ffortid Program Ver 3.0 Decomposition Manual

Harry I. Hornreich

Technion - Israel Institute of Technology

ffortid Program Ver 3.0 Decomposition Manual

Harry I. Hornreich

Technion - Israel Institute of Technology

ABSTRACT

This manual describes the decomposition of the ffortid dtroff postprocessor program into Software Units (SWU). Each SWU has a page describing its name, number, type, source code, scope diagram, capabilities, interface and service flow diagram (SFD). A SWU is any functional piece of software code. It can provide different kinds of services to other SWU depending on the software language semantics it is written in. Example services (in C) are declarations, definitions, global variables, functions & procedures. The SWU page captures its origin (its source code), its scope (the SWU it is composed of), its capabilities i.e. the services it offers to other SWU, its interface i.e. how its services can be accessed and how they can, might or should affect the enviroment in which they are used. The SFD captures graphically the relationships between the sub-units the SWU is composed of and its enviroment. The SFD shows not only the flow of data but also the use of declarations, definitions, procedure calls and any other kind of software service.

1. Overview

1.1. ffortid History

The first author of ffortid was Cary Buchman, an M.Sc. student at UCLA, and the first version was written during the years 1983-1984. That version could handle only Hebrew though it did have some hooks for Arabic that proved to be useless. The first external customer was the Hebrew University. Mulli Bahr a guru from HU modified the code to optimize the output in 1986 during a visit to UCLA. Johny Srouji extended ffortid for Arabic in 1989-1991.

Version	Years	Author	From	Major Modification
1	1983-1984	Cary Buchman	UCLA	Hebrew
2	1986	Mulli Bahr	HU	Output Optimization
3	1989-1991	Johny Srouji	Technion	Arabic

An up to date manual page of ffortid can be found at the end of this manual.

1.2. ffortid File Statistics

Num	File	Length (lines)	Functions
1	lex.h	30	-
2	lex.dit	37	-
3	token.h	34	-
4	macros.h	20	-
5	connect.h	256	-
6	table.h	18	-
7	dump.c	704	10
8	lines.c	296	6
9	main.c	506	1
10	misc.c	129	5
11	width.c	480	10
Total		2510	32

2. ffortid Program Software Units Summary

Num	Name	Type	Size (lines)	Low-Level
1	ffortid	Program	3422	
2	Dump	Module	1044	
3	Lines	Module	398	
4	Main	Module	1299	
5	Misc	Module	201	
6	Width	Module	480	
7	token.h	Declarations source file	34	*
8	lex.h	Definitions source file	30	*
9	macros.h	Definitions source file	20	*
10	connect.h	Data file	256	*
11	dump.c	Source file	704	
12	table.h	Declarations source file	18	*
13	lines.c	Source file	296	
14	lexer	Lex generated source file	691	
15	main.c	Source file	506	
16	misc.c	Source file	129	
17	width.c	Source file	480	
18	dump_defin	Definitions block	34	*
19	dump_line	Procedure	103	*
20	reverse_line	Procedure	83	*

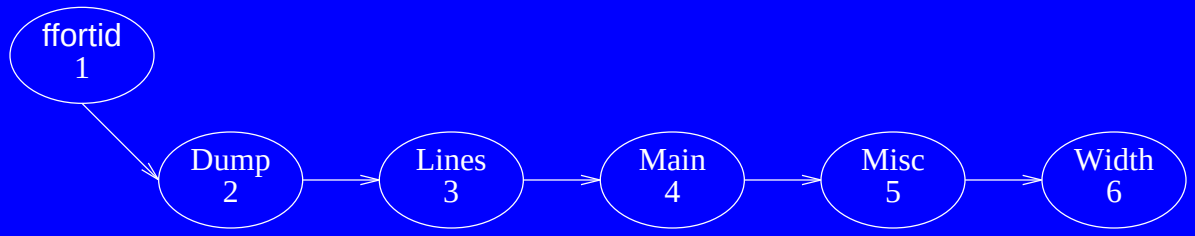
Table continued on next page ...

Num	Name	Type	Size (lines)	Low-Level
21	recalc_horiz	Function group	463	
22	print_line	Procedure	21	*
23	lines_defin	Definitions block	35	*
24	new_free_token	Function group	85	*
25	insert_tokens	Procedure group	52	*
26	put_tokens	Procedure group	124	*
27	lexer.dit	Lex source file	37	*
28	main_defin	Definitions block	58	*
29	main	Function	448	*
30	new_font	Procedure	41	*
31	font_info	Procedure	41	*
32	out_of_memory	Procedure	17	*
33	yywrap	Function	13	*
34	width	Function	17	*
35	width_defin	Definitions block	47	*
36	init_dev_font	Procedure group	229	*
37	width_calc	Function group	122	*
38	debug_error	Procedure group	82	*
39	recalc_horiz_2	Procedure	53	*
40	calc_total	Function	48	*
41	stretch	Function group	361	*

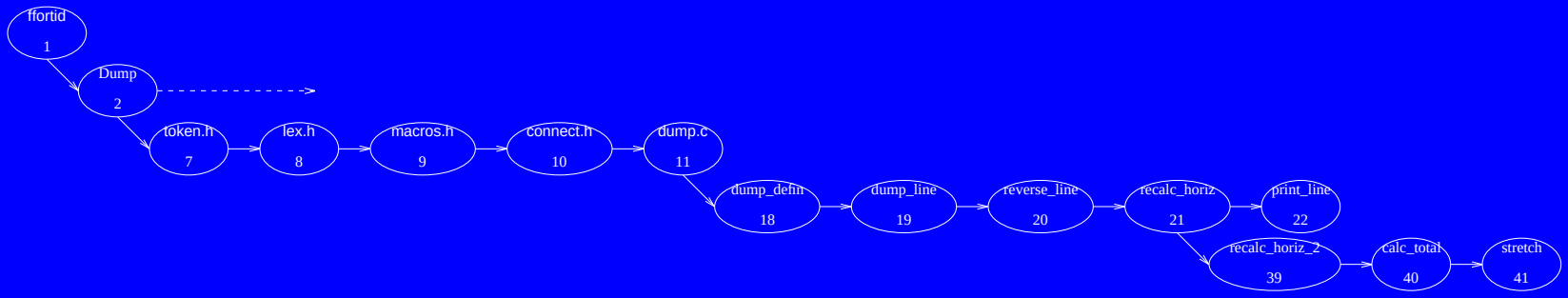
3. Software Units Overview and Pages

In the following pages is an overview of all the SWU in the decomposition followed by a SWU page describing each one in depth. The numbers used in the overview are the SWU numbers given in the previous page. The notation followed in the SFD in the SWU pages is described in Appendix A.

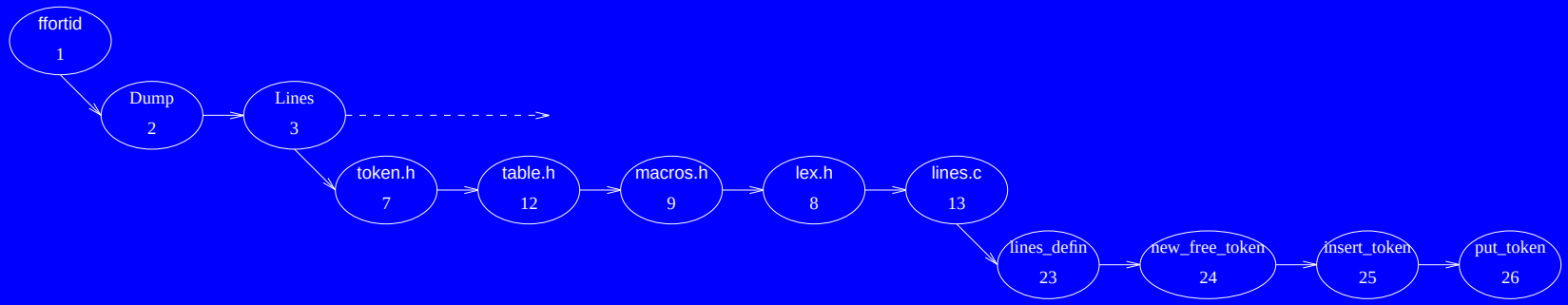
ffortid Software Units Decomposition - Top Level



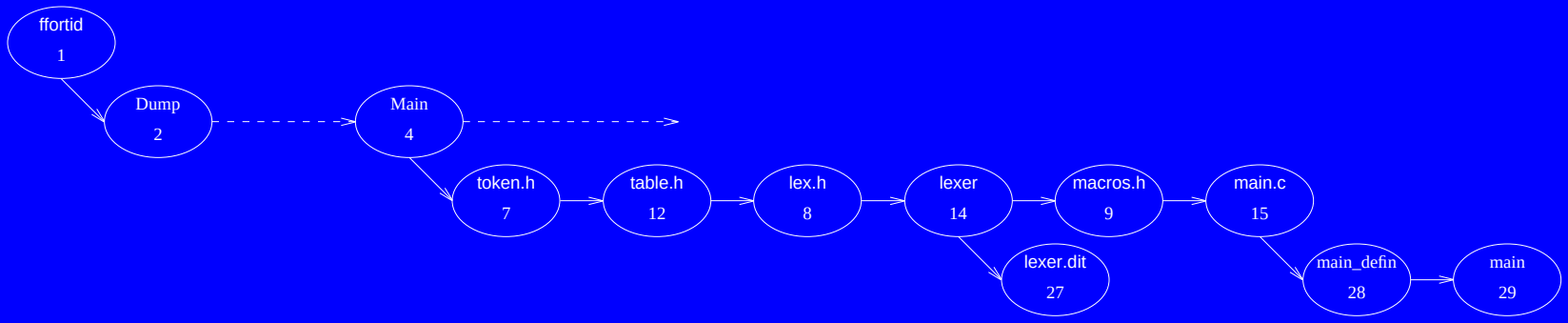
ffortid Software Units Decomposition - Cont 1/5



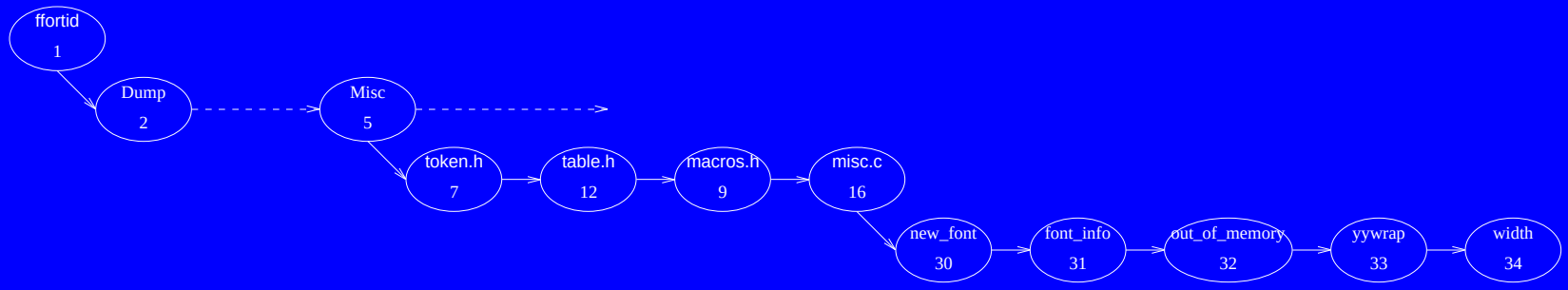
ffortid Software Units Decomposition - Cont 2/5



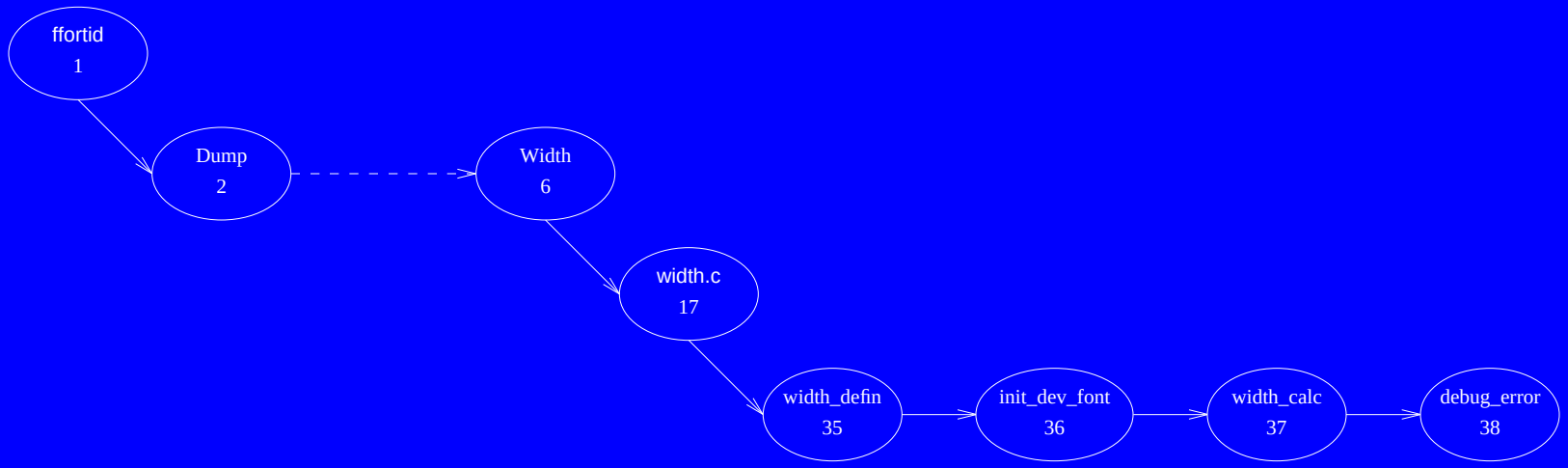
ffortid Software Units Decomposition - Cont 3/5



ffortid Software Units Decomposition - Cont 4/5



ffortid Software Units Decomposition - Cont 5/5

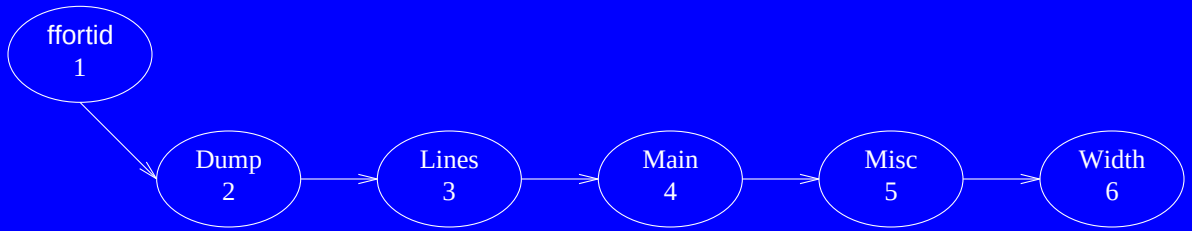


Software Unit #1 — ffortid

1.1 Software Unit Type

Program. (lex.h, lex.dit, token.h, macros.h, connect.h, table.h, dump.c, lines.c, main.c, misc.c, width.c)

1.2 Scope Diagram



1.3 Capabilities

ffortid takes from its standard input dtroff output, which is formatted strictly from left-to-right, finds occurrences of text in a right-to-left font and rearranges each line so that the text in each font is written in its proper direction. Additionaly, ffortid left and right justifys lines containing Arabic & Persian fonts by stretching connections in the words instead of inserting extra white space between the words in the lines.

1.4 Interface

command line options:

ffortid [–**r***font-position-list*] ... [–**w***paperwidth*] [–**a***font-position-list*] ...
[–**s**[**n**|**f**|**l**|**a**]] ...

The **-r***font-position-list* argument is used to specify which font positions are to be considered right-to-left. The **-w***paperwidth* argument is used to specify the width of the paper, in inches, on which the the document will be printed. The **-a***font-position-list* argument is used to indicate which font positions, generally a subset of those designated as right-to-left (but not necessarily), contain fonts for Arabic, Persian or related languages. The **-s** argument specifies the kind of stretching to be done for all fonts designated in the **-a***font-position-list*

1. **-sn** — Do no stretching at all for all the fonts.
2. **-sf** — Stretch the last stretchable word on each line.
3. **-sl** — Stretch the last stretchable word on each line up to a maximum length.
4. **-sa** — Stretch all stretchable words on the line by the same amount.

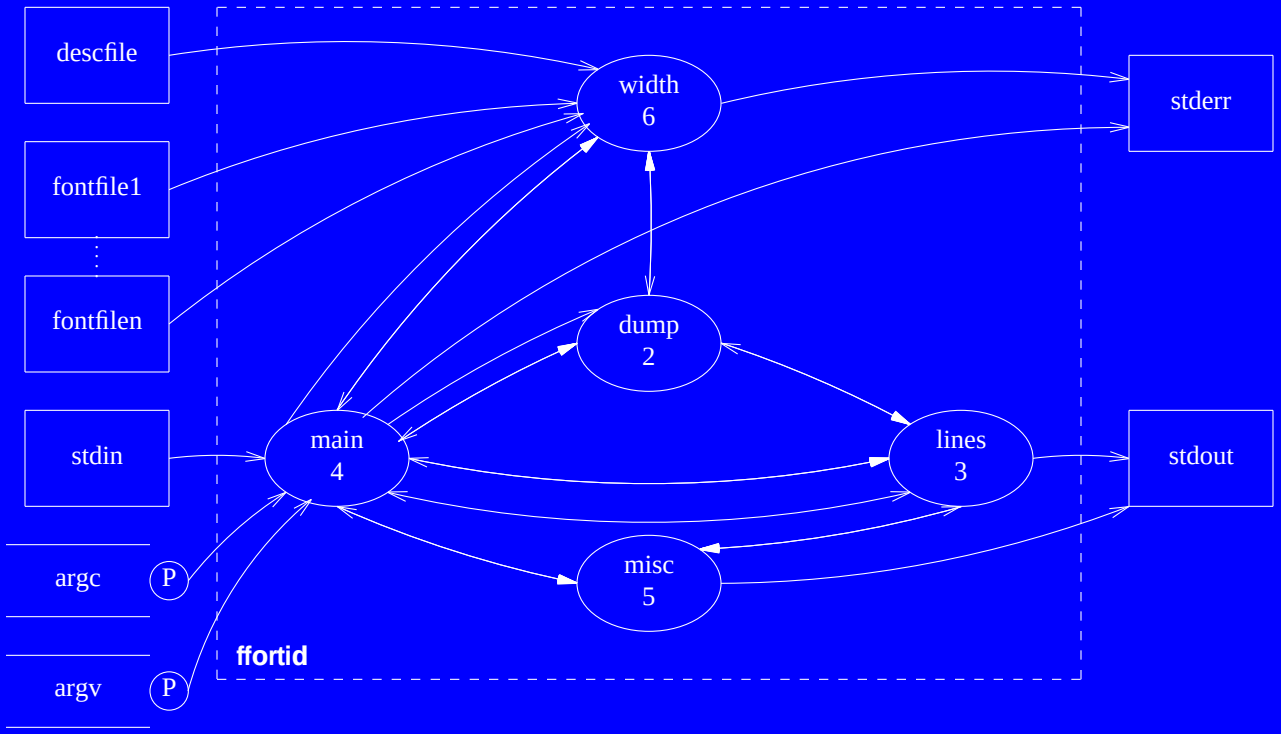
The default is no stretching at all.

Manual connection stretching can be achieved by using explicitly the base-line filler character **\(hy** in the dtroff input. It can be repeated as many times as necessary to achieve the desired connection length.

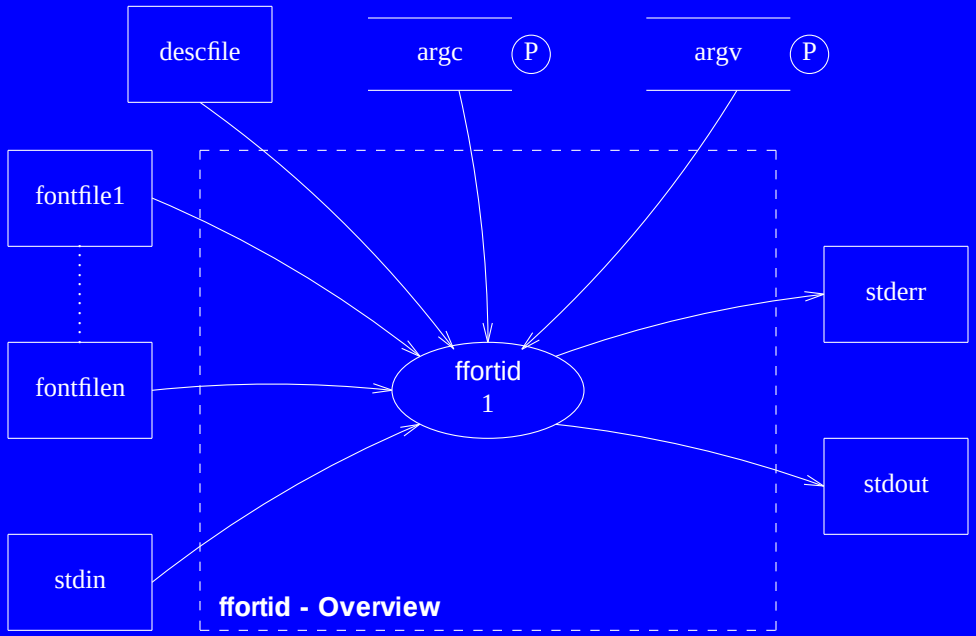
Side effects:

1. ffortid reads dtroff output from stdin and prints dtroff output to stdout.
2. ffortid prints encountered errors to stderr and halts program.
3. ffortid allocates and frees memory from the heap. If out of heap memory ffortid prints a ``out of memory´´ message to stdout and halts program.

1.5 Service Flow Diagram



1.6 Service Flow Overview Diagram

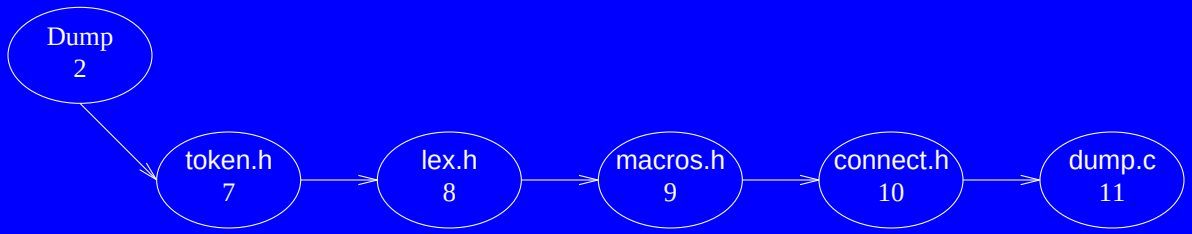


Software Unit #2 — Dump

2.1 Software Unit Type

Module. (token.h, lex.h, macros.h, connect.h, dump.c)

2.2 Scope Diagram



2.3 Capabilities

Contains routines that dump and reverse internal token lines while taking care of such issues as stretching and text direction.

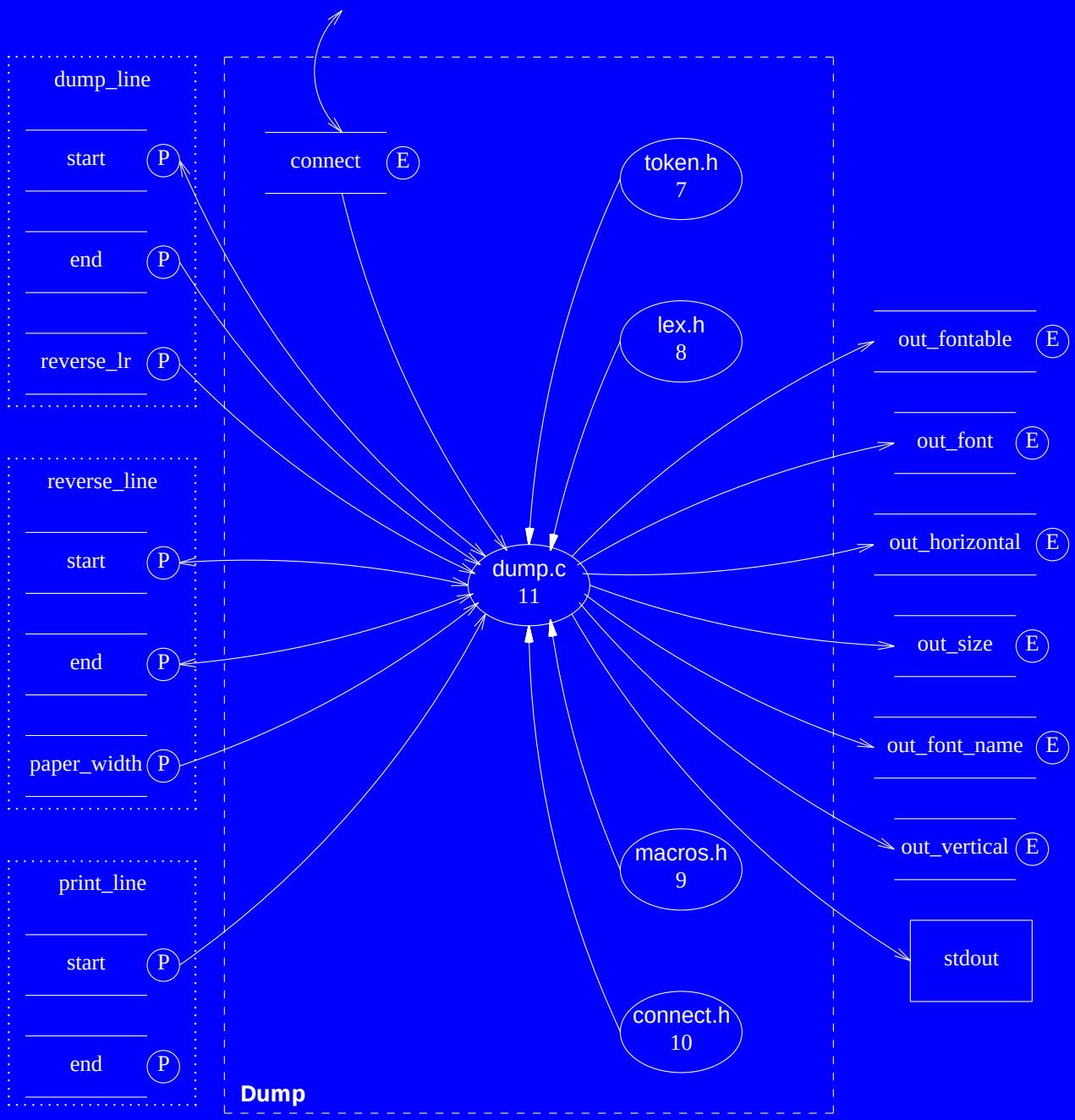
2.4 Interface

Globals:
connect - array of all **cnct** structures.

Functions:
dump_line - stretches and dumps an internal token line while reversing tokens of the specified lr direction.
reverse_line - reverses an internal token line while preserving zero width characters position.
print_line - prints an internal token line to stdout. Used for debugging.

Side effects:
1. **dump_line** prints passed token line to stdout and frees the heap memory used by it.
2. **dump_line** changes the values of external vars: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
3. **reverse_line** changes the tokens in the passed token line.
4. **print_line** prints the passed token line to stdout.

2.5 Service Flow Diagram

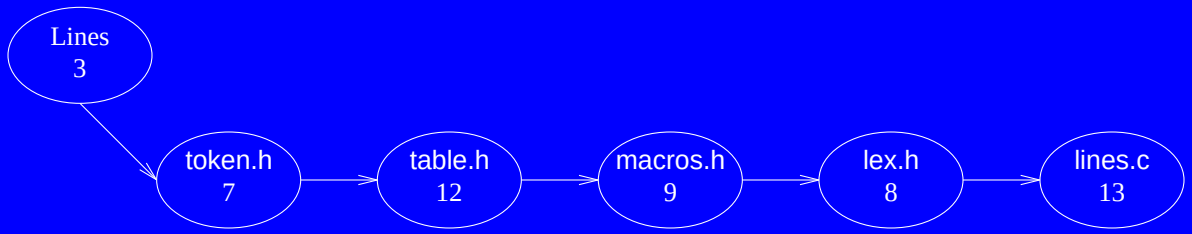


Software Unit #3 — Lines

3.1 Software Unit Type

Module. (token.h, table.h, macros.h, lex.h, lines.c)

3.2 Scope Diagram



3.3 Capabilities

Contains routines to allocate, free, insert and print internal tokens.

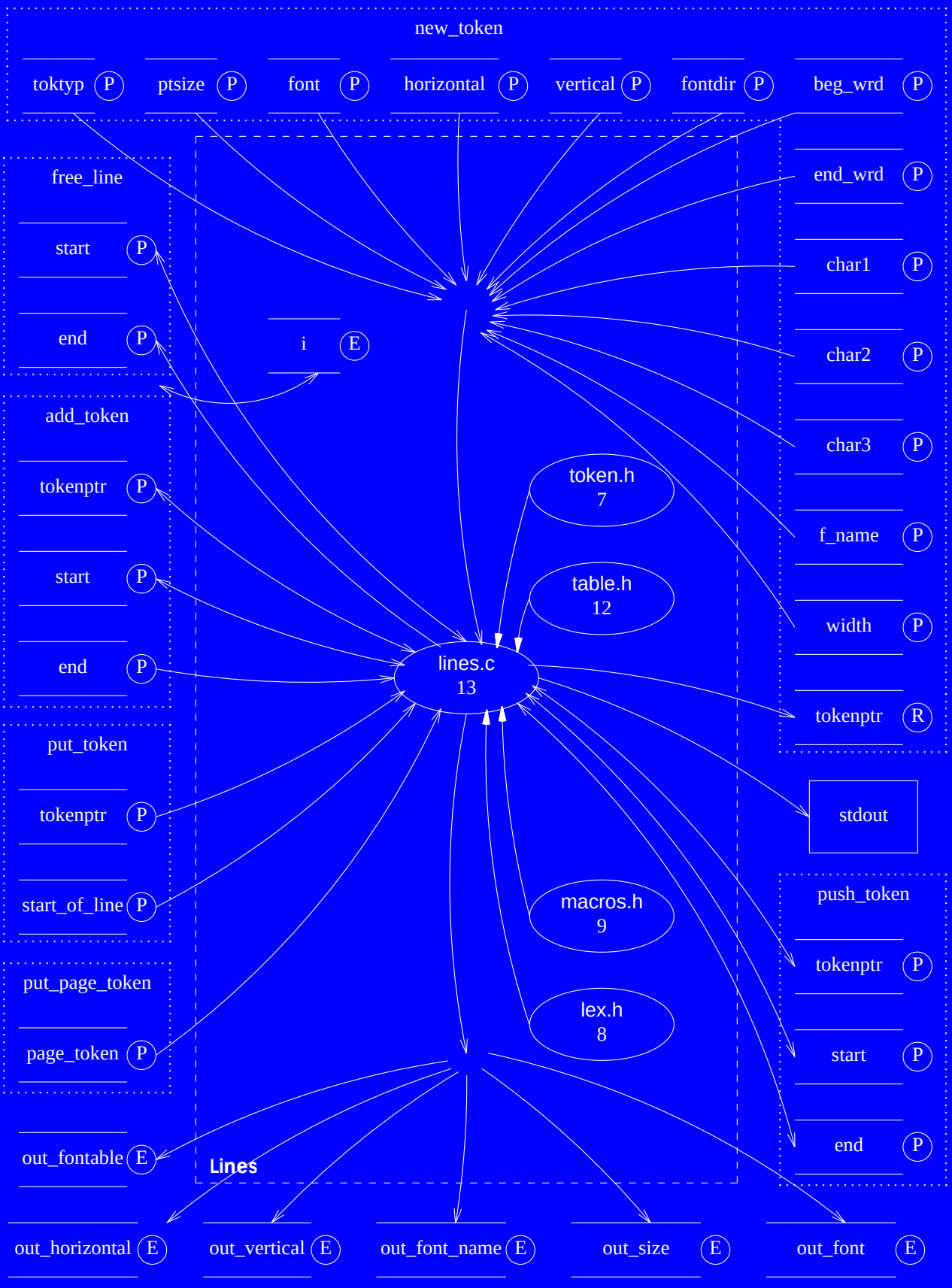
3.4 Interface

Globals:
i - general use index.

Functions:
new_token - allocates, initializes and returns a new internal token.
free_line - frees the memory allocated to a line of tokens.
add_token - adds a token to the end of a line.
push_token - pushes a token onto the front of a line.
put_token - outputs an internal token to stdout.
put_page_token - outputs a new page token and causes next **put_token** call to print font and point sizes.

Side effects:
1. **new_token** allocates memory from the heap. If memory allocation fails then an ``out of memory´´ message is printed to stdout and the program halts.
2. **free_line** frees allocated memory to the heap.
3. **add_token** and **push_token** change the passed token line.
4. **put_token** and **put_page_token** print tokens to stdout.
5. **put_token** changes the following external variables: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
6. **put_page_token** changes the following external variables: **out_size**, **out_font_name**, **out_vertical**.

3.5 Service Flow Diagram

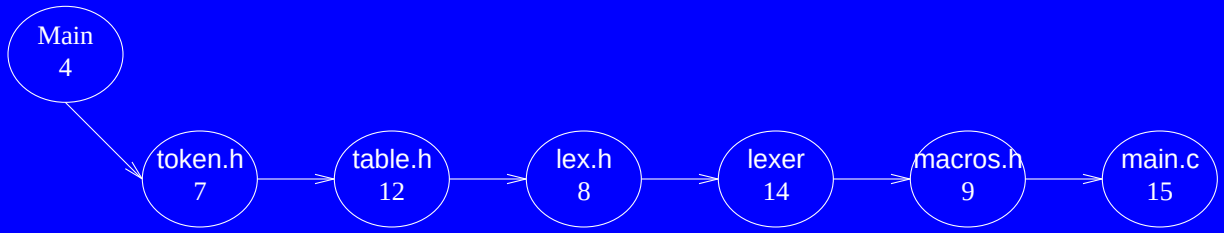


Software Unit #4 — Main

4.1 Software Unit Type

Module. (token.h, table.h, lex.h, lex.dit, macros.h, main.c)

4.2 Scope Diagram



4.3 Capabilities

Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

4.4 Interface

Globals:
in_font - current input font.
in_size - current input point size.
in_horizontal - current input horizontal position.
in_vertical - current input vertical position.
in_font_name - current input font name.
in_lr - current input font direction.
in_fontable - current input font table.
out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.
out_lr - current output font direction.
out_fontable - current output font table.
direction_table - formatting direction of fonts table.
arabic fonts - boolean table stating which font is arabic.
stretch_stage - the stretching type to be preformed.
device - name of output device.
c - general use char for flushing postscript and psfig text.

