log_ctc41

Generated by Doxygen 1.8.17

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

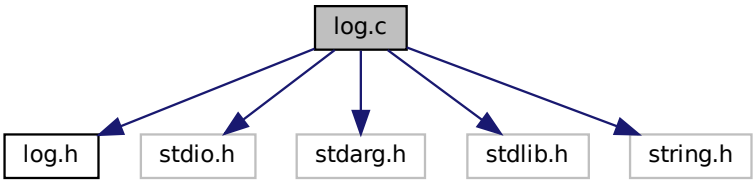
Chapter 2

File Documentation

2.1 log.c File Reference

```
#include "log.h"
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for log.c:



Functions

- void splitFileName (const char ∗fullFileName, char ∗path, char ∗fileName, char ∗extension)

  aux func: split fullFileName (with full path) into path/fileName/extension
- void initializePrinter (const char ∗path, const char ∗baseName, FileDestination files2open)

  open the files specified by files2open in the directory specified by path, with the basename specified
- void closePrinter ()

  closes all opened files
- void doneLEXstartSYN ()

  sets the curent compilation stage to SYN (syntatic analysis)
- void doneSYNstartTAB ()

  sets the curent compilation stage to TAB (symbol table)
- void doneTABstartGEN ()

  sets the curent compilation stage to GEN (code generation)

- void fflushc ()

  flushes all opened files.
- void pc (const char *format,...)

  prints in CURRENT output file AND stdout
- void pce (const char *format,...)

  prints in CURRENT output file AND stdout AND error file (3-way)
- void pp (FileDestination destination, const char *format,...)

  prints in both chosen output file(s) AND stdout

Variables

- FILE * fileER_

  error output
- FILE * fileLEX

  lexical analysis output
- FILE * fileSYN

  syntatic analysis output
- FILE * fileTAB

  symbol table output
- FILE * fileGEN

  generated code output
- FileDestination filesOpened

  sets which files will be opened. e.g. if you will only implement up to symbol table generation, do not open the file to output the generated code.
- FileDestination currentState

  marks the current stage of the compilation, used for pc and pce functions

2.1.1 Function Documentation

2.1.1.1 closePrinter()

```
void closePrinter ( )
```

closes all opened files

2.1.1.2 doneLEXstartSYN()

```
void doneLEXstartSYN ( )
```

sets the curent compilation stage to SYN (syntatic analysis)

2.1.1.3 doneSYNstartTAB()

```
void doneSYNstartTAB ( )
```

sets the curent compilation stage to TAB (symbol table)

2.1.1.4 doneTABstartGEN()

```
void doneTABstartGEN ( )
```

sets the curent compilation stage to GEN (code generation)

2.1.1.5 fflushc()

```
void fflushc ( )
```

flushes all opened files.

2.1.1.6 initializePrinter()

```
void initializePrinter (
            const char * path,
            const char * baseName,
            FileDestination files2open )
```

open the files specified by files2open in the directory specified by path, with the basename specified

- sets currentState to LEX - the first stage of compilation is always lexical analysis
- the supplied main code already calls this function adequately. Probably, students will not need to change it.

Parameters

path	directory for detailed output files
basename	the radical part of the file name
files2open	choose which files to open. If not all compilation stages will be run, you may choose to not open all files

example usage:

```
initializePrinter(detailpath, pgm, LOGALL); // open detailed output files
```

into the path given by detailpath, with the basename given by pgm, and LO↩
GALL means all files will be opened Here is the call graph for this function:

2.1.1.7 pc()

```
void pc (
            const char * format,
             ... )
```

prints in CURRENT output file AND stdout

- use this to **print usual output messages into the current compilation stage output**.
- this function will be used most of the time.

Parameters

format	variadic parameters, to be used as fprintf

example usage:

```
pc("my message n.  %i is %s",var_int,var_pointerchar); // prints on stdout
AND the output corresponding to the current compilation stage.
```

2.1.1.8 pce()

