TomPiler

Generated by Doxygen 1.9.3

Chapter 1

TomPiler

Version

0.2.5

1.0.1 Useful Pages

- compfiles.h
- file_util.h
- dfa.h
- tokens.h
- scan.h
- TCompFiles
- Scanner

1.0.2 About

Created by Group 3 for CSC-460, Language Translations with Dr. Pyzdrowski, at PennWest California.

Chapter 2

changelog

2/15/2023: Karl

- Finished debugging full DFA.
- Rewrote scanner in scan.h. No more using memory allocation and TokenCatch structures. It reads from file and writes directly to the listing and output files now.
- Deleted scanner.c, scanner.h, scanner_util.c, scanner_util.h and associated test files.
- Deleted the recognize keyword token function and the associated dfa, since the new dfa covers everything.
- If we need those files and features back, we can revert to an earlier commit.

2/14/2023: Karl

- Created a FULL dfa planned to replace all current logic.
- Moved all test header files to one header file "test.h".

2/13/2023: Karl and thomas

- fixed listing file not loading
- fixed extractInt AND extractInt tests (they were using extractWord)!
- running error count and print errors
- fixed detect SCANEOF
- fixed null terminates at end of buffer for no overflow print
- fixed last line being ignored
- several types of token catch initializers
- token catch allocates memory; can use the tokens later in the parser
- scanner printLine fixes
- print error count
- Token_GetOpRaw
- formatting line printing

2/13/2023: All Group Members

- Extract op
- Fleshed out the switch statement for take Action
- printLine always happens at the end of populateBuffer now
- extractOperator and Scanner_ExtractOperator.... extractOperator is in Scanner not scanner_util because it is dependent
- Token_CatchOp Token_CatchError

2/12/2023: Karl

- Added recognizers for trueop, nullop, falseop to the state transition table, which I had missed before.
- Skipwhitespace now returns the number of characters missed. This can be useful if we extract a number and it isn't followed by a whitespace (skipwhitespace will produce 0.) This may be a cause for an error print. (Worth asking)
- Fixed extractWord errors and added Scanner_ExtractWord
- Added extractInteger and Scanner_ExtractInteger
- Added a boundries member to Tscanner. This is a list of all boundry characters that delimit words, identifiers, and number and it includes all operators plus whitespace and EOF. See Scanner_Init() for how it's constructed.
- Token Recognize now returns ERROR if there is a non number, non alphanumeric within the tested string. It also now allows for identifiers to have numbers.
- Created Token wrapping struct called TokenCatch that encapsulates info about the token such as the recognized raw string, line number, and so forth.
- Scanner now takes files on Scanner_Scan(files...) not on Scanner_Init()
- Init and DeInit functions in main
- Moved switch statement/dispatcher into a function Scanner_TakeAction(lookaheadResult)
- Made basic Scanner_Scan(); currently will print the listing file numbers and lines only

2/11/2023: Karl

- Added skipWhitespace general function in scanner_util and added tests for it
- Added charIn function in scanner_util which is used by extractWord function in scanner_util.

2/10/2023: Karl

- Added Scanner_populateBuffer() and tested it.
- Created Scanner_LookAhead() and put a switch statement in Scanner_Scan()

2/8/2023: Karl, Thomas, Anthony

- created scanner.h and scanner.c

- add struct to hold scanner info
- scanner lifecycle functions
- scanner buffer functions
- scanner_util , created buffer resize and refresh functions
- created tests for scanner_util

2/7/2023: Karl

- abstracted command line argument parse and calls to a new function, CompFiles_FileOpenFromCLIArgs, which also generates the Temp file.
- created Tokens_GetName, the tokens Map, the tokens enum.
- created a state transition table as a 3d array for a keyword recognizer Token_RecognizeKeyword. Used excel to design the table; .xlsx is in the /docs folder.

2/1/2023 : All Group Members

- used fileapi.h to create a getAbsolutePath function
- created checkIfSamePaths function to compare file name actual paths
- reworked the validate files functions to check for output/input name collisions
- adjusted some printing
- fixed tempfile bug

1/28/2023: Karl

- used doxygen to generate documentation

1/27/2023: Thomas, Karl

- wrote copy inputs to outputs function

1/26/2023 : All group members

- refactored file_util into two files: compfiles and file_util
- worked on logic for validating an output file name
- auto-generate temp file
- validate listing file in a similar way to output file
- combined validation functions into one validate func; just pass it the command line arguments

1/25/2023 : Thomas

- promptOutputFile()
- Modified getString() to use realloc

1/24/2023 : All group members

- worked on main logic
- changed CompFiles struct to be a state machine
- created promptFilename

1/23/2023 : Thomas and klm127

- changed Author comment to include e-mail and class name.
- removed old addExtension function, old promptFilename function, and closeFile function.
- added promptFilename and getString function(not yet covered by unit tests)
- removed all of the stdin swapping to a `separate repo`, and tested it, due to nagging bugs.
  - NOTE: It turned out that the bug was that `dup2` closes a file and `fclose` was being called afterwards.

moved test dependencies to a sub folder `lib` and updated compilation commands to use this on the include path

1/22/2023 : thomas and klm127

- added removeExtension function and tests
- confirmed getchar will read an 'enter'.
- thomas fixed prompting function to accept alternate inputs
- added backupFile function and tests
- Included tests for filepaths with directories
- redid filenameHasExtension. It now allows for filenames like ".bob" and doesnt allow filenames that end in slashes. It does allow folders to have '.'s in them.

1/21/2023 : klm127

- added #pragma region directives to header files. This is basically just markup for VSCode. Each of these regions can now be folded in Visual Studio or VSCode. This does not affect -ansi compilation on MinGW-↩ W64 gcc; as far as I can tell. The purpose is to make the code much easier to navigate without relying on tab-based folding. `See Also: stackoverflow answer`
- Cleaned up comments, tab-based folding, etc.
- Fixed up the `addExtension` to use malloc to create a longer, concatenated string out of its inputs. Added unit tests for `addExtension`.
- Refactored std swapping test utility functions. The best way to test a prompter is now to use is to call `set↩ STDin3`, get the value, then dont forget to call `restoreSTD3()` *before* making a test-based assertion.

1/20/2023 : All group members in collaboration

- created `promptUserOverwriteSelection`.
- created tests for `promptUserOverwriteSelection`. This was quite an involved task because we had to figure out how to temporarily replace stdin and stdout with alternative files so that we could test functionalities like `scanf`. Ultimately we were able to figure it out.

1/19/2023 : klm127

- changed directory structure, added docs, src, and tests
- created changelog, included CuTest's readme in the docs
- updated tasks.json in .vscode to configure code generation
- output file is now `fileopen.exe` due to interpretation of video instructions
- added .gitignore so we can exclude executables from github
- Added the testing suite CuTest. More info `here`
- Added the functions `fileExists` and `filenameHasExtension`
- Added unit tests for `fileExists` and `filenameHasExtension`

Chapter 3

VSCode setup instructions

VSCode provides a decent environment to work in C with its highly customizable features, low overhead, and rich extension options.

The folder `.vscode` configures the workspace for use with VSCode.

`tasks.json` describes build and run commands.

Ctrl+Shift+B will build and run the programs.

You may have to change compilerPath in c_cpp_properties.json to your own compiler.

I'm using GCC 8.1 (came with CodeBlocks) with the -ansi flag.

I referenced this article when setting up the VSCode environment. `Medium Article`

I referenced the `gcc documentation` while setting up the compiler.

Chapter 4

Tompiler Readme

Tompiler will be a relatively simple compiler built for educational and explorative purposes.

4.1 Compiling

Compiler configurations are stored in the .bat files. There are two of them.

- runTests.bat compiles and runs the tests.
- compile.bat compiles and runs the code.

4.2 Using

Running `compile.bat` will run the compiler after executing. You can also find the executable, `fileopen.exe`, in your `bin` directory.

It takes up to two command line arguments. The first argument can be an input file path while the second argument can be an output file path.

Place the `bin` directory on your system path if you want to be able to run tompiler from anywhere.

4.3 Folder and file Descriptions

- .vscode : Contains vscode configurations.
- docs : Contains additional documentation
- src : Contains source code
  - main.c : Program entry point
  - compfiles.c / .h : struct for managing input output file access
  - file_util.c / .h : file i/o helpersfor the compiler
  - dfa.c / .h : The DFA which drives the scanning process.

  –

- tests : Contains source code for tests
  - lib: Contains test dependencies
    - CuTest.c / .h : CuTest micro test framework
    - std_swapper.c / .h : For swapping stdin and out with files.
  - file_util_test.c : tests for file util
  - dfa_test.c : tests for dfa.
  - tokens_test.c : test for token functions.
  - main_test.c : entry point for test compilation
  - tests.h : each test file has one exported member, a function that returns the testing suite. They are all declared here.

4.4 Included 3rd party library, CuTest.

Link to Cutest page

This is a small bit of code (only 340 lines!) that provides a unit testing skeleton.

4.5 Credits

- Tom Terhune
- Karl Miller
- Anthony Stepich

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Data Structure Documentation

7.1 Scanner Struct Reference

`#include <scan.h>`

Data Fields

- int line_no
- int col_no
- int errors
- FILE * in
- FILE * out
- FILE * temp
- FILE * listing

7.1.1 Detailed Description

Scanner struct holds references to the files being read and keeps track of the line and column position. It is a singleton.

