FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61–6402

Technical Information

KFA-ZAM-TKI-0305
Thomas Eickermann, Wolfgang Frings, Anke Häming, Birgit Reuter
14 May 1997

Generation of Nassi-Shneiderman Diagrams
under Unix with *nassi*

---

1 General

Nassi-Shneiderman diagrams serve to synoptically represent algorithms and program sequences by creating a graph of the elements of structured programming (loops and multi-way decisions) which can give a better overview in the case of complicated algorithms. The representation of algorithms using Nassi-Shneiderman diagrams is rather needed for small-scale programming, where it is a good alternative to flow diagrams.

The *nassi* program developed at ZAM serves to generate such Nassi-Shneiderman diagrams under Unix. *nassi* can be used to convert programs and algorithms available in C pseudocode or in C or PASCAL language into a graph. In this first version, *nassi* only supports a subset of the C programming language. A restriction is that the C parser of *nassi* does not recognize any definitions of variables within functions and is thus unable to exclude these from the diagram. However, since in most cases the diagrams are used in documentations where C pseudocode should be used, the restriction to the analysis of C pseudocode is not particularly relevant. In generating the graph, each function in the input is converted into a separate diagram.

For representation and postprocessing *nassi* provides a convenient interface with which single diagrams can be selected, represented, issued in different output formats and printed. The interface permits all menu-variable global options to be stored in profiles or directly in the source.

A graphics editor allows layout changes of the whole diagram or of single statements or control structures. If diagrams are too crowded, statements and whole structures can be hidden or shifted as a block to a separate diagram. Such changes to the layout and structure of diagrams can be stored again in the source code via special comments and are then directly available for further treatment of the source code.

For output purposes, *nassi* also provides the option of generating source data of the Tgif and Xfig graphics editors in addition to screen output, Encapsulated PostScript graphics and printable PostScript files. This makes it possible to also change and extend diagrams far beyond the functionality of the built-in graphics editor. For the rapid generation of diagrams *nassi* provides a batch option which generates diagrams in the desired output format independent of the interface.

2 Interface

2.1 Call

The program is called with

`nassi` *`[options] [source file]`*

In the current version, C and PASCAL source files can be specified. The options are described under 2.7 on the facing page.

2.2 Usage in batch mode

The call

`nassi -batch` *`[options] source file`*

causes *nassi* to be executed in batch mode. The diagrams are generated and issued without any further user inputs. The desired output format can be specified in a profile (7.1 on page 14).

2.3 Usage under X

After calling nassi, a window appears showing several menus, a subroutine browser displaying the subroutines of a source file, a graphics editor and a status browser.

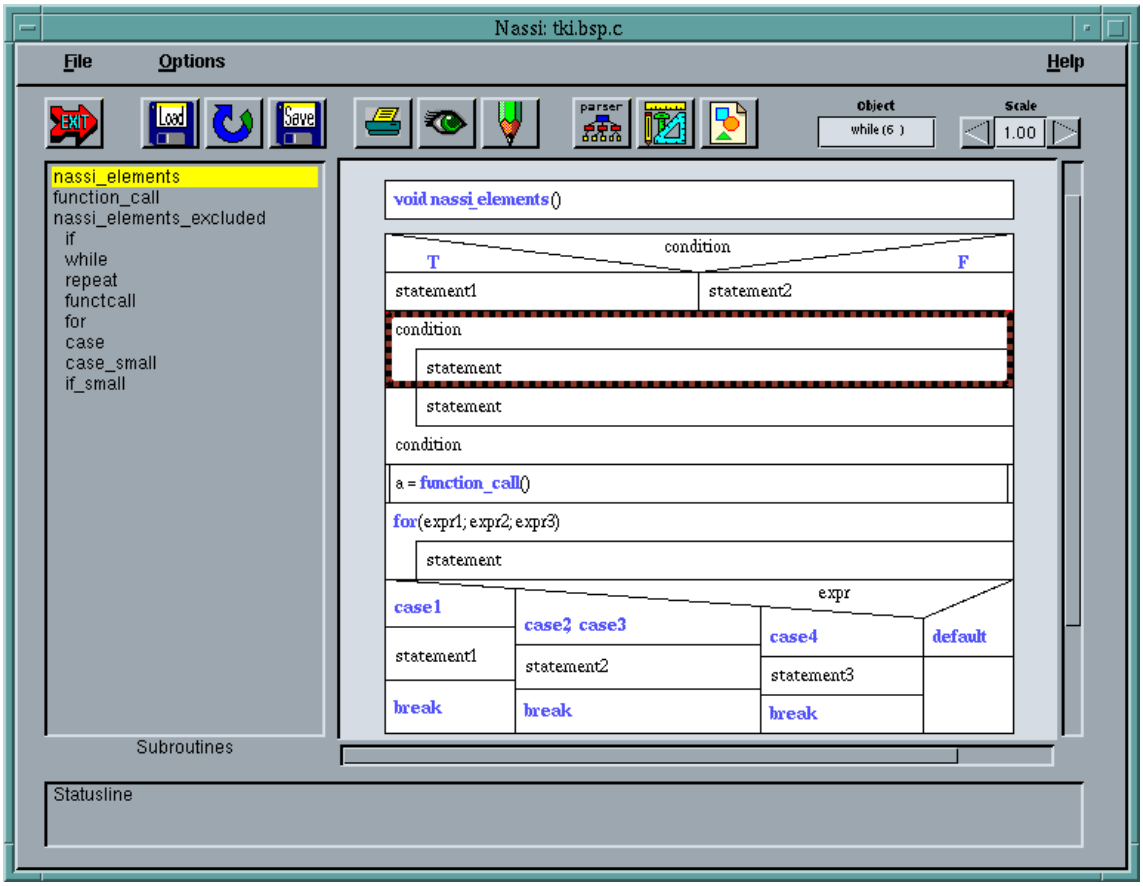


Figure 1: Main window of *nassi*

2.3.1 File menu

Load Source loads a source file that can be specified in a file browser.