```
void pce (
            const char * format,
             ... )
```

prints in CURRENT output file AND stdout AND error file (3-way)

use this to **print error messages into the current compilation stage output**.

Parameters

format	variadic parameters, to be used as fprintf

example usage:

```
pce("my message n.  %i is %s",var_int,var_pointerchar); // prints on stdout
AND error AND the output corresponding to the current compilation stage.
```

2.1.1.9 pp()

```
void pp (
            FileDestination destination,
            const char * format,
             ...  )
```

prints in both chosen output file(s) AND stdout

this function is intended for **hardcoded fine control over output files**.

- it is a *variadic function* which accepts a variable number of arguments. It repasses its arguments to fprintf, printing into the output files indicated by the destination argument.
- it does not care about the current state.
- It checks if the file was set to be opened before printing

Parameters

destination	bitflag setting which output files will be used
format	after the destination flag, this function should be used as fprintf.

example usage (check the possible flags on the enum fileDestination) :

```
pp(SYN, "my message n.  %i is %s",var_int,var_pointerchar); // prints on
stdout and sintatic analysis output

pp(SYN | LEX, "my message n.  %i is %s",var_int,var_pointerchar); // prints
on stdout, lexic and sintatic analysis outputs

pp(TAB | ERR, "my message n.  %i is %s",var_int,var_pointerchar); // prints
on stdout, symbol table and error outputs
```

2.1.1.10 splitFileName()

```
void splitFileName (
            const char * fullFileName,
            char * path,
            char * fileName,
            char * extension )
```

aux func: split fullFileName (with full path) into path/fileName/extension

students usually will not need to use this function

Here is the caller graph for this function:

2.1.2 Variable Documentation

2.1.2.1 currentState

```
FileDestination currentState
```

marks the current stage of the compilation, used for pc and pce functions

2.1.2.2 fileER_

```
FILE* fileER_
```

error output

2.1.2.3 fileGEN

```
FILE* fileGEN
```

generated code output

2.1.2.4 fileLEX

```
FILE* fileLEX
```

lexical analysis output

2.1.2.5 filesOpened

```
FileDestination filesOpened
```

sets which files will be opened. e.g. if you will only implement up to symbol table generation, do not open the file to output the generated code.

2.1.2.6 fileSYN

```
FILE* fileSYN
```

syntatic analysis output

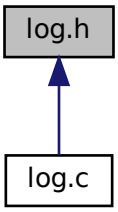
2.1.2.7 fileTAB

```
FILE* fileTAB
```

symbol table output

2.2 log.h File Reference

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum fileDestination FileDestination

  bitmask to select output files

Enumerations

- enum fileDestination {
  ER_ = 0x1, LEX = 0x2, LER = 0x3, SYN = 0x4,
  SER = 0x5, TAB = 0x8, TER = 0x9, GEN = 0x10,
  GER = 0x11, UP2SYN = 0x7, UP2TAB = 0xF, LOGALL = 0x1F }

  bitmask to select output files

Functions

- void initializePrinter (const char *path, const char *baseName, FileDestination files2open)

  open the files specified by files2open in the directory specified by path, with the basename specified
- void pp (FileDestination destination, const char *format,...)

  prints in both chosen output file(s) AND stdout
- void doneLEXstartSYN ()

  sets the curent compilation stage to SYN (syntatic analysis)
- void doneSYNstartTAB ()

  sets the curent compilation stage to TAB (symbol table)
- void doneTABstartGEN ()

  sets the curent compilation stage to GEN (code generation)
- void pc (const char *format,...)

  prints in CURRENT output file AND stdout
- void pce (const char *format,...)

  prints in CURRENT output file AND stdout AND error file (3-way)
- void fflushc ()

  flushes all opened files.
- void closePrinter ()

  closes all opened files

2.2.1 Typedef Documentation

2.2.1.1 FileDestination

```
typedef enum fileDestination FileDestination
```

bitmask to select output files

2.2.2 Enumeration Type Documentation

2.2.2.1 fileDestination