7.1.2 Field Documentation

7.1.2.1 col_no

`int col_no`

7.1.2.2 errors

```
int errors
```

7.1.2.3 in

```
FILE* in
```

7.1.2.4 line_no

```
int line_no
```

7.1.2.5 listing

```
FILE* listing
```

7.1.2.6 out

```
FILE* out
```

7.1.2.7 temp

```
FILE* temp
```

The documentation for this struct was generated from the following file:

- src/scan.h

7.2 TCompFiles Struct Reference

Manages input and output files.

```
#include <compfiles.h>
```

Data Fields

- FILE * in
- FILE * out
- FILE * temp
- FILE * listing
- short input_file_state
- short output_file_state
- short listing_file_state
- short terminate_requested
- short has_requested_default_filename
- char * input_file_name
- char * output_file_name
- char * listing_file_name
- char * temp_file_name

7.2.1 Detailed Description

Manages input and output files.

CompFiles is a globally accesible struct which maintains references to the loaded files.

It has a number of functions closely associated to it. In that way it is a class-like, but a singleton. There is only one CompFiles that ever should exist.

7.2.2 Field Documentation

7.2.2.1 has_requested_default_filename

```
short has_requested_default_filename
```

1 indicates that a user has requested to use a default ouput filename already. This is so that if the user selects this twice, they will automatically exit instead of looping the prompt.

7.2.2.2 in

```
FILE* in
```

A file pointer to an open input file.

7.2.2.3 input_file_name

```
char* input_file_name
```

The input filename.

7.2.2.4 input_file_state

```
short input_file_state
```

Determines the status of input file validation.

7.2.2.5 listing

```
FILE* listing
```

A file pointer to an open listing file.

7.2.2.6 listing_file_name

```
char* listing_file_name
```

The listing filename

7.2.2.7 listing_file_state

```
short listing_file_state
```

Determines the status of listing file validation.

7.2.2.8 out

```
FILE* out
```

A file pointer to an open output file.

7.2.2.9 output_file_name

```
char* output_file_name
```

The output filename,

7.2.2.10 output_file_state

```
short output_file_state
```

Determines the status of output file validation.

7.2.2.11 temp

```
FILE* temp
```

A file pointer to an open tmp file.

7.2.2.12 temp_file_name

```
char* temp_file_name
```

The temp filename

7.2.2.13 terminate_requested

```
short terminate_requested
```

1 indicates that a user requested to terminate the program.

The documentation for this struct was generated from the following file:

- src/compfiles.h

7.3 TokenCatch Struct Reference

```
#include <tokens.h>
```

Data Fields

- short token
- char * raw
- int line_no
- int col_no

7.3.1 Detailed Description

7.3.1.1 Note: TokenCatch is no longer used. It was used in an earlier version of this program. It may be revived in the future depending on the needs of the parser.

7.3.2 Field Documentation

7.3.2.1 col_no

```
int col_no
```

7.3.2.2 line_no

```
int line_no
```

7.3.2.3 raw

```
char* raw
```

7.3.2.4 token

```
short token
```

The documentation for this struct was generated from the following file:

- src/tokens.h

Chapter 8

File Documentation

8.1 docs/changelog.md File Reference

8.2 docs/VSCode.md File Reference

8.3 Readme.md File Reference

8.4 src/compfiles.c File Reference

```
#include "compfiles.h"
```

Functions

- void CompFiles_Init ()
- void CompFiles_GenerateTempFile ()
- void CompFiles_DeInit ()
- TCompFiles * CompFiles_GetFiles ()
- void CompFiles_LoadInputFile (FILE *newInputFile)
- void CompFiles_LoadOutputFile (FILE *newOutputFile)
- void CompFiles_LoadTempFile (FILE *newTempFile)
- void CompFiles_LoadListingFile (FILE *newListingFile)
- char * CompFiles_promptInputFilename ()
- void CompFiles_CopyInputToOutputs ()
- short CompFiles_Open (int argc, char *argv[])
- short CompFiles_AcquireValidatedFiles (char *inputFilename, const char *outputFilename)
- short CompFiles_AcquireValidatedInputFile (char *filename)
- short CompFiles_AcquireValidatedOutputFile (const char *filename)
- short CompFiles_AcquireValidatedListingFile (const char *filename)
- char * CompFiles_promptOutputFilename ()
- short CompFiles_promptUserOverwriteSelection ()

8.4.1 Function Documentation

8.4.1.1 CompFiles_AcquireValidatedFiles()

```
short CompFiles_AcquireValidatedFiles (
            char * inputFilename,
            const char * outputFilename )
```

Loops and prompts until all input and output files are set correctly or until terminate is requested. After the input, output, and listing files are generated, CompFiles_AcquireValidatedFiles also generates a temp file.

Parameters

inputFilename	a filename with which to begin input validation with or NULL
outputFilename	a filename with which to begin output validation with or NULL

Returns

1 if terminate was requested. Otherwise, 0.

Author

klm127

Date

1/26/2023

8.4.1.2 CompFiles_AcquireValidatedInputFile()

```
short CompFiles_AcquireValidatedInputFile (
            char * filename )
```

Validates an input file name and sets the value in the struct. It will continue looping until the user has supplied a valid filename or elected to quit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if the input file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.4.1.3 CompFiles_AcquireValidatedListingFile()

```
short CompFiles_AcquireValidatedListingFile (
            const char * filename )
```

Validates a listing file name and sets the value in the struct.
Called by CompFiles_ValidateOutputFile after an output file has been fully validated. The parameter passed will be the name of the output file with the extension 'list' instead.
If this file happens to exist, a similar loop will occur as when a user attempts to load an extant output file. The user will be prompted to enter a new file until one is validated or they elect to exit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if an output file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.4.1.4 CompFiles_AcquireValidatedOutputFile()

```
short CompFiles_AcquireValidatedOutputFile (
```

```
const char * filename )
```

Validates an output file name and sets the value in the struct. It will continue looping until the user has supplied a valid filename or elected to quit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if an output file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.4.1.5 CompFiles_CopyInputToOutputs()

```
void CompFiles_CopyInputToOutputs ( )
```

CompFiles_CopyInputToOutputs copies all the data from the input file to each of the output files. After execution, all output files (tmp, list, and out) will have text identical to the input files.

Warning

Precondition: All CompFiles file pointers must be open and ready to read/write.

Author

Thomas, Karl

Date

1/27/2023

8.4.1.6 CompFiles_DeInit()

```
void CompFiles_DeInit ( )
```

Closes any open files and returns CompFiles to the default values. Deletes the temp file.

8.4.1.7 CompFiles_GenerateTempFile()

```
void CompFiles_GenerateTempFile ( )
```

Generates a temporary file with a unique name. This file will be destroyed when CompFiles_DeInit() is called.

Author

klm127

Date

1/26/2023

8.4.1.8 CompFiles_GetFiles()

```
TCompFiles * CompFiles_GetFiles ( )
```

Gets the CompFiles struct so that the validated files can be used elsewhere in the program.

Returns

A TCompFiles struct.

8.4.1.9 CompFiles_Init()

```
void CompFiles_Init ( )
```

Initializes CompFiles struct to default values.

Note

Covered by unit tests.

8.4.1.10 CompFiles_LoadInputFile()

```
void CompFiles_LoadInputFile (
            FILE * newInputFile )
```

CompFiles_LoadInputFile loads a new file pointer as the input file. If there is a file already loaded, it closes that file first.

Parameters

newInputFile	A pointer to an open file in read mode.

8.4.1.11 CompFiles_LoadListingFile()

```
void CompFiles_LoadListingFile (
            FILE * newListingFile )
```

CompFiles_LoadListingFile loads a new file pointer as the listing file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.4.1.12 CompFiles_LoadOutputFile()

```
void CompFiles_LoadOutputFile (
            FILE * newOutputFile )
```

CompFiles_LoadOutputFile loads a new file pointer as the output file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.4.1.13 CompFiles_LoadTempFile()

```
void CompFiles_LoadTempFile (
            FILE * newTempFile )
```

CompFiles_LoadTempFile loads a new file pointer as the temp file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.4.1.14 CompFiles_Open()

```
short CompFiles_Open (
            int argc,
            char * argv[] )
```

Parses the command line args and calls functions to acquire validated filenames.

Parameters

argc	The argument count.
argv	The argument array.

Returns

1 if terminate was requested. Otherwise, 0.

Author

klm127

Date

2/7/2023

8.4.1.15 CompFiles_promptInputFilename()

```
char * CompFiles_promptInputFilename ( )
```

Calls the function getString() to recieve a filename from the user and returns it. It will set the 'terminate requested' flag in CompFiles if the user inputs only a \n.

Returns

char * inputfilename to be verified

Author

thomaserh99

Date

1/23/2023

Note

Covered by Unit Tests

8.4.1.16 CompFiles_promptOutputFilename()

```
char * CompFiles_promptOutputFilename ( )
```

Calls the function getString() to recieve a filename from the user and returns it. It will set the 'terminate requested' flag in CompFiles if the user inputs only a \n.

Warning

This should not be called until the input filename has been set. The user may elect to generate an output filename based on the input file. (inputfilename + .out)

Returns

A malloced string of an output filename to be verified.