Reload Source loads the current source code again.

Save Source saves the source code with the options set by the user, which are filed in the form of comments in the source code. Optionally, all options or only those set for single statements can be stored.

Save Profile	optionally saves the options changed by the user in `$HOME/.nassirc`, `./nassirc` or in a file to be specified. Options that reside in `./nassirc` or `$HOME/.nassirc` are automatically loaded upon call, searching first in the current directory.
Load Profile	loads a profile that can be specified in a file browser.
Print	enables output of the graph on a PostScript printer or to a file.
Preview	displays all marked subroutines with ghostview.
Generate Output	generates an output file corresponding to the output options or, if necessary, several files.
Exit	terminates nassi.

2.3.2 Options menu

This menu serves to specify options for the parser, generator and output modules. The menu items are described in section 3 on the following page.

2.3.3 Help menu

This menu contains the following help texts: *About*, *Introduction* (TKI-0305), *Changes* and *Problems*.

2.4 Button bar

The most common menu items are provided underneath the menu bar in the form of icons that can be activated by clicking on them with any of the mouse buttons. The button row is divided into four groups:

1. *Exit* terminates nassi without any further question
2. *Load, Reload, Save* for loading and saving source code
3. *Print, Preview, Generate* for printing, viewing and generating output files
4. *Parser, Generator, Output options* for setting the global options

2.5 Subroutine browser

After loading a source file, all subroutines are listed in the subroutine browser. Subroutines must be selected for displaying a diagram in the graphics editor and for generating output. A subroutine is selected by clicking with the left mouse button (**M1**) on the corresponding line in the subroutine browser. In order to select several subroutines, keep the shift key pressed (**S-M1**) while clicking on the subroutines. In this case, the graphics editor only displays the subroutine selected first.

2.6 Status browser

The status browser serves to record all actions, warnings and error messages. The number of lines of the browser can be increased through the option `-statuslines`.

2.7 Options and X resources

The following options can be specified when calling the program:

-batch	Program is executed in batch mode
-statuslines *n*	Number of lines of the status browser
-rec_file *file*	If this option was specified, the position and size of the windows are stored in the specified file for the next call. The file name specified is preceded by the path of the user's HOME directory.

-profile *string*	Name of the profile to be loaded
-fontsize *n*	Character size in pixels
-width *n*	Width of the window
-height *n*	Height of the window
-geometry *string*	Size and position of the window

Options can also be specified in the form of X resources in addition to the command line. The corresponding lines of the resources file are built up as follows:

```
nassi*option:  value
```

Logic values as in the `-batch` option are specified as *true* or *false*.
The following additional information may be given in the X resources for storing the geometry data or changing the colors:

Background	background color
BorderColor1	border color (left and upper border)
BorderColor2	border color (right and lower border)
InputColor	color of the input fields
CanvasColor	color of the canvas

An example of the specification of X resources can be found under
`/usr/local/nassi/KFAappdefaults/Nassi.ad` .

3 Options menus

3.1 Options of the parser

The menu for the parser options is activated either via the options menu or directly by the parser button. At first, a window appears in which a parser can be selected. In the current version, parsers are available for C pseudocode and PASCAL. In addition, it is possible to have the parser automatically selected as a function of the file name. This is done by the option ”Autoselect Language” which is active by default. A submenu exists for each parser, in which options can be set that are specific to the respective parser. It is only possible to branch into the menu of the currently active parser. For each parser, a search pattern (”File-pattern”) can be specified in this menu, which is used for automatic parser selection. The PASCAL parser additionally has the option ”hide empty statements”. It defines whether empty statements at the end of a block are to appear in the diagram. Such empty statements are the consequence of a ';' after the last statement in a block. This option is active by default.

3.2 Generator options

When the button for the options of the generator is pressed, a window appears in which the type of generator can be selected. In the current version, however, there are only Nassi- Shneiderman diagrams available for selection. Further options can be changed in the next window that appears when pressing the button with the text ”Nassi-Shneiderman diagram” (see Fig. 2).
The first three options concern the distance of the text from the border of the respective structure element. The values for these distances are specified in points, 1 point corresponding to approx. 0.3 mm. The values can be increased or lowered by 3 / 1 point using the double / single arrows, respectively.
The option *Line skip* switches between single and double line spacing.
The next option *use alternative switch* defines the limit at which the normal representation of a multi-way decision switches to the alternative representation.