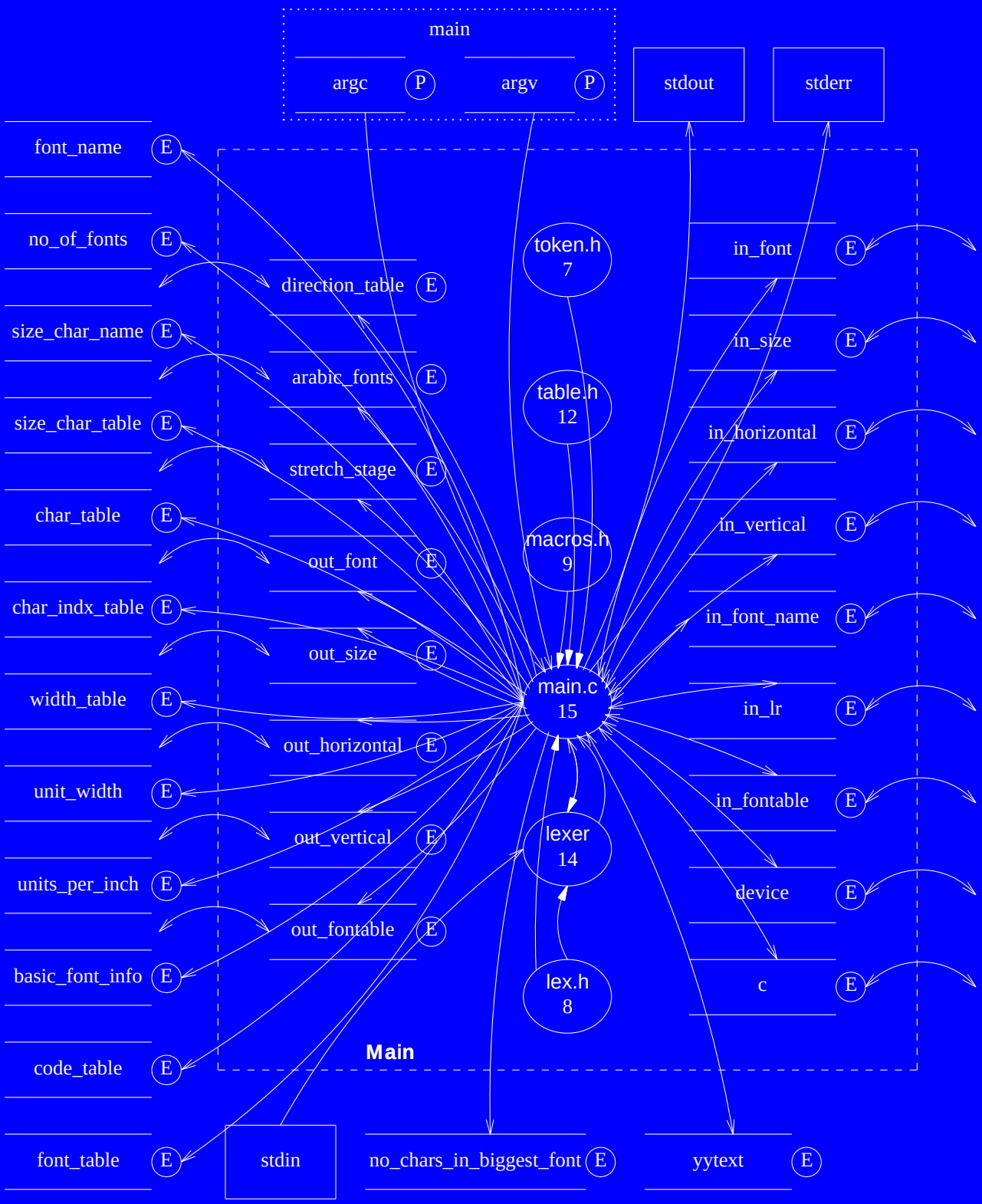
Functions:
main - main function for complete program including ffortid main driver.

4.4 Interface - Cont

Side effects:

1. **main** reads dtroff output from stdin and prints dtroff output to stdout.
2. **main** prints encountered errors to stderr and halts program.
3. **main** allocates and frees memory from the heap. If out of heap memory **main** prints a ``out of memory´´ message to stdout and halts program.
4. **main** changes the following external variables: **font_name**, **no_of_fonts**, **size_char_name**, **size_char_table**, **char_table**, **char_indx_table**, **width_table**, **unit_width**, **units_per_inch**, **basic_font_info**, **code_table**, **font_table**, **no_chars_in_biggest_font**, **yytext**.

4.5 Service Flow Diagram

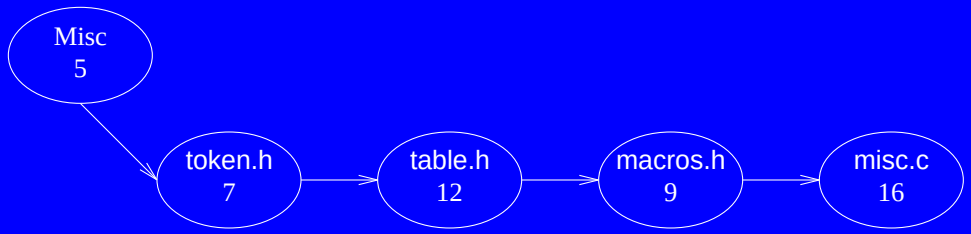


Software Unit #5 — Misc

5.1 Software Unit Type

Module (token.h, table.h, macros.h, misc.c)

5.2 Scope Diagram



5.3 Capabilities

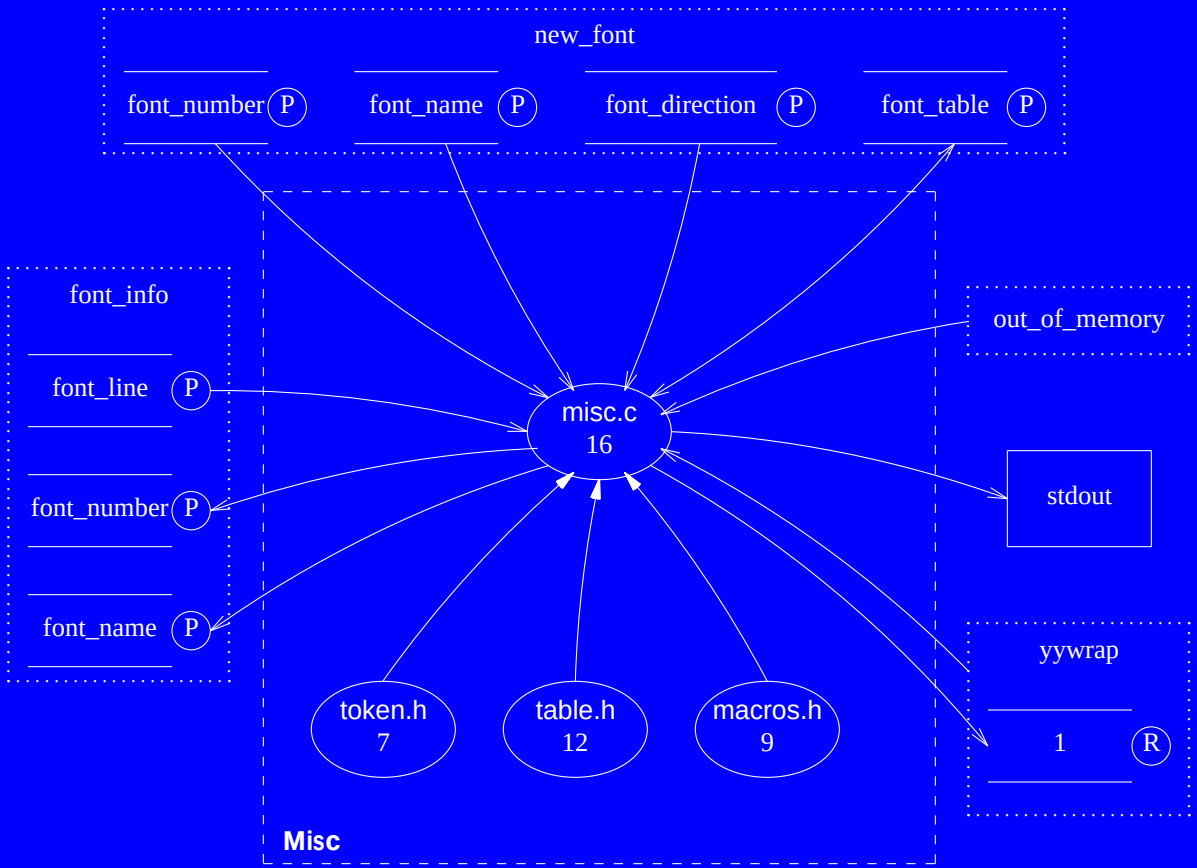
Contains a number of general support routines.

5.4 Interface

Functions:
new_font - adds a new font to the font table.
font_info - extracts a font number and name from a font token string.
out_of_memory - prints an ``out of memory´´ error message and halts execution.
yywrap - standard lex library function called whenever lex reaches an end-of-file.

Side effects:
1. **new_font** changes values in the passed **font_table**.
2. **font_info** returns through **font_number** the font token number and through **font_name** the font token name.
3. **out_of_memory** prints ``out of memory´´ error message to stdout and causes program to halt.

5.5 Service Flow Diagram

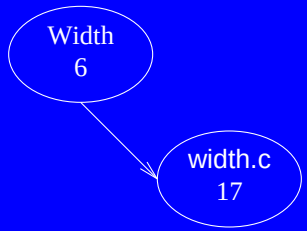


Software Unit #6 — Width

6.1 Software Unit Type

Module (width.c)

6.2 Scope Diagram



6.3 Capabilities

Contains globals that store the device and font width tables and routines to initialize them and return character widths based on them.

6.4 Interface

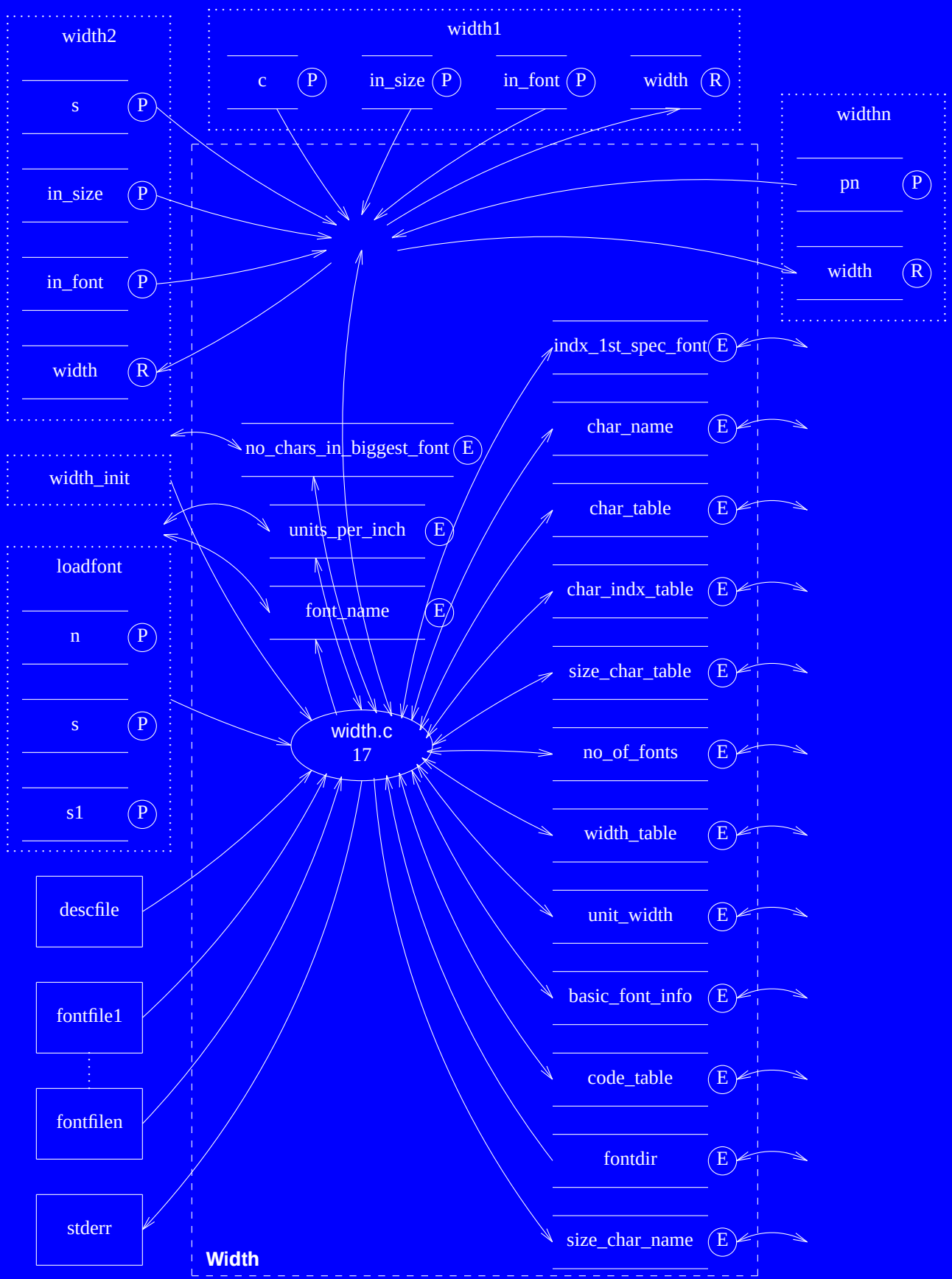
Globals:
basic_font_info - array of all fonts information.
font_name - array of all font names.
no_of_fonts - number of fonts initially mounted on the device.
indx_1st_spec_font - index of first special font.
size_char_table - size of character table in device.
unit_width - basic unit width in device.
units_per_inch - number of units per inch in device.
no_chars_in_biggest_font - number of chars in biggest font in device.
size_char_name - size of character name in device.
char_name - array of all character names in device.
char_table - array of indexes of characters in char_name.
char_indx_table - array of indexes of ascii characters in each font.
code_table - array of number codes for each char in each font.
width_table - array of widths for each char in each font.
fontdir - font files directory.

Functions:
width_init - initializes the device and font tables.
loadfont - loads a single font table. Currently body commented out.
width2 - returns the width of a specified funny character.
width1 - returns the width of a specified character.
widthn - returns the width of a character specified with its code.

Side effects:

1. **width_init** allocates memory from the heap.
2. Any error found in **width_init** is printed to stderr and the program halts.

6.5 Service Flow Diagram

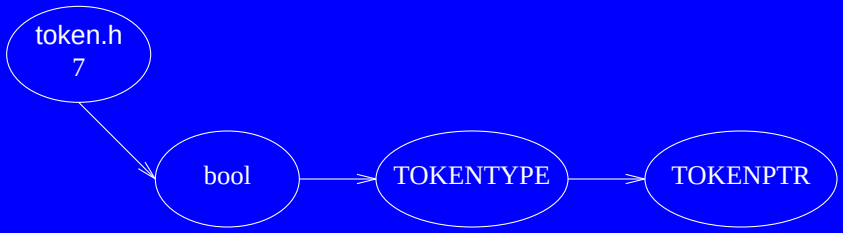


Software Unit #7 — token.h

7.1 Software Unit Type

Declerations source file. (token.h)

7.2 Scope Diagram



7.3 Capabilities

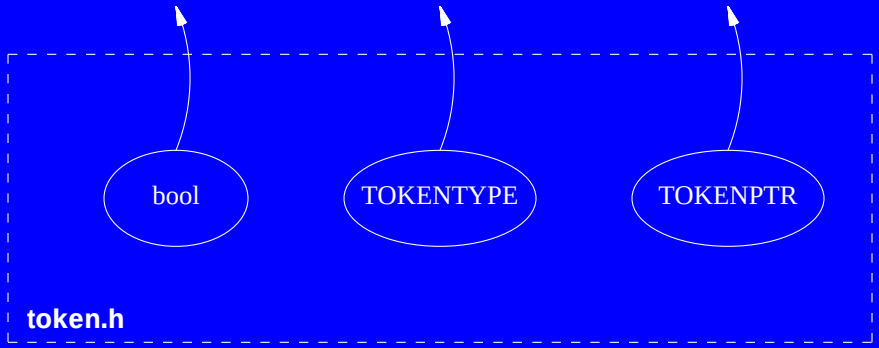
Contains the type declerations of the internal token representation structure and of **`bool`**.

7.4 Interface

Types:
`bool` - boolean values type.
`TOKENTYPE` - decleration of internal token representation structure.
`TOKENPTR` - decleration of pointer to **`TOKENTYPE`**.

Side effects:
None.

7.5 Service Flow Diagram

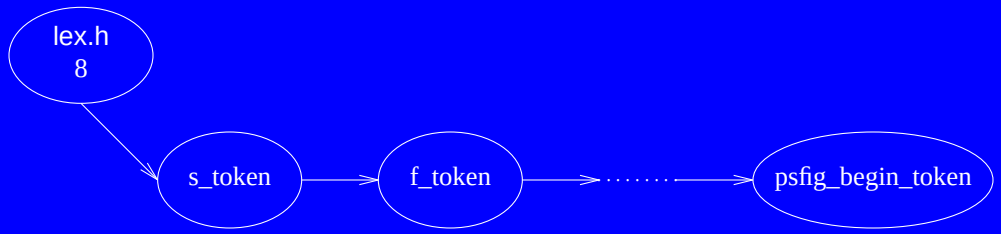


Software Unit #8 — lex.h

8.1 Software Unit Type

Definitions source file. (lex.h)

8.2 Scope Diagram



8.3 Capabilities

Contains 30 constant token definitions for lexical analyser.

8.4 Interface

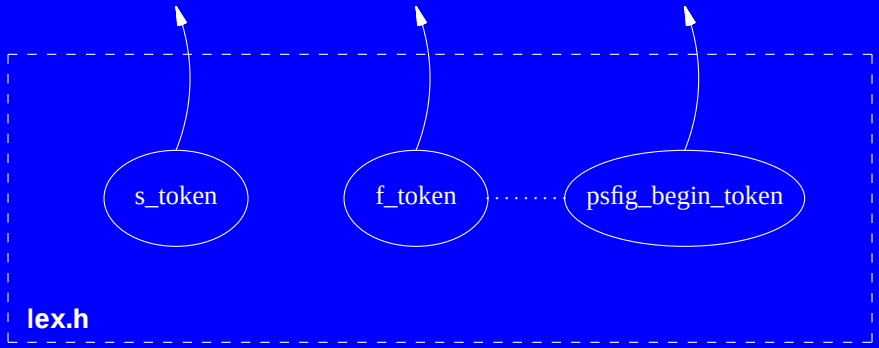
Constants:
s_token - dtroff s command token.
f_token - dtroff f command token.
c_token - dtroff c command token.
C_token - dtroff C command token.
H_token - dtroff H command token.
V_token - dtroff V command token.
h_token - dtroff h command token.
v_token - dtroff v command token.
hc_token - dtroff hc command token.
n_token - dtroff n command token.
w_token - dtroff w command token.
p_token - dtroff p command token.
trail_token - dtroff trail command token.
stop_token - dtroff stop command token.
dev_token - dtroff device command token.
res_token - dtroff resolution command token.
init_token - dtroff initialization command token.
font_token - dtroff font command token.
pause_token - dtroff pause command token.
height_token - dtroff height command token.
slant_token - dtroff slant command token.
newline_token - dtroff newline command token.
PR_token - dtroff page right-to-left command token.
PL_token - dtroff page left-to-right command token.

8.4 Interface - Cont

D_token - dtroff draw command token.
N_token - dtroff N command token.
include_token - dtroff include command token.
control_token - dtroff control command token.
postscript_begin_token - dtroff postscript begin command token.
psfig_begin_token - dtroff psfig begin command token.

Side effects:
None.

8.5 Service Flow Diagram

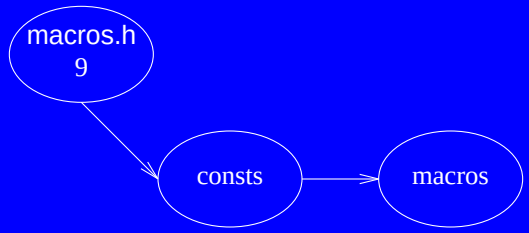


Software Unit #9 — macros.h

9.1 Software Unit Type

Definitions source file. (macros.h)

9.2 Scope Diagram



9.3 Capabilities

Contains general constant and macro definitions.

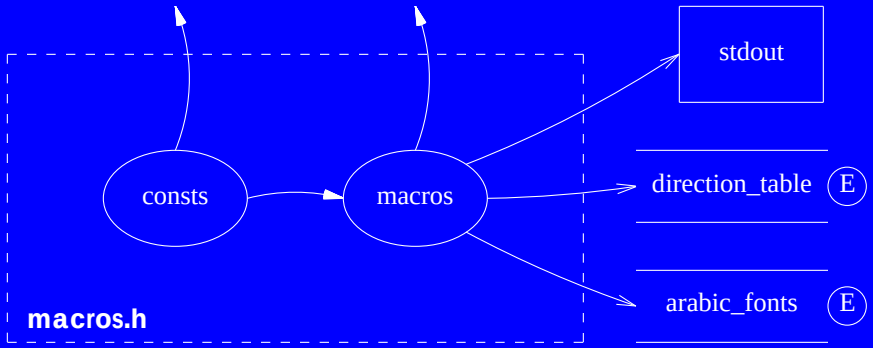
9.4 Interface

Constants:
BEGINING - token word begin constant.
NOT_BEGIN - token not word begin constant.
LEFT_TO_RIGHT - direction left to right constant.
RIGHT_TO_LEFT - direction right to left constant.
END - token word end constant.
NOT_END - token not word end constant.
TRUE - boolean true constant.
FALSE - boolean false constant.
NOFILLERS - token nofillers constant.
ARABIC - font arabic constant.

Macros:
DUMP_LEX - dump string to stdout as is.
SET_DIRECTION - set font direction.
FONT_DIRECTION - return font direction.
SET_AR_FONT - set font as arabic.
RESET_AR_FONT - set font as non-arabic.

Side effects:
1. **DUMP_LEX** prints to stdout.
2. **SET_DIRECTION** and **FONT_DIRECTION** change **direction_table**.
3. **SET_AR_FONT** and **RESET_AR_FONT** change **arabic_fonts**.

9.5 Service Flow Diagram

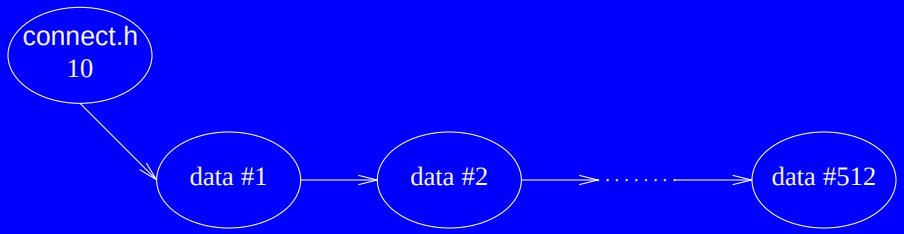


Software Unit #10 — connect.h

10.1 Software Unit Type

Data file. (connect.h)

10.2 Scope Diagram



10.3 Capabilities

Contains data for initalization of **`cnct`** structure (declared in dump.c). Data is inserted by using **`#include`**.

10.4 Interface

None.

10.5 Service Flow Diagram

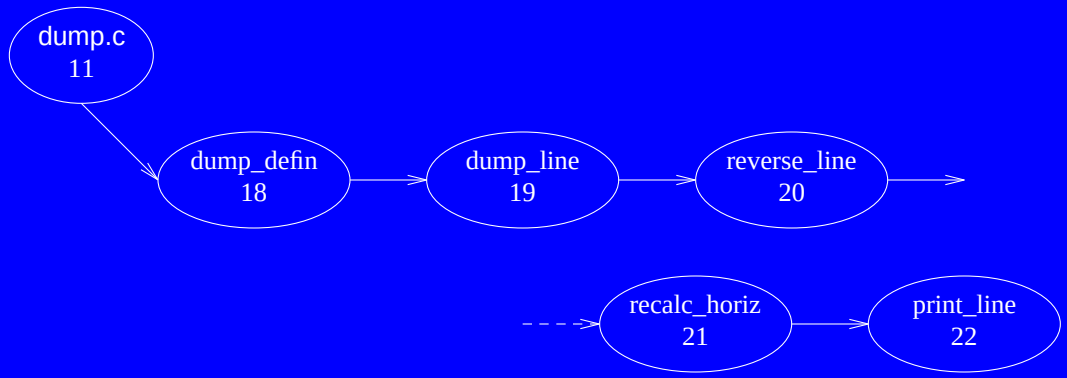
None.

Software Unit #11 — dump.c

11.1 Software Unit Type

Source file. (dump.c)

11.2 Scope Diagram



11.3 Capabilities

Contains routines that dump and reverse internal token lines while taking care of such issues as stretching and text direction.

11.4 Interface

Constants:
MAXZWC - maximum number of respective zero width characters.
NFONT - maximum number of fonts.

Macros:
max - maximum of two values.

Globals:
connect - array of all **cnct** structures.

Externals:
arabic fonts - boolean table stating which font is arabic.
stretch_stage - the stretching type to be performed.
width_table - array of widths for each char in each font.
unit_width - basic unit width in device.
new_token() - allocates, initializes and returns a new internal token.

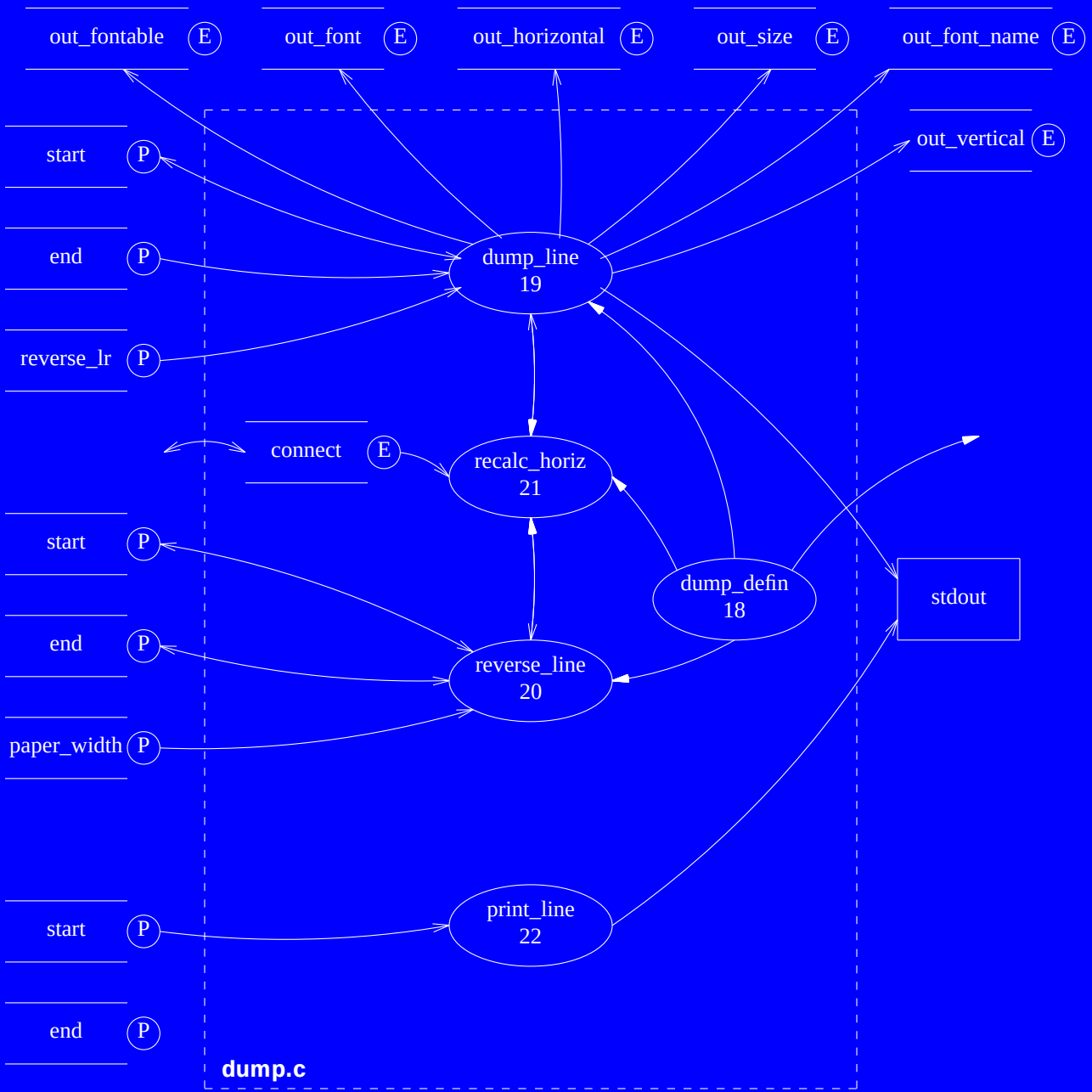
Functions:
dump_line - stretches and dumps an internal token line while reversing tokens of the specified lr direction.
reverse_line - reverses an internal token line while preserving zero width characters position.
print_line - prints an internal token line to stdout. Used for debugging.

11.4 Interface - Cont

Side effects:

1. **dump_line** prints passed token line to stdout and frees the heap memory used by it.
2. **dump_line** changes the values of external vars: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
3. **reverse_line** changes the tokens in the passed token line.
4. **print_line** prints the passed token line to stdout.

11.5 Service Flow Diagram

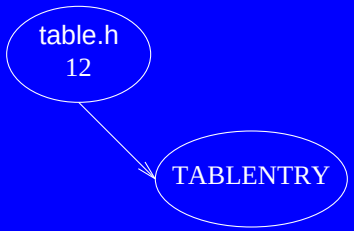


Software Unit #12 — table.h

12.1 Software Unit Type

Declerations source file. (table.h)

12.2 Scope Diagram



12.3 Capabilities

Contains the type decleration of the internal font table entry structure.

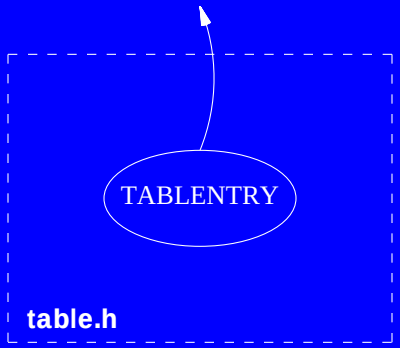
12.4 Interface

Types:
TABLENTRY - internal font table entry structure.

Side effects:

None.

12.5 Service Flow Diagram



Software Unit #13 — lines.c

13.1 Software Unit Type

Source file. (lines.c)

13.2 Scope Diagram

13.3 Capabilities

Contains routines to allocate, free, insert and print internal tokens.

13.4 Interface

Externals:
out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.
out_fontable - current output font table.
in_font - current input font.
in_size - current input point size.
in_horizontal - current input horizontal position.
in_vertical - current input vertical position.
in_font_name - current input font name.
in_fontable - current input font table.
in_lr - current input font direction.
direction_table - table of fonts formatting direction.

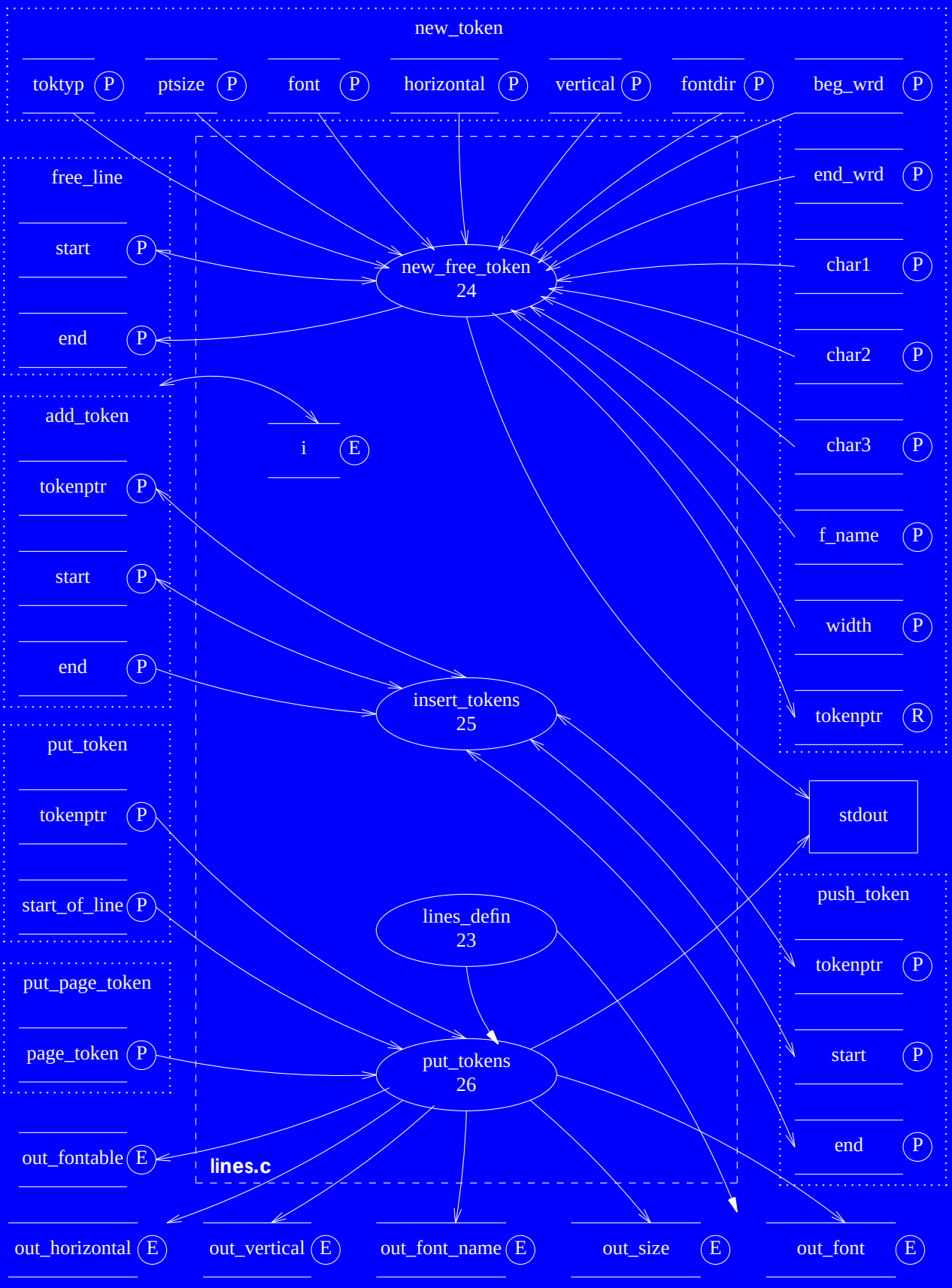
Globals:
i - general use index.