Author

thomaserh99

Date

Created On: 1/23/2023

Note

Covered by Unit Tests

8.4.1.17 CompFiles_promptUserOverwriteSelection()

```
short CompFiles_promptUserOverwriteSelection ( )
```

Prompts the user as to what they want to do about an output file already existing. It prints a prompt and parses the user response to one of the USER_OUTPUT_OVERWRITE_SELECTION enums. It does NOT loop.

Returns

short corresponding to one of the enums of USER_OTUPUT_OVERWRITE_SELECTION

Author

klm127, thomasterh99, anthony91501

Date

1/20/2023

Note

Covered by Unit Tests

8.5 src/compfiles.h File Reference

CompFiles struct and "methods" definitions.

```
#include <stdio.h>
#include "file_util.h"
#include <string.h>
#include <stdlib.h>
```

Data Structures

- struct TCompFiles

  Manages input and output files.

Enumerations

- enum COMPFILES_STATE { COMPFILES_STATE_NO_NAME_PROVIDED = 0 , COMPFILES_STATE_NAME_NEEDS_VALID = 1 , COMPFILES_STATE_NAME_VALIDATED = 2 }
- enum USER_OUTPUT_OVERWRITE_SELECTION { USER_OUTPUT_OVERWRITE_REENTER_FILENAME_SELECTED = 1 , USER_OUTPUT_OVERWRITE_OVERWRITE_EXI = 2 , USER_OUTPUT_OVERWRITE_DEFAULT_FILENAME = 3 , USER_OUTPUT_TERMINATE_PROGRAM = 4 , USER_OUTPUT_TERMINATE_INVALID_ENTRY = -1 }

Functions

- void CompFiles_Init ()
- void CompFiles_DeInit ()
- void CompFiles_GenerateTempFile ()
- TCompFiles * CompFiles_GetFiles ()
- void CompFiles_LoadInputFile (FILE *newInputFile)
- void CompFiles_LoadOutputFile (FILE *newOutputFile)
- void CompFiles_LoadTempFile (FILE *newTempFile)
- void CompFiles_LoadListingFile (FILE *newListingFile)
- short CompFiles_Open (int argc, char *argv[ ])
- short CompFiles_AcquireValidatedFiles (char *inputFilename, const char *outputFilename)
- short CompFiles_AcquireValidatedInputFile (char *filename)
- short CompFiles_AcquireValidatedOutputFile (const char *filename)
- short CompFiles_AcquireValidatedListingFile (const char *filename)
- char * CompFiles_promptInputFilename ()
- char * CompFiles_promptOutputFilename ()
- short CompFiles_promptUserOverwriteSelection ()
- void CompFiles_CopyInputToOutputs ()

Variables

- TCompFiles CompFiles

8.5.1 Detailed Description

CompFiles struct and "methods" definitions.
CompFiles struct and "methods".
CompFiles is a struct which holds pointers to the compilation input and output files. It also tracks their names and their validation status. It provides methods for prompting the user for valid file names until terminate is requested or all files are validated.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

January 2023

8.5.2 Enumeration Type Documentation

8.5.2.1 COMPFILES_STATE

`enum COMPFILES_STATE`
Describes the state of a filename validation process

Enumerator

COMPFILES_STATE_NO_NAME_PROVIDED	
COMPFILES_STATE_NAME_NEEDS_VALIDATION	
COMPFILES_STATE_NAME_VALIDATED	

8.5.2.2 USER_OUTPUT_OVERWRITE_SELECTION

```
enum USER_OUTPUT_OVERWRITE_SELECTION
```

Describes the possible selections a user may make when they elect to output to a file that already exists.

Enumerator

USER_OUTPUT_OVERWRITE_REENTER_FILENAME_SELECTED	
USER_OUTPUT_OVERWRITE_OVERWRITE_EXISTING_FILE	
USER_OUTPUT_OVERWRITE_DEFAULT_FILENAME	
USER_OUTPUT_TERMINATE_PROGRAM	
USER_OUTPUT_TERMINATE_INVALID_ENTRY	

8.5.3 Function Documentation

8.5.3.1 CompFiles_AcquireValidatedFiles()

```
short CompFiles_AcquireValidatedFiles (
            char * inputFilename,
            const char * outputFilename )
```

Loops and prompts until all input and output files are set correctly or until terminate is requested. After the input, output, and listing files are generated, CompFiles_AcquireValidatedFiles also generates a temp file.

Parameters

inputFilename	a filename with which to begin input validation with or NULL
outputFilename	a filename with which to begin output validation with or NULL

Returns

1 if terminate was requested. Otherwise, 0.

Author

klm127

Date

1/26/2023

8.5.3.2 CompFiles_AcquireValidatedInputFile()

```
short CompFiles_AcquireValidatedInputFile (
            char * filename )
```

Validates an input file name and sets the value in the struct. It will continue looping until the user has supplied a valid filename or elected to quit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if the input file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.5.3.3 CompFiles_AcquireValidatedListingFile()

```
short CompFiles_AcquireValidatedListingFile (
            const char * filename )
```

Validates a listing file name and sets the value in the struct.
Called by CompFiles_ValidateOutputFile after an output file has been fully validated. The parameter passed will be the name of the output file with the extension 'list' instead.
If this file happens to exist, a similar loop will occur as when a user attempts to load an extant output file. The user will be prompted to enter a new file until one is validated or they elect to exit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if an output file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.5.3.4 CompFiles_AcquireValidatedOutputFile()

```
short CompFiles_AcquireValidatedOutputFile (
            const char * filename )
```

Validates an output file name and sets the value in the struct. It will continue looping until the user has supplied a valid filename or elected to quit the program.

Parameters

filename	a filename with which to begin input validation with or NULL

Returns

0 if an output file was validated and loaded into the struct. 1 if the user requested to terminate the program.

8.5.3.5 CompFiles_CopyInputToOutputs()

```
void CompFiles_CopyInputToOutputs ( )
```

CompFiles_CopyInputToOutputs copies all the data from the input file to each of the output files. After execution, all output files (tmp, list, and out) will have text identical to the input files.

Warning

Precondition: All CompFiles file pointers must be open and ready to read/write.

Author

Thomas, Karl

Date

1/27/2023

8.5.3.6 CompFiles_DeInit()

```
void CompFiles_DeInit ( )
```

Closes any open files and returns CompFiles to the default values. Deletes the temp file.

8.5.3.7 CompFiles_GenerateTempFile()

```
void CompFiles_GenerateTempFile ( )
```

Generates a temporary file with a unique name. This file will be destroyed when CompFiles_DeInit() is called.

Author

klm127

Date

1/26/2023

8.5.3.8 CompFiles_GetFiles()

```
TCompFiles * CompFiles_GetFiles ( )
```

Gets the CompFiles struct so that the validated files can be used elsewhere in the program.

Returns

A TCompFiles struct.

8.5.3.9 CompFiles_Init()

```
void CompFiles_Init ( )
```

Initializes CompFiles struct to default values.

Note

Covered by unit tests.

8.5.3.10 CompFiles_LoadInputFile()

```
void CompFiles_LoadInputFile (
            FILE * newInputFile )
```

CompFiles_LoadInputFile loads a new file pointer as the input file. If there is a file already loaded, it closes that file first.

Parameters

newInputFile	A pointer to an open file in read mode.

8.5.3.11 CompFiles_LoadListingFile()

```
void CompFiles_LoadListingFile (
            FILE * newListingFile )
```

CompFiles_LoadListingFile loads a new file pointer as the listing file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.5.3.12 CompFiles_LoadOutputFile()

```
void CompFiles_LoadOutputFile (
            FILE * newOutputFile )
```

CompFiles_LoadOutputFile loads a new file pointer as the output file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.5.3.13 CompFiles_LoadTempFile()

```
void CompFiles_LoadTempFile (
            FILE * newTempFile )
```

CompFiles_LoadTempFile loads a new file pointer as the temp file. If there is a file already loaded, it closes that file first.

Parameters

newOutputFile	A pointer to an open file in write mode.

8.5.3.14 CompFiles_Open()

```
short CompFiles_Open (
            int argc,
            char * argv[] )
```

Parses the command line args and calls functions to acquire validated filenames.

Parameters

argc	The argument count.
argv	The argument array.

Returns

1 if terminate was requested. Otherwise, 0.

Author

klm127

Date

2/7/2023

8.5.3.15 CompFiles_promptInputFilename()

```
char * CompFiles_promptInputFilename ( )
```

Calls the function getString() to recieve a filename from the user and returns it. It will set the 'terminate requested' flag in CompFiles if the user inputs only a \n.

Returns

char * inputfilename to be verified

Author

thomaserh99

Date

1/23/2023

Note

Covered by Unit Tests

8.5.3.16 CompFiles_promptOutputFilename()

```
char * CompFiles_promptOutputFilename ( )
```

Calls the function getString() to recieve a filename from the user and returns it. It will set the 'terminate requested' flag in CompFiles if the user inputs only a \n.

Warning

This should not be called until the input filename has been set. The user may elect to generate an output filename based on the input file. (inputfilename + .out)

Returns

A malloced string of an output filename to be verified.