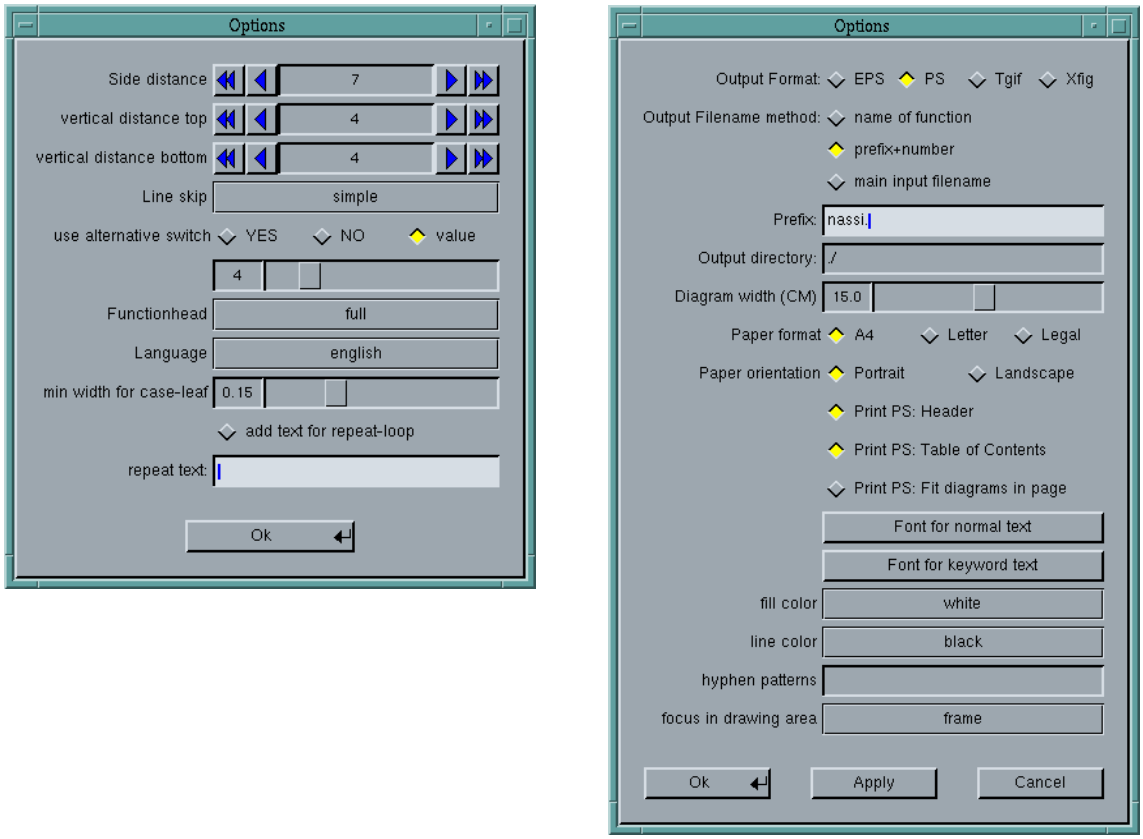


Figure 2: Options for the generator (left) and output (right)

The *function head* can either not be represented at all (*none*), or in a simplified form (*small*) or completely (*full*). In the simplified version, only the function name is plotted, whereas in the complete version all the parameters of the function also appear.
Language selection only relates to the words TRUE and FALSE in the ”if” head.
The next option relates to the minimum width of a case leaf and specifies a percentage value for the total width of the respective case structure to be plotted. The last two options provide the possibility of replacing the keyword WHILE by any arbitrary text in repeat loops.
The default values are as follows:

Side Distance	7	Functionhead	full
vertical distance top	7	Language	english
vertical distance bottom	7	min width for case leaf	0.15
Line skip	simple		

3.3 Options for the output

The third menu contains options which concern the output of the diagrams (see Fig. 2).
Four output formats are currently supported and can be selected in the top menu item.

EPS For each diagram, a separate file is produced containing Encapsulated PostScript code. These output files can be used for incorporating the diagrams into other word processing systems.

PS A PostScript file of all diagrams selected is produced and can be directly printed. In this case, one diagram is generated per page and can be reduced in size, if necessary, so that

it fits on one DIN A4 page. The pages are provided with a current header featuring the page number, name of the input file, date of generation and name of the current function.

Tgif A separate *obj* file is produced for each diagram and can be loaded and further processed with the *Tgif* graphics editor. The representation of the diagram within *Tgif* varies slightly from the print version (PS) since alternate fonts are used again in *Tgif*.

Xfig A separate *fig* file is produced for each diagram and can be loaded and further processed with the *Xfig* graphics editor.

There are again three possibilities for generating the output file name:

name of function With the *EPS* and *Tgif* output formats, the name of the plotted function or of the exclude block is used for the individual files. Blanks are replaced by an underscore (_).

prefix+number The prefix specified in the next input field with an appended serial number is used for the name of the output file. The serial number is re-initialized with 1 for each output call.

main input filename The name of the input file (without extension), to which a serial number is appended again, is used for the name of the output file. The serial number is re-initialized with 1 for each output call.

The input field *Prefix* is given the name which, provided with a serial number, is used for generating the file name using the method *prefix+number*.
The next input field *Output directory* describes the directory in which the output files are stored. In this field, both a relative (e.g. `./out`) and an absolute path (e.g. `/tmp/out`) can be specified.
The option *Diagram width (CM)* describes the desired width of the diagrams in centimetres. The height of the diagrams depends on the contents of the functions.
The next option *Paper format* allows to specify a paper format for output. The selection of the german DIN format *A4* and the formats *letter* and *legal* is possible. The following option *Paper orientation* switches between the upright *portrait* and horizontal *landscape* orientation.
The next three options are only valid, if the output format *PS* is selected.
The first one (*Print PS: Header*) controls the printing of a header on each page which includes the input file name, the function name, the page number, and the current date.
The second one *Print PS: Table of contents* causes *nassi* to insert a table of contents at the beginning of the output.
The third option *Print PS: Fit diagram in page* switches between the scaling and the splitting of a diagram. The first choice scales the diagram down unless it fits on the output page. In the other case the diagram will be split to more pages without any scaling.
The next two buttons (*Font for normal/keyword text*) open new submenus in which the font name, font format, font size and font color can be set.
The attributes *fill color* and *line color* adjust the background color and the line color of the diagrams.
The input field *hyphen patterns* allows the specification of hyphenation proposals to be used for all the texts in the diagrams. The words can be set either directly or via the local options menu of the graphics editor using the switch *save hyphen in global hyphens*.
The last option *focus in drawing area* does not describe an option for the output, but the way of marking the structure element currently focused with the mouse in the graphics editor.
All the attributes of this menu have a global effect, i.e. they apply to all diagrams and can only be overwritten by the local attributes for single statements.