13.4 Interface - Cont

Functions:
new_token - allocates, initializes and returns a new internal token.
free_line - frees the memory allocated to a line of tokens.
add_token - adds a token to the end of a line.
push_token - pushes a token onto the front of a line.
put_token - outputs an internal token to stdout.
put_page_token - outputs a new page token and causes next **put_token** call to print font and point sizes.

Side effects:
1. **new_token** allocates memory from the heap. If memory allocation fails then an ``out of memory´´ message is printed to stdout and the program halts.
2. **free_line** frees allocated memory to the heap.
3. **add_token** and **push_token** change the passed token line.
4. **put_token** and **put_page_token** print tokens to stdout.
5. **put_token** changes the following external variables: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
6. **put_page_token** changes the following external variables: **out_size**, **out_font_name**, **out_vertical**.

13.5 Service Flow Diagram

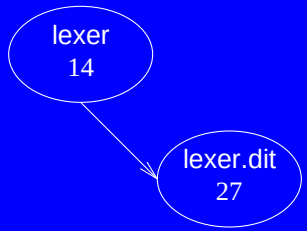


Software Unit #14 — lexer

14.1 Software Unit Type

Lex generated source file. (lex.dit)

14.2 Scope Diagram



14.3 Capabilities

Lexicaly parses dtroff output into tokens.

14.4 Interface

Globals:
yytext - points to the actual string matched by the lexical analyser.

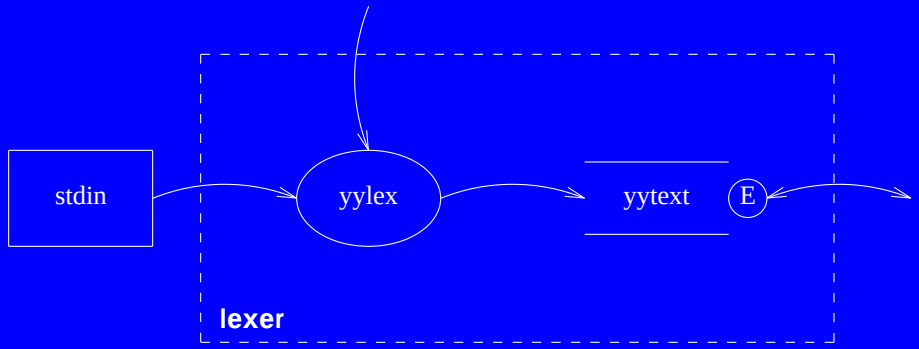
Functions:

yylex - returns next token matched by the lexical analyser.

14.4 Interface - Cont

Side effects:
Reads in dtroff output from stdin.

14.5 Service Flow Diagram

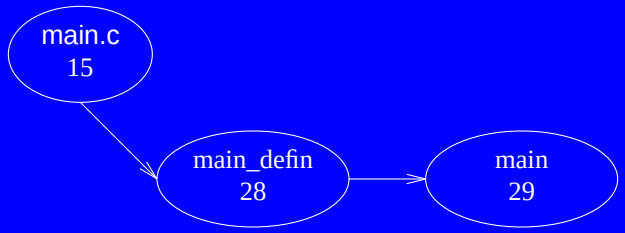


Software Unit #15 — main.c

15.1 Software Unit Type

Source file. (main.c)

15.2 Scope Diagram



15.3 Capabilities

Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

15.4 Interface

Constants:
USAGE - command line usage explanation string.

Macros:
MARK_PREVIOUS_END - marks the last token in the current input line as ending a word.
ADD_CHAR1 - creates a new token from 1 char and adds it to end of current input line.
ADD_CHAR2 - creates a new token from 2 chars and adds it to end of current input line.
ADD_CHARN - creates a new token of from 3 chars and adds it to end of current input line.

Static Globals:
copyright - string holding copyright information.

Globals:
in_font - current input font.
in_size - current input point size.
in_horizontal - current input horizontal position.
in_vertical - current input vertical position.
in_font_name - current input font name.
in_lr - current input font direction.
in_fontable - current input font table.
out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.

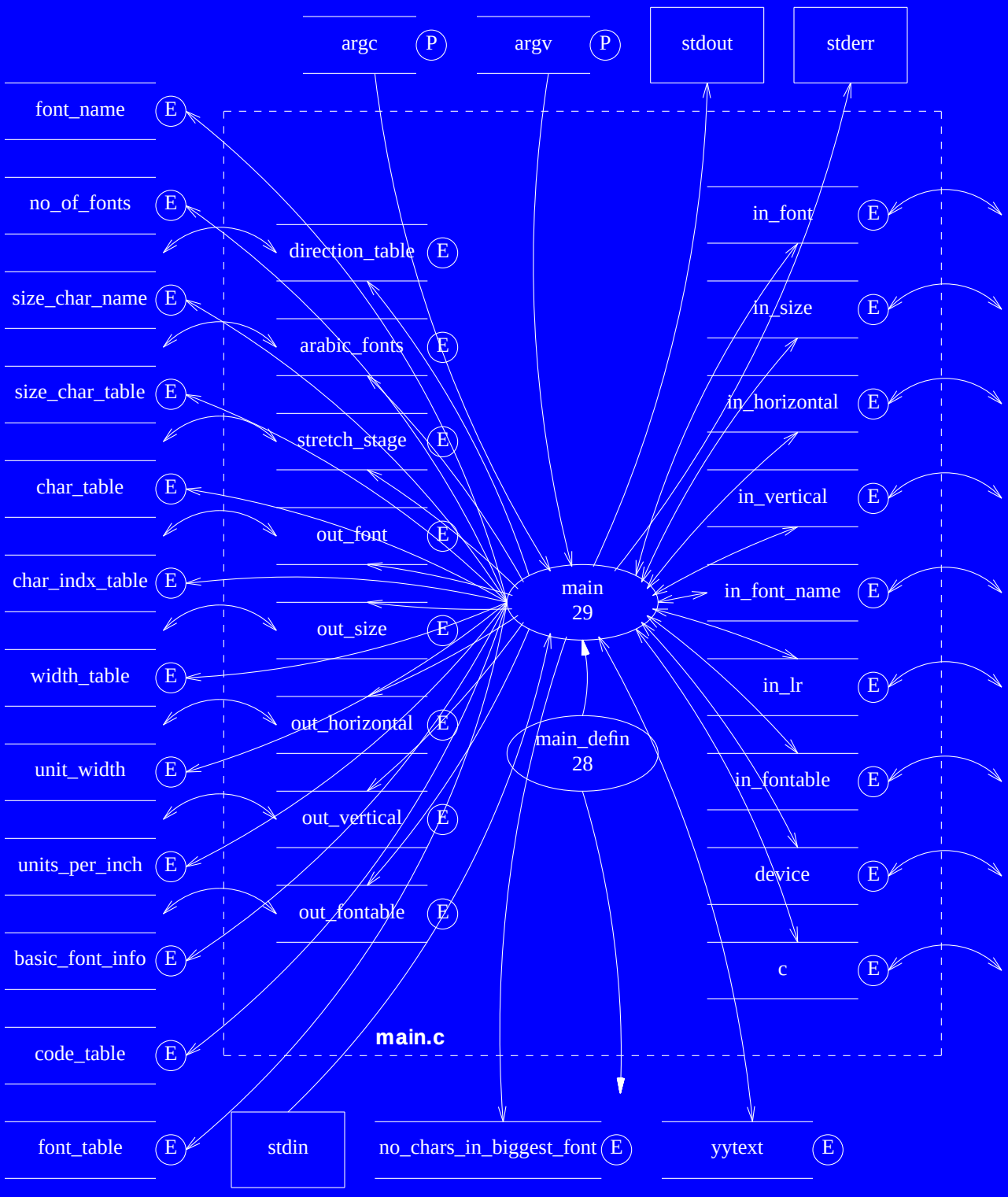
15.4 Interface - Cont

out_lr - current output font direction.
out_fontable - current output font table.
direction_table - formatting direction of fonts table.
arabic fonts - boolean table stating which font is arabic.
stretch_stage - the stretching type to be preformed.
device - name of output device.
c - general use char for flushing postscript and psfig text.

Functions:
main - main function for complete program including ffortid main driver.

Side effects:
1. **MARK_PREVIOUS_END** changes the token pointed by **in_end**.
2. **ADD_CHAR1**, **ADD_CHAR2** and **ADD_CHARN** create a new token allocated from the heap and add it to the token line pointed to by **in_start** and **in_end**.
3. **main** reads dtroff output from stdin and prints dtroff output to stdout.
4. **main** prints encountered errors to stderr and halts program.
5. **main** allocates and frees memory from the heap. If out of heap memory **main** prints a ``out of memory´´ message to stdout an halts program.
6. **main** changes the following external variables: **font_name**, **no_of_fonts**, **size_char_name**, **size_char_table**, **char_table**, **char_indx_table**, **width_table**, **unit_width**, **units_per_inch**, **basic_font_info**, **code_table**, **font_table**, **no_chars_in_biggest_font**, **yytext**.

15.5 Service Flow Diagram

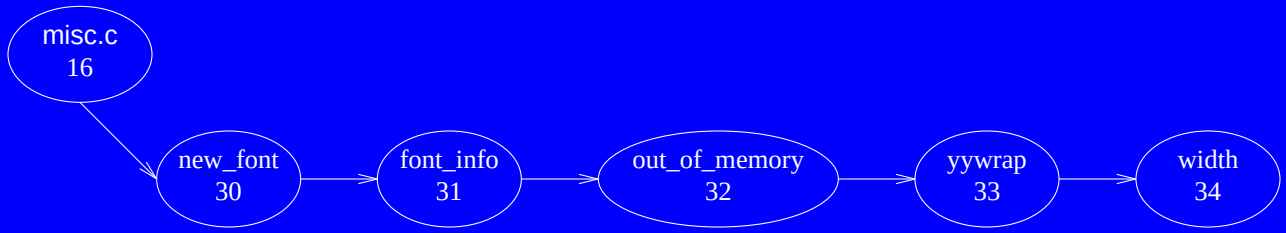


Software Unit #16 — misc.c

16.1 Software Unit Type

Source file. (misc.c)

16.2 Scope Diagram



16.3 Capabilities

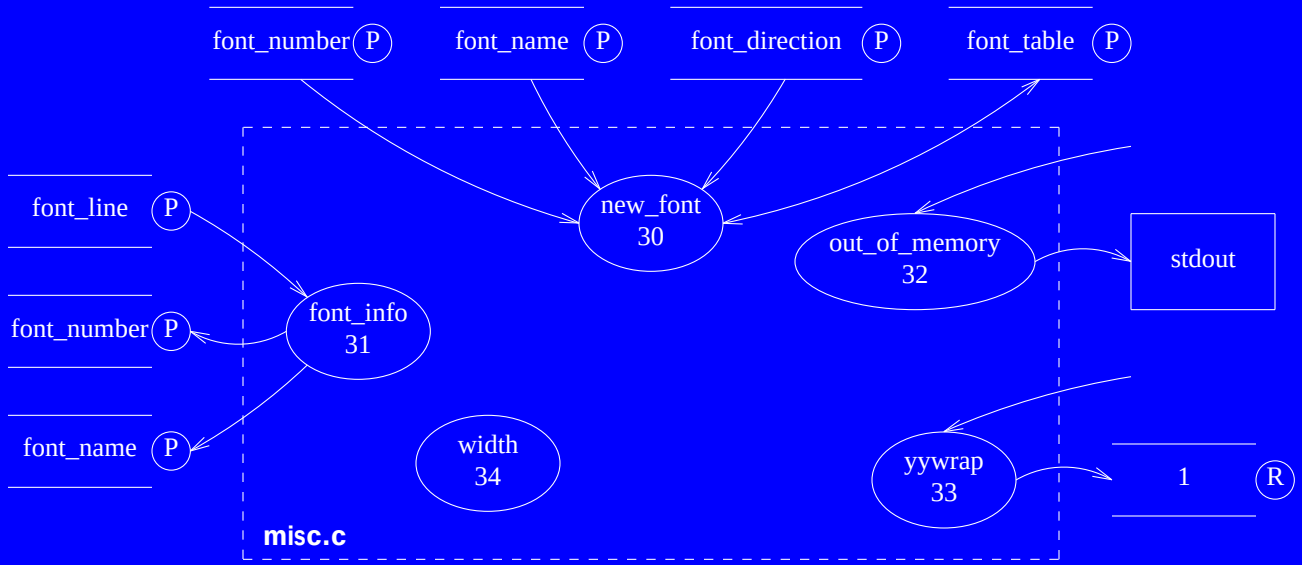
Contains a number of general support routines.

16.4 Interface

Functions:
new_font - adds a new font to the font table.
font_info - extracts a font number and name from a font token string.
out_of_memory - prints an ``out of memory´´ error message and halts execution.
yywrap - standard lex library function called whenever lex reaches an end-of-file.

Side effects:
1. **new_font** changes values in the passed **font_table**.
2. **font_info** returns through **font_number** the font token number and through **font_name** the font token name.
3. **out_of_memory** prints ``out of memory´´ error message to stdout and causes program to halt.

16.5 Service Flow Diagram

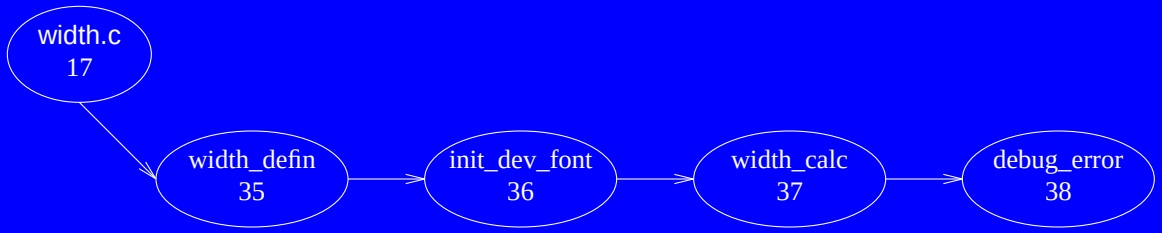


Software Unit #17 — width.c

17.1 Software Unit Type

Source file. (width.c)

17.2 Scope Diagram



17.3 Capabilities

Contains globals that store the device and font width tables and routines to initialize them and return character widths based on them.

17.4 Interface

Constants:
MAXNOFONTS - max number of fonts.
MAXWIDENTRIES - max width entries.
NOCHARSINBIGGESTFONT - no of characters in biggest font in device description.
MAXNOCHARS - max number of chars with with two letters or --- names.
SIZECHARINDXTABLE - size of character index table including ascii chars but not non-graphics.
FATAL - passed to error procedure to signal fatal error.
BYTEMASK - mask used to make character numbers positive.

Types:
Fontinfo - single font information structure.

Globals:
basic_font_info - array of all fonts information.
font_name - array of all font names.
no_of_fonts - number of fonts initially mounted on the device.
indx_1st_spec_font - index of first special font.
size_char_table - size of character table in device.
unit_width - basic unit width in device.
units_per_inch - number of units per inch in device.
no_chars_in_biggest_font - number of chars in biggest font in device.
size_char_name - size of character name in device.
char_name - array of all character names in device.
char_table - array of indexes of characters in char_name.
char_indx_table - array of indexes of ascii characters in each font.

17.4 Interface - Cont

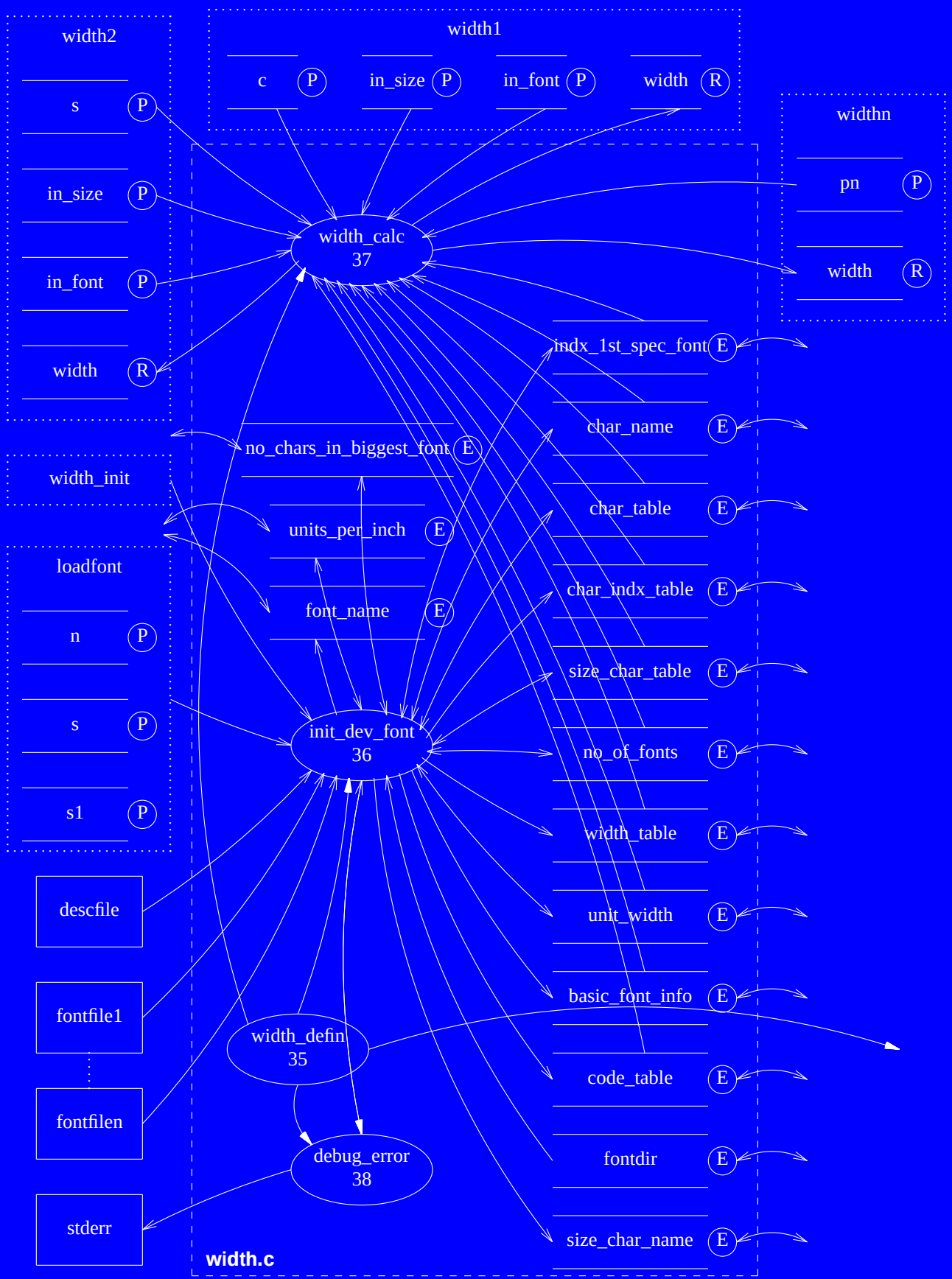
code_table - array of number codes for each char in each font.
width_table - array of widths for each char in each font.
fontdir - font files directory.

Externals:
in_size - current input font point size.
in_font - current input font number.
device - name of output device.

Functions:
width_init - initializes the device and font tables.
loadfont - loads a single font table. Currently body commented out.
width2 - returns the width of a specified funny character.
width1 - returns the width of a specified character.
widthn - returns the width of a character specified with its code.

Side effects:
1. **width_init** allocates memory from the heap.
2. Any error found in **width_init** is printed to stderr and the program halts.

17.5 Service Flow Diagram

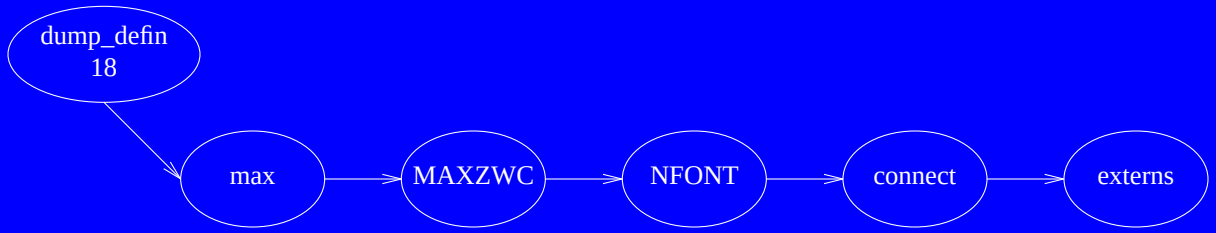


Software Unit #18 — dump_defin

18.1 Software Unit Type

Definitions block. (dump.c: 1-34)

18.2 Scope Diagram



18.3 Capabilities

Contains definitions used by the functions in dump.c.

18.4 Interface

Constants:
MAXZWC - maximum number of respective zero width characters.
NFONT - maximum number of fonts.

Macros:
max - maximum of two values.

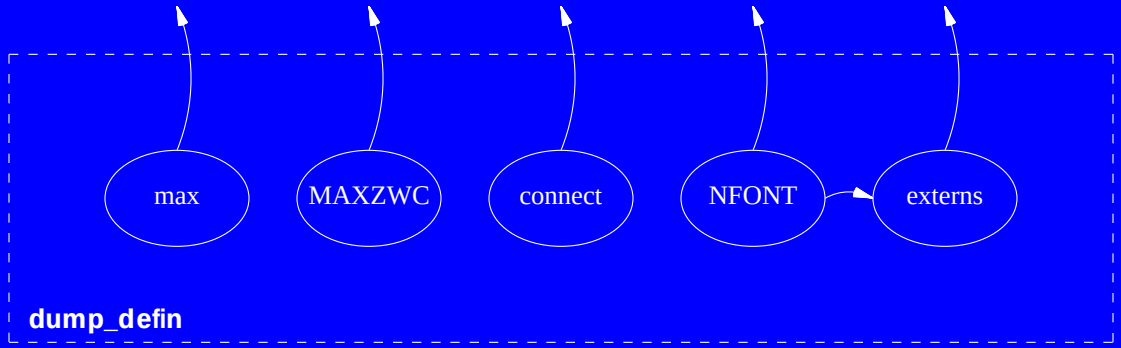
Types:
cnct - structure holding each arabic chars code & connect before value.

Globals:
connect - array of all **cnct** structures.

Externals:
arabic fonts - boolean table stating which font is arabic.
stretch_stage - the stretching type to be performed.
width_table - array of widths for each char in each font.
unit_width - basic unit width in device.
new_token() - allocates, initializes and returns a new internal token.

Side effects:
None

18.5 Service Flow Diagram

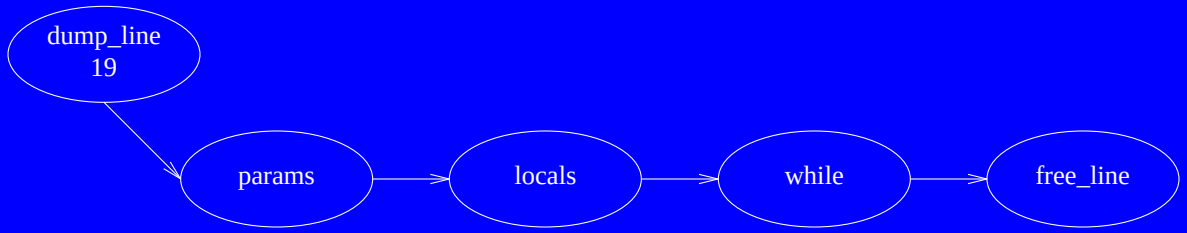


Software Unit #19 — dump_line

19.1 Software Unit Type

Procedure. (dump.c: 35-137)

19.2 Scope Diagram



19.3 Capabilities

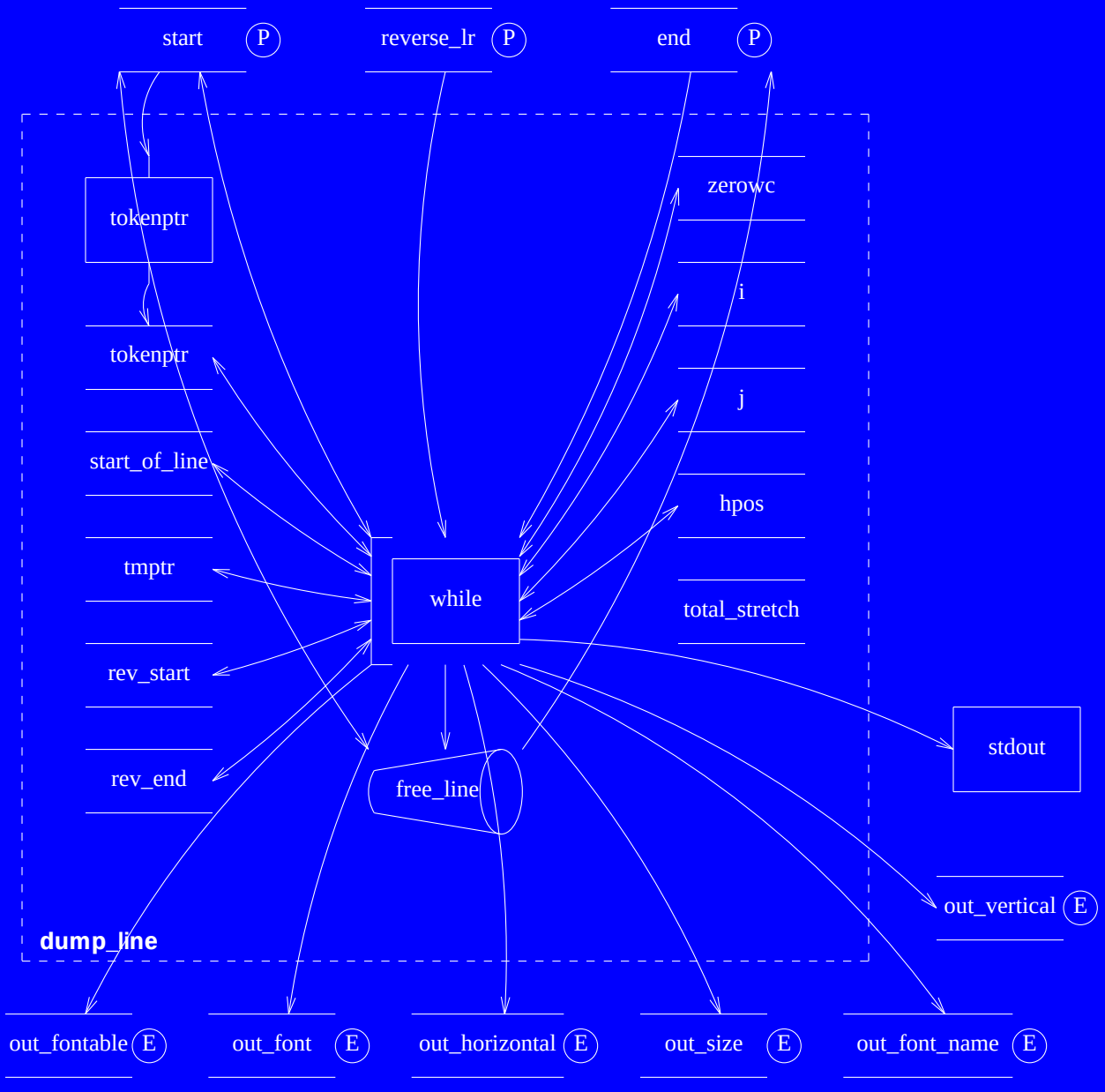
Stretches and dumps the specified internal token line to stdout while reversing the tokens of the specified lr direction. Deals also with zero width characters and zero horizontal movements.

19.4 Interface

Parameters:
start - pointer to first token in line.
end - pointer to last token in line.
reverse_lr - boolean specifying tokens of which direction are to be reversed.

Side effects:
1. Prints dumped line to stdout.
2. Changes the values of external vars: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
3. Frees the heap memory used by the passed token line.

19.5 Service Flow Diagram

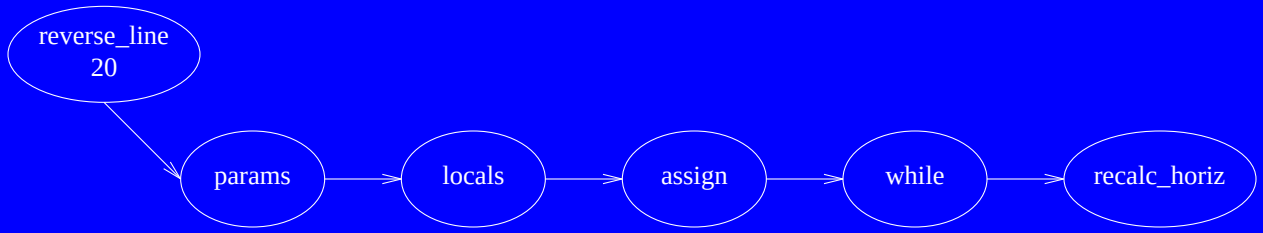


Software Unit #20 — reverse_line

20.1 Software Unit Type

Procedure. (dump.c: 138-220)

20.2 Scope Diagram



20.3 Capabilities

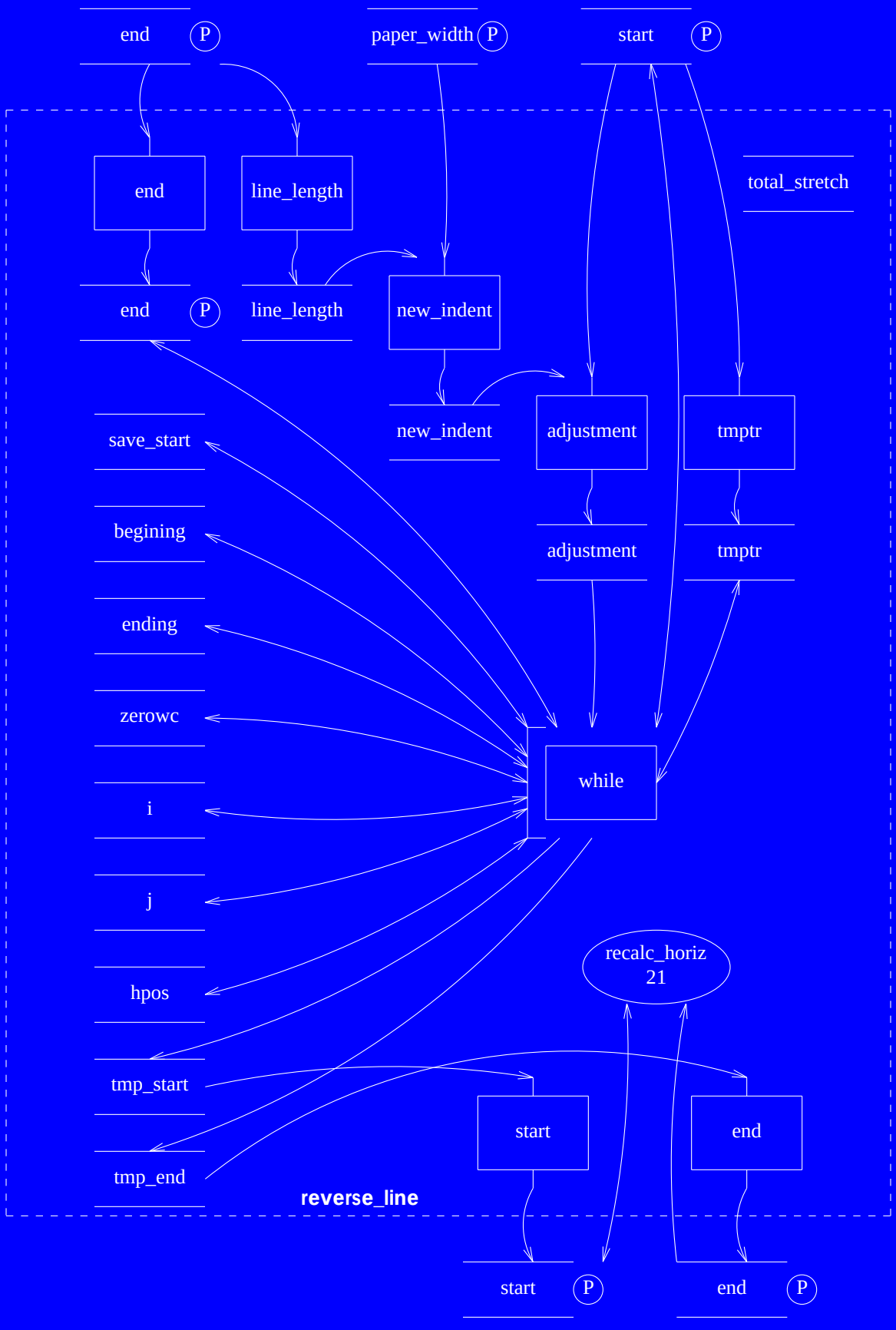
Reverses the specified internal token line while preserving the order of zero width characters with their next letter.

20.4 Interface

Parameters:
start - pointer to first token in line.
end - pointer to last token in line.
paper_width - paper width in points.

Side effects:
Changes the tokens in the passed token line.