```
enum fileDestination
```

bitmask to select output files

Enumerator

ER_	NOT STDERR!!! JUST A FILE TO STORE YOUR ERR MSGS!!
LEX	lexical
LER	LEX AND ERR.
SYN	syntatic analisis
SER	syntatic analisis AND error
TAB	symbol table
TER	symbol table AND error
GEN	code generation
GER	code generation AND error
UP2SYN	ER + LEX + SYN - all up to synthatic analisis AND ERROR.
UP2TAB	ER + LEX + SYN + TAB - all up to symbol table AND ERROR.
LOGALL	ER + LEX + SYN + TAB + GEN - everything, including code generation AND ERROR.

2.2.3 Function Documentation

2.2.3.1 closePrinter()

```
void closePrinter ( )
```

closes all opened files

2.2.3.2 doneLEXstartSYN()

```
void doneLEXstartSYN ( )
```

sets the curent compilation stage to SYN (syntatic analysis)

2.2.3.3 doneSYNstartTAB()

```
void doneSYNstartTAB ( )
```

sets the curent compilation stage to TAB (symbol table)

2.2.3.4 doneTABstartGEN()

```
void doneTABstartGEN ( )
```

sets the curent compilation stage to GEN (code generation)

2.2.3.5 fflushc()

```
void fflushc ( )
```

flushes all opened files.

2.2.3.6 initializePrinter()

```
void initializePrinter (
            const char * path,
            const char * baseName,
            FileDestination files2open )
```

open the files specified by files2open in the directory specified by path, with the basename specified

- sets currentState to LEX - the first stage of compilation is always lexical analysis
- the supplied main code already calls this function adequately. Probably, students will not need to change it.

Parameters

path	directory for detailed output files
basename	the radical part of the file name
files2open	choose which files to open. If not all compilation stages will be run, you may choose to not open all files

example usage:

```
initializePrinter(detailpath, pgm, LOGALL); // open detailed output files
into the path given by detailpath, with the basename given by pgm, and LO↩
GALL means all files will be opened
```

Here is the call graph for this function:

2.2.3.7 pc()

```
void pc (
            const char * format,
             ... )
```

prints in CURRENT output file AND stdout

- use this to **print usual output messages into the current compilation stage output**.
- this function will be used most of the time.

Parameters

format	variadic parameters, to be used as fprintf

example usage:

```
pc("my message n.  %i is %s",var_int,var_pointerchar); // prints on stdout
AND the output corresponding to the current compilation stage.
```

2.2.3.8 pce()

```
void pce (
            const char * format,
             ... )
```

prints in CURRENT output file AND stdout AND error file (3-way)

use this to **print error messages into the current compilation stage output**.

Parameters

format	variadic parameters, to be used as fprintf

example usage:

```
pce("my message n.  %i is %s",var_int,var_pointerchar); // prints on stdout
AND error AND the output corresponding to the current compilation stage.
```

2.2.3.9 pp()

```
void pp (
            FileDestination destination,
            const char * format,
             ... )
```

prints in both chosen output file(s) AND stdout

this function is intended for **hardcoded fine control over output files**.

- it is a *variadic function* which accepts a variable number of arguments. It repasses its arguments to fprintf, printing into the output files indicated by the destination argument.
- it does not care about the current state.
- It checks if the file was set to be opened before printing

Parameters

destination	bitflag setting which output files will be used
format	after the destination flag, this function should be used as fprintf.

example usage (check the possible flags on the enum fileDestination) :

```
pp(SYN, "my message n.  %i is %s",var_int,var_pointerchar); // prints on
stdout and sintatic analysis output

pp(SYN | LEX, "my message n.  %i is %s",var_int,var_pointerchar); // prints
on stdout, lexic and sintatic analysis outputs

pp(TAB | ERR, "my message n.  %i is %s",var_int,var_pointerchar); // prints
on stdout, symbol table and error outputs
```

Index