Author

thomaserh99

Date

Created On: 1/23/2023

Note

Covered by Unit Tests

8.5.3.17 CompFiles_promptUserOverwriteSelection()

```
short CompFiles_promptUserOverwriteSelection ( )
```

Prompts the user as to what they want to do about an output file already existing. It prints a prompt and parses the user response to one of the USER_OUTPUT_OVERWRITE_SELECTION enums. It does NOT loop.

Returns

short corresponding to one of the enums of USER_OTUPUT_OVERWRITE_SELECTION

Author

klm127, thomasterh99, anthony91501

Date

1/20/2023

Note

Covered by Unit Tests

8.5.4 Variable Documentation

8.5.4.1 CompFiles

`TCompFiles CompFiles`
The CompFiles singleton.

8.6 compfiles.h

Go to the documentation of this file.

```
1
2 #ifndef compfiles_h
3 #define compfiles_h
4
5 #include <stdio.h>
6 #include "file_util.h"
7 #include <string.h>
8 #include <stdlib.h>
9
21 /*
22 -----------------
23 CompFiles typedef
24 -----------------
25 */
26 #pragma region structs
28 enum COMPFILES_STATE {
29     COMPFILES_STATE_NO_NAME_PROVIDED = 0,
30     COMPFILES_STATE_NAME_NEEDS_VALIDATION = 1,
31     COMPFILES_STATE_NAME_VALIDATED = 2
32 };
33
41 typedef struct {
43     FILE * in;
45     FILE * out;
47     FILE * temp;
49     FILE * listing;
51     short input_file_state;
53     short output_file_state;
55     short listing_file_state;
57     short terminate_requested;
59     short has_requested_default_filename;
61     char * input_file_name;
63     char * output_file_name;
65     char * listing_file_name;
67     char * temp_file_name;
68 } TCompFiles;
69
71 TCompFiles CompFiles;
72
73 #pragma endregion structs
74
75 /*
76 -------------------
77 CompFiles lifecycle
78 -------------------
79 */
80 #pragma region lifecycle
81
85 void CompFiles_Init();
87 void CompFiles_DeInit();
94 void CompFiles_GenerateTempFile();
95
100 TCompFiles* CompFiles_GetFiles();
```

```
101
102 #pragma endregion lifecycle
103
104 /*
105 -----------------
106 CompFiles setters
107 -----------------
108 */
109 #pragma region setters
110
114 void CompFiles_LoadInputFile(FILE * newInputFile);
115
119 void CompFiles_LoadOutputFile(FILE * newOutputFile);
120
124 void CompFiles_LoadTempFile(FILE * newTempFile);
125
129 void CompFiles_LoadListingFile(FILE * newListingFile);
130
131 #pragma endregion setters
132
133 /*
134 ------------------
135 CompFiles prompts
136 ------------------
137 */
138 #pragma region prompts
139
140
150 short CompFiles_Open(int argc, char *argv[]);
151
152
162 short CompFiles_AcquireValidatedFiles(char * inputFilename, const char * outputFilename);
163
164
171 short CompFiles_AcquireValidatedInputFile(char * filename);
172
179 short CompFiles_AcquireValidatedOutputFile(const char * filename);
180
191 short CompFiles_AcquireValidatedListingFile(const char * filename);
192
202 char * CompFiles_promptInputFilename();
203
215 char * CompFiles_promptOutputFilename();
216
220 enum USER_OUTPUT_OVERWRITE_SELECTION {
221     USER_OUTPUT_OVERWRITE_REENTER_FILENAME_SELECTED = 1,
222     USER_OUTPUT_OVERWRITE_OVERWRITE_EXISTING_FILE = 2,
223     USER_OUTPUT_OVERWRITE_DEFAULT_FILENAME = 3,
224     USER_OUTPUT_TERMINATE_PROGRAM = 4,
225     USER_OUTPUT_TERMINATE_INVALID_ENTRY = -1
226 };
238 short CompFiles_promptUserOverwriteSelection();
239
240 #pragma endregion prompts
241
242 /*
243 --------------------
244 CompFiles operations
245 --------------------
246 */
247 #pragma region operations
248
258 void CompFiles_CopyInputToOutputs();
259
260 #pragma endregion operations
261
262
263 #endif
264
```

8.7 src/dfa.c File Reference

The DFA and related logic definitions.

```
#include "tokens.h"
#include "dfa.h"
#include <string.h>
#include <stdio.h>
```

Enumerations

- enum DFA_STATES {
  STATE_START , STATE_ID , STATE_ERROR , STATE_B ,
  STATE_BE , STATE_BEG , STATE_BEGI , STATE_BEGIN ,
  STATE_E , STATE_EN , STATE_END , STATE_R ,
  STATE_RE , STATE_REA , STATE_READ , STATE_I ,
  STATE_IF , STATE_T , STATE_TH , STATE_THE ,
  STATE_THEN , STATE_EL , STATE_ELS , STATE_ELSE ,
  STATE_ENDI , STATE_ENDIF , STATE_ENDW , STATE_ENDWH ,
  STATE_ENDWHI , STATE_ENDWHIL , STATE_ENDWHILE , STATE_W ,
  STATE_WH , STATE_WHI , STATE_WHIL , STATE_WHILE ,
  STATE_F , STATE_FA , STATE_FAL , STATE_FALS ,
  STATE_FALSE , STATE_TR , STATE_TRU , STATE_TRUE ,
  STATE_N , STATE_NU , STATE_NUL , STATE_NULL ,
  STATE_LPAR , STATE_RPAR , STATE_SEMIC , STATE_COMMA ,
  STATE_COLON , STATE_COLONEQUALS , STATE_PLUS , STATE_MINUS ,
  STATE_MULTIPLY , STATE_DIV , STATE_NOT , STATE_LESS ,
  STATE_LESSEQ , STATE_GREAT , STATE_GREATEQ , STATE_EQ ,
  STATE_NOTEQ , STATE_INT , STATE_EOF , STATE_WR ,
  STATE_WRI , STATE_WRIT , STATE_WRITE }
- enum DFA_CHARS {
  CH_A , CH_B , CH_C , CH_D ,
  CH_E , CH_F , CH_G , CH_H ,
  CH_I , CH_J , CH_K , CH_L ,
  CH_M , CH_N , CH_O , CH_P ,
  CH_Q , CH_R , CH_S , CH_T ,
  CH_U , CH_V , CH_W , CH_X ,
  CH_Y , CH_Z , CH_WSPC , CH_LPRN ,
  CH_RPRN , CH_SEMIC , CH_COMM , CH_COLON ,
  CH_EQU , CH_PLUS , CH_MINUS , CH_STAR ,
  CH_DIV , CH_NOT , CH_LT , CH_GT ,
  CH_NUM , CH_EOF , CH_NOTINSET , CH_NLINE }

Functions

- char * GetStateString (int n)
- char * GetDFAColString (int n)
- short GetDFAColumn (char c)
- int GetNextToken (FILE *file, int *charsRead)
- int GetNextTokenInBuffer (char *buffer, int *bufIndex, int *charsRead)
- void printCell (int row, int col)
- void printStateAndChar (int row, int col)

Variables

- short DFA [71][44][3]

8.7.1 Detailed Description

The DFA and related logic definitions.
The DFA is a 3 dimensional array that maps a given state and character input to a result consisting of the next state, token, and whether reading should continue.
The DFA was created in Excel, and the excel file is available in docs/fullDFA.xlsx.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.7.2 Enumeration Type Documentation

8.7.2.1 DFA_CHARS

`enum DFA_CHARS`

Enumerator

CH_A	
CH_B	
CH_C	
CH_D	
CH_E	
CH_F	
CH_G	
CH_H	
CH_I	
CH_J	
CH_K	
CH_L	
CH_M	
CH_N	
CH_O	
CH_P	
CH_Q	
CH_R	
CH_S	
CH_T	
CH_U	
CH_V	
CH_W	
CH_X	
CH_Y	
CH_Z	
CH_WSPC	
CH_LPRN	
CH_RPRN	
CH_SEMIC	
CH_COMM	
CH_COLON	
CH_EQU	
CH_PLUS	
CH_MINUS	
CH_STAR	
CH_DIV	
CH_NOT	
CH_LT	
CH_GT	

Enumerator

CH_NUM	
CH_EOF	
CH_NOTINSET	
CH_NLINE	

8.7.2.2 DFA_STATES

`enum DFA_STATES`

Enumerator

STATE_START	
STATE_ID	
STATE_ERROR	
STATE_B	
STATE_BE	
STATE_BEG	
STATE_BEGI	
STATE_BEGIN	
STATE_E	
STATE_EN	
STATE_END	
STATE_R	
STATE_RE	
STATE_REA	
STATE_READ	
STATE_I	
STATE_IF	
STATE_T	
STATE_TH	
STATE_THE	
STATE_THEN	
STATE_EL	
STATE_ELS	
STATE_ELSE	
STATE_ENDI	
STATE_ENDIF	
STATE_ENDW	
STATE_ENDWH	
STATE_ENDWHI	
STATE_ENDWHIL	
STATE_ENDWHILE	
STATE_W	
STATE_WH	
STATE_WHI	
STATE_WHIL	
STATE_WHILE	
STATE_F	
STATE_FA	

Enumerator

STATE_FAL	
STATE_FALS	
STATE_FALSE	
STATE_TR	
STATE_TRU	
STATE_TRUE	
STATE_N	
STATE_NU	
STATE_NUL	
STATE_NULL	
STATE_LPAR	
STATE_RPAR	
STATE_SEMIC	
STATE_COMMA	
STATE_COLON	
STATE_COLONEQUALS	
STATE_PLUS	
STATE_MINUS	
STATE_MULTIPLY	
STATE_DIV	
STATE_NOT	
STATE_LESS	
STATE_LESSEQ	
STATE_GREAT	
STATE_GREATEQ	
STATE_EQ	
STATE_NOTEQ	
STATE_INT	
STATE_EOF	
STATE_WR	
STATE_WRI	
STATE_WRIT	
STATE_WRITE	

8.7.3 Function Documentation

8.7.3.1 GetDFAColString()

```
char * GetDFAColString (
            int n )
```

GetDFAColString returns the column name associated with a given number. It is used only for debugging the DFA.