20.5 Service Flow Diagram

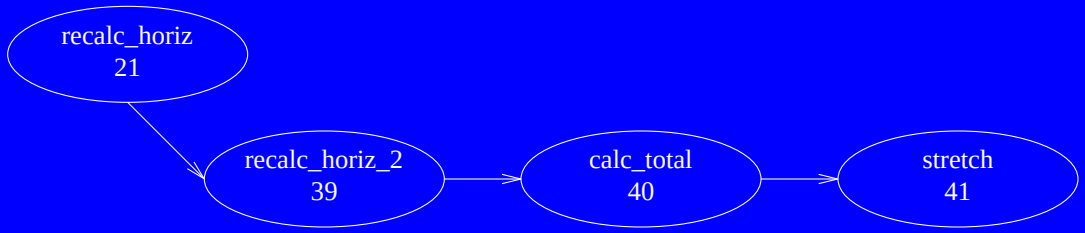


Software Unit #21 — recalc_horiz

21.1 Software Unit Type

Function group. (dump.c: 221-683)

21.2 Scope Diagram



21.3 Capabilities

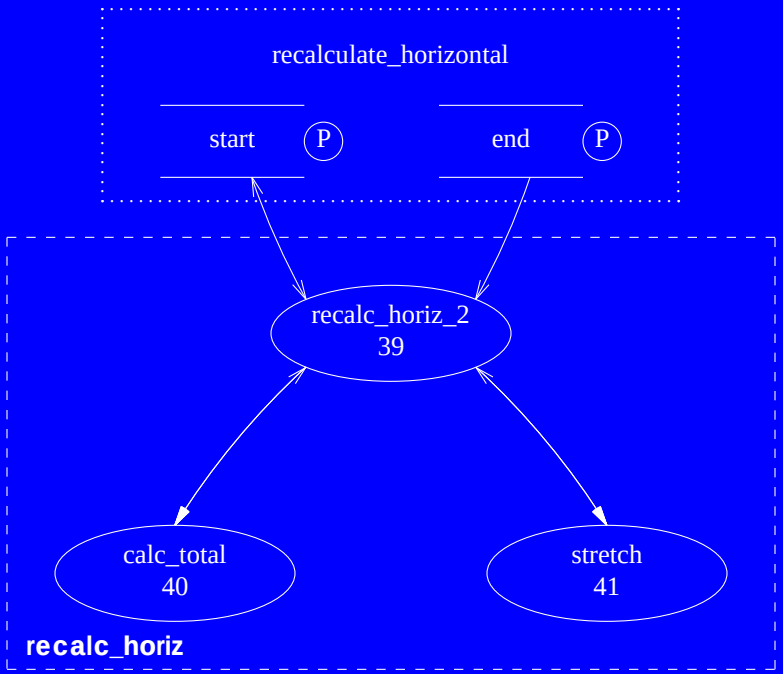
Recalculates the horizontal motion and stretches the specified token line.

21.4 Interface

Procedures:
`recalculate_horizontal` - recalculates the horizontal motion and stretches a line.

Side effects:
Changes the tokens in the passed token line.

21.5 Service Flow Diagram

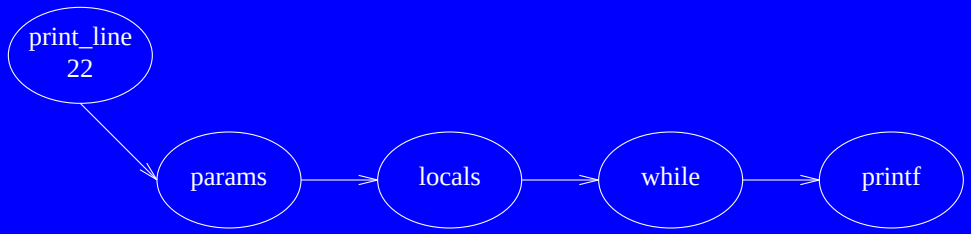


Software Unit #22 — print_line

22.1 Software Unit Type

Procedure. (dump.c: 684-704)

22.2 Scope Diagram



22.3 Capabilities

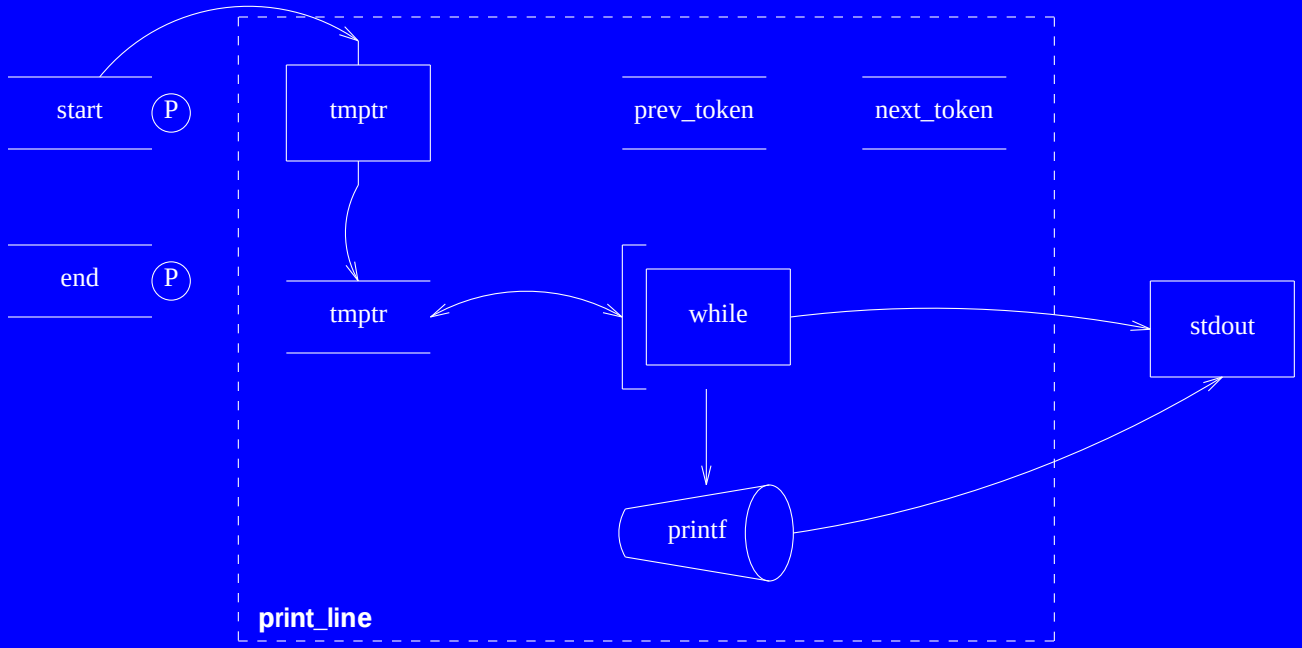
Prints the specified internal token line to stdout for debugging.

22.4 Interface

Parameters:
start - pointer to first token in line.
end - pointer to last token in line.

Side effects:
Prints the passed token line to stdout.

22.5 Service Flow Diagram

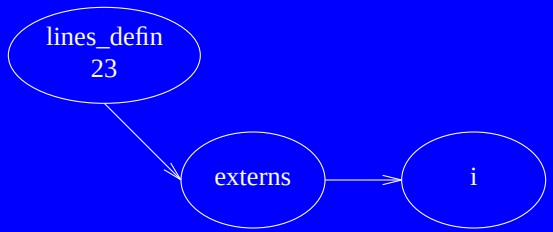


Software Unit #23 — lines_defin

23.1 Software Unit Type

Definitions block. (lines.c: 1-35)

23.2 Scope Diagram



23.3 Capabilities

Contains definitions used by the lines.c functions.

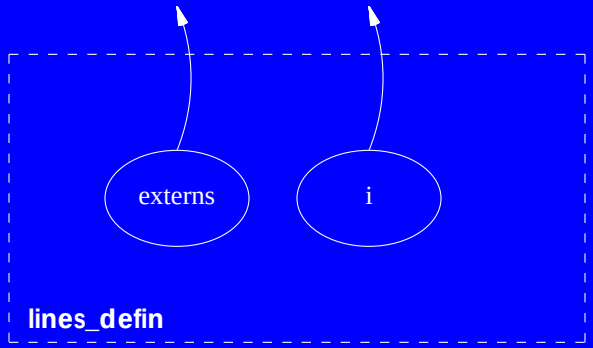
23.4 Interface

Externals:
out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.
out_fontable - current output font table.
in_font - current input font.
in_size - current input point size.
in_horizontal - current input horizontal position.
in_vertical - current input vertical position.
in_font_name - current input font name.
in_fontable - current input font table.
in_lr - current input font direction.
direction_table - table of fonts formatting direction.

Globals:
i - general use index.

Side effects:
None.

23.5 Service Flow Diagram

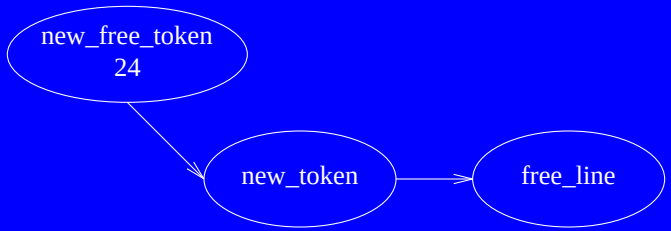


Software Unit #24 — new_free_token

24.1 Software Unit Type

Function group. (lines.c: 36-91 & 268-296)

24.2 Scope Diagram



24.3 Capabilities

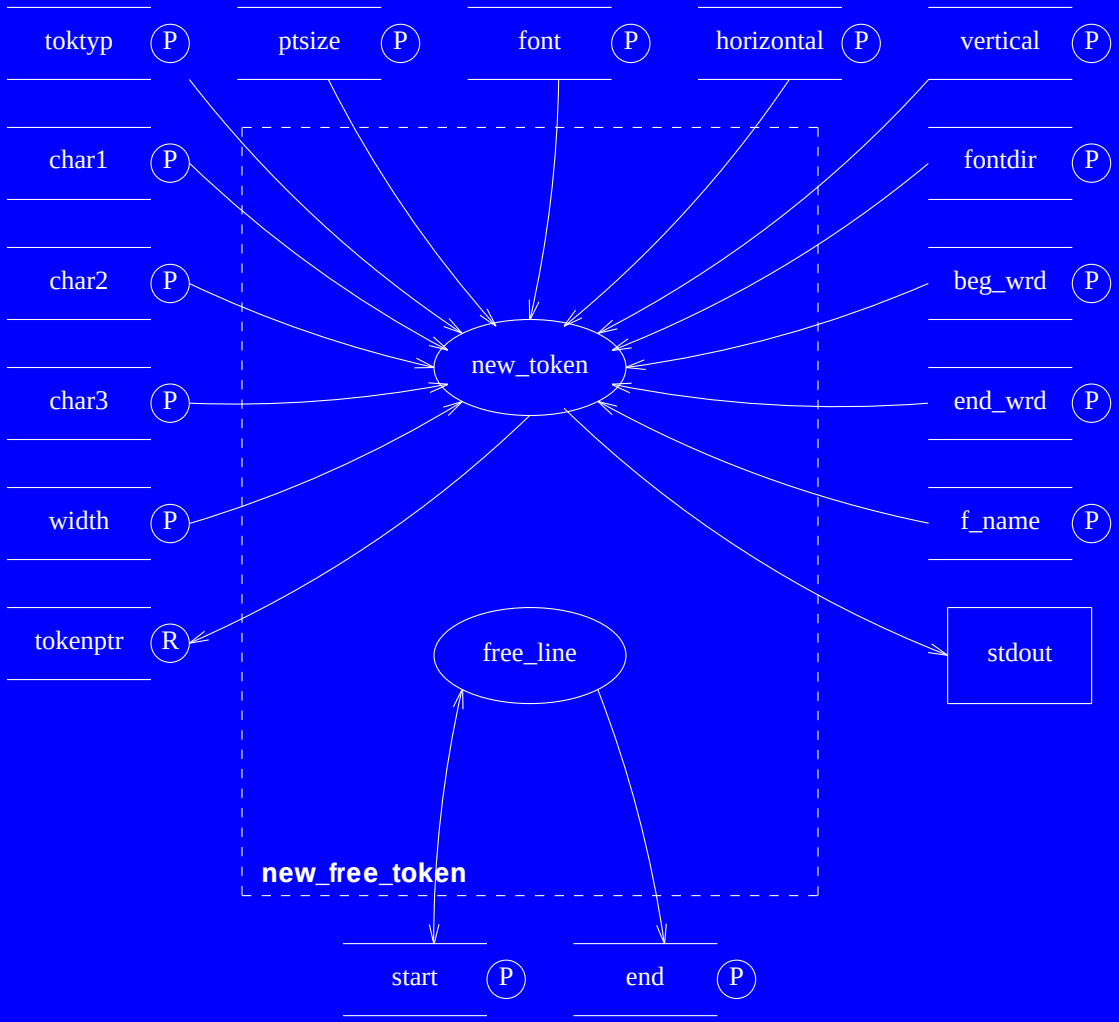
Contains routines to allocate new tokens and to free lines of tokens.

24.4 Interface

Functions:
new_token - allocates, initializes and returns a new internal token.
free_line - frees the memory allocated to a line of tokens.

Side effects:
1. **new_token** allocates memory from the heap.
2. If memory allocation fails in **new_token** then an ``out of memory´´ message is printed to stdout and the program halts.
3. **free_line** frees allocated memory to the heap.

24.5 Service Flow Diagram

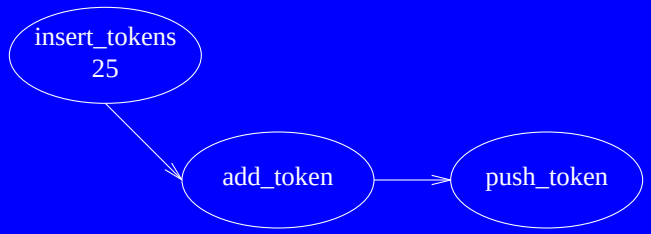


Software Unit #25 — insert_tokens

25.1 Software Unit Type

Procedure group. (lines.c: 92-143)

25.2 Scope Diagram



25.3 Capabilities

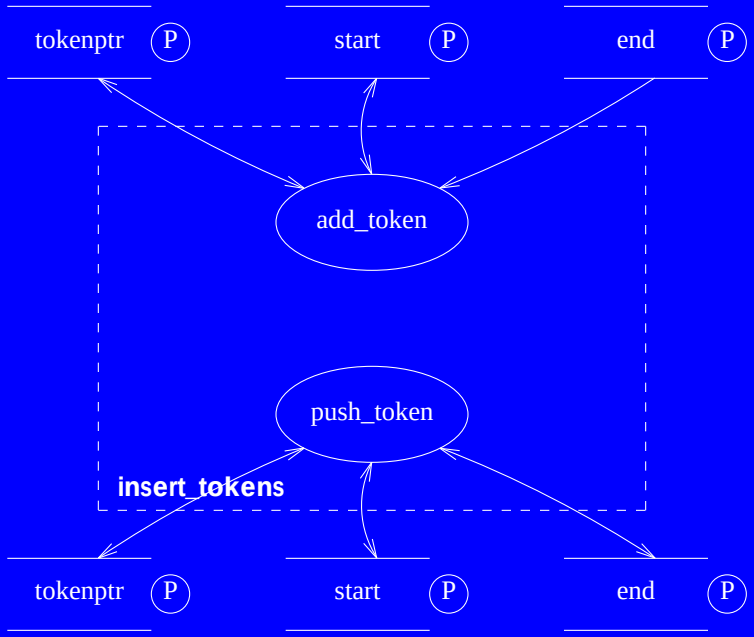
Contains routines to add tokens to the end & front of a token line.

25.4 Interface

Procedures:
`add_token` - adds a token to the end of a line.
`push_token` - pushes a token onto the front of a line.

Side effects:
Both procedures change the passed token line.

25.5 Service Flow Diagram

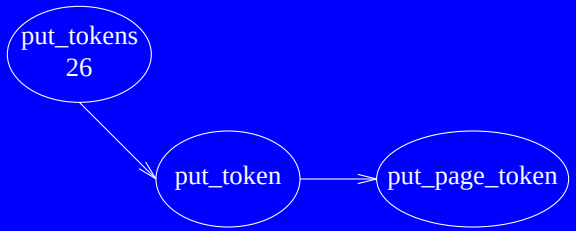


Software Unit #26 — put_tokens

26.1 Software Unit Type

Procedure group. (lines.c: 144-267)

26.2 Scope Diagram



26.3 Capabilities

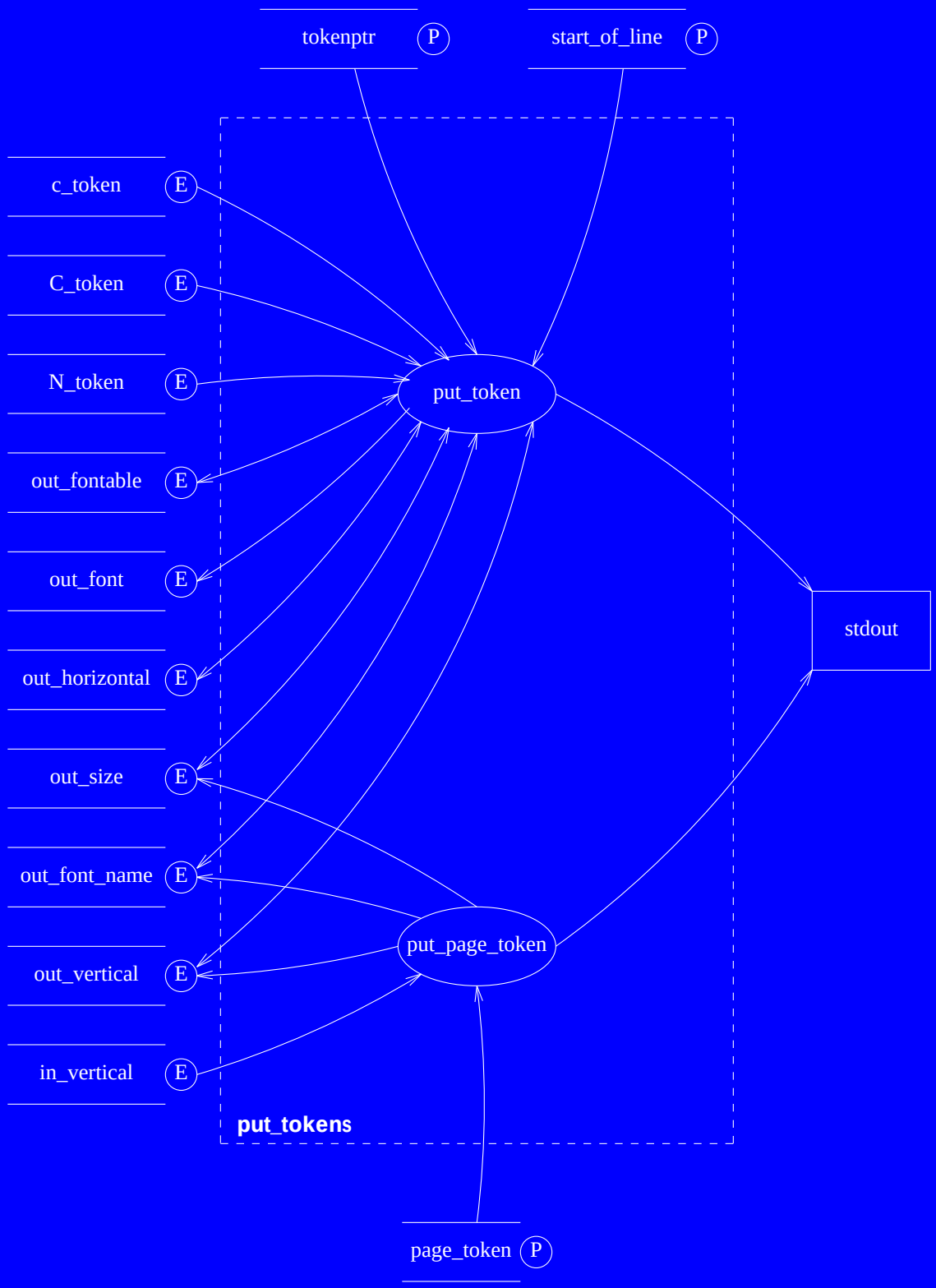
Contains routines to output internal and new page tokens to stdout.

26.4 Interface

Procedures:
put_token - outputs an internal token to stdout.
put_page_token - outputs a new page token and causes next **put_token** call to print font and point sizes.

Side effects:
1. Both procedures print tokens to stdout.
2. **put_token** changes the following external variables: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
3. **put_page_token** changes the following external variables: **out_size**, **out_font_name**, **out_vertical**.

26.5 Service Flow Diagram

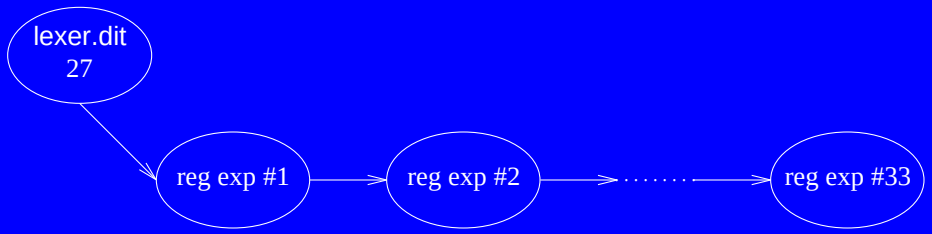


Software Unit #27 — lexer.dit

27.1 Software Unit Type

Lex source file. (lex.dit)

27.2 Scope Diagram



27.3 Capabilities

Contains regular expressions to recognize dtroff output commands, and has associated with each expression an action returning a distinct token.

27.4 Interface

None.

27.5 Service Flow Diagram

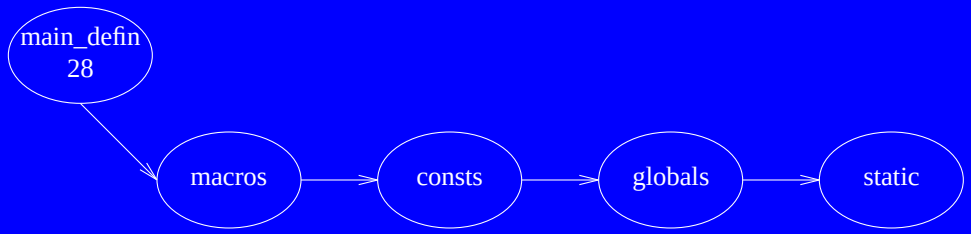
None.

Software Unit #28 — main_defin

28.1 Software Unit Type

Definitions block. (main.c: 1-58)

28.2 Scope Diagram



28.3 Capabilities

Contains definitions used by **`main`** & complete program.

28.4 Interface

Constants:
USAGE - command line usage explanation string.

Macros:
MARK_PREVIOUS_END - marks the last token in the current input line as ending a word.
ADD_CHAR1 - creates a new token from 1 char and adds it to end of current input line.
ADD_CHAR2 - creates a new token from 2 chars and adds it to end of current input line.
ADD_CHARN - creates a new token of from 3 chars and adds it to end of current input line.

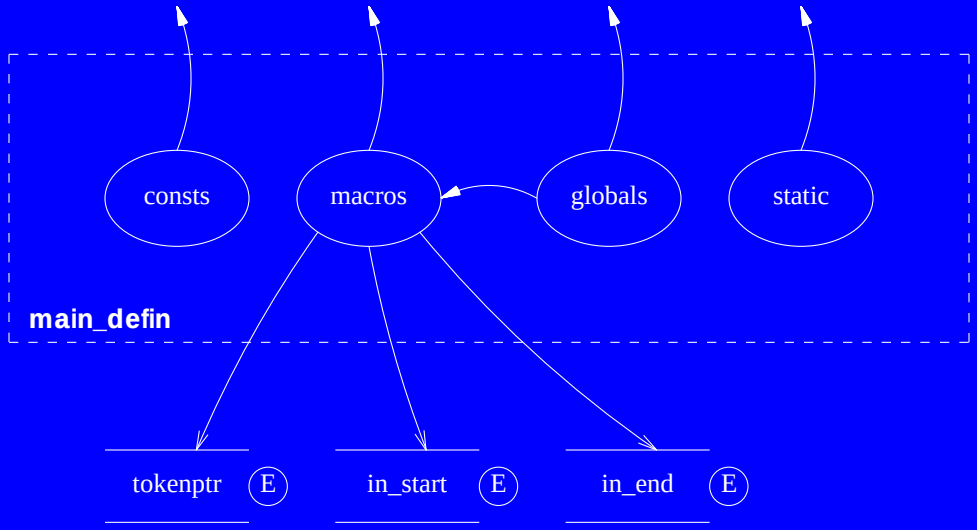
Static Globals:
copyright - string holding copyright information.

28.4 Interface - Cont

Globals:
in_font - current input font.
in_size - current input point size.
in_horizontal - current input horizontal position.
in_vertical - current input vertical position.
in_font_name - current input font name.
in_lr - current input font direction.
in_fontable - current input font table.
out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.
out_lr - current output font direction.
out_fontable - current output font table.
direction_table - formatting direction of fonts table.
arabic fonts - boolean table stating which font is arabic.
stretch_stage - the stretching type to be preformed.
device - name of output device.
c - general use char for flushing postscript and psfig text.

Side effects:
1. **MARK_PREVIOUS_END** changes the token pointed by **in_end**.
2. **ADD_CHAR1**, **ADD_CHAR2** and **ADD_CHARN** create a new token allocated from the heap and add it to the token line pointed to by **in_start** and **in_end**.

28.5 Service Flow Diagram

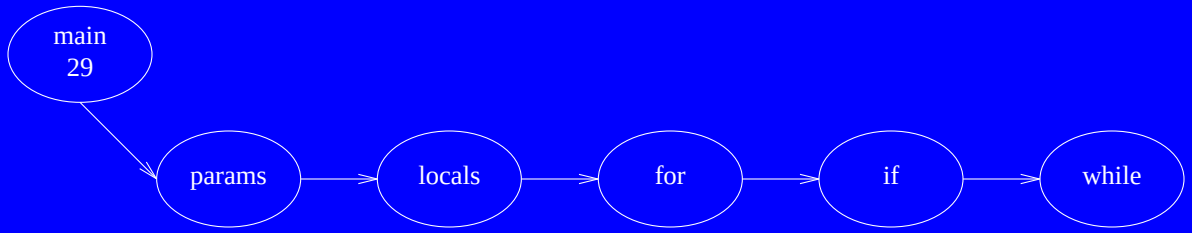


Software Unit #29 — main

29.1 Software Unit Type

Function. (main.c: 59-506)

29.2 Scope Diagram



29.3 Capabilities

Program main function. Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

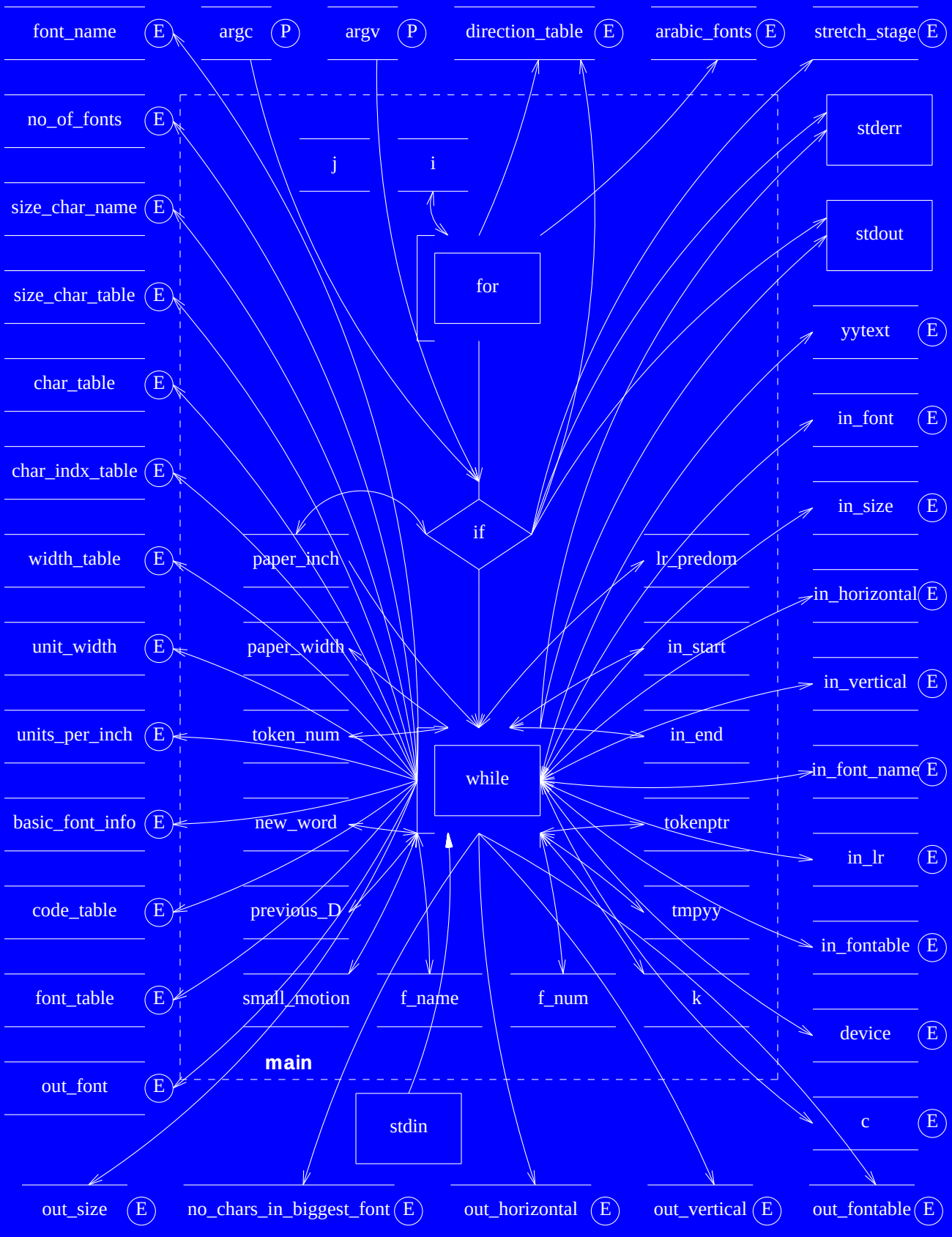
29.4 Interface

Parameters:
argc - number of command line arguments.
argv - array of pointers to command line arguments.

Return value:
Program exit status.

Side effects:
1. Reads dtroff output from stdin.
2. Prints dtroff output to stdout.
3. Prints encountered errors to stderr and halts program.
4. Allocates and frees memory from the heap.
5. If out of heap memory prints ``out of memory´´ message to stdout and halts program.
6. Changes the following external variables: **in_font**, **in_size**, **in_horizontal**, **in_vertical**, **in_font_name**, **in_lr**, **in_fontable**, **out_font**, **out_size**, **out_horizontal**, **out_vertical**, **out_fontable**, **direction_table**, **arabic_fonts**, **stretch_stage**, **device**, **c**, **font_name**, **no_of_fonts**, **size_char_name**, **size_char_table**, **char_table**, **char_indx_table**, **width_table**, **unit_width**, **units_per_inch**, **basic_font_info**, **code_table**, **font_table**, **no_chars_in_biggest_font**, **yytext**.

29.5 Service Flow Diagram

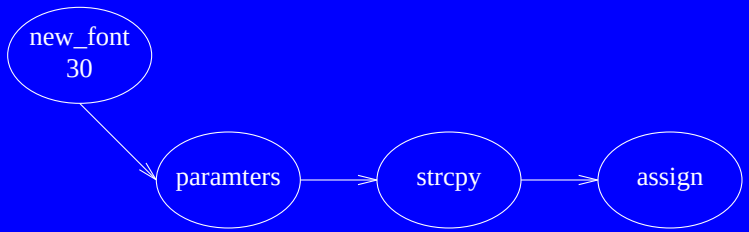


Software Unit #30 — new_font

30.1 Software Unit Type

Procedure. (misc.c: 1-41)

30.2 Scope Diagram



30.3 Capabilities

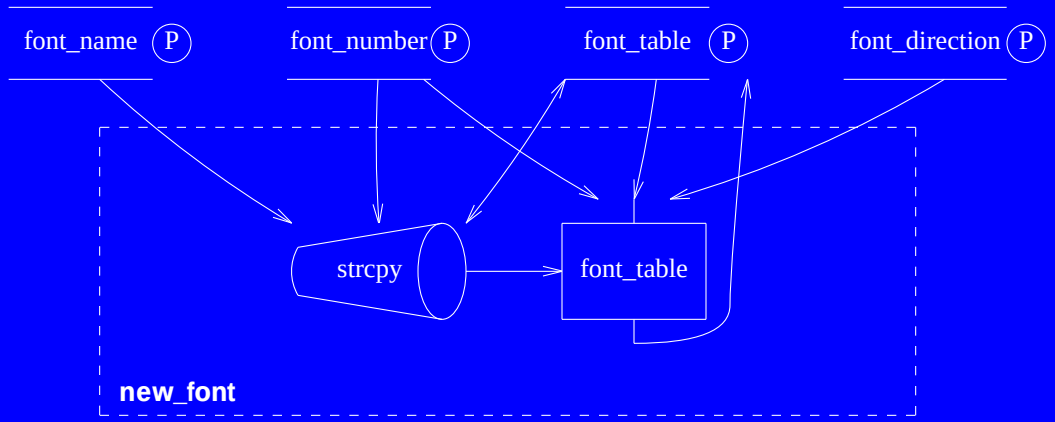
Adds a new font to the font table.

30.4 Interface

Parameters:
font_number - number of new font in font table.
font_name - string holding font name.
font_direction - direction of new font.
font_table - the font table to to add the font to.

Side effects:
Changes values in the passed font table.

