictionaries,
Proc. Logic of Programs (E.Engeler ed.), LNCS 125, Springer, Berlin 1981 pp.145-168

Müldner, T., Salwicki, A.,
On algorithmic Properties of Concurrent Programs,
in: Proc. Logic of Programs (E.Engeler ed.), LNCS 125, Springer, Berlin 1981 pp.170-193

Mirkowska,G., Salwicki, A.,
On applications of Algorithmic Logic,
in: Proc. CAAP'86 (P. Franchi-Zanetacci ed.) Springer, 1986 pp.288-306

Mirkowska,G., Salwicki, A.,
Axiomatic definability of programming language semantics,
in: Proc. IFIP Working Conf on Formal Description of Programming Concepts
Ebberup 1986 (M. Wirsing ed.)
Noth Holland, Amsterdam, 1986, pp1-15

Mirkowska,G., Salwicki, A.,
On Axiomatic Definition of Max-model of concurrency,
in Proc. Advanced School on Mathematical Models of Parallelism Rome 1986
(M. Venturini-Zilli ed.) LNCS Springer Berlin

Salwicki, A.,
Algorithmic Theories of Data Structures,
in Proc. ICALP'82 (M.Nilsen, E.Schmidt eds.) LNCS 140 Springer, Berlin, 1982, pp. 458-472

Related literature

on object programming is immense.

Let us quote a few books:

E. Horowitz,
Fundamentals of Programming Languages,
Springer, New York, 1983

O.-J. Dahl, B. Myhrhaug, K. Nygaard,
Simula 67 Common Base Language,
Norwegian Computing Center, Oslo, 1970 the mother of object languages!!

B. Meyer,
Object-oriented software construction,
Prentice Hall, 1988

B. Stroustrup
The C++ Programming Language,
Addison-Wesley, Reading, Mass., 1991

on logics of programs:

see a survey

D. Kozen, J. Tiuryn
Logics of Programs,
in: Handbook of Theoretical Computer Science, vol.B, Formal Models and Semantics
Elsevier, Amsterdam, 1990, pp. 789-998