8.7.3.2 GetDFAColumn()

```
short GetDFAColumn (
            char c )
```

Translates a given character into a column index of the state-transition table.

8.7.3.3 GetNextToken()

```
int GetNextToken (
            FILE * file,
            int * charsRead )
```

Gets the next token from the file. Skips leading whitespace. Sets charsRead to the number of characters read, not including whitespace skipped.

Parameters

file	A file to read for tokens.
charsRead	A pointer to an int. The value at charsRead will be overwritten with the number of chars read.

Returns

An int representing a token. See tokens.c.

8.7.3.4 GetNextTokenInBuffer()

```
int GetNextTokenInBuffer (
            char * buffer,
            int * bufIndex,
            int * charsRead )
```

Gets the next token from a buffer. Skips leading whitespace. Sets charsRead to the number of characters read, not including whitespace skipped.

Note

This function primarily exists to test the DFA itself against buffers rather than passing in files.

Parameters

buffer	a character buffer
bufIndex	

8.7.3.5 GetStateString()

```
char * GetStateString (
            int n )
```

8.7.3.6 printCell()

```
void printCell (
            int row,
            int col )
```

A debug function for printing a cell in the DFA.

8.7.3.7 printStateAndChar()

```
void printStateAndChar (
            int row,
            int col )
```

A debug function for printing a state (row name) and column (char name)

8.7.4 Variable Documentation

8.7.4.1 DFA

```
short DFA[71][44][3]
```

The DFA drives the scanner logic. Each of the 71 rows in this state transition table corresponds to a state, or a node on a DFA graph.
Each column corresponds to an edge, with columns 0-25 being 'a' through 'z'. There are also operator characters. The ASCII code of a character is not it's column position and characters must be converted to column numbers before they can index this state transition table. At each cell, there are three values. First is the next state to transition to. Second is the token. Third is a signal to the DFA driver whether to continue reading or not. It says, 'this character is a boundary character for this state'.
The DFA was generated in Excel, and that .xlsx file is available in the /docs folder of this project.

8.8 src/dfa.h File Reference

The DFA and related logic declarations.

```
#include <stdio.h>
```

Functions

- short GetDFAColumn (char c)
- int GetNextToken (FILE ∗file, int ∗charsRead)
- int GetNextTokenInBuffer (char ∗buffer, int ∗bufIndex, int ∗charsRead)
- void printCell (int row, int col)
- void printStateAndChar (int row, int col)

8.8.1 Detailed Description

The DFA and related logic declarations.
The DFA is a 3 dimensional array that maps a given state and character input to a result consisting of the next state, token, and whether reading should continue.
The DFA was created in Excel, and the excel file is available in docs/fullDFA.xlsx.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.8.2 Function Documentation

8.8.2.1 GetDFAColumn()

```
short GetDFAColumn (
            char c )
```

Translates a given character into a column index of the state-transition table.

8.8.2.2 GetNextToken()

```
int GetNextToken (
            FILE * file,
            int * charsRead )
```

Gets the next token from the file. Skips leading whitespace. Sets charsRead to the number of characters read, not including whitespace skipped.

Parameters

file	A file to read for tokens.
charsRead	A pointer to an int. The value at charsRead will be overwritten with the number of chars read.

Returns

An int representing a token. See tokens.c.

8.8.2.3 GetNextTokenInBuffer()

```
int GetNextTokenInBuffer (
            char * buffer,
            int * bufIndex,
            int * charsRead )
```

Gets the next token from a buffer. Skips leading whitespace. Sets charsRead to the number of characters read, not including whitespace skipped.

Note

This function primarily exists to test the DFA itself against buffers rather than passing in files.

Parameters

buffer	a character buffer
bufIndex	

8.8.2.4 printCell()

```
void printCell (
            int row,
            int col )
```

A debug function for printing a cell in the DFA.

8.8.2.5 printStateAndChar()

```
void printStateAndChar (
            int row,
            int col )
```

A debug function for printing a state (row name) and column (char name)

8.9 dfa.h

Go to the documentation of this file.

```
1 #ifndef dfa_h
2 #define dfa_h
3
```

```
4 #include <stdio.h>
21 short GetDFAColumn(char c);
22
29 int GetNextToken(FILE * file, int * charsRead);
30
31
40 int GetNextTokenInBuffer(char * buffer, int * bufIndex, int * charsRead);
41
42
44 void printCell(int row, int col);
45
47 void printStateAndChar(int row, int col);
48
49 #endif
```

8.10 src/file_util.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "file_util.h"
#include <windows.h>
```

Functions

- short fileExists (const char *filename)
- void backupFile (const char *filename)
- int filenameHasExtension (const char *filename)
- char * addExtension (const char *filename, const char *extension)
- char * removeExtension (const char *filename)
- char * generateAbsolutePath (const char *filename)
- short checkIfSamePaths (const char *filename1, const char *filename2)
- char * getString ()

8.10.1 Function Documentation

8.10.1.1 addExtension()

```
char * addExtension (
            const char * filename,
            const char * extension )
```

addExtension modifies the string given by filename by concatenating the string given by extension.
addExtension returns a pointer to a new, concatenated string. This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Parameters

filename	the char array to modify
extension	the char array to append

Authors

thomasterh99, klm127

Date

1/18/2023

Note

Covered by Unit Tests

8.10.1.2 backupFile()

```
void backupFile (
            const char * filename )
```

Renames an existing file, adding the extension '.bak' to the end of it. For example 'outFile.out' will become 'out↩ File.out.bak'.
If the backup file exists already, the new file will have additional '.bak's appended until a name is found that does not collide.

Author

klm127

Date

1/22/2023

Note

Covered by Unit Tests

8.10.1.3 checkIfSamePaths()

```
short checkIfSamePaths (
            const char * filename1,
            const char * filename2 )
```

checkIfSamePaths uses generateAbsolutePath to see if two filenames have the same resulting path.

Precondition

both filenames should be validated to be possible filenames.

Parameters

filename1	the first filename to check.
filename2	the second filename to check.

Returns

1 if they are the same path, 0 otherwise.

Author

karl

Date

2/1/2023

8.10.1.4 fileExists()

```
short fileExists (
            const char * filename )
```

fileExists checks whether a file with name filename exists.

Parameters

filename	: the filename to check.

Returns

short:

- 1 if the file exists
- 0 if it does not.
- -1 if file cant exist

Authors

klm127

Date

1/19/2023

Note

Covered by Unit Tests

8.10.1.5 filenameHasExtension()

```
int filenameHasExtension (
            const char * filename )
```

filenameHasExtension checks whether a filename has an extension. It validates that a string would be a valid path but with one additional condition: it must have a period in the file name portion of the path followed by at least one character.

Parameters

filename	the string to check

Returns

int:

- the index of the `.` character in the string if it exists. otherwise, one of the negative `FILE_EXTENSION_PARSE` enums indicating why the filename is invalid;
  - (-1) means there was no period.
  - (-2) means it ended in a period.
  - (-3) means it is only a period.
  - (-4) means it ends in a slash and is a directory.

Author

klm127

Date

1/19/2023

Note

Covered by Unit Tests

8.10.1.6 generateAbsolutePath()

```
char * generateAbsolutePath (
            const char * filename )
```

generateAbsolutePath uses a fileapi.h call to generate the absolute path for a given filename.

Precondition

filename has already been validated to have an extension

Parameters

filename	the filename to create an absolute path for

Returns

a malloced string for a full path name

Warning

ensure the returned string is freed when you are done to avoid memory leaks

Authors

karl, anthony, thomas

Date

2/1/2023

8.10.1.7 getString()

```
char * getString ( )
```

getString scans a string character by character until recieving a null termination character or a new line

Returns

a pointer to a new character array given by the user with a size of the number of characters + 4 for the possible extension This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Author

thomaserh99

Date

1/23/2023

Note

Covered by Unit Tests

8.10.1.8 removeExtension()

```
char * removeExtension (
            const char * filename )
```

removeExtension modifices the string given in parameters by copying the characters of the string up to the index of the last period.

Precondition

filename has been validated to have a correct extension (not leading with a '.', not ending with a '.')

Parameters

filename	the filename char* to remove the extension from.

Returns

a pointer to a new, extensionless string.

Warning

This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Authors

thomasterh99, klm127

Date

1/22/2023

Note

Covered by Unit Tests

8.11 src/file_util.h File Reference

Functions to assist with file operations.