30.5 Service Flow Diagram

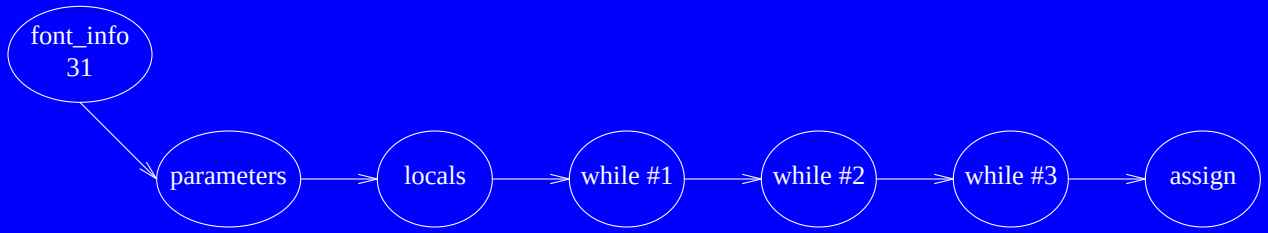


Software Unit #31 — font_info

31.1 Software Unit Type

Procedure. (misc.c: 59-99)

31.2 Scope Diagram



31.3 Capabilities

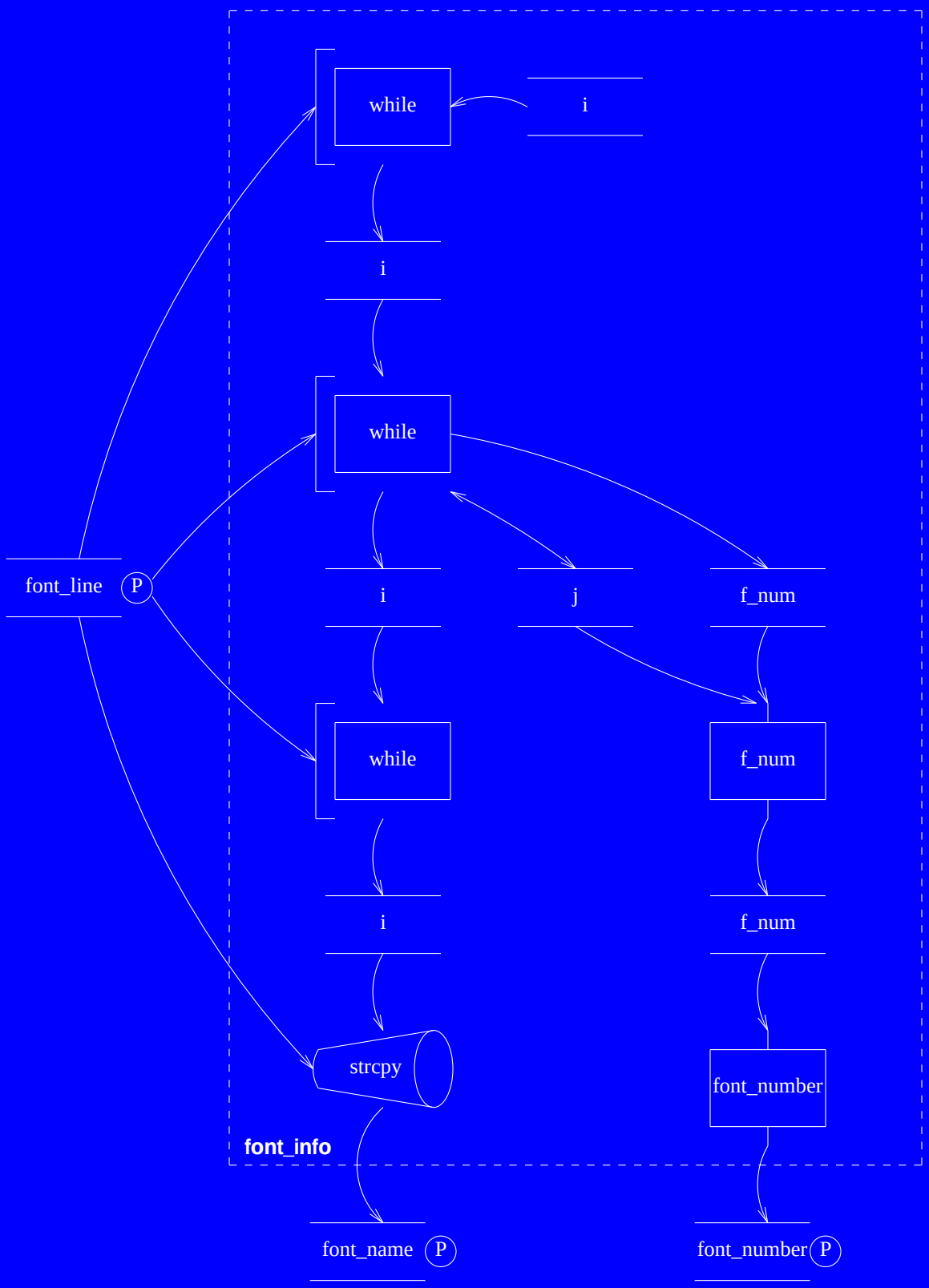
Extracts a font number and name from a font token string.

31.4 Interface

Parameters:
font_line - lex font input token line.
font_number - pointer to font number.
font_name - pointer to font name.

Side effects:
1. Returns through **font_number** the font token number.
2. Returns through **font_name** the font token name.

31.5 Service Flow Diagram

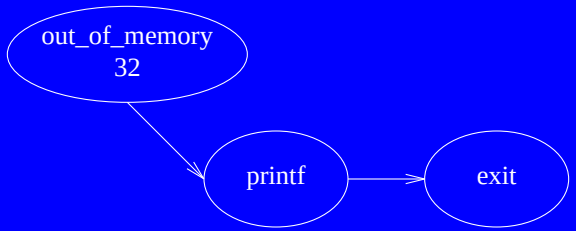


Software Unit #32 — out_of_memory

32.1 Software Unit Type

Procedure. (misc.c: 100-116)

32.2 Scope Diagram



32.3 Capabilities

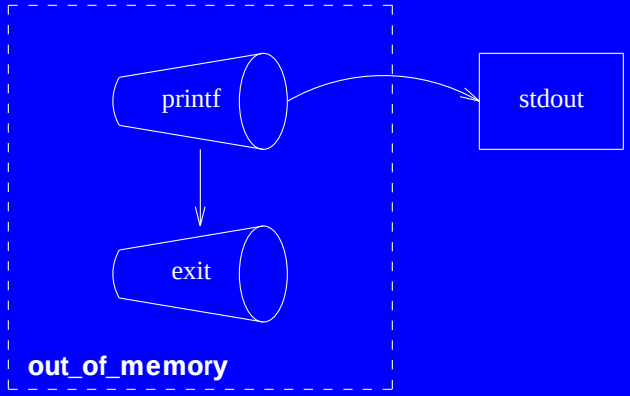
Prints an ``out of memory´´ error message and halts program execution.

32.4 Interface

Parameters:
None.

Side effects:
1. Prints ``out of memory´´ error message to stdout.
2. Causes program to halt.

32.5 Service Flow Diagram

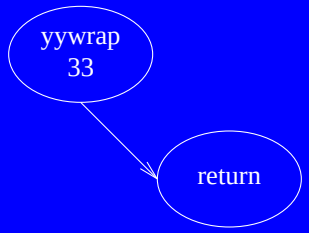


Software Unit #33 — yywrap

33.1 Software Unit Type

Function. (misc.c: 117-129)

33.2 Scope Diagram



33.3 Capabilities

Standard lex library function called whenever lex reaches an end-of-file. The default **`yywrap`** also always returns 1.

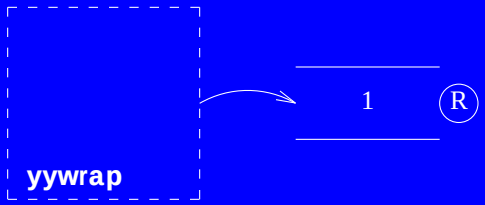
33.4 Interface

Parameters:
None.

Return value:
Always 1.

Side effects:
None.

33.5 Service Flow Diagram

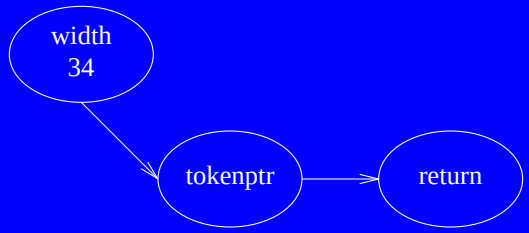


Software Unit #34 — width

34.1 Software Unit Type

Function. (misc.c: 42-58)

34.2 Scope Diagram



34.3 Capabilities

Returns 0 as the width of every token passed. Currently unused.

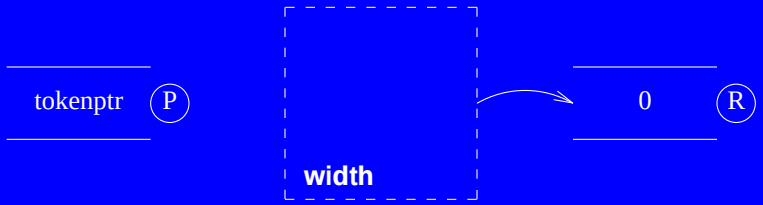
34.4 Interface

Parameters:
`tokenptr` - pointer to an internal token.

Return value:
Always 0.

Side effects:
None.

34.5 Service Flow Diagram

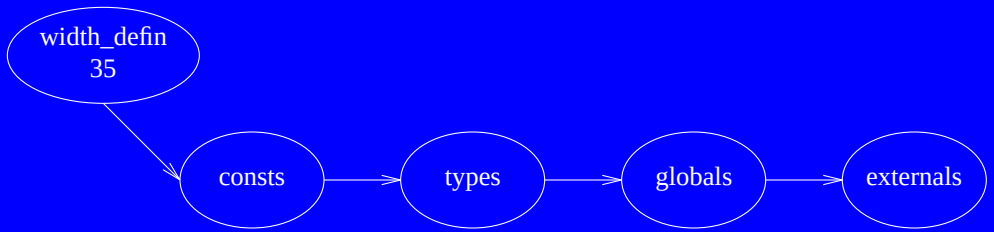


Software Unit #35 — width_defin

35.1 Software Unit Type

Definitions block. (width.c: 1-47)

35.2 Scope Diagram



35.3 Capabilities

Contains definitions and externals used by width.c functions.

35.4 Interface

Constants:
MAXNOFONTS - max number of fonts.
MAXWIDENTRIES - max width entries.
NOCHARSINBIGGESTFONT - no of characters in biggest font in device description.
MAXNOCHARS - max number of chars with with two letters or --- names.
SIZECHARINDXTABLE - size of character index table including ascii chars but not non-graphics.
FATAL - passed to error procedure to signal fatal error.
BYTEMASK - mask used to make character numbers positive.

Types:
Fontinfo - single font information structure.

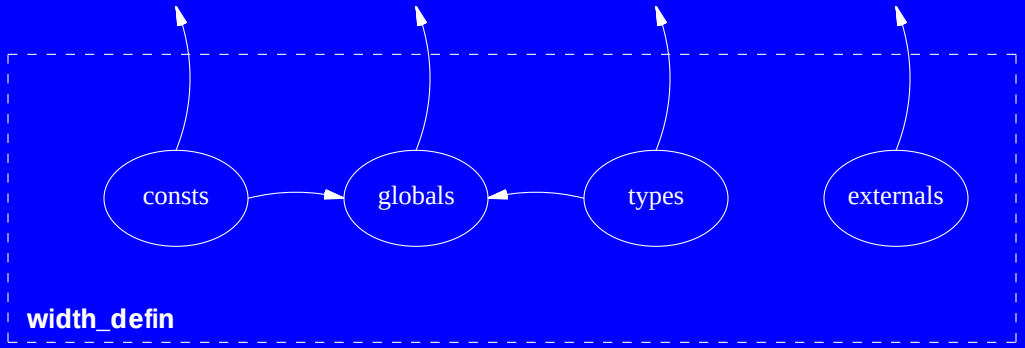
35.4 Interface - Cont

Globals:
basic_font_info - array of all fonts information.
font_name - array of all font names.
no_of_fonts - number of fonts initially mounted on the device.
indx_1st_spec_font - index of first special font.
size_char_table - size of character table in device.
unit_width - basic unit width in device.
units_per_inch - number of units per inch in device.
no_chars_in_biggest_font - number of chars in biggest font in device.
size_char_name - size of character name in device.
char_name - array of all character names in device.
char_table - array of indexes of characters in char_name.
char_indx_table - array of indexes of ascii characters in each font.
code_table - array of number codes for each char in each font.
width_table - array of widths for each char in each font.
fontdir - font files directory.

Externals:
in_size - current input font point size.
in_font - current input font number.
device - name of output device.

Side effects:
None

35.5 Service Flow Diagram

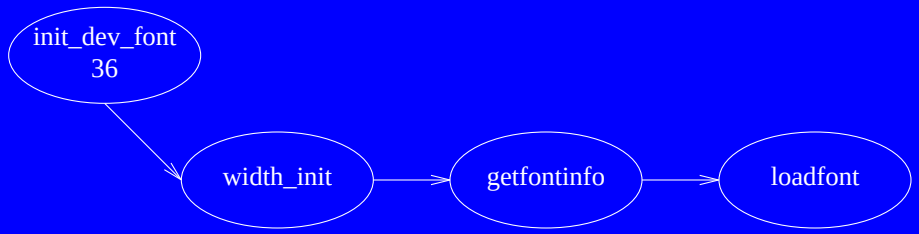


Software Unit #36 — init_dev_font

36.1 Software Unit Type

Procedure group. (width.c: 48-236 & 310-349)

36.2 Scope Diagram



36.3 Capabilities

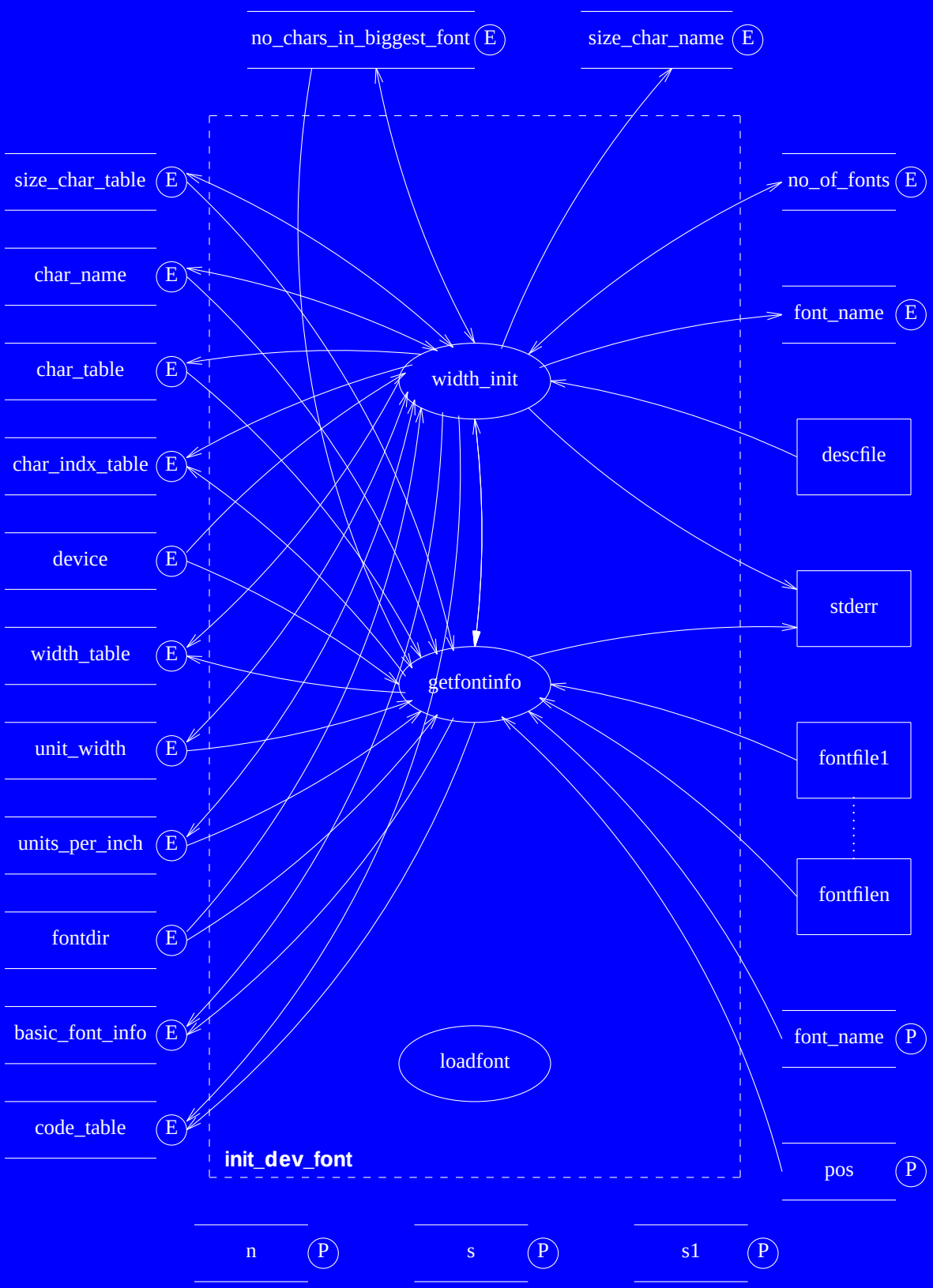
Contains routines to initialize and load device and font width tables.

36.4 Interface

Procedures:
width_init - initializes the global device and font tables.
getfontinfo - read in a single font table.
loadfont - loads a single font table. Currently body commented out.

Side effects:
1. **width_init** allocates memory from the heap.
2. Any error found in **width_init** is printed to stderr and the program halts.
3. **width_init** changes the following external variables: **size_char_table**, **char_name**, **char_table**, **char_indx_table**, **width_table**, **unit_width**, **units_per_inch**, **basic_font_info**, **code_table**, **no_chars_in_biggest_font**, **size_char_name**, **no_of_fonts**, **font_name**.
4. Any error found in **getfontinfo** is printed to stderr and the program halts.
5. **getfontinfo** changes the following external variables: **basic_font_info**, **width_table**, **code_table**, **char_indx_table**.

36.5 Service Flow Diagram

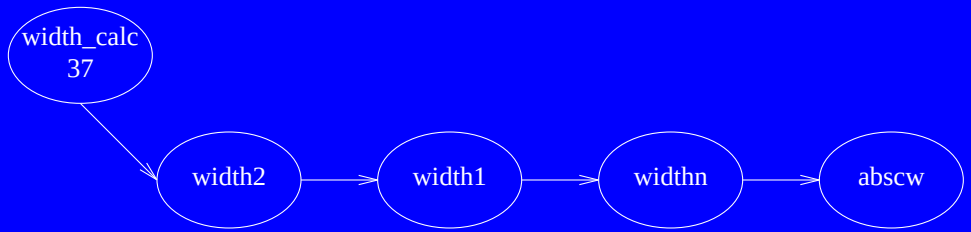


Software Unit #37 — width_calc

37.1 Software Unit Type

Function group. (width.c: 359-480)

37.2 Scope Diagram



37.3 Capabilities

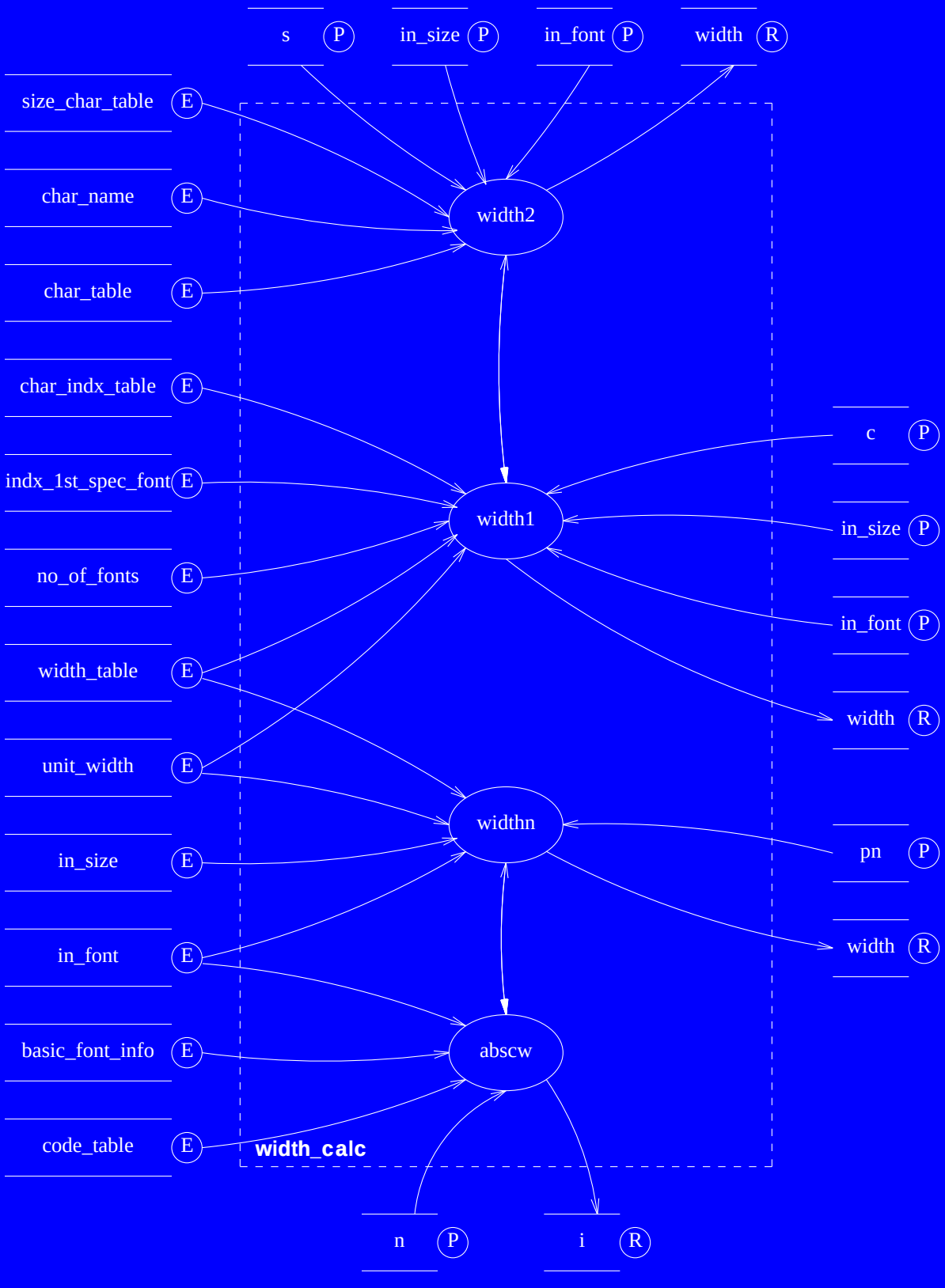
Contains routines to determine the width of different kinds of characters.

37.4 Interface

Functions:
width2 - returns the width of a specified funny character.
width1 - returns the width of a specified character.
widthn - returns the width of a character specified with its code.
abscw - returns the index of char with absolute number **n** in current **in_font**.

Side effects:
None.

37.5 Service Flow Diagram

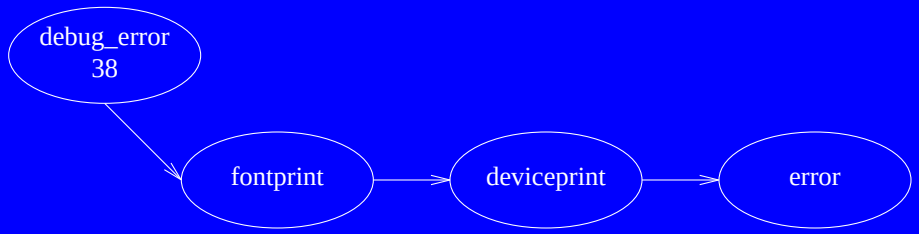


Software Unit #38 — debug_error

38.1 Software Unit Type

Procedure group. (width.c: 237-309 & 350-358)

38.2 Scope Diagram



38.3 Capabilities

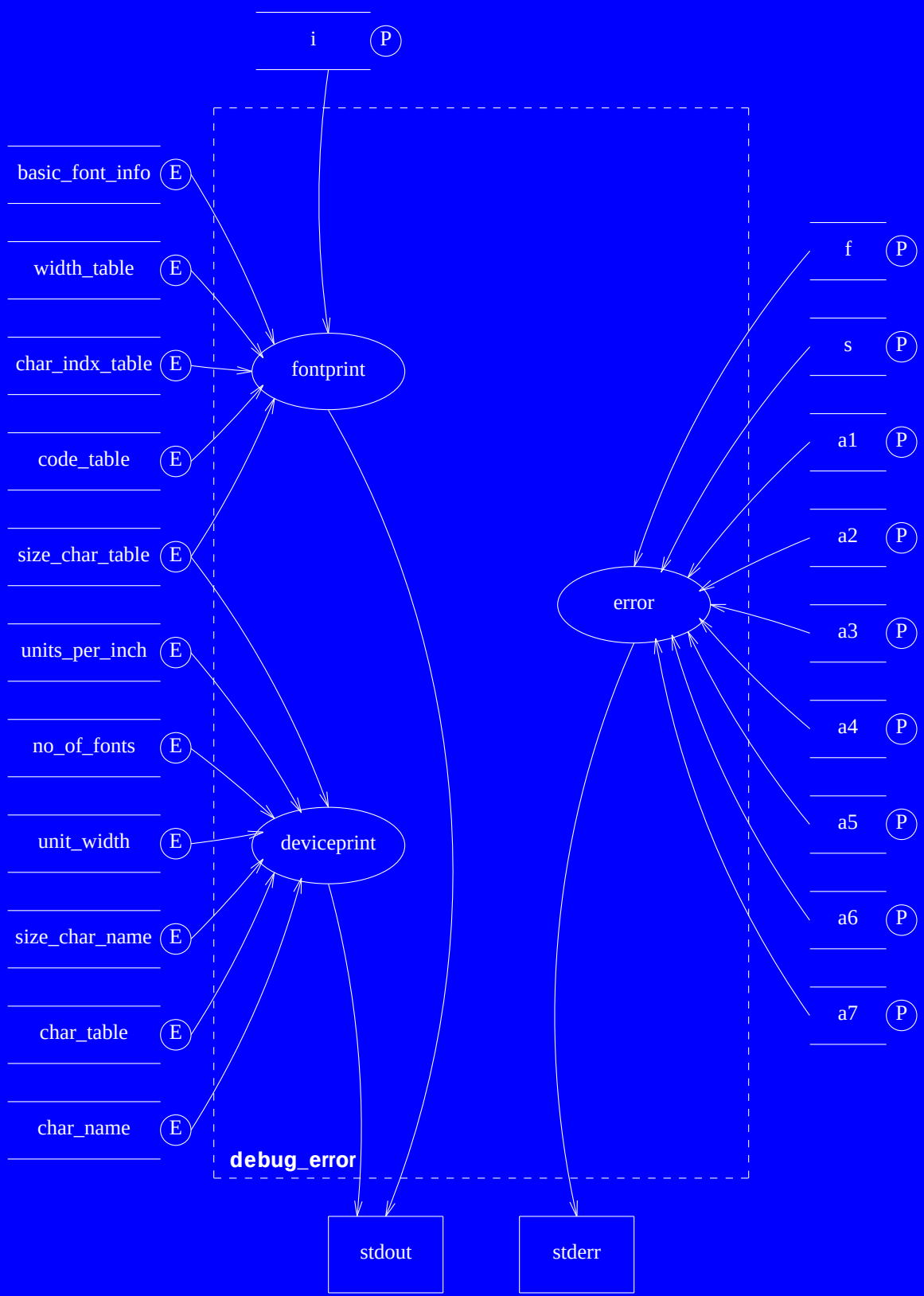
Contains routines that print font width and device tables for debugging and an error routine to print errors and halt program execution if they are fatal.

38.4 Interface

Functions:
fontprint - prints a font's width table.
deviceprint - prints the device table.
error - prints error message to stderr and halts program if fatal.

Side effects:
1. **fontprint** and **deviceprint** print to stdout font width & device tables.
2. **error** prints to stderr error information
3. **error** can halt program execution depending on the **f** parameter.

38.5 Service Flow Diagram

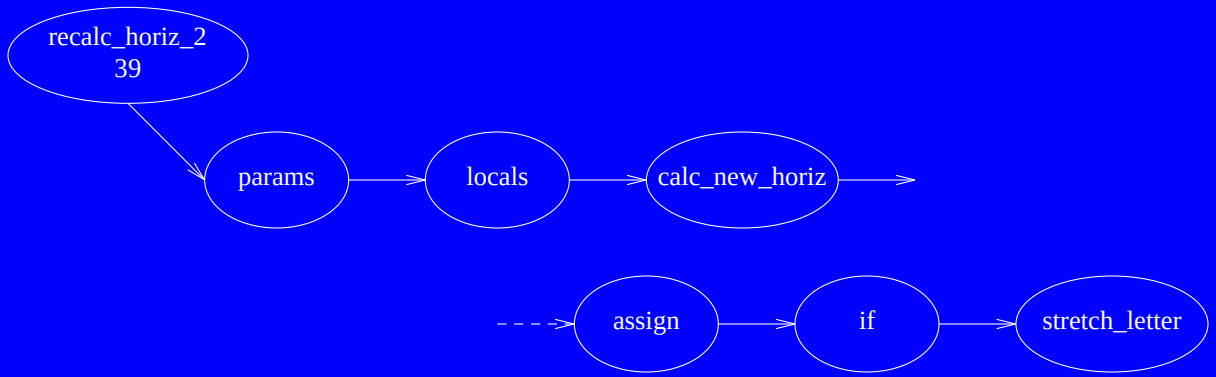


Software Unit #39 — recalc_horiz_2

39.1 Software Unit Type

Procedure. (dump.c: 221-273)

39.2 Scope Diagram



39.3 Capabilities

Recalculates the horizontal motion and stretches the specified token line.

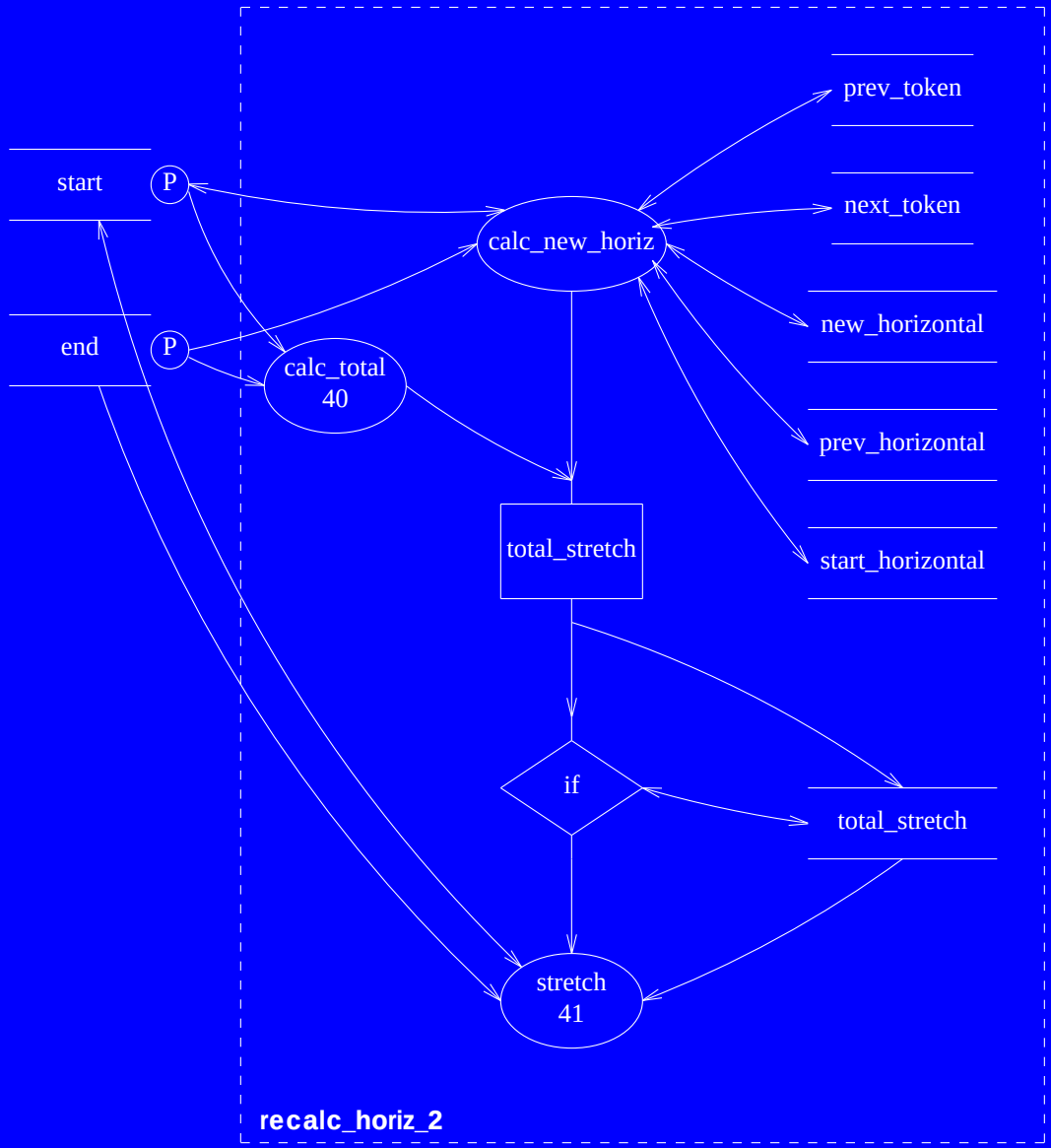
39.4 Interface

Procedure name:
`recalculate_horizontal`

Parameters:
`start` - pointer to first token in the line.
`end` - pointer to last token in the line.

Side effects:
Changes the tokens in the passed token line.

39.5 Service Flow Diagram

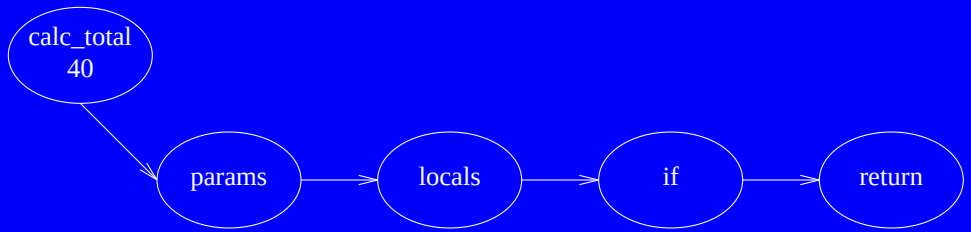


Software Unit #40 — calc_total

40.1 Software Unit Type

Function. (dump.c: 274-321)

40.2 Scope Diagram



40.3 Capabilities

Returns the total stretching amount in a line.