The default values for the individual attributes are as follows:

Output Format	EPS	width of nassi	15.0 cm
Output Filename method	prefix	max height of nassi	25.0 cm
Prefix	`./nassi`	Print PS header	yes
Output directory	`./`		
normal font name	Times	normal font size	11
normal font format	roman	normal font color	black
keyword font name	Times	keyword font size	11
keyword font format	bold	keyword font color	black
fill color	white	line color	black
hyphen patterns		focus in drawing area	frame

4 Graphics Editor

In the right-hand part of the *nassi* window there is an area serving to display and process the diagrams generated. This area always shows the diagram selected in the list of functions. If several diagrams are selected there, only the first diagram is shown.

4.1 Display

Since not all of the fonts are available under X11, alternate fonts may be used in displaying the diagrams, which can have a different size and form. If a font is not available in the desired size, the next smallest font of the same type is displayed. If the character font is not available, Courier is taken as an alternative. The editor only shows the coarse structure of the diagram with respect to fonts, a diagram with the proper fonts and font sizes is obtained using the preview button.
The scroll bars at the right and bottom of the canvas can be used to displace the diagram. Moreover, the keys **PageUp** and **PageDown** and the **Cursor** keys are assigned for displacing the diagram.
The diagram can be directly displaced in the canvas by pressing the middle mouse button (**M2**) and keeping it pressed; every movement of the mouse then shifts the diagram accordingly.
The fields **Object** and **Scale** at the right above the canvas show the type of structure on which the mouse is pointing and the current scaling factor for representation.
Scaling can be varied in a range from 0.25 to 4 using the two buttons beside the field and also by the key combinations **C-M1** and **C-M3** (mouse buttons). This scaling has only an effect on the display in the canvas and no significance for the output.

4.2 Local changes to structures

A local options menu is available for changing the layout and structure for statements or whole control structures of the diagram.
In order to change one or more statements, these must first be selected.
Structures can be selected using the left mouse button (**M1**), and the selected structures are then provided with a (red) mark at the corner points. The selection of the structure is cancelled by pressing this button again. The left mouse button can only be used to select one structure. The selection of another structure cancels all previous selections.
In contrast, additional structures can be selected with the key combination Shift left mouse button (**S-M1**). If the current structure has not been selected up to now, it is added to the set of selected structures, otherwise it is removed again from the set.

The right mouse button can now be used to call various submenus via a small PopUp menu. If only one structure was selected, these submenus show its attributes. Attributes which so far have not been changed in this structure are initialized with *inherit* in the menus.
The submenus do not have to be closed after each change. If a change is acknowledged with the **Apply** button instead of the **ok** button, the window remains open. The open windows are updated when selecting other structures.
The values of the individual inputs can either be chosen from a selection list using the left mouse button (**M1**) or be increased or reduced using the middle (**M2**) or right mouse button (**M3**).
The menus *structure*, *font / color* and *layout / misc* are available.

4.2.1 *structure* menu

This menu (Fig. 3) contains options which change the structural representation of the diagrams. This makes it possible to exclude structures from individual diagrams (exclude) or hide them (hide).

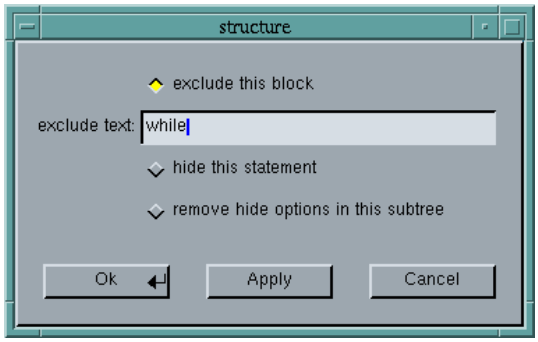


Figure 3: Local *structure* options menu in the graphics editor

Exclude blocks
The upper two options serve to administer Exclude blocks which are helpful in the case of diagrams being too long or too crowded. It is thus possible to remove individual statements or structures from this diagram and put them in a new separate diagram. The main diagram will then only show the scheme of a simple statement with the text `Block: exclude text`. The excluded block then appears as an additional input in the list of functions and can be further treated like a normal function. The block will be given the same heading in the lower-level diagram as in the higher-level one (`Block: exclude text`).
Such an exclusion can only be cancelled again in the local options menu of the higher-level diagram, since the attributes concerning Exclude and Hide are blocked for the function or block statement.
Hiding statements
Individual statements or structures can be hidden with the switch *hide this statement*. They will then not appear any more in the visible display of the diagram. However, since they are then no longer visible in the graphics editor either, this attribute cannot be cancelled from the editor any more. The hiding of structures can only be cancelled in the higher-level structures using the switch *remove hide options in this subtree*.

4.2.2 *fonts / colors* menu

This menu (Fig. 4 on the next page) can be used to change the fonts and colors of texts.
A distinction is made between fonts and colors for normal text and those for keywords. The options of font name, font format, font size and color are available for both of them.
The last two options determine the fill and frame color of the structure.