```
#include <stdbool.h>
#include <stdio.h>
```

Enumerations

- enum FILE_EXISTS_ENUM { FILE_CANT_EXIST = -1 , FILE_EXISTS = 1 , FILE_DOES_NOT_EXIST = 0 }
- enum FILENAME_EXTENSION_PARSE { FILENAME_HAS_NO_PERIOD = -1 , FILENAME_ENDS_IN_PERIOD = -2 , FILENAME_IS_ONLY_PERIOD = -3 , FILENAME_IS_DIRECTORY = -4 }

Functions

- void backupFile (const char *filename)
- short fileExists (const char *filename)
- int filenameHasExtension (const char *filename)
- char * addExtension (const char *filename, const char *extension)
- char * removeExtension (const char *filename)
- char * generateAbsolutePath (const char *filename)
- short checkIfSamePaths (const char *filename1, const char *filename2)
- char * getString ()

8.11.1 Detailed Description

Functions to assist with file operations.

Authors

Karl Miller, Tom Terhune, Anthony Stepich

8.11.2 Enumeration Type Documentation

8.11.2.1 FILE_EXISTS_ENUM

`enum FILE_EXISTS_ENUM`
Alias for true false, 1, 0

Enumerator

FILE_CANT_EXIST	
FILE_EXISTS	
FILE_DOES_NOT_EXIST	

8.11.2.2 FILENAME_EXTENSION_PARSE

`enum FILENAME_EXTENSION_PARSE`
The enum FILENAME_EXTENSION_PARSE describes possible return values from filenameHasExtension which indicate different ways which a filename may be invalid.

Enumerator

FILENAME_HAS_NO_PERIOD	
FILENAME_ENDS_IN_PERIOD	
FILENAME_IS_ONLY_PERIOD	
FILENAME_IS_DIRECTORY	

8.11.3 Function Documentation

8.11.3.1 addExtension()

```
char * addExtension (
            const char * filename,
            const char * extension )
```

addExtension modifies the string given by filename by concatenating the string given by extension.
addExtension returns a pointer to a new, concatenated string. This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Parameters

filename	the char array to modify
extension	the char array to append

Authors

thomasterh99, klm127

Date

1/18/2023

Note

Covered by Unit Tests

8.11.3.2 backupFile()

```
void backupFile (
            const char * filename )
```

Renames an existing file, adding the extension '.bak' to the end of it. For example 'outFile.out' will become 'out↩ File.out.bak'.
If the backup file exists already, the new file will have additional '.bak's appended until a name is found that does not collide.

Author

klm127

Date

1/22/2023

Note

Covered by Unit Tests

8.11.3.3 checkIfSamePaths()

```
short checkIfSamePaths (
            const char * filename1,
            const char * filename2 )
```

checkIfSamePaths uses generateAbsolutePath to see if two filenames have the same resulting path.

Precondition

both filenames should be validated to be possible filenames.

Parameters

filename1	the first filename to check.
filename2	the second filename to check.

Returns

1 if they are the same path, 0 otherwise.

Author

karl

Date

2/1/2023

8.11.3.4 fileExists()

```
short fileExists (
            const char * filename )
```

fileExists checks whether a file with name filename exists.

Parameters

filename	: the filename to check.

Returns

short:

- 1 if the file exists
- 0 if it does not.
- -1 if file cant exist

Authors

klm127

Date

1/19/2023

Note

Covered by Unit Tests

8.11.3.5 filenameHasExtension()

```
int filenameHasExtension (
            const char * filename )
```

filenameHasExtension checks whether a filename has an extension. It validates that a string would be a valid path but with one additional condition: it must have a period in the file name portion of the path followed by at least one character.

Parameters

filename	the string to check

Returns

int:

- the index of the `.` character in the string if it exists. otherwise, one of the negative `FILE_EXTENSION_PARSE` enums indicating why the filename is invalid;
  - (-1) means there was no period.
  - (-2) means it ended in a period.
  - (-3) means it is only a period.
  - (-4) means it ends in a slash and is a directory.

Author

klm127

Date

1/19/2023

Note

Covered by Unit Tests

8.11.3.6 generateAbsolutePath()

```
char * generateAbsolutePath (
            const char * filename )
```

generateAbsolutePath uses a fileapi.h call to generate the absolute path for a given filename.

Precondition

filename has already been validated to have an extension

Parameters

filename	the filename to create an absolute path for

Returns

a malloced string for a full path name

Warning

ensure the returned string is freed when you are done to avoid memory leaks

Authors

karl, anthony, thomas

Date

2/1/2023

8.11.3.7 getString()

```
char * getString ( )
```

getString scans a string character by character until recieving a null termination character or a new line

Returns

a pointer to a new character array given by the user with a size of the number of characters + 4 for the possible extension This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Author

thomaserh99

Date

1/23/2023

Note

Covered by Unit Tests

8.11.3.8 removeExtension()

```
char * removeExtension (
            const char * filename )
```

removeExtension modifices the string given in parameters by copying the characters of the string up to the index of the last period.

Precondition

filename has been validated to have a correct extension (not leading with a '.', not ending with a '.')

Parameters

filename	the filename char* to remove the extension from.

Returns

a pointer to a new, extensionless string.

Warning

This string is allocated with `malloc`. When you are done with it, the memory should be cleared with `free` to avoid memory leaks.

Authors

thomasterh99, klm127

Date

1/22/2023

Note

Covered by Unit Tests

8.12 file_util.h

Go to the documentation of this file.

```
1 #ifndef file_util_h
2 #define file_util_h
9 #include <stdbool.h>
10 #include <stdio.h>
11
12 /*
13 ----------------
14 file operations
15 ----------------
16 */
17 #pragma region fileops
18
28 void backupFile(const char *filename);
29
31 enum FILE_EXISTS_ENUM
32 {
33     FILE_CANT_EXIST = -1,
34     FILE_EXISTS = 1,
35     FILE_DOES_NOT_EXIST = 0
36 };
51 short fileExists(const char *filename);
52
53 #pragma endregion fileops
54
55 /*
56 -------------------
57 filename functions
58 -------------------
59 */
60 #pragma region filenames
61
65 enum FILENAME_EXTENSION_PARSE
66 {
67     FILENAME_HAS_NO_PERIOD = -1,
68     FILENAME_ENDS_IN_PERIOD = -2,
69     FILENAME_IS_ONLY_PERIOD = -3,
70     FILENAME_IS_DIRECTORY = -4
71 };
72
91 int filenameHasExtension(const char *filename);
92
105 char *addExtension(const char *filename, const char *extension);
106
123 char *removeExtension(const char *filename);
124
```

```
136 char *generateAbsolutePath(const char *filename);
137
150 short checkIfSamePaths(const char *filename1, const char *filename2);
151
152 #pragma endregion filenames
153
154 /*
155 ---------------------------
156 prompt assistance functions
157 ---------------------------
158 */
159 #pragma region prompts
160
172 char *getString();
173
174 #pragma endregion prompts
175
176 #endif
```

8.13 src/main.c File Reference

Program entry point.

```
#include "file_util.h"
#include "compfiles.h"
#include "scan.h"
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
```

Functions

- void Init ()
- void Execute (int argc, char *argv[])
- void DeInit ()
- int main (int argc, char *argv[])

8.13.1 Detailed Description

Program entry point.

Authors

Anthony Stepich

Tom Terhune

Karl Miller

8.13.2 Program 1 - fileopen

8.13.2.1 Group 3

8.13.2.1.1 CSC 460 - Language Translation

8.13.3 Function Documentation

8.13.3.1 DeInit()

```
void DeInit ( )
```

8.13.3.2 Execute()

```
void Execute (
            int argc,
            char * argv[] )
```

8.13.3.3 Init()

```
void Init ( )
```

Start the program by initializing the needed modules in the correct order.

8.13.3.4 main()

```
int main (
            int argc,
            char * argv[] )
```

Program entry point.

8.14 src/scan.c File Reference

Scanner struct and 'methods' definitions.

```
#include "dfa.h"
#include "tokens.h"
#include "scan.h"
```

Enumerations

- enum LHEAD_RESULT { LH_CLEAR , LH_NLINE , LH_EOF , LH_COMMENT }

Functions

- void Scanner_Init ()
- void Scanner_DeInit ()
- short Scanner_Lookahead ()
- void Scanner_AdvanceLine ()
- int Scanner_SkipWhitespace ()
- void Scanner_ScanAndPrint (FILE *input, FILE *output, FILE *listing, FILE *temp)
- void Scanner_PrintLine ()
- void Scanner_BackprintIdentifier (int nchars)
- void Scanner_PrintTokenFront (int token)
- void Scanner_PrintErrorListing ()
- void Scanner_PrintErrorSummary ()

Variables

- struct Scanner scanner

8.14.1 Detailed Description

Scanner struct and 'methods' definitions.
Scanner is responsible for tokenizing an input file. It uses the dfa defined in dfa.c to do so. It prints lines an errors to a listing file and token results to an output file.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.14.2 Enumeration Type Documentation

8.14.2.1 LHEAD_RESULT

```
enum LHEAD_RESULT
```

Enumerator

LH_CLEAR	
LH_NLINE	
LH_EOF	
LH_COMMENT	

8.14.3 Function Documentation

8.14.3.1 Scanner_AdvanceLine()

```
void Scanner_AdvanceLine ( )
```

Advances the file pointer until the start of the next line. Increments the line-number counter in scanner.

8.14.3.2 Scanner_BackprintIdentifier()