40.4 Interface

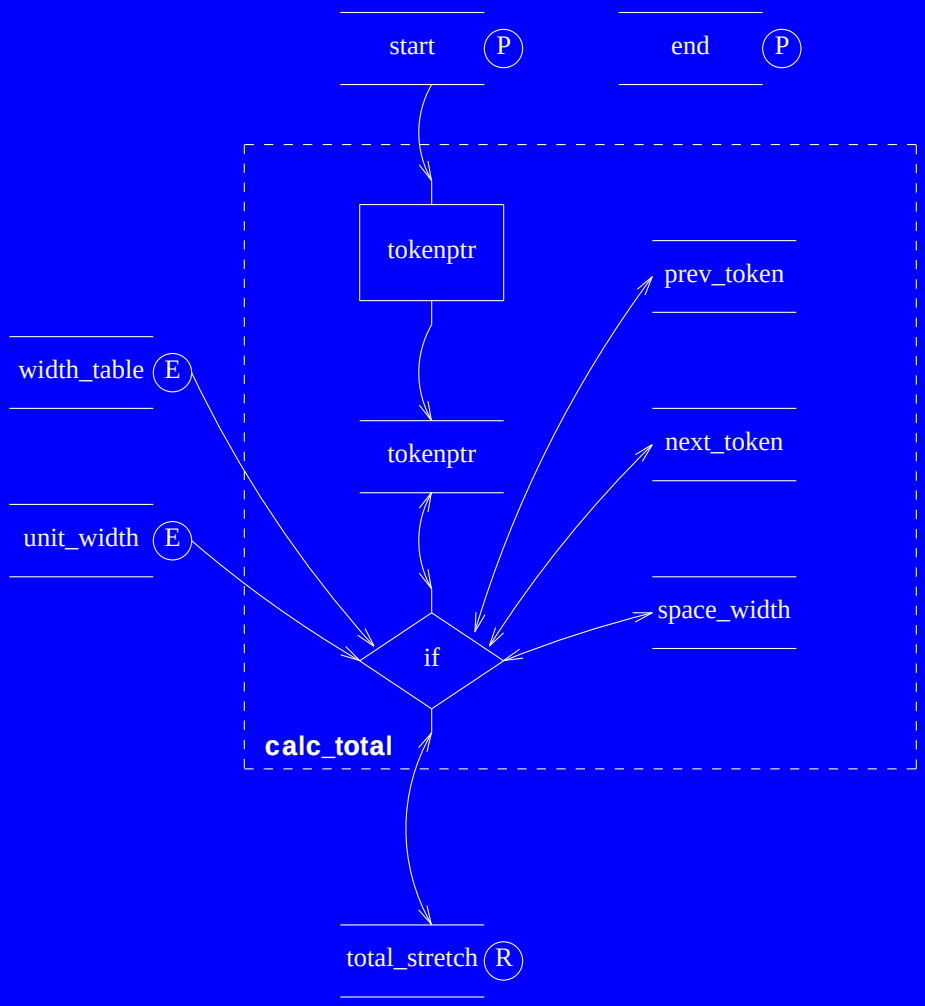
Function name:
`calc_total_stretching`

Parameters:
`start` - pointer to first token in the line.
`end` - pointer to last token in the line.

Return value:
`total_stretch` - total stretch possible in the line.

Side effects:
None.

40.5 Service Flow Diagram

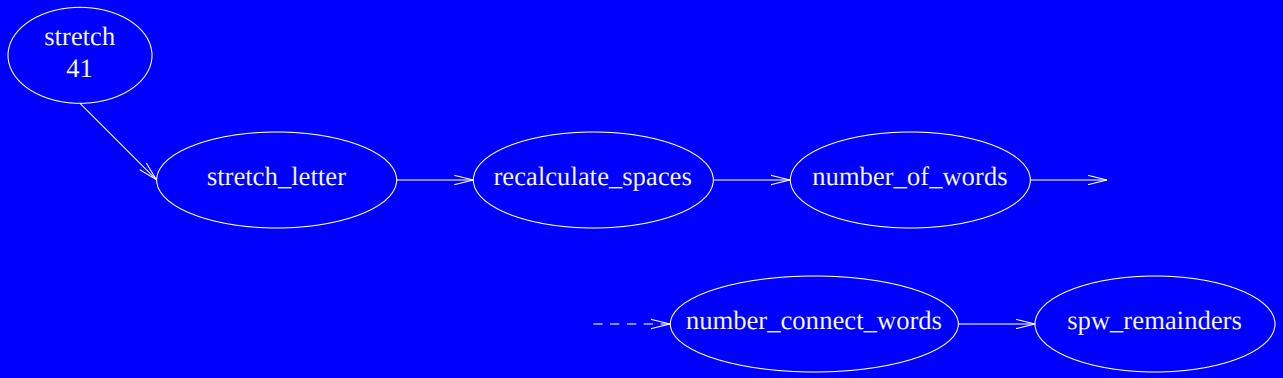


Software Unit #41 — stretch

41.1 Software Unit Type

Function group. (dump.c: 322-683)

41.2 Scope Diagram



41.3 Capabilities

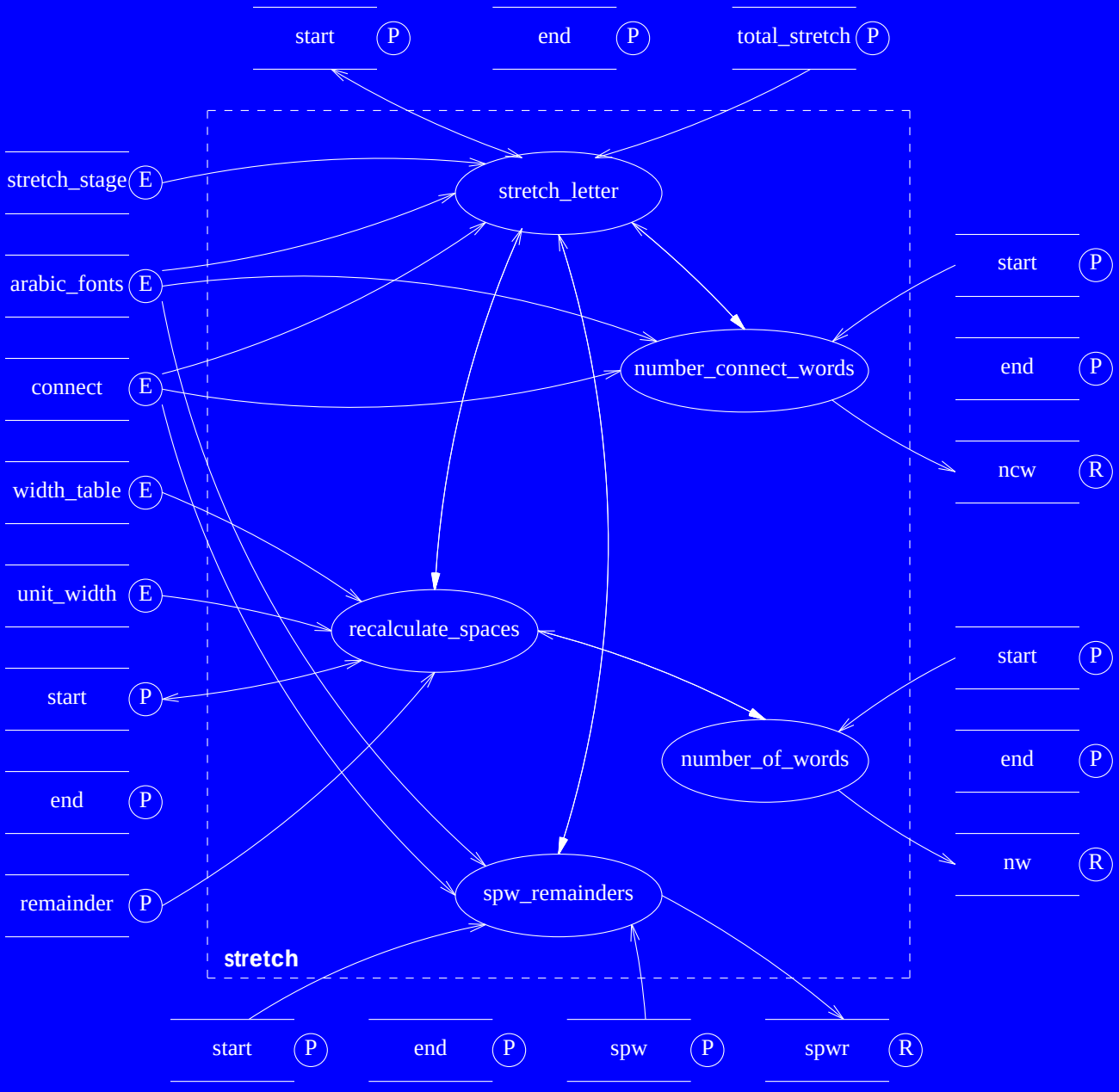
Stretches the appropriate letters in a line according to the stretch stage.

41.4 Interface

Functions:
stretch_letter - stretches the appropriate letters in a line according to the stretch stage.
recalculate_spaces - recalculates the inter words space after stretching a letter or letters.
number_of_words - returns the number of words in a line.
number_connect_words - returns the number of connectable words in a line.
spw_remainders - returns the total remainders from all of the stretchable words in a line.

Side effects:
1. **stretch_letter** changes the tokens in the passed token line.
2. **recalculate_spaces** changes the tokens in the passed token line.

41.5 Service Flow Diagram



4. Acknowledgments

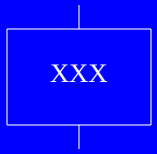
I wish to thank Prof. Daniel Berry for his helpful comments and assistance in preparing this manual. I would also like to thank Prof. Noah Prywes for his guidance and advice.

Appendix A - Service Flow Diagram Icons

Appendix B - ffortid Manual

Appendix C - ffortid Source Files

Appendix A - Service Flow Diagram Icons

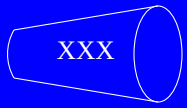


Assignment

a=b+c

XXX is the name of the variable on the left hand side of the assignment.

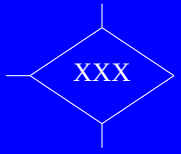
Procedure/Function Call



my_procedure(arg1,arg2)

XXX is the procedure name.

Condition



if (my_var)...else... *switch(c)*

XXX is either IF or SWITCH.

Simple Condition

if (my_var)

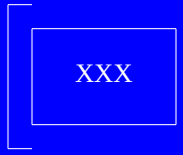
XXX is always IF without an else.

IO File

FILE fd;*

XXX is the name of the variable or the name of the file in quotes.

Loop



for(i=0;i<n;i++)... while (cond) do ... do statement while (cond)

XXX is the type of statment, e.g. FOR, WHILE or DO.

Software Unit

A single Software Unit.

XXX is the name of the SWU. If it has a number the number n is shown.

Local Variable

type name;

XXX is the name of the variable.

Parameter Variable

func(type name);

XXX is the name of the parameter variable.

Return Variable

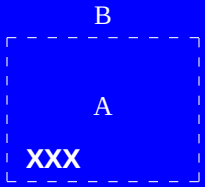
type func(arg1, arg2);

XXX is the name of the return variable.

External Variable

extern type name;

XXX is the name of the external var;



Software Unit Borderline

A is internal to SWU XXX. B is external.

XXX is the name of the SWU. All SWU in the scope of XXX are in the box.

Function/Procedure parameters group

proc is the name of the function/procedure. YYY is a parameter.

Groups function/procedure parameters and return value for SWU entry point.

Scope relationship

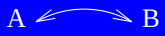
SWU A precedes SWU B in a block.

Captures precedence of SWU within a block.

Data Flow Relationship

Data flows between SWU A and SWU B.

Relationship between a consumer and producer of data.



Bi-Directional Data Flow Relationship

Data flows between SWU A and SWU B and vice-versa.

Bi-Directional relationship between a consumer and producer of data.

Call Relationship

SWU A calls a function or procedure in SWU B.

Relationship between a function/procedure caller and the callee.

Use relationship

SWU B uses declerations or definitions in SWU A.

Relationship between decleration/definition in a SWU and it’s use in another SWU.

Appendix B - ffortid Manual Page

NAME

ffortid – in dtroff output, find and reverse all text in designated right-to-left fonts and carry out stretching in Arabic and Farsi text

SYNOPSIS

ffortid [–**r***font-position-list*] ... [–**w***paperwidth*] [–**a***font-position-list*] ...
[–**s**[**n**|**f**|**l**|**a**]] [*file*] ...

DESCRIPTION

ffortid's job is to take the *dtroff*(1) output which is formatted strictly left-to-right, to find occurrences of text in a right-to-left font and to rearrange each line so that the text in each font is written in its proper direction. *ffortid* deals exclusively with *dtroff* output, it does not know and does not need to know anything about any of *dtroff*'s preprocessors. Therefore, the results of using *ffortid* with any of *dtroff*'s preprocessors depends only on the *dtroff* output generated as a result of the input to *dtroff* from the preprocessors. Furthermore, the output of *ffortid* goes on to the same device drivers to which the *dtroff* output would go; therefore, *ffortid*'s output is in the same form as that of *dtroff*.

In the command line, the –**r***font-position-list* argument is used to specify which font positions are to be considered right-to-left. A *font-position-list* is a list of font positions separated by white space, but with no white space at the beginning. *ffortid*, like *ditroff*, recognizes up to 256 possible font positions (0-255). The actual number of available font positions depends only on the typesetting device and its associated *ditroff* device driver. The default font direction for all possible font positions is left-to-right. Once the font direction is set, either by default or with the –**r***font-position-list* argument, it remains in effect throughout the entire document. Observe then that *ffortid*'s processing is independent of what glyphs actually get printed for the mounted fonts. It processes the designated fonts as right-to-left

fonts even if, in fact, the alphabet is that of a left-to-right language. In fact, it is possible that the same font be mounted in two different positions, only one of which is designated as a right-to-left font position. This is how a single font can be printed left-to-right and right-to-left in the same document.

In addition to the specified font directions, the results of *ffortid*'s reformatting also depends on the document's *current formatting direction*, which can be either left-to-right or right-to-left. The default formatting direction is left-to-right and can be changed by the user at any point in the document through the use of the

```
x X PL
```

and

```
x X PR
```

commands in the *dtroff* output. These commands set the current formatting direction to left-to-right and right-to-left, respectively. These commands are either meaningless or erroneous to *dtroff* device drivers; therefore they are removed by *ffortid* as they are obeyed. These commands can be generated by use of

```
\X'PL'
```

and

```
\X'PR'
```

escapes in the *dtroff* input. For the convenience of the user, two macros

```
.PL
```

and

```
.PR
```

are defined in the *–mX2* and *–mXP* macro packages, that cause generation of the proper input to *ffortid*. They are defined by

```
..de PL
\\X'PL'
..
.de PR
\\X'PR'
..
```

If the current formatting direction is left-to-right, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from left to right. In each *dtroff* output line, any sequence of contiguous right-to-left font characters is reversed in place.

If the current formatting direction is right-to-left, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from right to left. Each *dtroff* output line is reversed, including both the left and right margins. Then, any sequence of contiguous left-to-right font characters is reversed in place.

The **–w***paperwidth* argument is used to specify the width of the paper, in inches, on which the document will be printed. As explained later, *ffortid* uses the specified paper width to determine the width of the right margin. The default paper width is 8.5 inches and like the font directions, it remains in effect throughout the entire document.

It is important to note that *ffortid* uses the specified paper width to determine the margin widths in the reformated output line. For instance, suppose that a document is formatted for printing on paper 8.5 inches wide with a left margin (page offset) of 1.5 inches and a line length of 6 inches. This results in a right margin of 1 inch. Suppose then that the text specifies a current formatting direction of right-to-left. Then, *ffortid*'s reformatting of the output line results in left and right margins of 1 and 1.5 inches, respectively. This margin calculation works well for documents formatted entirely in one direction. However, as a document's formatting direction changes, the resulting margins widths are exchanged. Thus, **.PL**'s right and left margins end up not being the same as **.PR**'s right and left margins. The user can make *ffortid* preserve the left and right margins by specifying, with the **–w***paperwidth* argument, a paper width other than the actual paper width. This artificial paper width should be chosen so that both margins will appear to *ffortid* to be as wide as the desired left margin. For example, for the document mentioned above, a specified paper width of 9.0 inches results in reformatted left and right margins of 1.5 inches each. The resulting excess in the right margin is just white space that effectively falls of the edge of the paper and does not effect the formatting of the document.

There is one exception to these simple rotation rules in that *ffortid*, at present, makes no attempt to reverse any of *dtroff*'s drawing functions, such as those used by *pic*(1) and *ideal*(1) (which are also available directly to the user). It is therefore suggested that these drawing functions, and thus *pic* and *ideal*, be used only when the current formatting direction is left-to-right. Additionally, due to the cleverness of the *dtroff* output generated by most substantial *eqn*(1) equations, it is suggested that *eqn*'s use also be limited to a left-to-right formatting direction for all but the simplest forms of equations. These rules do not in any way restrict the use of right-to-left fonts in the text dealt with by any of the preprocessors, but simply suggest that these particular preprocessors be used only when the formatting direction is left-to-right.

An additional point to keep in mind when preparing input both for *dtroff*'s preprocessors and for *dtroff* itself is that *ffortid* rotates, as a unit, each sequence of characters of the same direction. In order to force *ffortid* to rotate parts of a sequence independently, as for a *tbl*(1) table, one must artificially separate them with a change to a font of the opposite direction.

The **–a***font-position-list* argument is used to indicate which fonts positions, generally a subset of those designated as right-to-left, contain fonts for Arabic, Farsi, or related languages. For these fonts, left and right justification of a line is achieved by stretching instead of inserting extra white space between the words in the line. Stretching is done on a line only if the line contains at least one word in a **–a** designated font. If so, stretching is used in place of extra white space insertion for the entire line. There are several kinds of stretching, and which is in effect for all **–a** designated fonts is specified with the **–s** option, described below. If it is desired not to stretch a particular Arabic, Farsi, or other font, while still stretching others, then the particular font should not be listed in the **–a***font-position-list*. Words in such fonts will not be stretched and will be spread with extra white space if the containing line is spread with extra white space. The **–r** and the **–a** specifications are independent. If a font is in the **–a***font-position-list* but not in the **–r***font-position-list*, then its text will be stretched but not

reversed. This independence can be used to advantage when it is necessary to designate a particular Arabic, Farsi, or other font as left-to-right for examples or to get around the above mentioned limitations in the use of *eqn*, *ideal*, *pic*, or *tbl*.

The kind of stetching to be done for all fonts designated in the –**a***font-position-list* is indicated by the –**s** argument. The choices are:

–sn

Do no stretching at all for all the fonts.

–sf

Stretch the last stretchable word on each line. A stretchable word is a word containing a stretchable character (if the font is dynamic) or a stretchable connection to a character (if the font has a straight base line). *Currently only stretchable connections to characters are handled; a future version will deal with dynamic fonts*. If no stretchable word exists on the line, then spread the words in the line as does *dtroff*.

–sl

Stretch the last stretchable word on each line. If the amount of stretch for that word is longer than a limit equal to the current point size times the length of the base line filler used to achieve the stretched connection, then stretch the penultimate stretchable word up to that limit, and if necessary, then stretch the stretchable word before that, etc. If no stretchable word exists on the line, or some extra stretch is left after stretching all stretchable words to the limit, then spread the words in the line as does *dtroff*.

–sa

Stretch all stretchable words on each line by the same amount (different amount for each line). If

no stretchable word exists on the line, then spread the words in the line as does *dtroff*. This is the default for all –**a** designated fonts.

FILES

/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/font/dev*/*	device description and font width tables

SEE ALSO

Cary Buchman, Daniel M. Berry, *User's Manual for Ditroff/Ffortid, An Adaptation of the UNIX Ditroff for Formatting Bi-Directional Text*,
Johny Srouji, Daniel M. Berry, *An Adaptation of the UNIX Ditroff for Arabic Formatting*
troffort(l), ptrn(l)