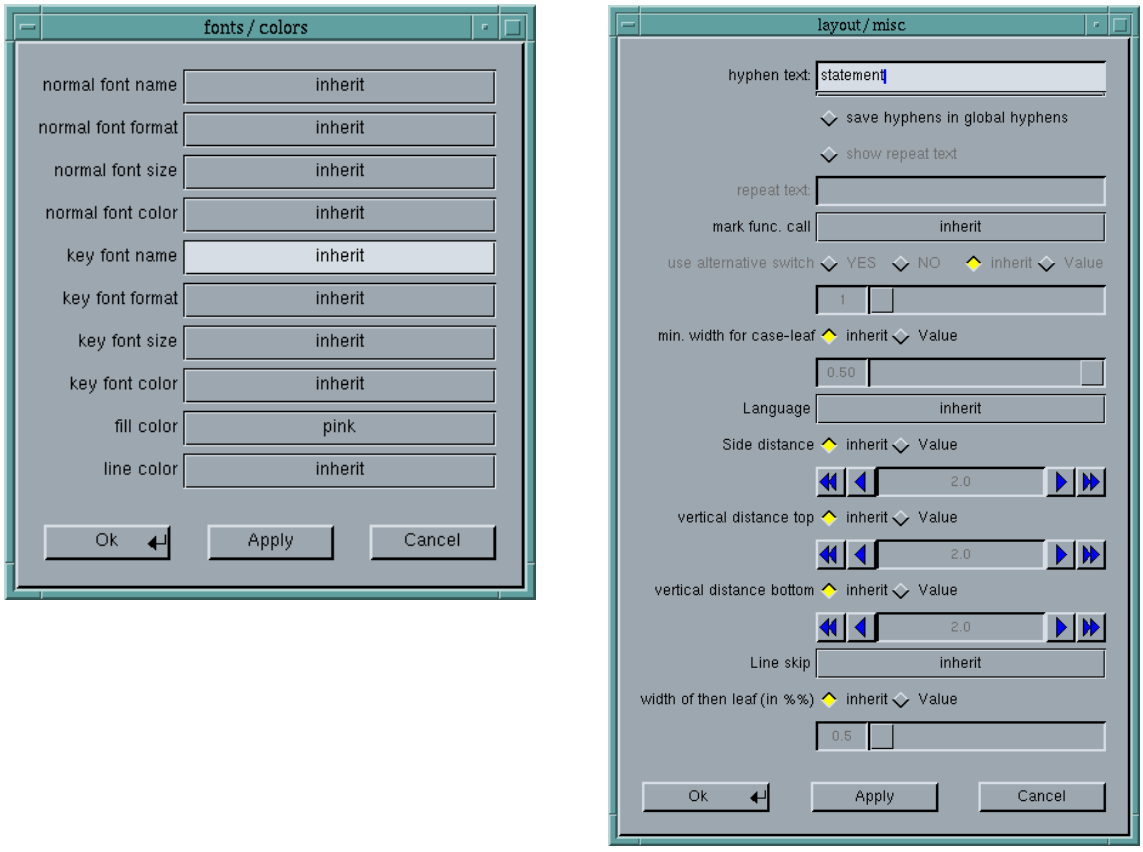


Figure 4: Local *layout / misc (fonts / colors)* options menu in the graphics editor

4.2.3 *layout / misc* menu

This menu (Fig. 4) provides options determining the layout and display of the individual structures. The first two options can be used to insert breaks in the existing text of a statement (see also chapter 5 on the next page).
The next two options determine the representation of the keyword `WHILE` for repeat loops. The option *mark func. call* can decide on whether the structure is to be marked with lateral double lines as a function call. The value *NO_inherit* specifies that neither this nor the structures contained therein are to be marked as a function. The values *auto* and *auto_inherit* turn on the automatic recognition of a function call for this structure (where applicable, with the structures contained therein).
The next option *use alternative switch* defines the limit at which the normal representation of a multi-way decision switches to the alternative representation.
The option *min. width for case-leaf* serves to set a minimum column width for a case leaf.
The option *Language* describes the representation of the keywords `TRUE` and `FALSE` in an *if* structure. The next three options describe the distances of text from the border of the structure. The last option can be used to change line spacing.

4.2.4 Changing the layout of an if-statement

If an if-statement is selected, *nassi* displays an additional marker (in green) at the top of the vertical line between the *then* and *else* leaf of the if-statement. This button can be dragged to a new position. While dragging you see the new triangle for the if-condition. After releasing the mouse button the new diagram layout will be computed and displayed. The hint `COLSPEC_IF` describes the layout of an if-statement in the source code. The value of this hint is the portion of the horizontal width

reserved for the *then* leaf.

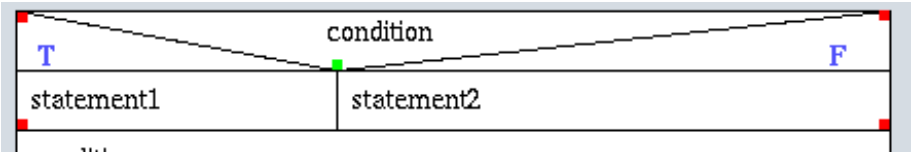


Figure 5: Changing the layout of an if-statement: *the green button rearranges the layout of the if-statement*

4.2.5 Navigation through the call tree of a program

Statements marked as function call with a double line are selectable with a double click of the left mouse button (**Double-M1**). After selecting *nassi* switches to the diagram of the function called in this statement. Additionally you can select block exclude statements in the same way.
The function displayed before this migration is stored in an internal stack. You can go back to the last one with a double click on the whole diagram (e.g. on the head of the diagram). So you can move up and down through the call tree of the input program. A selection in the subroutine browser flushes the internal stack.
In the current version of *nassi* the same functionality is assigned to the **Return** (Enter) key.