```
void Scanner_BackprintIdentifier (
            int nchars )
```

Moves the file pointer of scanner.in back nchars and prints that many chars to the output file and possibly the console. Used for printing the actual text of a token.

Parameters

nchars	The number of characters to backprint.

8.14.3.3 Scanner_DeInit()

```
void Scanner_DeInit ( )
```

De-initializes scanner values, setting file pointers to NULL (but not closing files.)

8.14.3.4 Scanner_Init()

```
void Scanner_Init ( )
```

Initializes scanner values to zero.

8.14.3.5 Scanner_Lookahead()

```
short Scanner_Lookahead ( )
```

Looks ahead to determine if there are any more tokens on the line, or if there is a comment at the end of the line. Resets the fileposition after looking ahead.

Returns

0 = Clear to Scan, 1 = Newline next, 2 = EOF next, 3 = Comment next,

8.14.3.6 Scanner_PrintErrorListing()

```
void Scanner_PrintErrorListing ( )
```

Prints an error message to the listing file and possibly the console. Example: \nError. & not recognized.

8.14.3.7 Scanner_PrintErrorSummary()

```
void Scanner_PrintErrorSummary ( )
```

Prints the total error count to the listing file and possibly the console.

8.14.3.8 Scanner_PrintLine()

```
void Scanner_PrintLine ( )
```

Prints a line with a line number to the listing file. Will print newlines but will not print EOFs. Resets the file pointer to its original position after printing the line.

8.14.3.9 Scanner_PrintTokenFront()

```
void Scanner_PrintTokenFront (
            int token )
```

8.14.3.10 Scanner_ScanAndPrint()

```
void Scanner_ScanAndPrint (
            FILE * input,
            FILE * listing,
            FILE * output,
            FILE * temp )
```

Scans a file for tokens and prints detailed information to the listing and output files.

Parameters

input	An input file pointer, already opened for reading.
listing	An listing file pointer, already opened for writing.
output	An output file pointer, already opened for writing.
temp	An temp file pointer, already opened for writing.

8.14.3.11 Scanner_SkipWhitespace()

```
int Scanner_SkipWhitespace ( )
```

Advances the file pointer until a nonwhitespace character (not space or tab) and returns the number of characters skipped.
While the DFA can skip whitespace independently, using this method allows tracking the number of characters that were skipped to maintain an accurate column number.

Returns

The number of whitespace characters skipped.

8.14.4 Variable Documentation

8.14.4.1 scanner

```
struct Scanner scanner
```

8.15 src/scan.h File Reference

Scanner struct and 'methods' declarations.

```
#include <stdio.h>
```

Data Structures

- struct Scanner

Macros

- #define SCANNER_PRINTS_LINES_TO_CONSOLE 1
- #define SCANNER_PRINTS_TOKENS_TO_CONSOLE 1

Functions

- void Scanner_Init ()
- void Scanner_DeInit ()
- void Scanner_ScanAndPrint (FILE *input, FILE *listing, FILE *output, FILE *temp)
- short Scanner_Lookahead ()
- void Scanner_AdvanceLine ()
- int Scanner_SkipWhitespace ()
- void Scanner_PrintLine ()
- void Scanner_BackprintIdentifier (int nchars)
- void Scanner_PrintTokenFront ()
- void Scanner_PrintErrorListing ()
- void Scanner_PrintErrorSummary ()

8.15.1 Detailed Description

Scanner struct and 'methods' declarations.
Scanner is responsible for tokenizing an input file. It uses the dfa defined in dfa.c to do so. It prints lines an errors to a listing file and token results to an output file.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.15.2 Macro Definition Documentation

8.15.2.1 SCANNER_PRINTS_LINES_TO_CONSOLE

```
#define SCANNER_PRINTS_LINES_TO_CONSOLE 1
```

8.15.2.2 SCANNER_PRINTS_TOKENS_TO_CONSOLE

```
#define SCANNER_PRINTS_TOKENS_TO_CONSOLE 1
```

8.15.3 Function Documentation

8.15.3.1 Scanner_AdvanceLine()

```
void Scanner_AdvanceLine ( )
```

Advances the file pointer until the start of the next line. Increments the line-number counter in scanner.

8.15.3.2 Scanner_BackprintIdentifier()

```
void Scanner_BackprintIdentifier (
            int nchars )
```

Moves the file pointer of scanner.in back nchars and prints that many chars to the output file and possibly the console. Used for printing the actual text of a token.

Parameters

nchars	The number of characters to backprint.

8.15.3.3 Scanner_DeInit()

```
void Scanner_DeInit ( )
```

De-initializes scanner values, setting file pointers to NULL (but not closing files.)

8.15.3.4 Scanner_Init()

```
void Scanner_Init ( )
```

Initializes scanner values to zero.

8.15.3.5 Scanner_Lookahead()

```
short Scanner_Lookahead ( )
```

Looks ahead to determine if there are any more tokens on the line, or if there is a comment at the end of the line. Resets the fileposition after looking ahead.

Returns

0 = Clear to Scan, 1 = Newline next, 2 = EOF next, 3 = Comment next,

8.15.3.6 Scanner_PrintErrorListing()

```
void Scanner_PrintErrorListing ( )
```

Prints an error message to the listing file and possibly the console. Example: \nError. & not recognized.

8.15.3.7 Scanner_PrintErrorSummary()

```
void Scanner_PrintErrorSummary ( )
```

Prints the total error count to the listing file and possibly the console.

8.15.3.8 Scanner_PrintLine()

```
void Scanner_PrintLine ( )
```

Prints a line with a line number to the listing file. Will print newlines but will not print EOFs. Resets the file pointer to its original position after printing the line.

8.15.3.9 Scanner_PrintTokenFront()

```
void Scanner_PrintTokenFront ( )
```

Prints the token output to the output file and possibly the console. Example: \ntoken number: 0 token type: BEGIN actual token: After calling, Scanner_BackprintIdentifier should be called to print the actual token.

8.15.3.10 Scanner_ScanAndPrint()

```
void Scanner_ScanAndPrint (
            FILE * input,
            FILE * listing,
            FILE * output,
            FILE * temp )
```

Scans a file for tokens and prints detailed information to the listing and output files.

Parameters

input	An input file pointer, already opened for reading.
listing	An listing file pointer, already opened for writing.
output	An output file pointer, already opened for writing.
temp	An temp file pointer, already opened for writing.

8.15.3.11 Scanner_SkipWhitespace()

```
int Scanner_SkipWhitespace ( )
```

Advances the file pointer until a nonwhitespace character (not space or tab) and returns the number of characters skipped.
While the DFA can skip whitespace independently, using this method allows tracking the number of characters that were skipped to maintain an accurate column number.

Returns

The number of whitespace characters skipped.

8.16 scan.h

Go to the documentation of this file.

```
1 #ifndef scan_h
2 #define scan_h
3
4 #include <stdio.h>
16 /*
17 --------------------
18 Flags
19 --------------------
20 */
21
22 #ifndef SCANNER_PRINTS_LINES_TO_CONSOLE
23 #define SCANNER_PRINTS_LINES_TO_CONSOLE 1
24 #endif
```

```
25
26 #ifndef SCANNER_PRINTS_TOKENS_TO_CONSOLE
27 #define SCANNER_PRINTS_TOKENS_TO_CONSOLE 1
28 #endif
29
30 /*
31 ------------------
32 Scanner lifecycle
33 ------------------
34 */
35 #pragma region lifecycle
39 struct Scanner {
40     /* The line number being scanned.  */
41     int line_no;
42     /* The column number.  */
43     int col_no;
44     /* The error count.  */
45     int errors;
46     /* File pointers.  */
47     FILE * in;
48     FILE * out;
49     FILE * temp;
50     FILE * listing;
51 };
52
54 void Scanner_Init();
56 void Scanner_DeInit();
57
58 #pragma endregion lifecycle
59
60 /*
61 ------------------
62 Scanning methods
63 ------------------
64 */
65 #pragma region scanning
73 void Scanner_ScanAndPrint(FILE *input, FILE *listing, FILE *output, FILE *temp);
74
83 short Scanner_Lookahead();
84
88 void Scanner_AdvanceLine();
89
97 int Scanner_SkipWhitespace();
98 #pragma endregion scanning
99
100 /*
101 -------------------
102 Printing methods
103 -------------------
104 */
105 #pragma region printing
106
110 void Scanner_PrintLine();
111
116 void Scanner_BackprintIdentifier(int nchars);
117
121 void Scanner_PrintTokenFront();
122
126 void Scanner_PrintErrorListing();
127
131 void Scanner_PrintErrorSummary();
132
133 #pragma endregion printing
134
135
136 #endif
```

8.17 src/tokens.c File Reference

Token map and related functions.