Appendix C - ffortid Source Files

```
1    # define s_token             1
2    # define f_token             2
3    # define c_token             3
4    # define C_token             4
5    # define H_token             5
6    # define V_token             6
7    # define h_token             7
8    # define v_token             8
9    # define hc_token            9
10   # define n_token            10
11   # define w_token            11
12   # define p_token            12
13   # define trail_token        13
14   # define stop_token         14
15   # define dev_token          15
16   # define res_token          16
17   # define init_token         17
18   # define font_token         18
19   # define pause_token        19
20   # define height_token       20
21   # define slant_token        21
22   # define newline_token      22
23   # define PR_token           23
24   # define PL_token           24
25   # define D_token            25
26   # define N_token            26
27   # define include_token      27
28   # define control_token      28
29   # define postscript_begin_token  29
30   # define psfig_begin_token  30
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3    %%
4    s[0-9]+                                  {return (s_token);}
5    f[0-9]+                                  {return (f_token);}
6    c.                                       {return (c_token);}
7    C..                                      {return (C_token);}
8    N[0-9]+                                  {return (N_token);}
9    H[0-9]+                                  {return (H_token);}
10   V[0-9]+                                  {return (V_token);}
11   h[0-9]+                                  {return (h_token);}
12   v[0-9]+                                  {return (v_token);}
13   n[0-9]+" "[0-9]+                         {return (n_token);}
14   [0-9][0-9].                              {return (hc_token);}
15   w                                        {return (w_token);}
16   p[0-9]+                                  {return (p_token);}
17   x" "trailer                              {return (trail_token);}
18   x" "stop                                 {return (stop_token);}
19   x" "T" ".+                               {return (dev_token);}
20   x" "r(es)?" "[0-9]+" "[0-9]+" "[0-9]+    {return (res_token);}
21   x" "i(nit)?                              {return (init_token);}
22   x" "f(ont)?" "[0-9]+" ".+                {return (font_token);}
23   x" "p(ause)?                             {return (pause_token);}
24   x" "H" "[0-9]+                           {return (height_token);}
25   x" "S" "[0-9]+                           {return (slant_token);}
26   "\n"                                     {return (newline_token);}
27   x" "PR(\\)?                              {return (PR_token);}
28   x" "PL(\\)?                              {return (PL_token);}
29   x" "X" "PR                               {return (PR_token);}
30   x" "X" "PL                               {return (PL_token);}
31   x" "X" "p.+(\\)?$                        {return (psfig_begin_token);}
```

```
32   x" "TS(\\)?                {return (control_token);}
33   x" "TE(\\)?                {return (control_token);}
34   D.*$                       {return (D_token);}
35   &.+\\$                     {return (include_token);}
36   \%PB(\\)?                  {return (postscript_begin_token);}
37   %%
```

```
1   /*****************************************************************
2   *****************************************************************
3   **                                                             **
4   **   this file contains the type definition of the internal token  **
5   **   representation structure                                  **
6   **                                                             **
7   *****************************************************************
8   *****************************************************************/
9
10   typedef int bool;
11
12   typedef struct tokn
13   {
14     int          token_type;              /* token type                */
15     int          point_size;              /* point size                */
16     int          font      ;              /* font number               */
17     char         font_name[3];            /* font name                 */
18     bool         lr        ;              /* font direction            */
19     int          width     ;              /* character width           */
20     int          vertical_pos;            /* vertical position         */
21     int          horizontal_pos;          /* horizontal position       */
22     bool         begining  ;              /* begining of word indicator */
23     bool         ending    ;              /* end of word indicator     */
24     int          fillers_num;             /* how many fillers to put   */
25     int          filler_width;            /* the filler width          */
26     char         char1     ;              /* 1st character of (abs)char token*/
27     char         char2     ;              /* 2nd character of (abs)char token*/
28     char         char3     ;              /* 3rd character of abs char token*/
29     struct tokn  *next     ;              /* next token                */
30   } TOKENTYPE, *TOKENPTR;
31
```

```
32
33   /*****************************************************************************/
34   /*****************************************************************************/
```

```
1
2    /*
3         this file contains general constant and macro definitions
4                                                                    */
5
6    #define  BEGINING            1
7    #define  NOT_BEGIN           0
8    #define  LEFT_TO_RIGHT       1
9    #define  RIGHT_TO_LEFT       0
10   #define  END                 1
11   #define  NOT_END             0
12   #define  TRUE                1
13   #define  FALSE               0
14   #define  NOFILLERS          0
15   #define  ARABIC             1
16   #define  DUMP_LEX(TOKN)      printf("%s\n",TOKN)
17   #define  SET_DIRECTION(FNUM,DIR)  direction_table[FNUM] = DIR
18   #define  FONT_DIRECTION(FNUM)     direction_table[FNUM]
19   #define  SET_AR_FONT(FNUM)        arabic_fonts[FNUM] = TRUE
20   #define  RESET_AR_FONT(FNUM)      arabic_fonts[FNUM] = FALSE
```

```
1    "000" ,  FALSE ,
2    "001" ,  FALSE ,
3    "002" ,  FALSE ,
4    "003" ,  FALSE ,
5    "004" ,  FALSE ,
6    "005" ,  FALSE ,
7    "006" ,  FALSE ,
8    "007" ,  FALSE ,
9    "010" ,  FALSE ,
10   "011" ,  FALSE ,
11   "012" ,  FALSE ,
12   "013" ,  FALSE ,
13   "014" ,  FALSE ,
14   "015" ,  FALSE ,
15   "016" ,  FALSE ,
16   "017" ,  FALSE ,
17   "020" ,  FALSE ,
18   "021" ,  FALSE ,
19   "022" ,  FALSE ,
20   "023" ,  FALSE ,
21   "024" ,  TRUE  ,
22   "025" ,  TRUE  ,
23   "026" ,  TRUE  ,
24   "027" ,  TRUE  ,
25   "030" ,  FALSE ,
26   "031" ,  FALSE ,
27   "032" ,  FALSE ,
28   "033" ,  FALSE ,
29   "034" ,  FALSE ,
30   "035" ,  FALSE ,
31   "036" ,  FALSE ,
```

```
32   "037" ,  FALSE ,
33   "040" ,  FALSE ,
34   "041" ,  FALSE ,
35   "042" ,  FALSE ,
36   "043" ,  FALSE ,
37   "044" ,  FALSE ,
38   "045" ,  FALSE ,
39   "046" ,  FALSE ,
40   "047" ,  FALSE ,
41   "050" ,  FALSE ,
42   "051" ,  FALSE ,
43   "052" ,  FALSE ,
44   "053" ,  FALSE ,
45   "054" ,  FALSE ,
46   "055" ,  FALSE ,
47   "056" ,  FALSE ,
48   "057" ,  FALSE ,
49   "060" ,  FALSE ,
50   "061" ,  FALSE ,
51   "062" ,  FALSE ,
52   "063" ,  FALSE ,
53   "064" ,  FALSE ,
54   "065" ,  FALSE ,
55   "066" ,  FALSE ,
56   "067" ,  FALSE ,
57   "070" ,  FALSE ,
58   "071" ,  FALSE ,
59   "072" ,  FALSE ,
60   "073" ,  FALSE ,
61   "074" ,  FALSE ,
62   "075" ,  FALSE ,
```

```
63   "076" ,  FALSE ,
64   "077" ,  FALSE ,
65   "0100"   ,  FALSE ,
66   "0101"   ,  FALSE ,
67   "0102"   ,  FALSE ,
68   "0103"   ,  FALSE ,
69   "0104"   ,  FALSE ,
70   "0105"   ,  FALSE ,
71   "0106"   ,  FALSE ,
72   "0107"   ,  FALSE ,
73   "0110"   ,  FALSE ,
74   "0111"   ,  FALSE ,
75   "0112"   ,  FALSE ,
76   "0113"   ,  FALSE ,
77   "0114"   ,  FALSE ,
78   "0115"   ,  FALSE ,
79   "0116"   ,  FALSE ,
80   "0117"   ,  FALSE ,
81   "0120"   ,  FALSE ,
82   "0121"   ,  FALSE ,
83   "0122"   ,  FALSE ,
84   "0123"   ,  FALSE ,
85   "0124"   ,  FALSE ,
86   "0125"   ,  FALSE ,
87   "0126"   ,  FALSE ,
88   "0127"   ,  FALSE ,
89   "0130"   ,  FALSE ,
90   "0131"   ,  FALSE ,
91   "0132"   ,  FALSE ,
92   "0133"   ,  FALSE ,
93   "0134"   ,  FALSE ,
```

```
94   "0135"   ,  FALSE ,
95   "0136"   ,  FALSE ,
96   "0137"   ,  FALSE ,
97   "0140"   ,  FALSE ,
98   "0141"   ,  FALSE ,
99   "0142"   ,  FALSE ,
100  "0143"   ,  FALSE ,
101  "0144"   ,  FALSE ,
102  "0145"   ,  FALSE ,
103  "0146"   ,  FALSE ,
104  "0147"   ,  FALSE ,
105  "0150"   ,  FALSE ,
106  "0151"   ,  FALSE ,
107  "0152"   ,  FALSE ,
108  "0153"   ,  FALSE ,
109  "0154"   ,  FALSE ,
110  "0155"   ,  FALSE ,
111  "0156"   ,  FALSE ,
112  "0157"   ,  FALSE ,
113  "0160"   ,  FALSE ,
114  "0161"   ,  FALSE ,
115  "0162"   ,  FALSE ,
116  "0163"   ,  FALSE ,
117  "0164"   ,  FALSE ,
118  "0165"   ,  FALSE ,
119  "0166"   ,  FALSE ,
120  "0167"   ,  FALSE ,
121  "0170"   ,  FALSE ,
122  "0171"   ,  FALSE ,
123  "0172"   ,  FALSE ,
124  "0173"   ,  FALSE ,
```

```
125  "0174"   ,  FALSE ,
126  "0175"   ,  FALSE ,
127  "0176"   ,  FALSE ,
128  "0177"   ,  FALSE ,
129  "0200"   ,  FALSE ,
130  "0201"   ,  FALSE ,
131  "0202"   ,  FALSE ,
132  "0203"   ,  FALSE ,
133  "0204"   ,  FALSE ,
134  "0205"   ,  FALSE ,
135  "0206"   ,  FALSE ,
136  "0207"   ,  FALSE ,
137  "0210"   ,  FALSE ,
138  "0211"   ,  FALSE ,
139  "0212"   ,  FALSE ,
140  "0213"   ,  FALSE ,
141  "0214"   ,  FALSE ,
142  "0215"   ,  FALSE ,
143  "0216"   ,  FALSE ,
144  "0217"   ,  FALSE ,
145  "0220"   ,  FALSE ,
146  "0221"   ,  FALSE ,
147  "0222"   ,  FALSE ,
148  "0223"   ,  FALSE ,
149  "0224"   ,  FALSE ,
150  "0225"   ,  TRUE  ,
151  "0226"   ,  TRUE  ,
152  "0227"   ,  TRUE  ,
153  "0230"   ,  TRUE  ,
154  "0231"   ,  TRUE  ,
155  "0232"   ,  TRUE  ,
```

```
156  "0233"   ,  TRUE  ,
157  "0234"   ,  TRUE  ,
158  "0235"   ,  TRUE  ,
159  "0236"   ,  TRUE  ,
160  "0237"   ,  TRUE  ,
161  "0240"   ,  TRUE  ,
162  "0241"   ,  TRUE  ,
163  "0242"   ,  TRUE  ,
164  "0243"   ,  TRUE  ,
165  "0244"   ,  FALSE ,
166  "0245"   ,  TRUE  ,
167  "0246"   ,  TRUE  ,
168  "0247"   ,  TRUE  ,
169  "0250"   ,  TRUE  ,
170  "0251"   ,  TRUE  ,
171  "0252"   ,  TRUE  ,
172  "0253"   ,  TRUE  ,
173  "0254"   ,  TRUE  ,
174  "0255"   ,  TRUE  ,
175  "0256"   ,  TRUE  ,
176  "0257"   ,  TRUE  ,
177  "0260"   ,  TRUE  ,
178  "0261"   ,  TRUE  ,
179  "0262"   ,  TRUE  ,
180  "0263"   ,  TRUE  ,
181  "0264"   ,  TRUE  ,
182  "0265"   ,  TRUE  ,
183  "0266"   ,  TRUE  ,
184  "0267"   ,  FALSE ,
185  "0270"   ,  TRUE  ,
186  "0271"   ,  TRUE  ,
```

```
187  "0272"   ,  TRUE  ,
188  "0273"   ,  TRUE  ,
189  "0274"   ,  TRUE  ,
190  "0275"   ,  TRUE  ,
191  "0276"   ,  TRUE  ,
192  "0277"   ,  TRUE  ,
193  "0300"   ,  TRUE  ,
194  "0301"   ,  TRUE  ,
195  "0302"   ,  TRUE  ,
196  "0303"   ,  TRUE  ,
197  "0304"   ,  TRUE  ,
198  "0305"   ,  TRUE  ,
199  "0306"   ,  TRUE  ,
200  "0307"   ,  TRUE  ,
201  "0310"   ,  TRUE  ,
202  "0311"   ,  TRUE  ,
203  "0312"   ,  TRUE  ,
204  "0313"   ,  TRUE  ,
205  "0314"   ,  TRUE  ,
206  "0315"   ,  TRUE  ,
207  "0316"   ,  TRUE  ,
208  "0317"   ,  TRUE  ,
209  "0320"   ,  TRUE  ,
210  "0321"   ,  TRUE  ,
211  "0322"   ,  TRUE  ,
212  "0323"   ,  TRUE  ,
213  "0324"   ,  TRUE  ,
214  "0325"   ,  TRUE  ,
215  "0326"   ,  TRUE  ,
216  "0327"   ,  TRUE  ,
217  "0330"   ,  TRUE  ,
```

```
218  "0331"   ,  TRUE  ,
219  "0332"   ,  FALSE ,
220  "0333"   ,  FALSE ,
221  "0334"   ,  TRUE  ,
222  "0335"   ,  FALSE ,
223  "0336"   ,  TRUE  ,
224  "0337"   ,  FALSE ,
225  "0340"   ,  TRUE  ,
226  "0341"   ,  FALSE ,
227  "0342"   ,  TRUE  ,
228  "0343"   ,  FALSE ,
229  "0344"   ,  TRUE  ,
230  "0345"   ,  FALSE ,
231  "0346"   ,  TRUE  ,
232  "0347"   ,  FALSE ,
233  "0350"   ,  FALSE ,
234  "0351"   ,  TRUE  ,
235  "0352"   ,  FALSE ,
236  "0353"   ,  TRUE  ,
237  "0354"   ,  FALSE ,
238  "0355"   ,  FALSE ,
239  "0356"   ,  FALSE ,
240  "0357"   ,  FALSE ,
241  "0360"   ,  FALSE ,
242  "0361"   ,  FALSE ,
243  "0362"   ,  FALSE ,
244  "0363"   ,  FALSE ,
245  "0364"   ,  FALSE ,
246  "0365"   ,  FALSE ,
247  "0366"   ,  FALSE ,
248  "0367"   ,  FALSE ,
```

```
249  "0370"   ,  FALSE ,
250  "0371"   ,  FALSE ,
251  "0372"   ,  FALSE ,
252  "0373"   ,  FALSE ,
253  "0374"   ,  FALSE ,
254  "0375"   ,  FALSE ,
255  "0376"   ,  FALSE ,
256  "0377"   ,  FALSE
```

```
1    /*****************************************************************
2    *****************************************************************
3    **                                                             **
4    **   this file contains the type definition of the internal font   **
5    **   table structure                                           **
6    **                                                             **
7    *****************************************************************
8    *****************************************************************/
9
10    typedef struct fntable
11    {
12      char         name[3];              /* font name      */
13      bool         direction;            /* font direction */
14    } TABLENTRY;;
15
16
17   /*****************************************************************/
18   /*****************************************************************/
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3    /*************************************************************************
4    *************************************************************************
5    **                                                                     **
6    ** this file includes routines that dump the current internal line     **
7    **                                                                     **
8    *************************************************************************
9    *************************************************************************/
10
11   #include <stdio.h>
12   #include "TOKEN.h"
13   #include "lex.h"
14   #include "macros.h"
15
16   #define  max(a,b)       ((a<b) ? b : a)
17   #define  MAXZWC         100        /* maximum number pf respective zero
18                                         width characters        */
19   #define  NFONT          255        /* defined also in width.c */
20   TOKENPTR new_token();              /* function return type    */
21
22   struct cnct
23   {
24       char code[6];
25       bool connect_before;
26   } connect[257] =
27           {
28   #include "connect.h"
29           };
30
31   extern   bool arabic_fonts[256];
```

```
32   extern   char *width_table[NFONT+1];
33   extern   char stretch_stage;
34   extern   int unit_width;
35
36   /*************************************************************************
37   *************************************************************************
38   **                                                                     **
39   ** this routine dumps the current internal line while reversing        **
40   ** the tokens of the specified lr direction                            **
41   ** Was enhanced in order to deal with ZERO width characters, or        **
42   ** ZERO horizontal movements (via the usage of '\z' for example).      **
43   ** It was also optimized and made more efficient.                      **
44   **                                                                     **
45   ** Johny                                                               **
46   *************************************************************************
47   *************************************************************************/
48
49   dump_line(start,end,reverse_lr)
50
51     TOKENPTR *start;
52     TOKENPTR *end;
53     bool     reverse_lr;
54
55   {
56
57     bool     start_of_line = TRUE;
58
59     TOKENPTR tmptr;
60     TOKENPTR rev_start = NULL;      /* ptr to start of reversed token line */
61     TOKENPTR rev_end   = NULL;      /* ptr to end of reversed token line   */
62     TOKENPTR tokenptr  = *start;    /* next token to dump                  */
```

```
63      TOKENPTR zerowc[MAXZWC];
64      int      total_stretch = 0;
65      int      i,j,hpos;
66
67      while(tokenptr != NULL)
68      {
69        if  (tokenptr->lr != reverse_lr)
70        {                              /* if the language is written in it's
71                                          natural direction                  */
72          if (tokenptr->next == NULL)
73            tokenptr->ending = NOT_END;
74          put_token(tokenptr,start_of_line);
75          start_of_line = FALSE;
76          tokenptr = tokenptr->next;
77        }
78        else
79        {
80          while ((tokenptr != NULL) && (tokenptr->lr == reverse_lr))
81          {
82            if (tokenptr->ending) {            /* remark word endings */
83                tokenptr->ending = NOT_END;
84                tokenptr->begining = BEGINING;
85            }
86            if (tokenptr->begining) {
87                tokenptr->ending = END;
88                tokenptr->begining = NOT_BEGIN;
89            }
90            hpos = tokenptr->horizontal_pos;
91            i = 0;
92            while ((tokenptr->next != NULL) &&
93                   (tokenptr->next->horizontal_pos == hpos))
```

```
94          {
95              zerowc[i++] = tokenptr;
96              tokenptr = tokenptr->next;
97          }
98          tmptr = tokenptr->next;
99          if (i>0)        /* a ZERO-width character was detected */
100         {
101           int temp;
102
103           zerowc[i] = tokenptr;
104           for (j=i; j>=0; j--) {
105             zerowc[j]->width = tokenptr->width;
106             push_token(zerowc[j],&rev_start,&rev_end);
107           }
108           temp = zerowc[i]->ending;
109           zerowc[i]->ending = zerowc[0]->ending;
110           zerowc[0]->ending = NOT_END;
111           zerowc[0]->begining = temp;
112         }
113         else push_token(tokenptr,&rev_start,&rev_end);
114         tokenptr = tmptr;
115       }
116
117       recalculate_horizontal(rev_start,rev_end);
118
119       if (tokenptr == NULL)           /* unmark word end at end of line */
120         rev_end->ending = NOT_END;
121
122       tmptr = rev_start;      /* dump reversed line */
123       while (tmptr != NULL)
124       {
```

```
125           put_token(tmptr,start_of_line);
126           start_of_line = FALSE;
127           tmptr = tmptr->next;
128         }
129
130         free_line(&rev_start,&rev_end);
131       }
132     }
133
134     free_line(start,end);
135
136     return;
137   }
138
139   /*************************************************************************
140   *************************************************************************
141   **                                                                     **
142   ** this routine reverses the specified internal line                   **
143   ** It also recognizes ZERO width characters and preserves their        **
144   ** order with the next letter.                                         **
145   **                                                                     **
146   ** Johny                                                               **
147   *************************************************************************
148   *************************************************************************/
149
150
151   reverse_line(start,end,paper_width)
152
153   TOKENPTR *start;
154   TOKENPTR *end;
155   int      paper_width;
```

```
156
157  {
158    int       line_length = (*end)->horizontal_pos + (*end)->width;
159    int       new_indent  = paper_width - line_length - 1;
160    int       adjustment  = new_indent - (*start)->horizontal_pos;
161    TOKENPTR  tmptr = (*start);
162    TOKENPTR  save_start;
163    TOKENPTR  tmp_start   = NULL;
164    TOKENPTR  tmp_end     = NULL;
165    bool      begining;
166    bool      ending;
167    int       total_stretch;
168    TOKENPTR  zerowc[MAXZWC];
169    int       i,j,hpos;
170
171    (*end)->ending = END;
172
173    while ((tmptr = (*start)) != NULL)
174    {
175      *start = (*start)->next;
176
177      begining = tmptr->begining;
178      ending   = tmptr->ending;
179      tmptr->begining = ending;
180      tmptr->ending   = begining;
181      hpos = tmptr->horizontal_pos;
182      i = 0;
183     /* enter the equaly hpos tokens into an array */
184      while ((tmptr->next != NULL) && (tmptr->next->horizontal_pos == hpos))
185      {
186          zerowc[i++] = tmptr;
```

```
187             tmptr = tmptr->next;
188         }
189         if (i>0) {
190             int temp;
191
192             zerowc[i] = tmptr;
193             for (j=i-1; j>=0; j--)      /* update the start pointer */
194                 *start = (*start)->next;
195             save_start = *start;
196           /* update the width for each one of them */
197             for (j=i; j>=0; j--) {
198                 zerowc[j]->width = tmptr->width;  /* take the last token width */
199                 zerowc[j]->horizontal_pos += adjustment;
200                 push_token(zerowc[j],&tmp_start,&tmp_end);
201             }
202             temp = zerowc[i]->ending;
203             zerowc[i]->ending = zerowc[0]->ending;
204             zerowc[0]->ending = NOT_END;
205             zerowc[0]->begining = temp;
206             *start = save_start;
207         }
208         else if (tmptr != NULL) {
209             tmptr->horizontal_pos += adjustment;
210             push_token(tmptr,&tmp_start,&tmp_end);
211         }
212     }
213
214     *start = tmp_start;
215     *end   = tmp_end;
216
217     recalculate_horizontal(*start,*end);
```

```
218
219     return;
220   }
221
222   /*************************************************************************
223   *************************************************************************
224   **                                                                     **
225   ** this routine recalculates the horizontal motion in the specified    **
226   ** internal line                                                       **
227   **                                                                     **
228   *************************************************************************
229   *************************************************************************/
230
231   recalculate_horizontal(start,end)
232
233     TOKENPTR  start;                /* ptr to start of line */
234     TOKENPTR  end  ;                /* ptr to end of line   */
235
236   {
237
238     TOKENPTR prev_token;
239     TOKENPTR next_token;
240     int      new_horizontal;
241     int      prev_horizontal;
242     int      start_horizontal;
243     int      total_stretch;
244
245     start_horizontal = start->horizontal_pos;
246     prev_token = start;
247     prev_horizontal = end->horizontal_pos;
248     new_horizontal  = prev_horizontal;
```

```
249
250    if(prev_token->next != NULL)
251    {
252       next_token = prev_token->next;
253       while (next_token != NULL)
254       {
255         new_horizontal = new_horizontal + (prev_token->horizontal_pos -
256                          next_token->horizontal_pos) +
257                          (prev_token->width) - (next_token->width);
258         prev_token->horizontal_pos = prev_horizontal;
259         prev_token = next_token;
260         prev_horizontal = new_horizontal;
261         next_token = prev_token->next;
262       }
263     }
264     prev_token->horizontal_pos = new_horizontal;
265
266     /* stretch Arabic words and reposition letters .. Johny */
267     total_stretch = calculate_total_stretching(start,end);
268     if (total_stretch < 0)       /* the line was shrinked!! */
269         total_stretch = 0;
270     stretch_letter(start,end,total_stretch);
271
272     return;
273  }
274
275  /*************************************************************************
276  *************************************************************************
277  **                                                                     **
278  ** This function should return the total stretching amount.            **
279  ** It does that, by calculating the total amount of extra interword    **
```

```
280  ** spaces.
281  **                                                                    **
282  ** Johny                                                              **
283  ************************************************************************
284  ************************************************************************/
285
286  int
287  calculate_total_stretching (start,end)
288
289  TOKENPTR  start;                /* ptr to start of line */
290  TOKENPTR  end  ;                /* ptr to end of line   */
291
292  {
293    TOKENPTR prev_token;
294    TOKENPTR next_token;
295    TOKENPTR tokenptr  = start;
296    int      total_stretch = 0;
297    int      space_width;
298
299    if ((tokenptr != NULL) && (tokenptr->next != NULL))
300    {
301        prev_token = tokenptr;
302        next_token = tokenptr->next;
303        while (next_token != NULL)
304        {
305          /* we have to calculate the space width for each word, as the
306             point size can change at any point */
307          space_width = (int) ((width_table[prev_token->font][0] *
308                        prev_token->point_size / unit_width) + 0.5);
309          if (prev_token->ending) {
310             total_stretch = total_stretch + (next_token->horizontal_pos -
```

```
311                             prev_token->horizontal_pos) -
312                             (prev_token->width + space_width);
313           }
314           prev_token = next_token;
315           next_token = prev_token->next;
316         }
317         tokenptr = next_token;
318     }
319
320     return(total_stretch);
321   }
322
323   /*************************************************************************
324   *************************************************************************
325   **                                                                     **
326   ** stretch the appropriate letter, based on the stretching stage.      **
327   ** stretch_stage can be one of the following:                    **
328   **    n : dont stretch at all.                                  **
329   **    f : stretch the final word only.                          **
330   **    l : stretch the last word(s). the difference from the 'f'      **
331   **        stage, is when the stretching amount is too big for one    **
332   **       word ... in that case, the remainder will be given to the   **
333   **       previous one.                                        **
334   **    a : stretch all of the Arabic words in the line, equally.      **
335   **                                                        **
336   ** Johnny.                                                           **
337   **                                                                     **
338   *************************************************************************
339   *************************************************************************/
340
341   stretch_letter (start,end,total_stretch)
```

```
342
343  TOKENPTR  start;                  /* ptr to start of line */
344  TOKENPTR  end  ;                  /* ptr to end of line   */
345  int       total_stretch;
346
347  {
348    int  filler_width;
349
350    if (total_stretch == 0)
351      return;
352    switch (stretch_stage)
353    {
354    case 'n' :    /* NO stretching is done */
355      {
356        break;
357      }
358    case 'f' :    /* stretch the LAST connectable letter in the last word */
359    case 'l' :
360      {
361        bool      finished = FALSE;
362        TOKENPTR  tokenptr = start;
363
364        while ((tokenptr != NULL) && !finished)
365        {
366          /* skip till the first arabic letter, or start of line */
367          while ((tokenptr != NULL) &&
368                 (arabic_fonts[tokenptr->font] != ARABIC))
369                        tokenptr = tokenptr->next;
370          while ((tokenptr != NULL) && !finished &&
371                 (arabic_fonts[tokenptr->font] == ARABIC))
372          {
```

```
373           int  remainder = 0;
374           char letter[4];
375
376           sprintf(letter,"%c%c%c",tokenptr->char1,tokenptr->char2,
377                                   tokenptr->char3);
378           filler_width = width2("hy",tokenptr->point_size,tokenptr->font);
379         /* check if the letter is stretchable */
380           if (connect[atoi(letter)].connect_before)
381           {
382             int  fillers_num = (int) (total_stretch / filler_width);
383             int  limit = tokenptr->point_size;
384
385             if ((stretch_stage == 'l') && (fillers_num > limit))
386             {
387                 TOKENPTR tmptr = tokenptr;  /* save it. Usefull, in case
388                                                we stretch the last (first)
389                                                word in the line         */
390                 int      tmpstretch = total_stretch;
391
392                 tokenptr->fillers_num = limit;
393                 tokenptr->filler_width = filler_width;
394                 total_stretch = (tmpstretch - (limit*filler_width));
395                 /* skip to the next word, and execute another iteration */
396                 while ((tokenptr != NULL) && !(tokenptr->ending))
397                     tokenptr = tokenptr->next;
398                 if (tokenptr == NULL)   /* this was the last word, so we
399                                            can't do anything           */
400                 {
401                     tmptr->fillers_num = fillers_num;
402                     tmptr->filler_width = filler_width;
403                     remainder = tmpstretch % filler_width;
```

```
404                     recalculate_spaces(start,end,remainder);
405                     finished = TRUE;
406                     break;
407                 }
408             }
409             else {
410                 tokenptr->fillers_num = fillers_num;
411                 tokenptr->filler_width = filler_width;
412                 remainder = total_stretch % filler_width;
413                 recalculate_spaces(start,end,remainder);
414                 finished = TRUE;
415                 break;
416             }
417           }
418           tokenptr = tokenptr->next;
419         }
420       }
421
422       break;
423     }
424   case 'a' :    /* spread the stretching among the words */
425     {
426
427       int      ncw = 0;              /* number of connectable words     */
428       int      spw = 0;              /* stretching per word             */
429       int      last_word_remainder;  /* we stretch the last word more   */
430       int      remainder = 0;
431       bool     last_word = TRUE;     /* are we stretching the last word */
432       TOKENPTR tokenptr = start;
433
434       if ((ncw = number_connect_words(start,end)) == 0)
```

```
435           break;                        /* there are NO stretchable words  */
436         spw = (int) (total_stretch / ncw);
437         while (tokenptr != NULL)
438         {
439           while ((tokenptr != NULL) &&
440                  (arabic_fonts[tokenptr->font] != ARABIC))
441                         tokenptr = tokenptr->next;
442           while ((tokenptr != NULL) &&
443                  (arabic_fonts[tokenptr->font] == ARABIC))
444           {
445             char letter[4];
446
447             sprintf(letter,"%c%c%c",tokenptr->char1,tokenptr->char2,
448                                     tokenptr->char3);
449             if (connect[atoi(letter)].connect_before)
450             {
451               filler_width = width2("hy",tokenptr->point_size,tokenptr->font);
452               if (last_word) {
453                 /* the total sum of remainders will be added to stretch
454                    the last word */
455                 last_word_remainder = (total_stretch % ncw) +
456                                       spw_remainders(start,end,spw) -
457                                       (spw % filler_width);
458                 tokenptr->fillers_num = (int) ((spw+last_word_remainder) /
459                                                filler_width);
460                 remainder = (spw + last_word_remainder) % filler_width;
461                 last_word = FALSE;
462               }
463               else tokenptr->fillers_num = (int) (spw / filler_width);
464               tokenptr->filler_width = filler_width;
465               while ((tokenptr != NULL) && !(tokenptr->ending))
```

```
466                 tokenptr = tokenptr->next;
467             }
468             if (tokenptr != NULL)
469                 tokenptr = tokenptr->next;
470           }
471         }
472         recalculate_spaces(start,end,remainder);
473 
474         break;
475       }
476     }
477 
478     return;
479 
480 }
481 
482 
483 /*************************************************************************
484 *************************************************************************
485 **                                                                     **
486 ** recalculates the inter words spaces, after stretching a specif-     **
487 ** ied letter. It simply removes the  extra spaces added by  troff     **
488 ** and spread the amount of remainder, which is the amount of poi-     **
489 ** nts less that the filler width.                                     **
490 ** initialization: hmove = 0                                           **
491 ** when reaching an end of word: hmove = hmove - extra_space           **
492 ** when reaching a letter with a filler: hmove = hmove + filler        **
493 **                                                                     **
494 ** Johny.                                                             **
495 **                                                                     **
496 *************************************************************************
```

```
497  ************************************************************************/
498
499  recalculate_spaces (start,end,remainder)
500
501  TOKENPTR  start;                  /* ptr to start of line */
502  TOKENPTR  end  ;                  /* ptr to end of line   */
503  int       remainder;
504
505  {
506    TOKENPTR prev_token;
507    TOKENPTR next_token;
508    TOKENPTR tokenptr  = start;
509    int      space_width=0, filler_width=0;
510    int      extra_space=0, hmove=0;
511    int      prev_horizontal;
512    int      nw, rpw=0;                /* remainder per word   */
513    bool     last_word = TRUE;
514
515    if ((nw = number_of_words(start,end)) == 0)
516      return;
517    rpw = (int) (remainder / nw);
518
519    if ((tokenptr != NULL) && (tokenptr->next != NULL))
520    {
521        prev_token = tokenptr;
522        next_token = tokenptr->next;
523        prev_horizontal = prev_token->horizontal_pos;
524        while (next_token != NULL)
525        {
526          space_width = (int) ((width_table[prev_token->font][0] *
527                                prev_token->point_size / unit_width) + 0.5);
```

```
528          filler_width = width2("hy",next_token->point_size,next_token->font);
529          if (prev_token->ending) {
530             extra_space = (next_token->horizontal_pos -
531                            prev_horizontal) -
532                           (prev_token->width + space_width);
533             hmove = hmove - extra_space + rpw;
534             if (last_word) {
535                hmove = hmove + (remainder % nw);
536                last_word = FALSE;
537             }
538          }
539          else if (prev_token->fillers_num > 0)
540             hmove = hmove + prev_token->fillers_num * filler_width;
541          prev_horizontal = next_token->horizontal_pos;
542          next_token->horizontal_pos += hmove;
543          prev_token = next_token;
544          next_token = next_token->next;
545          if (next_token == NULL)
546              prev_token->horizontal_pos += rpw;
547       }
548       tokenptr = next_token;
549    }
550 }
551
552 /*************************************************************************
553 *************************************************************************
554 **                                                                     **
555 ** return the number of words in the current line.                     **
556 ** Johny.                                                              **
557 **                                                                     **
558 *************************************************************************
```

```
559  ************************************************************************/
560
561  int
562  number_of_words (start,end)
563
564  TOKENPTR  start;                /* ptr to start of line */
565  TOKENPTR  end  ;                /* ptr to end of line   */
566  {
567    TOKENPTR  tokenptr = start;
568    int       nw = 0;             /* the number of Arabic words in line */
569
570    while (tokenptr != NULL) {
571      if (tokenptr->ending)
572          nw++;
573      tokenptr = tokenptr->next;
574    }
575    nw++;
576    return(nw);
577  }
578
579  /************************************************************************
580  ************************************************************************
581  **                                                                    **
582  ** return the number of connectable words in the current line.        **
583  ** a connectable word, is a word which contains a connectable         **
584  ** letter.                                                            **
585  ** Johny.                                                             **
586  **                                                                    **
587  ************************************************************************
588  ************************************************************************/
589
```

```
590  int
591  number_connect_words (start,end)
592
593  TOKENPTR  start;                  /* ptr to start of line */
594  TOKENPTR  end  ;                  /* ptr to end of line   */
595  {
596    TOKENPTR  tokenptr = start;
597    int       ncw = 0;
598
599
600    while (tokenptr != NULL) {
601      bool    connectable = FALSE;
602      char    letter[4];
603
604      sprintf(letter,"%c%c%c",tokenptr->char1,tokenptr->char2,
605                              tokenptr->char3);
606      if ((arabic_fonts[tokenptr->font]) &&
607          (connect[atoi(letter)].connect_before)) {
608          connectable = TRUE;
609          while ((tokenptr != NULL) && !(tokenptr->ending))
610              tokenptr = tokenptr->next;
611          if (tokenptr == NULL) {
612              ncw++;
613              break;
614          }
615      }
616      if ((tokenptr->ending) && (arabic_fonts[tokenptr->font]) && connectable) {
617          ncw++;
618          connectable = FALSE;
619      }
620      tokenptr = tokenptr->next;
```

```
621     }
622     return(ncw);
623   }
624
625   /*************************************************************************
626   *************************************************************************
627   **                                                                     **
628   ** returns the total remainders from all of the stretchable words.     **
629   ** a word remainder is the word_stretchable_value modulu the filler    **
630   ** width at that point.                                                **
631   **                                                                     **
632   ** Johny                                                               **
633   *************************************************************************
634   *************************************************************************/
635
636   int
637   spw_remainders (start,end,spw)
638
639   TOKENPTR  start;                  /* ptr to start of line */
640   TOKENPTR  end  ;                  /* ptr to end of line   */
641   int       spw  ;
642   {
643     TOKENPTR  tokenptr = start;
644     int       spwr = 0;             /* stretch-per-word remainders */
645     int       filler_width;
646
647     while (tokenptr != NULL)
648     {
649       bool     connectable = FALSE;
650       char     letter[4];
651       TOKENPTR prev_token;
```

```
652
653     sprintf(letter,"%c%c%c",tokenptr->char1,tokenptr->char2,
654                             tokenptr->char3);
655     if ((arabic_fonts[tokenptr->font]) &&
656         (connect[atoi(letter)].connect_before))
657     {
658         connectable = TRUE;
659         while ((tokenptr != NULL) && !(tokenptr->ending) &&
660                (arabic_fonts[tokenptr->font]))
661         {
662             prev_token = tokenptr;
663             tokenptr = tokenptr->next;
664         }
665         if (tokenptr == NULL)
666         {
667             if (connectable) {
668                 filler_width = width2("hy",prev_token->point_size,
669                                           prev_token->font);
670                 spwr += (spw % filler_width);
671             }
672             break;
673         }
674     }
675     if ((tokenptr->ending) && (arabic_fonts[tokenptr->font]) && connectable) {
676         filler_width = width2("hy",tokenptr->point_size,tokenptr->font);
677         spwr += (spw % filler_width);
678         connectable = FALSE;
679     }
680     tokenptr = tokenptr->next;
681   }
682   return(spwr);
```

```
683  }
684
685
686  /* used for debugging ... Johny */
687  print_line(start,end)
688  TOKENPTR  *start;                /* ptr to start of line */
689  TOKENPTR  *end  ;                /* ptr to end of line   */
690  {
691
692    TOKENPTR prev_token;
693    TOKENPTR next_token;
694    TOKENPTR tmptr = *start;
695
696    while (tmptr != NULL) {
697      printf("horiz:%d , vert:%d , fill_num:%d , char:%c%c%c , width:%d\n",
698              tmptr->horizontal_pos,tmptr->vertical_pos,
699              tmptr->fillers_num,tmptr->char1,
700           tmptr->char2,tmptr->char3,tmptr->width);
701      tmptr = tmptr->next;
702    }
703    printf("\n");
704  }
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3    /*******************************************************
4    *******************************************************
5    **                                                   **
6    ** this file contains a number of routines that work on **
7    ** internal tokens and token lines                   **
8    **                                                   **
9    *******************************************************
10   *******************************************************/
11
12   #include  <stdio.h>
13   #include  "TOKEN.h"
14   #include  "TABLE.h"
15   #include  "macros.h"
16   #include  "lex.h"
17
18
19
20   extern int  out_font;                /* current output font */
21   extern int  out_size;                /* current output size */
22   extern int  out_horizontal;          /* current output horizontal position */
23   extern int  out_vertical;            /* current output vertical position */
24   extern char out_font_name[];         /* current output font name */
25   extern TABLENTRY out_fontable[];     /* currnet output font table */
26   extern int  in_font;                 /* current input font */
27   extern int  in_size;                 /* current input size */
28   extern int  in_horizontal;           /* current input horizontal position */
29   extern int  in_vertical;             /* current input vertical position */
30   extern char in_font_name[];          /* current input font name */
31   extern TABLENTRY in_fontable[];      /* currnet input font table */
```

```
32  extern bool in_lr;                    /* current input font direction */
33  extern bool direction_table[];        /* direction of font i */
34
35  int i;
36
37  /*************************************************************
38  *************************************************************
39  **                                                         **
40  ** this routine allocates and initializes a new internal token **
41  **                                                         **
42  *************************************************************
43  *************************************************************/
44
45  TOKENPTR
46  new_token (toktyp,ptsize,font,horizontal,vertical,fontdir,beg_wrd,end_wrd,char1,
47             char2,char3,f_name,width)
48
49    int          toktyp;         /* token type                */
50    int          ptsize;         /* point size                */
51    int          font;           /* font number               */
52    int          horizontal;     /* horizontal position       */
53    int          vertical;       /* vertical position         */
54    bool         fontdir;        /* font direction            */
55    bool         beg_wrd;        /* begining of word indicator */
56    bool         end_wrd;        /* end of word indicator     */
57    char         char1;          /* 1st char of (abs) character token*/
58    char         char2;          /* 2nd char of (abs) character token*/
59    char         char3;          /* 3nd char of abs character token*/
60    char         *f_name;        /* font name                 */
61    int          width;          /* character width           */
62
```

```
63   {
64
65     char  *calloc();
66     TOKENPTR tokenptr = (TOKENTYPE *) calloc(1,sizeof(TOKENTYPE));
67
68     if (tokenptr != NULL)
69     {
70       tokenptr->token_type    = toktyp;
71       tokenptr->point_size    = ptsize;
72       tokenptr->font          = font;
73       tokenptr->horizontal_pos= horizontal;
74       tokenptr->vertical_pos  = vertical;
75       tokenptr->lr            = fontdir;
76       tokenptr->begining      = beg_wrd;
77       tokenptr->ending        = end_wrd;
78      tokenptr->fillers_num   = NOFILLERS;   /* initialization ... Johny */
79       tokenptr->char1         = char1;
80       tokenptr->char2         = char2;
81       tokenptr->char3         = char3;
82       strcpy(tokenptr->font_name,f_name);
83       tokenptr->width         = width;
84       tokenptr->next          = NULL;
85     }
86     else
87     {
88       out_of_memory();
89     }
90    return (tokenptr);
91   }
92
93
```

```
94  /*************************************************************************/
95  /*************************************************************************/
96
97
98  /*
99      this routine adds a token to the end of a line
100                                                        */
101
102 add_token (tokenptr,start,end)
103
104   TOKENPTR     tokenptr;          /* token to be added        */
105   TOKENPTR     *start;            /* ptr to start of line ptr */
106   TOKENPTR     *end;              /* ptr to end of line ptr   */
107
108 {
109   if (*start == NULL)
110     *start = tokenptr;
111   else {
112     (*end)->next = tokenptr;
113    tokenptr->next = NULL;        /* better to make sure it's NULL */
114   }
115   *end = tokenptr;
116   return;
117 }
118
119 /*************************************************************************/
120 /*************************************************************************/
121
122 /*
123     this routine pushes a token onto the front of a line
124                                                          */
```

```
125  push_token (tokenptr,start,end)
126
127    TOKENPTR     tokenptr;            /* token to be push         */
128    TOKENPTR     *start;              /* ptr to start of line ptr */
129    TOKENPTR     *end;                /* ptr to end of line ptr   */
130
131  {
132
133    if (*end == NULL) {
134                             /* reset the prev and next pointers ...