4.2.6 Key assignment within the graphics editor

Key	Description
PageUp	moves diagram downwards by half a page
PageDown	moves diagram upwards by half a page
Cursor $\leftarrow, \rightarrow, \uparrow, \downarrow$	moves diagram by one unit in the corresponding direction
left mouse button (M1)	selects / deselects structure
Shift-M1	selects further structure and deselects previously selected structure
middle mouse button (M2)	shifts diagram according to mouse movement
right mouse button (M3)	calls local options menu
Control-M1	reduces diagram size by one increment
Control-M3	enlarges diagram by one increment
Double-M1, Return	goes to the function/block called in this statement

5 Hyphenation of Text

The text of a statement often does not fit in one line so that proper hyphenation must be performed. In principle, lines are only broken between two tokens of the programming language or in the case of blanks within strings.
The user can insert additional optional breaks for texts, which are only observed as required. So-called local breaks can be entered, which only apply to the next structure, and so-called global breaks applying to the entire source code.
A distinction is made between breaks with and without hyphen. In order to obtain a unique identification for the input of breaks, two different symbols are used (# without break symbol, $\sim$ with break symbol).

5.1 Input of breaks

The user has several possibilities for the input of breaks.
Local breaks can be inserted in the source code via comments (`HYPHEN`) or using a special menu in the graphics editor. In this menu, the user can only insert breaks, but not change text, since this is the task of a source-code editor.
Global breaks can be specified in the output options, using also a comment (`G_HYPHEN`) at the beginning of the source code or by a specification in the profile (`G_HYPHEN`).

5.2 Internal treatment of breaks

The graphics editor provides a menu which displays the text pieces belonging to the current structure. Breaks can be inserted in this menu.
The generator only treats these breaks when a line must be broken. It then checks whether a hyphen should and can be inserted. If this hyphen does not fit in the line, the last complete piece of text is written to the next line. If a character string does not have any breaks and does not fit into the line, a hard division is made leaving as many characters as possible (at least 2) in the old line. A hyphen is inserted when a break was made between two letters.

6 Parser

The current version is able to analyse C and PASCAL programs. The aim was to process both ”real” programs and pseudocode. For PASCAL this aim was achieved by a ”pseudocode” syntax, which is an extension of the standard PASCAL syntax. C does not allow this due to a few particularities of the syntax. For this reason, a pseudocode parser was implemented and an ANSI-C parser is planned for a future version of *nassi*.

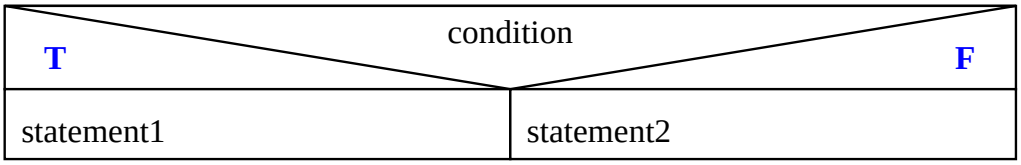
6.1 C–pseudocode

The preprocessor appertaining to C and the special features of the syntax require a very detailed analysis of the source code if an interpretation is desired which is correct in all points. As a rule, however, such an analysis will not accept ”pseudocode”. In order to be able to analyse both pseudocode and at least simple C programs with a parser, a syntax with the following features was defined:
A largely ANSI-C-conforming text is expected outside function definitions (i.e. for variables, type and function declarations). Within functions, on the other hand, a relatively free input is permissible.
The restrictions and particular features involved in this compromise will be described in the following.

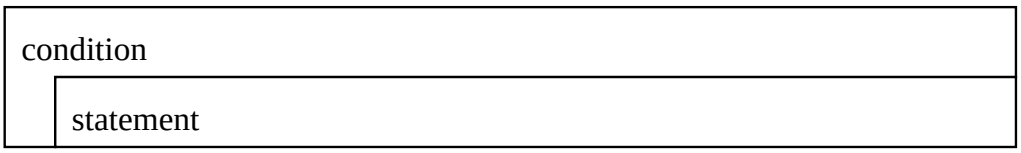
6.1.1 Mapping the C-structure elements on the structogram elements

The programming language C contains types of statements which do not directly correspond to the Nassi-Shneiderman structure elements. Although a correct conversion is basically possible, it may lead to extremely unclear diagrams in some cases (especially with the `goto` statement), so that a different approach was selected.
The following types of C statements essentially correspond to the Nassi-Shneiderman structure elements:

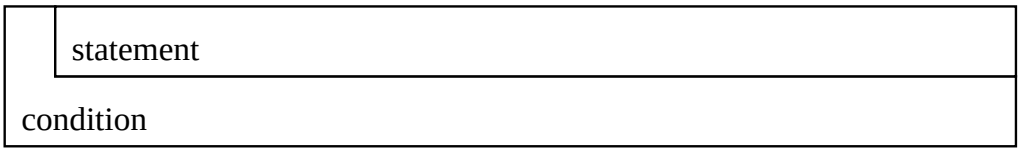
- `if( condition ) statement1 else statement2`



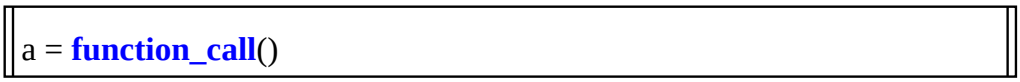
- `while( condition ) statement`



- `do statement while( condition )`
  It should be noted here that the Nassi-Shneiderman structure element corresponds to a ”`do ... until`” . The condition can be preceded by an arbitrary text using the generator option ”add text for repeat-loop”, e.g. by a negating ”`!`”.

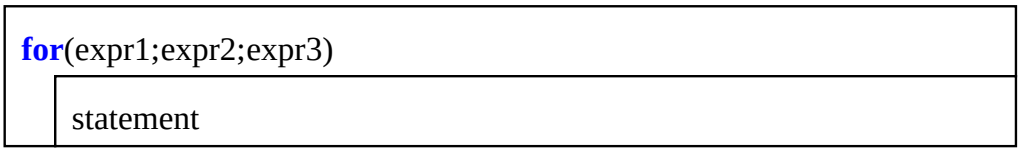


- Calls for functions defined within the program are marked accordingly.

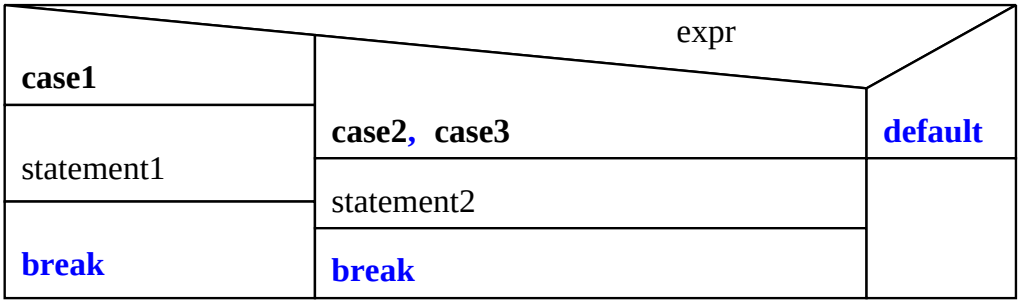


In order to correctly represent the following types of statements, they must be generally decomposed into several statements. This was omitted for the sake of clarity.

- The `for` loop: `for( expr1; expr2; expr3)` statement is represented like the `while` loop with a starting condition.



- The `switch` statement in C does not correspond to the multiple choice structure element according to Nassi-Shneiderman , but is rather related to the ”computed-goto” of FORTRAN. However, since it is generally used like a multiple choice by inserting the `break` statement, it is always represented in the diagram as such.



There is no direct equivalent of the statements `break` `continue` and `goto`. The conversion

into Nassi-Shneiderman elements depends on the context in which they are used. These statements are represented as simple instructions in the diagrams.

6.1.2 Restrictions and special features

The following restrictions apply to the analysis of ”real” C programs:

- As in ANSI-C, the type of a function must be explicitly specified in declarations/definitions (no implicit *int* as in K&R-C).
- Preprocessor directives are disregarded, i.e.:
  - macros are not expanded
  - include-files are ignored
  - conditional compilation is not performed
- Declarations of variables within functions cannot be distinguished from statements. They are therefore represented as statements, but can be suppressed by the graphics editor (see ”hide” in 4.2.1 on page 8).
- If a pointer is used on a function in a statement, this statement appears as a function call in the diagram. The same applies to variables whose name is the same as that of a function. If necessary, errors can be corrected in the graphics editor using the option *mark.func.call* (see 4.2.3 on page 9).

The following should be noted when generating pseudocode:

- Keywords, braces and semicolons retain their normal meaning.
- Brackets (`( ) [ ]`) must be used in pairs and nested correctly.
- Double and single quotes (`" '`) enclose strings that are not further analysed.
- The format of C statements is free — except for the above three items.
- Keywords and semicolons preceded by a \ can be used in the text.
- German umlauts can be entered directly or generated in a similar way as in LaTeX, that is e.g. `\"A` for ”Ä” oder `\"s` for ß.
- The sequence `\.`*X* generates the character *X*. In this way, for example, brackets not occurring in pairs can be generated or blanks forced in a diagram.
- The sequence `\\` is replaced by a line break in the diagram.

6.2 PASCAL

PASCAL also contains types of statements which do not directly correspond to the Nassi- Shneiderman structure elements. They are treated similarly to the corresponding C statements.
The following types of PASCAL statements correspond exactly to the Nassi-Shneiderman structure elements(cf. Figures in 6.1 on page 11):

- `IF condition THEN statement1 ELSE statement2`
- `WHILE condition DO statement`
- `REPEAT statement_list UNTIL condition`
- `CASE expression OF ... END`
  Although not defined by the standard, the specification of a default leaf (with the keyword `OTHERWISE` or `ELSE`) allowed in many compiler implementations is also accepted here.
- Calls for functions and procedures defined within the program are marked accordingly.

In order to correctly represent the following types of statements, they would in general have to be decomposed into several statements. This was omitted for the sake of clarity.

- The `for` loop: `FOR var := val1 TO/DOWNTO val2 DO statement` is represented like the `WHILE` loop.
- The `WITH var_list DO statement` instruction is not a structure element, since it merely declares abbreviating notations for `statement`. Nevertheless, such constructs are represented here like a `WHILE` loop.

There is no direct equivalent of the `goto` instruction. The conversion into Nassi-Shneiderman elements depends on the context in which it is used. This statement is represented like a simple instruction in the diagrams.