135                                it's not sure that we call it ONLY
136                                after new_token.  Johny  */
137     (tokenptr)->next = NULL;
138      *end = tokenptr;
139    }
140    else tokenptr->next = (*start);
141    *start = tokenptr;
142    return;
143  }
144
145  /*****************************************
146  *****************************************
147  **                                     **
148  ** this routine outputs an internal token **
149  **                                     **
150  *****************************************
151  *****************************************/
152
153  put_token (tokenptr,start_of_line)
154
155    TOKENPTR tokenptr;          /* token to output */
```

```
156    bool     start_of_line;      /* indicates first character in line */
157
158  {
159    if (start_of_line)
160    {
161      printf("H%d\n",tokenptr->horizontal_pos);
162    }
163
164    if (tokenptr->point_size != out_size)
165    {
166      printf ("s%d\n",tokenptr->point_size);
167      out_size = tokenptr->point_size;
168    }
169    if (strcmp(tokenptr->font_name,out_font_name))
170    {
171      if (strcmp(tokenptr->font_name,out_fontable[tokenptr->font].name))
172      {
173        printf("x font %d %s\n",tokenptr->font,tokenptr->font_name);
174        new_font(tokenptr->font,tokenptr->font_name,tokenptr->lr,
175                 out_fontable);
176      }
177      printf ("f%d\n",tokenptr->font);
178      out_font = tokenptr->font;
179      strcpy(out_font_name,tokenptr->font_name);
180    }
181
182    if (tokenptr->vertical_pos != out_vertical)
183    {
184      printf ("V%d\n",tokenptr->vertical_pos);
185      out_vertical = tokenptr->vertical_pos;
186    }
```

```
187
188    if (!start_of_line)
189    {
190      if (tokenptr->horizontal_pos < out_horizontal)
191      {
192        printf("H%d\n",tokenptr->horizontal_pos);
193      }
194      else if (tokenptr->horizontal_pos > out_horizontal) {
195        int j =(tokenptr->horizontal_pos - out_horizontal);
196        if (j < 100 && j > 9 && tokenptr->token_type == c_token){
197                printf( "%d%c", j, tokenptr->char1);
198                goto compress;
199            }
200        printf("h%d",(tokenptr->horizontal_pos - out_horizontal));
201      }
202    }
203
204
205    switch (tokenptr->token_type) {
206      case c_token: printf("c%c\n",tokenptr->char1); break;
207      case C_token: printf("C%c%c\n",tokenptr->char1,tokenptr->char2); break;
208      case N_token: printf("N%c%c%c\n",tokenptr->char1,tokenptr->char2,
209                    tokenptr->char3); break;
210    }
211  compress: /* thanks to Mulli Bahr hnncx compressed to nnx */
212    out_horizontal = tokenptr->horizontal_pos;
213
214    /* added to insert the fillers ... Johny */
215    if (tokenptr->fillers_num > 0) {
216     int i;
217     printf("h%d",tokenptr->width);
```

```
218     printf("Chy\n");
219     for (i=0; i<(tokenptr->fillers_num)-1; i++)
220        printf("h%dChy\n",tokenptr->filler_width);
221     printf("h%d",tokenptr->filler_width);
222     out_horizontal += tokenptr->fillers_num * tokenptr->filler_width +
223                   tokenptr->width;
224   }
225
226   if (tokenptr->ending == END)
227     printf("w");
228
229   return;
230 }
231
232
233 /*************************************************************
234 *************************************************************
235 **                                                         **
236 ** this routine outputs a new page token and its associated **
237 ** font and point size tokens and resets its font and size  **
238 ** to null and 0 so as to force output of f and s commands  **
239 **                                                         **
240 *************************************************************
241 *************************************************************/
242
243 put_page_token(page_token)
244
245   char  *page_token;          /* lex page token value */
246
247 {
248
```

```
249    extern char  yytext[];     /* lex token value */
250
251    printf("V%d\n",in_vertical);
252              /* ditroff guarantees and thus drivers
253              require that before page_token, a V0 token!
254              Mulli Bahr, Oct, 86*/
255    out_vertical = in_vertical;
256    printf("%s\n",page_token);
257
258    out_size = 0;                          /* reset out_size to zero ... */
259    out_font_name[0] = '\0';               /* reset out_font_name to empty
260           string  so that the next token print will be forced to dump out s and f
261           commands. On some devices the current size and page are not remembered
262           across page boundaries, especially if there's a bunch of "x font XX"s.
263           Besides the ditroff output always seems to have the s and f issued just
264           before the text on each page */
265
266    return;
267  }
268
269
270  /*****************************************************************/
271  /*****************************************************************/
272
273
274  /*
275      this routine deallocates a line of tokens
276                                                  */
277
278  free_line(start,end)
279
```

```
280    TOKENPTR  *start;           /* ptr to start of line */
281    TOKENPTR  *end;             /* ptr to end of line   */
282
283  {
284    TOKENPTR  tokenptr;         /* next element in line */
285
286    while( (tokenptr = *start) != NULL)
287    {
288      *start = tokenptr->next;
289      cfree(tokenptr);
290    }
291    *end = NULL;
292    return;
293  }
294
295  /*************************************************************************/
296  /*************************************************************************/
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3
4    /*****************************************************
5    *****************************************************
6    **                                                 **
7    ** this file includes the main ffortid driver routine **
8    **                                                 **
9    *****************************************************
10   *****************************************************/
11
12   #include <stdio.h>
13   #include "TOKEN.h"
14   #include "TABLE.h"
15   #include "lex.h"
16   #include "lexer"
17   #include "macros.h"
18
19   /*************************************************************************/
20   /*************************************************************************/
21
22   #define  MARK_PREVIOUS_END   in_end->ending = END
23
24   #define  ADD_CHAR1(TOKTYP,BEGN,CHAR1) tokenptr = new_token(TOKTYP,in_size,in_font,
     in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,NULL,NULL,in_font_name,width1(C
     HAR1,in_size,in_font)); add_token(tokenptr, &in_start, &in_end )
25
26   #define  ADD_CHAR2(TOKTYP,BEGN,CHAR1,CHAR2) tokenptr = new_token(TOKTYP,in_size,in
     _font,in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,CHAR2,NULL,in_font_name,w
     idth2(yytext+1,in_size,in_font)); add_token(tokenptr, &in_start, &in_end )
27
```

```
28  /*****************************************************************************/
29  #define  ADD_CHARN(TOKTYP,BEGN,CHAR1,CHAR2,CHAR3) tokenptr = new_token(TOKTYP,in_s
    ize,in_font,in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,CHAR2,CHAR3,in_font
    _name,widthn(yytext+1)); add_token(tokenptr, &in_start, &in_end )
30
31  /*****************************************************************************/
32  /* I think this is better ... Johny                                          */
33  #define  USAGE  "Usage: %s [-wpaperwidth] [-rfont_position_list] [-afont_position_
    list] [-s[nfla]]\n"
34  /*****************************************************************************/
35
36    int   in_font=1;                  /* current input font                    */
37    int   in_size=0;                  /* current input size                    */
38    int   in_horizontal=0;            /* current input horizontal position     */
39    int   in_vertical=0;              /* current input vertical position       */
40    char  in_font_name[3];            /* current input font name               */
41    bool  in_lr=LEFT_TO_RIGHT;        /* current input font direction          */
42    TABLENTRY in_fontable[256];       /* current input font table              */
43
44    int   out_font=1;                 /* current output font                   */
45    int   out_size=0;                 /* current output size                   */
46    int   out_horizontal=0;           /* current output horizontal position    */
47    int   out_vertical=0;             /* current output vertical position      */
48    char  out_font_name[3];           /* current output font name              */
49    TABLENTRY out_fontable[256];      /* current output font table             */
50
51    bool  direction_table[256];       /* formatting direction of font i        */
52    bool   arabic_fonts[256];          /* whether font i is arabic or not       */
53    char  stretch_stage='a';          /* the stretching stage. default is NOT
54                                      to stretch.                               */
55    char  *device;                    /* output device                         */
```

```
56      char  c;                              /* for flushing included postscript
57                                            and psfig text    */
58      static char copyright[]="(c) Copyright 1987 Berry Computer Scientists, Ltd.";
59
60    /*************************************************************************/
61    /*************************************************************************/
62
63    main (argc,argv)
64      int           argc;
65      char          *argv[];
66
67    {
68
69      extern char  yytext[];    /* lex token string                          */
70
71                                /* indicates the predominate formatting direction */
72      bool          lr_predom = LEFT_TO_RIGHT;
73      TOKENPTR      in_start  = NULL;       /* ptr to start of internal input line */
74      TOKENPTR      in_end    = NULL;       /* ptr to end of internal input line   */
75
76      double        paper_inch=8.5;         /* paper width in inches         */
77      int           paper_width;            /* paper width in points         */
78      int           token_num;              /* current lex token                 */
79      bool          new_word=TRUE;          /* indicates word token was encountered*/
80      bool          previous_D=FALSE;       /* indicates a D_token was just done   */
81      char          small_motion[3];        /* motion from hc_token              */
82      char          f_name[3];              /* new font name                     */
83      int           f_num;                  /* new font number                   */
84      TOKENPTR      tokenptr;               /* ptr to new internal tokens        */
85      TOKENPTR      new_token();            /* ptr to new internal tokens        */
86      char          *calloc();
```

```
87      double        atof();
88      int           i,j,k;                  /* counter index  */
89      char          *tmpyy;                 /* temp ptr for yytext */
90
91    /*************************************************************************/
92    /*************************************************************************/
93
94      for (i=0;i<=255;i++) {      /* johny */
95        SET_DIRECTION(i,LEFT_TO_RIGHT);
96       RESET_AR_FONT(i);
97      }
98
99      if (--argc > 0)
100     {
101       if (argv[1][0] == '-')
102       {
103         i=1;
104         while(i<= argc)
105         {
106           switch (argv[i][1])
107
108           {
109           /* specify the fonts to be reversed */
110             case 'r':
111             case 'R':
112               {
113                 if (strlen(argv[i]) > 2)
114                   SET_DIRECTION(atoi(argv[i++]+2),RIGHT_TO_LEFT);
115                 else
116                   i++;
117                 while( (i<=argc) && (argv[i][0] != '-') )
```

```
118                 SET_DIRECTION(atoi(argv[i++]),RIGHT_TO_LEFT);
119               break;
120             }
121           /* specify the Arabic Fonts. Only Arabic Fonts are stretched */
122             case 'a':
123             case 'A':
124               {
125                 if (strlen(argv[i]) > 2)
126                   SET_AR_FONT(atoi(argv[i++]+2));
127                 else
128                   i++;
129                 while( (i<=argc) && (argv[i][0] != '-') )
130                   SET_AR_FONT(atoi(argv[i++]));
131                 break;
132               }
133           /* specify paper width */
134             case 'w':
135             case 'W':
136               {
137                 if (strlen(argv[i]) > 2)
138                   paper_inch = atof(argv[i++]+2);
139                 else
140                 {
141                   i++;
142                   paper_inch = atof(argv[i++]);
143                 }
144                 break;
145               }
146           /* specify the stretching style.
147             n: dont stretch (default0
148             f: stretch final Arabic word.
```

```
149           l: stretch last Arabic word(s).
150           a: stretch all the Arabic words in the line, equally */
151           case 's':
152           case 'S':
153          {
154            stretch_stage = *(argv[i++]+2);
155            if (stretch_stage == '\0')
156              stretch_stage = 'a';
157            if ( (stretch_stage != 'n') && (stretch_stage != 'f') &&
158                 (stretch_stage != 'l') && (stretch_stage != 'a') )
159            {
160             fprintf(stderr,"%s: incorrect stretch_stage\n",argv[0]);
161             exit(1);
162             }
163               break;
164          }
165           default:
166             {
167               fprintf(stderr,"FFORTID: illegal argument: %s\n",argv[i]);
168               exit(1);
169               break;
170             }
171        }
172      }
173    }
174    else
175    {
176      printf(USAGE,argv[0]);
177      exit(1);
178    }
179  }
```

```
180
181  /*************************************************************************/
182  /*************************************************************************/
183
184    while ( (token_num=yylex()) ) /* get each token until the end-of-file */
185    {
186
187
188      switch (token_num)                /* process based on token type */
189      {
190
191        case s_token:
192             {
193               in_size = atoi(yytext+1);
194               break;
195             }
196
197        case f_token:
198             {
199               in_font = atoi(yytext+1);
200               in_lr   = in_fontable[in_font].direction;
201               strcpy(in_font_name,in_fontable[in_font].name);
202               break;
203             }
204        case c_token:
205             {
206               if (new_word)
207               {
208                 ADD_CHAR1(c_token,BEGINING,*(yytext+1));
209                 new_word =FALSE;
210               }
```

```
211             else
212             {
213               ADD_CHAR1(c_token,NOT_BEGIN,*(yytext+1));
214             }
215             break;
216           }
217         case C_token:
218           {
219             if (new_word)
220             {
221               ADD_CHAR2(C_token,BEGINING,*(yytext+1),*(yytext+2));
222               new_word = FALSE;
223             }
224             else
225             {
226               ADD_CHAR2(C_token,NOT_BEGIN,*(yytext+1),*(yytext+2));
227             }
228             break;
229           }
230         case N_token:
231           {
232             if (new_word)
233             {
234               ADD_CHARN(N_token,BEGINING,*(yytext+1),*(yytext+2),*(yytext+3));
235               new_word = FALSE;
236             }
237             else
238             {
239               ADD_CHARN(N_token,NOT_BEGIN,*(yytext+1),*(yytext+2),*(yytext+3));
240             }
241             break;
```

```
242             }
243         case H_token:
244             {
245               in_horizontal = atoi(yytext+1);
246               break;
247             }
248         case V_token:
249             {
250               in_vertical = atoi(yytext+1);
251               break;
252             }
253         case h_token:
254             {
255               in_horizontal = in_horizontal + atoi(yytext+1);
256               break;
257             }
258         case v_token:
259             {
260               in_vertical = in_vertical + atoi(yytext+1);
261               break;
262             }
263         case hc_token:
264             {
265               small_motion[0] = *(yytext);
266               small_motion[1] = *(yytext+1);
267               small_motion[2] = '\0';
268               in_horizontal = in_horizontal + atoi(small_motion);
269               if (new_word)
270               {
271                 ADD_CHAR1(c_token,BEGINING,*(yytext+2));
272                 new_word = FALSE;
```

```
273             }
274             else
275             {
276               ADD_CHAR1(c_token,NOT_BEGIN,*(yytext+2));
277             }
278             break;
279           }
280       case n_token:
281           {
282             if (!previous_D)
283             {
284               if(in_end != NULL)
285               {
286                 MARK_PREVIOUS_END;
287                 if (lr_predom)
288                   dump_line(&in_start,&in_end,RIGHT_TO_LEFT);
289                 else
290                 {
291                   reverse_line(&in_start,&in_end,paper_width);
292                   dump_line(&in_start,&in_end,LEFT_TO_RIGHT);
293                 }
294               }
295             }
296             else
297               previous_D = FALSE;
298             DUMP_LEX(yytext);
299             break;
300           }
301       case w_token:
302           {
303             if(in_end != NULL)
```

```
304                 MARK_PREVIOUS_END;
305               new_word = TRUE;
306               break;
307             }
308         case p_token:
309             {
310               put_page_token(yytext);
311               break;
312             }
313         case trail_token:
314         case pause_token:
315         case height_token:
316         case slant_token:
317         case include_token:
318         case control_token:
319             {
320               DUMP_LEX(yytext);
321               break;
322             }
323         case res_token:
324             {
325               DUMP_LEX(yytext);
326               for(tmpyy=yytext;(*tmpyy<'0')||(*tmpyy>'9');tmpyy++);
327               paper_width = paper_inch * atoi(tmpyy);
328               break;
329             }
330         case init_token:
331             {
332               DUMP_LEX(yytext);
333               width_init();
334               break;
```

```
335             }
336         case stop_token:
337             {
338               printf("V%d\n",in_vertical);
339               DUMP_LEX(yytext);
340               break;
341             }
342         case newline_token:
343             {
344               break;
345             }
346         case dev_token:
347             {
348               device = calloc(1,strlen(yytext)-3);
349               strcpy(device,yytext+4);
350               DUMP_LEX(yytext);
351               break;
352             }
353         case font_token:
354             {
355               font_info(yytext,&f_num,f_name);
356               new_font(f_num,f_name,FONT_DIRECTION(f_num),in_fontable);
357               loadfont(f_num,f_name,NULL);
358               strcpy (out_fontable[f_num].name,in_fontable[f_num].name);
359               out_fontable[f_num].direction = in_fontable[f_num].direction;
360               DUMP_LEX(yytext);
361               break;
362             }
363         case PR_token:
364             {
365               lr_predom = RIGHT_TO_LEFT;
```

```
366             break;
367           }
368       case PL_token:
369           {
370             lr_predom = LEFT_TO_RIGHT;
371             break;
372           }
373       case postscript_begin_token:
374           {
375             DUMP_LEX(yytext);
376                /* copy everything up to and including 2 lines */
377                /* containing PE\ and .\ */
378                while ( ( c = getchar() ) != EOF ) {
379          putchar(c);
380          if ( c == 'P' ) {
381             if ( ( c = getchar() ) != EOF ) {
382                putchar(c);
383                if (c != 'E') goto not_end_PS_yet;
384             }
385             if ( ( c = getchar() ) != EOF ) {
386                putchar(c);
387                if (c != '\\') goto not_end_PS_yet;
388             }
389             if ( ( c = getchar() ) != EOF ) {
390                putchar(c);
391                if (c != '\n') goto not_end_PS_yet;
392             }
393             if ( ( c = getchar() ) != EOF ) {
394                putchar(c);
395                if (c != '.') goto not_end_PS_yet;
396             }
```

```
397             if ( ( c = getchar() ) != EOF ) {
398                putchar(c);
399                if (c != '\\') goto not_end_PS_yet;
400             }
401             if ( ( c = getchar() ) != EOF ) {
402                putchar(c);
403                if (c != '\n') goto not_end_PS_yet;
404             }
405             goto done_flushing_PS;
406          }
407       not_end_PS_yet:;
408                }
409              goto done;
410         done_flushing_PS:
411              break;
412           }
413       case psfig_begin_token:
414           {
415             printf("H%d\n",in_horizontal);
416             printf("V%d\n",in_vertical);
417             DUMP_LEX(yytext);
418                /* copy everything up to and including line */
419                /* containing "x X pendFig"*/
420                while ( ( c = getchar() ) != EOF ) {
421          putchar(c);
422          if ( c == 'x' ) {
423             if ( ( c = getchar() ) != EOF ) {
424                putchar(c);
425                if (c != ' ') goto not_end_psfig_yet;
426             }
427             if ( ( c = getchar() ) != EOF ) {
```

```
428             putchar(c);
429             if (c != 'X') goto not_end_psfig_yet;
430          }
431          if ( ( c = getchar() ) != EOF ) {
432             putchar(c);
433             if (c != ' ') goto not_end_psfig_yet;
434          }
435          if ( ( c = getchar() ) != EOF ) {
436             putchar(c);
437             if (c != 'p') goto not_end_psfig_yet;
438          }
439          if ( ( c = getchar() ) != EOF ) {
440             putchar(c);
441             if (c != 'e') goto not_end_psfig_yet;
442          }
443          if ( ( c = getchar() ) != EOF ) {
444             putchar(c);
445             if (c != 'n') goto not_end_psfig_yet;
446          }
447          if ( ( c = getchar() ) != EOF ) {
448             putchar(c);
449             if (c != 'd') goto not_end_psfig_yet;
450          }
451          if ( ( c = getchar() ) != EOF ) {
452             putchar(c);
453             if (c != 'F') goto not_end_psfig_yet;
454          }
455          if ( ( c = getchar() ) != EOF ) {
456             putchar(c);
457             if (c != 'i') goto not_end_psfig_yet;
458          }
```

```
459             if ( ( c = getchar() ) != EOF ) {
460                putchar(c);
461                if (c != 'g') goto not_end_psfig_yet;
462             }
463             if ( ( c = getchar() ) != EOF ) {
464                putchar(c);
465                if (c != '\n') goto not_end_psfig_yet;
466             }
467             goto done_flushing_psfig;
468          }
469       not_end_psfig_yet:;
470                }
471              goto done;
472         done_flushing_psfig:
473              break;
474            }
475       case D_token:
476            {
477              printf("H%d\n",in_horizontal);
478              printf("V%d\n",in_vertical);
479              printf("%s\n",yytext);
480              tmpyy = yytext+1;
481              if (*tmpyy == 'l' || *tmpyy == '-')
482              {
483                k = 0;
484                while( ((*tmpyy < '0') || (*tmpyy > '9')) && (*tmpyy != '-') )
485                  tmpyy++;
486                while (*tmpyy != '\0')
487                {
488                  if(k=1-k)
489                    in_horizontal += atoi(tmpyy);
```

```
490                 else
491                   in_vertical   += atoi(tmpyy);
492                 while(((*tmpyy >= '0') && (*tmpyy <= '9')) || (*tmpyy == '-') )
493                   tmpyy++;
494                 while( ((*tmpyy < '0') || (*tmpyy > '9')) && (*tmpyy != '-')
495                       && (*tmpyy != '\0') )
496                   tmpyy++;
497               }
498             }
499             out_horizontal = in_horizontal;
500             out_vertical = in_vertical;
501             previous_D = TRUE;
502           }
503     } /* end switch */
504   }  /* end while */
505   done:;
506 } /* end main() */
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3
4    /*****************************************************
5    *****************************************************
6    **                                                 **
7    ** this file contains a number of supporting routines **
8    **                                                 **
9    *****************************************************
10   *****************************************************/
11
12   #include "TOKEN.h"
13   #include "TABLE.h"
14   #include "macros.h"
15
16
17   /*********************************************************************/
18   /*********************************************************************/
19
20
21   /***********************************************
22   ***********************************************
23   **                                           **
24   ** this routine adds a new font to the font table **
25   **                                           **
26   ***********************************************
27   ***********************************************/
28
29   new_font(font_number,font_name,font_direction,font_table)
30
31     int       font_number;
```

```
32      char      *font_name;
33      bool      font_direction;
34      TABLENTRY font_table[];
35
36    {
37      strcpy(font_table[font_number].name,font_name);
38      font_table[font_number].direction = font_direction;
39
40      return;
41    }
42
43
44    /*********************************************************
45    *********************************************************
46    **                                                     **
47    ** this routine determines the width of a character token **
48    **                                                     **
49    *********************************************************
50    *********************************************************/
51
52    width(tokenptr)
53
54      TOKENPTR  tokenptr;
55
56    {
57      return(0);
58    }
59
60    /*********************************************************
61    *********************************************************
62    **                                                     **
```

```
63   ** this routine determines the width of a character token **
64   **                                                        **
65   ************************************************************
66   ************************************************************/
67
68   font_info(font_line,font_number,font_name)
69
70     char  *font_line;                /* LEX input token line */
71     int   *font_number;              /* new font number      */
72     char  *font_name[];              /* new font name        */
73
74   {
75
76     int   i = 0;
77     int   j = 0;
78     char  f_num[4];
79
80     while ((*(font_line+i) < '0') || (*(font_line+i) > '9'))
81       i++;
82
83     while ((*(font_line+i) >= '0') && (*(font_line+i) <= '9'))
84     {
85       f_num[j] = *(font_line+i);
86       i++;
87       j++;
88     }
89     f_num[j] = '\0';
90
91     while (*(font_line+i) == ' ')
92       i++;
93
```

```
94      strcpy(font_name,font_line+i);
95      *font_number = atoi(f_num);
96
97      return;
98
99    }
100
101   /*************************************************
102   *************************************************
103   **                                             **
104   ** this routine prints an error message and    **
105   ** halts execution upon OUT OF MEMORY condition **
106   **                                             **
107   *************************************************
108   *************************************************/
109
110   out_of_memory()
111   {
112
113     printf("\nFATAL ERROR: out of memory\n");
114     exit(1);
115
116   }
117
118
119   /*****************************************************************************/
120   /*****************************************************************************/
121
122
123   yywrap()
124   {
```

```
125     return 1;
126   }
127
128   /*************************************************************************/
129   /*************************************************************************/
```

```
1    /* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
2    /* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
3
4    #include <stdio.h>
5
6    typedef struct {
7       int   space_width;
8       int   no_width_entries;
9       char  is_special_font;
10      char  font_name[10];
11   } Fontinfo;
12
13   #define  MAXNOFONTS  255
14   #define  MAXWIDENTRIES  256
15   #define  NOCHARSINBIGGESTFONT MAXWIDENTRIES-1   /* for no biggestfont in DESC */
16   #define  MAXNOCHARS  512   /* characters with two-letter or --- names */
17   #define  SIZECHARINDXTABLE (MAXNOCHARS + 128-32)   /* includes ascii chars,
18              but not non-graphics */
19
20   Fontinfo basic_font_info[MAXNOFONTS+1];
21   char  font_name[MAXNOFONTS][10];
22
23   int   no_of_fonts;
24   int   indx_1st_spec_font;  /* index of first special font */
25   int   size_char_table;
26   int   unit_width;
27   int   units_per_inch;
28   int   no_chars_in_biggest_font = NOCHARSINBIGGESTFONT;
29
30   int   size_char_name;
31   char  char_name[5*MAXNOCHARS];   /* 2 or 3 letter character names,
```

```
32                  including \0 for each */
33      short char_table[MAXNOCHARS+1];  /* index of characters in char_name */
34
35      char  *char_indx_table[MAXNOFONTS+1]; /*fitab*/
36      char  *code_table[MAXNOFONTS+1]; /*codetab*/
37      char  *width_table[MAXNOFONTS+1];   /* widtab would be a better name */
38
39      #define  FATAL 1
40      #define  BYTEMASK 0377
41
42      /* char  *fontdir = "/usr/lib/font"; */
43      char  *fontdir = "/usr/dberry/usr/lib/font";
44
45      extern   int   in_size; /* current input point size */
46      extern   int   in_font; /* current input font */
47      extern   char  *device; /* output device */
48
49      /*************************************************************
50      *************************************************************
51      **                                                         **
52      ** this routine initializes the device and font width tables **
53      **                                                         **
54      *************************************************************
55      *************************************************************/
56
57      width_init()   /* read in font and code files, etc. */
58      {
59         int i, j;
60         char *malloc();
61         char temp[60];
62         FILE *descfile;
```

```
63      char word[100], *ptr;
64
65      sprintf(temp, "%s/dev%s/DESC", fontdir, device);
66      if ((descfile = fopen(temp, "r")) == NULL){
67         error(FATAL, "can't open DESC for %s\n", temp);
68      }
69      while (fscanf(descfile, "%s", word) != EOF) {
70         if (strcmp(word, "res") == 0) {
71            fscanf(descfile, "%d", &units_per_inch);
72         } else if (strcmp(word, "unitwidth") == 0) {
73            fscanf(descfile, "%d", &unit_width);
74         } else if (strcmp(word, "fonts") == 0) {
75            fscanf(descfile, "%d", &no_of_fonts);
76            if (no_of_fonts > MAXNOFONTS) {
77               error(FATAL,
78               "have more fonts, %d, than the maximum allowed, %d\n",
79                no_of_fonts, MAXNOFONTS);
80            }
81            /* start at 1 to leave 0 for default font */
82            for (i = 1; i <= no_of_fonts; i++) {
83               fscanf(descfile, "%s", font_name[i]);
84            }
85         } else if (strcmp(word, "biggestfont") == 0) {
86            fscanf(descfile, "%d", &no_chars_in_biggest_font);
87            if (no_chars_in_biggest_font > NOCHARSINBIGGESTFONT) {
88               error(FATAL,
89                  "biggestfont of DESC too big, %d\n",
90                  no_chars_in_biggest_font);
91            }
92         } else if (strcmp(word, "charset") == 0) {
93            ptr = char_name;
```

```
94            size_char_table = 0;
95            while (fscanf(descfile, "%s", ptr) != EOF) {
96               if (size_char_table == MAXNOCHARS-1) {
97                  error(FATAL,
98                  "have more chars than the maximum allowed, %d\n",
99                  MAXNOCHARS);
100              }
101              char_table[size_char_table++] = ptr - char_name;
102              while (*ptr++) /* skip to end of char name */
103                 ;
104           }
105           size_char_name = ptr - char_name;
106           char_table[size_char_table++] = 0; /* end with \0 */
107        } else
108        /* skip anything else */
109        while (getc(descfile) != '\n');
110     }
111     fclose(descfile);
112
113     width_table[0] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
114     code_table[0] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
115     for (j = 0; j <= MAXWIDENTRIES-1; j++) {
116        width_table[0][j] = 0;
117        code_table[0][j] = 0;
118     }
119     char_indx_table[0] =(char *) malloc(SIZECHARINDXTABLE * sizeof(char));
120     for (j = 0; j <= SIZECHARINDXTABLE-1; j++) {
121        char_indx_table[0][j] = 0;
122     }
123     basic_font_info[0].space_width = 0;
124     basic_font_info[0].no_width_entries = no_chars_in_biggest_font+1;
```

```
125     basic_font_info[0].is_special_font = 0;
126     basic_font_info[0].font_name[0] = '\0';
127
128
129  /* deviceprint(); /* debugging print of device */
130  /* fontprint(0); /*debugging print of font tables [0]*/
131
132     for (i = 1; i <= no_of_fonts; i++) {
133        width_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
134        code_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
135        for (j = 0; j <= MAXWIDENTRIES-1; j++) {
136           width_table[i][j] = 0;
137           code_table[i][j] = 0;
138        }
139        char_indx_table[i] =(char *) malloc(SIZECHARINDXTABLE *
140                 sizeof(char));
141        for (j = 0; j <= SIZECHARINDXTABLE-1; j++) {
142           char_indx_table[i][j] = 0;
143        }
144        getfontinfo(font_name[i],i);
145  /*    fontprint(i); /*debugging print of font tables [i]*/
146     }
147  }
148
149  getfontinfo(font_name,pos)
150  char *font_name;
151  int pos;
152  {
153     FILE *fontfile;
154     int i, no_width_entries, space_width;
155     char buffer[100], word[30],
```

```
156         char_c[10], wid[10], asc_des[10], code[10];
157   /* Fontinfo *ftemp = &basic_font_info[pos]; */
158
159      sprintf(buffer, "%s/dev%s/%s", fontdir, device, font_name);
160      if ((fontfile = fopen(buffer, "r")) == NULL){
161         error(FATAL, "can't open width table for %s\n", font_name);
162      }
163      while (fscanf(fontfile, "%s", word) != EOF) {
164         if (word[0] == '#')
165            while(getc(fontfile) != '\n');
166         else if (strcmp(word, "name") == 0)
167            fscanf(fontfile, "%s", basic_font_info[pos].font_name);
168         else if (strcmp(word, "special") == 0)
169            basic_font_info[pos].is_special_font = 1;
170         else if (strcmp(word, "spacewidth") == 0) {
171            fscanf(fontfile, "%d",&space_width);
172   /*       &((&basic_font_info[pos])->space_width) = space_width;*/
173            basic_font_info[pos].space_width = space_width;
174            if (space_width == 0) {
175               width_table[pos][0] = units_per_inch * unit_width / 72 / 3;
176            } else {
177               width_table[pos][0] = space_width;
178            }
179         } else if (strcmp(word, "charset") == 0) {
180            while(getc(fontfile) != '\n');
181            no_width_entries = 0;
182            /* widths are origin 1 so char_indx_table entry of
183            0 can mean not there */
184            while (fgets(buffer, 100, fontfile) != NULL) {
185               sscanf(buffer, "%s %s %s %s",
186                  char_c, wid, asc_des, code);
```

```
187             if (wid[0] != '"') { /* not a ditto */
188                no_width_entries++;
189                width_table[pos][no_width_entries] =
190                   atoi(wid);
191  /*
192                sscanf(wid, "%d",
193                   &width_table[pos][no_width_entries]);
194  */
195                if (code[0] == '0')
196                   sscanf(code, "%o", &i);
197                else
198                   sscanf(code, "%d", &i);
199                code_table[pos][no_width_entries] = i;
200             }
201             /* otherwise a synonym for previous character,
202             so leave previous values intact */
203             if (strlen(char_c) == 1)   /* it's ascii */
204                char_indx_table[pos][char_c[0]-32] =
205                   no_width_entries;
206                   /* char_indx_table origin
207                   omits non-graphics */
208             else if (strcmp(char_c, "---") != 0) {
209                   /* it has a 2-char name */
210                for (i = 0; i <= size_char_table; i++)
211                   if (strcmp(
212                       &char_name[char_table[i]],
213                       char_c) == 0) {
214                      char_indx_table[pos][i+128-32] =
215                         no_width_entries;
216                         /* starts after the ascii */
217                      break;
```

```
218                  }
219               if (i >= size_char_table)
220                  error(FATAL,
221                      "font %s: %s not in charset\n",
222                      font_name, char_c);
223            }
224         }
225         if (no_width_entries > MAXWIDENTRIES) {
226            error(FATAL,
227               "font has %d characters, too big\n",
228               no_width_entries);
229         }
230         basic_font_info[pos].no_width_entries =
231            no_chars_in_biggest_font+1;
232       }
233         else while(getc(fontfile) != '\n');
234    }
235    fclose(fontfile);
236 }
237
238 /*******************************************************
239 *******************************************************
240 **                                                   **
241 ** this routine prints the specified font width table **
242 **                                                   **
243 *******************************************************
244 *******************************************************/
245
246 fontprint(i)   /* debugging print of font i (0,...) */
247 {
248    int jj, kk, nn;
```

```
249
250     printf("font %d:\n", i);
251     nn = basic_font_info[i].no_width_entries;
252
253     printf("base=0xxxxxx, nchars=%d, spec=%d, name=%s, width_table=0xxxxxx, char_in
    dx_table=0xxxxxx, code_table=0xxxxxx\n",
254        nn, basic_font_info[i].is_special_font, basic_font_info[i].font_name);
255
256     printf("\nwidths:\n");
257     for (jj=0; jj <= nn; jj++) {
258        printf(" %2d", width_table[i][jj] & BYTEMASK);
259        if (jj % 20 == 19) printf("\n");
260     }
261
262     printf("\nchar_indx_table:\n");
263     for (jj=0; jj < size_char_table + 128-32; jj++) {
264        printf(" %2d", char_indx_table[i][jj] & BYTEMASK);
265        if (jj % 20 == 19) printf("\n");
266     }
267
268     printf("\ncode_table:\n");
269     for (jj=0; jj <= nn; jj++) {
270        printf(" %2d", code_table[i][jj] & BYTEMASK);
271        if (jj % 20 == 19) printf("\n");
272     }
273
274     printf("\n");
275  }
276
277  /*****************************************************
278  *****************************************************
```

```
279  **                                                  **
280  ** this routine prints the device table **
281  **                                                  **
282  ******************************************************
283  ******************************************************/
284
285  deviceprint()  /* debugging print of device */
286  {
287     int j;
288     int jj;
289
290     printf("device:\n");
291
292     printf("res=%d nfonts=%d nchtab=%d unitwidth=%d lchname=%d",
293        units_per_inch, no_of_fonts, size_char_table, unit_width,
294        size_char_name);
295
296     printf("\nchtab:\n");
297     for (jj=0; jj <= size_char_table-1; jj++) {
298        printf(" %2d", char_table[jj]);
299        if (jj % 20 == 19) printf("\n");
300     }
301
302     printf("\nchname:\n");
303     for (jj=0; jj <= size_char_table-1; jj++) {
304        printf(" %s", &char_name[char_table[jj]]);
305        if (jj % 20 == 19) printf("\n");
306     }
307
308     printf("\n");
309  }
```

```
310
311  /*****************************************************
312  *****************************************************
313  **                                                 **
314  ** this routine loads the specified font width table **
315  **                                                 **
316  *****************************************************
317  *****************************************************/
318
319
320  loadfont(n, s, s1)   /* load font info for font s on position n (0...) */
321  int n;
322  char *s, *s1;
323  {
324  /*
325     char temp[60];
326     int fin, nw, norig;
327
328
329     if (n < 0 || n > MAXNOFONTS)
330        error(FATAL, "illegal fp command %d %s", n, s);
331     if (strcmp(s, fontbase[n]->namefont) == 0)
332        return;
333     if (s1 == NULL || s1[0] == '\0')
334        sprintf(temp, "%s/dev%s/%s.out", fontdir, device, s);
335     else
336        sprintf(temp, "%s/%s.out", s1, s);
337     if ((fin = open(temp, 0)) < 0)
338        error(FATAL, "can't open font table %s", temp);
339     norig = fontbase[n]->nwfont & BYTEMASK;
340     read(fin, fontbase[n], 3 * norig + size_char_table+128-32 + sizeof(struct Font)
     );
```

```
341     if ((fontbase[n]->nwfont & BYTEMASK) > norig)
342        error(FATAL, "Font %s too big for position %d\n", s, n);
343     close(fin);
344     nw = fontbase[n]->nwfont & BYTEMASK;
345     o_width_table[n] = (char *) fontbase[n] + sizeof(struct Font);
346     o_char_indx_table[n] = (char *) o_width_table[n] + 3 * nw;
347     fontbase[n]->nwfont = norig;
348  */
349  }
350
351
352  error(f, s, a1, a2, a3, a4, a5, a6, a7) {
353     fprintf(stderr, "ffortid: ");
354     fprintf(stderr, s, a1, a2, a3, a4, a5, a6, a7);
355     fprintf(stderr, "\n");
356     if (f)
357           exit(1);
358  }
359
360  /*********************************************************************
361  *********************************************************************
362  **                                                                 **
363  ** this routine determines the width of the specified funny character **
364  **                                                                 **
365  *********************************************************************
366  *********************************************************************/
367
368
369  width2(s,in_size,in_font)  /* s is a funny char name */
370  char *s;
371  int  in_size;     /* Johny */
```

```
372  int    in_font;
373  {
374     int i;
375
376     for (i = 0; i < size_char_table; i++)
377  /*    if (strcmp(&o_char_name[o_char_table[i]], s) == 0) */
378        if (strcmp(&char_name[char_table[i]], s) == 0)
379           break;
380     if (i < size_char_table)
381           return(width1(i + 128,in_size,in_font));
382         else
383           return(width1(0,in_size,in_font));
384  }
385
386  /*************************************************************
387  *************************************************************
388  **                                                         **
389  ** this routine determines the width of the specified character **
390  **                                                         **
391  ** in_size is passed as a parameter as this procedure is used   **
392  ** to calculate the filler width for each word of the line.    **
393  ** it is necessary because the in_size at the start and end of **
394  ** the line may be different. Johny                  **
395  **                                          **
396  *************************************************************
397  *************************************************************/
398
399
400  width1(c,in_size,in_font)  /* output char c */
401  int c;
402  int in_size;
```

```
403  int in_font;
404  {
405     char *pw;
406     register char *p;
407     register int i, k;
408     int j, w, width;
409
410     c -= 32;
411     if (c <= 0) {
412  /*    return(o_width_table[in_font][0] ); */
413        return(width_table[in_font][0] );
414     }
415     k = in_font;
416  /* i = o_char_indx_table[in_font][c] & BYTEMASK; */
417     i = char_indx_table[in_font][c] & BYTEMASK;
418     if (i != 0) {  /* it's on this font */
419  /*    pw = o_width_table[in_font]; */
420        pw = width_table[in_font];
421     } else if (indx_1st_spec_font > 0) { /* on special (we hope) */
422        for (k=indx_1st_spec_font, j=0;
423             j < no_of_fonts; j++, k = k % no_of_fonts + 1)
424  /*       if ((i = o_char_indx_table[k][c] & BYTEMASK) != 0) { */
425           if ((i = char_indx_table[k][c] & BYTEMASK) != 0) {
426  /*          pw = o_width_table[k]; */
427              pw = width_table[k];
428              break;
429           }
430     }
431     if (i == 0 || j == no_of_fonts) {
432  /*    return(o_width_table[in_font][0] ); */
433        return(width_table[in_font][0] );
```

```
434     }
435     width = pw[i] & BYTEMASK;
436     width = (int) ((width * in_size / unit_width) + 0.5) ;
437          return(width);
438  }
439
440  /*************************************************************
441  *************************************************************
442  **                                                         **
443  ** this routine determines the width of the character whose **
444  ** code is n, i.e. specified Nn                            **
445  **                                                         **
446  *************************************************************
447  *************************************************************/
448
449
450  widthn(pn)  /* output char with abs code *pn */
451  char *pn;
452  {
453     char *pw;
454     register int i;
455     int n,width;
456
457     sscanf(pn, "%d", &n); /*get the string *pn and convert to integer*/
458
459     i= abscw (n);
460  /* pw=o_width_table[in_font]; */
461     pw=width_table[in_font];
462     width = pw[i] & BYTEMASK;
463     width = (int) ((width * in_size / unit_width) + 0.5) ;
464      return(width);
```

```
465  }
466
467  abscw(n) /*index of abs char n in o_width_table[], etc. */
468  int n;
469  {
470     register int i, ncf;
471
472  /* ncf= fontbase[in_font]->nwfont & BYTEMASK; */
473     ncf= basic_font_info[in_font].no_width_entries & BYTEMASK;
474     for (i=0; i< ncf; i++)
475  /*    if ((unsigned char)o_code_table[in_font][i] == n) */
476        if ((unsigned char)code_table[in_font][i] == n)
477        /* a bug fix for the \N'xxx' to work, when xxx > 128 */
478           return i;
479     return 0;
480  }
```