6.2.1 Restrictions and special features

The following restrictions apply to the analysis of ”real” PASCAL programs:

- If a variable whose name is the same as that of a function is used in a statement, the statement appears as a function call in the diagram (unless it is an assignment of the return value of the function itself). If necessary, errors can be corrected in the graphics editor using the option *mark func.call* (see 4.2.3 on page 9).

The following should be noted when generating pseudocode:

- Keywords, colons and semicolons retain their normal meaning.
- Brackets (`( ) [ ]`) must be used in pairs and nested correctly.
- Single quotes (`'`) enclose strings that are not further analysed.
- The format of PASCAL statements is free — except for the above three items.
- Keywords, colons and semicolons preceded by a \ can be used in the text.
- German umlauts can be entered directly or generated in a similar way as in LaTeX, that is e.g. `\"A` for ”Ä” or `\"s` for ”ß”.
- The sequence `\.`*X* generates the character *X*. In this way, for example, brackets not occurring in pairs can be generated or blanks forced in a diagram.
- The sequence `\\` is replaced by a line break in the diagram.

7 Options in the Profile and Source Codes

7.1 Profile

A possibly existing profile is loaded when calling *nassi*. The options contained therein apply to the entire session unless they are overwritten by other options. In Profile the options have the following appearance:

```
Keyword = value
Keyword = value
...
```

Each option is written in a separate line. If the value of an option contains blanks, it must be provided with ”...”.
The keywords with their values are described in 8 on page 16. If the Profile has the name `./nassirc` or `$HOME/.nassirc`, it is automatically loaded upon call, searching first in the current directory. Otherwise, it must be specified with the option *-profile* or via the menu item *File→Load Profile*.

7.2 Comments in the Source code

The user can change the appearance of individual statements by means of special comments in the Source code. These comments contain e.g. information about fonts and colors. The parser stores these comments in a list of hints, which is used by the generator.
However, the user must himself make sure that the values are correct.

7.2.1 Appearance of the comments

The comments begin and end like normal comments of the programming language. They are reconized as *Nassi special comments* (`NSC`) if the keyword `/* NSC:` or `/* NSC_GLOBAL:` is the first token inside the comment.

One or several pairs of keywords with corresponding values and separated by blanks must follow. The value may be enclosed in ".
A list of pairs is defined in 8.

7.2.2 Position and validity of comments

The NSC comments are always placed in front of the statement structure to which they are to apply. If they are placed within a statement, the parser will produce a syntax error message.
The comments cannot be nested.
If a comment is placed before a function name, it applies to the function head and the block below the function head. If it is placed between function name and block ("{" in C and "BEGIN" in PASCAL) it only applies to the block.
Comments preceded by NSC_GLOBAL: set options that are applicable to the entire Source code. Only one comment of this type is allowed. It must be placed before the first function in the Source file.

7.3 Hierarchy of the options

The different possibilities of specifying options require a defined hierarchy of execution and validity. The following list shows the order in which the options are read. If a keyword occurs several times, the value read last is valid.

1. Profile
2. starting area of the Source code
3. comments in the Source code

8 Annex: List of Keywords and Values

The following keywords are so far defined in order to set options:

Keyword	Possible values	Explanation
LINE_SKIP	SIMPLE / DOUBLE	line spacing
FUNCTIONHEAD	NONE / SMALL / FULL	type of function head
MINWIDTH	0.0,..,1.0	minimum width of a case or "if" leaf; percentage of total width
FONT_NAME	Helvetica, NewCenturySchlbk, Times,Courier	character set for normal text
FONT_FORMAT	roman,bold,italic, bold-italic	character set format for normal text
FONT_SIZE	8,..,34	character set size for normal text
FONT_COLOR	red, black,yellow,...	color for normal text
KEY_FONT_NAME	Helvetica,NewCenturySchlbk, Times,Courier	character set for keywords
KEY_FONT_FORMAT	roman,bold,italic, bold-italic	character set format for keywords
KEY_FONT_SIZE	8,..,34	character set size for keywords
KEY_FONT_COLOR	red, black,yellow,...	color for keywords
SIDE_DISTANCE	2,..,20	distance of text from lateral border (in points)
VERTICAL_DIST_BOT	2,..,20	distance of text from bottom border (in points)
VERTICAL_DIST_TOP	2,..,20	distance of text from top border (in points)
FILL_COLOR	red, black,yellow,...	color for background filling
LINE_COLOR	red, black,yellow,...	line color
HIDE	YES / NO	hiding the structure
EXCLUDE	YES / NO	structure is excluded
EXCLUDETEXT	any string	text for excluded structure
LANGUAGE	ENGLISH / GERMAN	language for keywords TRUE/FALSE or WAHR/FALSCH in an "if" structure
G_HYPHEN	any string	specification of global hyphenation proposals
HYPHEN	any string	specification of local hyphenation proposals
OUT_WIDTH	1.0,..,18.0	width of a diagram (in cm)
OUT_MAXHEIGTH	1.0,..,29.7	maximum height of a diagram, has no meaning in this version (in cm)
OUT_FORMAT	PS, EPS, TGIF	type of output
OUT_METHOD	"name of function", "prefix+number", "name of main input file"	type of output file name
OUT_PREFIX	"nassi."	prefix preceding the name of the output files
OUT_DIR	"./"	directory in which the output files are stored
PS_HEADER	YES / NO	adding a header for PostScript output
FOCUS	frame, invert, line, block, debug	way in which the statement under the mouse should be marked
COLSPEC_IF	0.0, ..., 100.0	percentage of the horizontal width reserved for the *then* leaf on an if-statement