```
#include "tokens.h"
#include <string.h>
#include <stdio.h>
```

Functions

- const char * Token_GetName (int id)
- struct TokenCatch * Token_Catch (short tokenType, char *raw_text_found, int line_found_at, int col_found↩

_at)
- char * Token_GetOpRaw (short tokenType)
- struct TokenCatch * Token_CatchOp (short tokenType, int line_found_at, int col_found_at)
- struct TokenCatch * Token_CatchError (char badChar, int line_found_at, int col_found_at)
- void Token_Destroy (struct TokenCatch *token)

Variables

- const char * tokensMap [ ]

8.17.1 Detailed Description

Token map and related functions.
The tokensMap maps a given token to a constant string, which is used by Token_GetName() to get the name of a token. The index of a token string in the tokensMap is the same as it's enumerated value. E.G, BEGIN is value 0 and "BEGIN" is at position 0 in the tokensMap array.
This file also contains definitions for TokenCatch methods, which are no longer used. In an earlier version of the program, a TokenCatch wrapped a given token with related data and was memory-allocated. The current version does not use TokenCatch, but it is retained here in case we need it for future parsing features.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.17.2 Function Documentation

8.17.2.1 Token_Catch()

```
struct TokenCatch * Token_Catch (
            short tokenType,
            char * raw_text_found,
            int line_found_at,
            int col_found_at )
```

8.17.2.2 Note: TokenCatch is no longer used. It was used in an earlier version of this program. It may be revived in the future depending on the needs of the parser.

8.17.2.3 Token_CatchError()

```
struct TokenCatch * Token_CatchError (
            char badChar,
            int line_found_at,
            int col_found_at )
```

Token_CatchError is called when an error is found. Whatever character is passed in will become the 'raw' member of a TokenCatch.

Parameters

tokenType	A type in enum TOKEN
raw_text_found	A char pointer to the raw text that caused this token to be identified as such.
line_found_at	The line in the file the token was found.
col_found_at	The column at which the token was found.

Returns

A pointer to a malloced TokenCatch encapsulating the parameter data.

8.17.2.4 Token_CatchOp()

```
struct TokenCatch * Token_CatchOp (
            short tokenType,
            int line_found_at,
            int col_found_at )
```

Token_Catch_Op is called when an op is found. It still produces a TokenCatch but it infers the text that was found based on the token type rather than needing the raw text, since there is not variation in how the operators can be written.

Parameters

tokenType	A type in enum TOKEN
raw_text_found	A char pointer to the raw text that caused this token to be identified as such.
line_found_at	The line in the file the token was found.
col_found_at	The column at which the token was found.

Returns

A new TokenCatch encapsulating the parameter data.

8.17.2.5 Token_Destroy()

```
void Token_Destroy (
            struct TokenCatch * token )
```

Token Destroy deallocates a token by first freeing the internal 'raw' string, then deallocating the token itself.

Parameters

token	A token to deallocate.

8.17.2.6 Token_GetName()

```
const char * Token_GetName (
            int id )
```

Token_GetName gets a character string representing a token.

Parameters

id	The token ENUM to retrieve.

Returns

const char* A string from a lookup table, e.g, "BEGIN". If the param is not a valid token, then it returns "NULL".

Author

klm127

Date

2/7/2023

Note

Covered By Unit Tests

8.17.2.7 Token_GetOpRaw()

```
char * Token_GetOpRaw (
            short tokenType )
```

Token_GetOpName gets a malloced string for assignment to raw representing what must have been found for an operator text given an enumerated operarator token. If its not one of the operators, it returns ':', which is the one case when a valid operator character was a syntactic error.

Parameters

tokenType	The operator token enumerated id

Returns

A malloced string containing the operator, e.g. "<=".

8.17.3 Variable Documentation

8.17.3.1 tokensMap

```
const char* tokensMap[]
```

TokensMap maps each token to the corresponding string.

Warning

If you change the order in the enum, you must also change the order in this map!

8.18 src/tokens.h File Reference

Token functions declarations.

```
#include <stdlib.h>
```

Data Structures

- struct TokenCatch

Enumerations

- enum TOKEN {
  BEGIN =0 , END , READ , WRITE ,
  IF , THEN , ELSE , ENDIF ,
  WHILE , ENDWHILE , ID , INTLITERAL ,
  FALSEOP , TRUEOP , NULLOP , LPAREN ,
  RPAREN , SEMICOLON , COMMA , ASSIGNOP ,
  PLUSOP , MINUSOP , MULTOP , DIVOP ,
  NOTOP , LESSOP , LESSEQUALOP , GREATEROP ,
  GREATEREQUALOP , EQUALOP , NOTEQUALOP , SCANEOF ,
  ERROR }

Functions

- const char * Token_GetName (int id)
- struct TokenCatch * Token_Catch (short tokenType, char *raw_text_found, int line_found_at, int col_found_at)
- char * Token_GetOpRaw (short tokenType)
- struct TokenCatch * Token_CatchOp (short tokenType, int line_found_at, int col_found_at)
- struct TokenCatch * Token_CatchError (char badChar, int line_found_at, int col_found_at)
- void Token_Destroy (struct TokenCatch *token)

8.18.1 Detailed Description

Token functions declarations.
The tokensMap maps a given token to a constant string, which is used by Token_GetName() to get the name of a token. The index of a token string in the tokensMap is the same as it's enumerated value. E.G, BEGIN is value 0 and "BEGIN" is at position 0 in the tokensMap array.
This file also contains declarations for TokenCatch methods, which are no longer used. In an earlier version of the program, a TokenCatch wrapped a given token with related data and was memory-allocated. The current version does not use TokenCatch, but it is retained here in case we need it for future parsing features.

Authors

Tom Terhune, Karl Miller, Anthony Stepich

Date

February 2023

8.18.2 Enumeration Type Documentation

8.18.2.1 TOKEN

`enum TOKEN`

Enumerator

BEGIN	
END	
READ	
WRITE	
IF	
THEN	
ELSE	
ENDIF	
WHILE	
ENDWHILE	
ID	
INTLITERAL	
FALSEOP	
TRUEOP	
NULLOP	
LPAREN	
RPAREN	
SEMICOLON	
COMMA	
ASSIGNOP	

Enumerator

PLUSOP	
MINUSOP	
MULTOP	
DIVOP	
NOTOP	
LESSOP	
LESSEQUALOP	
GREATEROP	
GREATEREQUALOP	
EQUALOP	
NOTEQUALOP	
SCANEOF	
ERROR	

8.18.3 Function Documentation

8.18.3.1 Token_Catch()

```
struct TokenCatch * Token_Catch (
            short tokenType,
            char * raw_text_found,
            int line_found_at,
            int col_found_at )
```

Token_Catch is called when an actual token has been found. It produces a TokenCatch struct which wraps the token type with other associated data, such as the raw text that was found and the line it was found at.

Parameters

tokenType	A type in enum TOKEN
raw_text_found	A char pointer to the raw text that caused this token to be identified as such.
line_found_at	The line in the file the token was found.
col_found_at	The column at which the token was found.

Returns

A new TokenCatch encapsulating the parameter data.

8.18.3.2 Note: TokenCatch is no longer used. It was used in an earlier version of this program. It may be revived in the future depending on the needs of the parser.

8.18.3.3 Token_CatchError()

```
struct TokenCatch * Token_CatchError (
            char badChar,
            int line_found_at,
            int col_found_at )
```

Token_CatchError is called when an error is found. Whatever character is passed in will become the 'raw' member of a TokenCatch.

Parameters

tokenType	A type in enum TOKEN
raw_text_found	A char pointer to the raw text that caused this token to be identified as such.
line_found_at	The line in the file the token was found.
col_found_at	The column at which the token was found.

Returns

A pointer to a malloced TokenCatch encapsulating the parameter data.

8.18.3.4 Token_CatchOp()

```
struct TokenCatch * Token_CatchOp (
            short tokenType,
            int line_found_at,
            int col_found_at )
```

Token_Catch_Op is called when an op is found. It still produces a TokenCatch but it infers the text that was found based on the token type rather than needing the raw text, since there is not variation in how the operators can be written.

Parameters

tokenType	A type in enum TOKEN
raw_text_found	A char pointer to the raw text that caused this token to be identified as such.
line_found_at	The line in the file the token was found.
col_found_at	The column at which the token was found.

Returns

A new TokenCatch encapsulating the parameter data.

8.18.3.5 Token_Destroy()

```
void Token_Destroy (
            struct TokenCatch * token )
```

Token Destroy deallocates a token by first freeing the internal 'raw' string, then deallocating the token itself.

Parameters

token	A token to deallocate.

8.18.3.6 Token_GetName()

```
const char * Token_GetName (
            int id )
```

Token_GetName gets a character string representing a token.

Parameters

id	The token ENUM to retrieve.

Returns

const char* A string from a lookup table, e.g, "BEGIN". If the param is not a valid token, then it returns "NULL".

Author

klm127

Date

2/7/2023

Note

Covered By Unit Tests

8.18.3.7 Token_GetOpRaw()

```
char * Token_GetOpRaw (
            short tokenType )
```

Token_GetOpName gets a malloced string for assignment to raw representing what must have been found for an operator text given an enumerated operarator token. If its not one of the operators, it returns ':', which is the one case when a valid operator character was a syntactic error.

Parameters

tokenType	The operator token enumerated id

Returns

A malloced string containing the operator, e.g. "<=".

8.19 tokens.h

Go to the documentation of this file.

```
1 #ifndef tokens_h
2 #define tokens_h
3
4 #include <stdlib.h>
5
19 enum TOKEN {
20     BEGIN=0, END, READ, WRITE, IF, THEN, ELSE, ENDIF, WHILE, ENDWHILE, ID, INTLITERAL, FALSEOP, TRUEOP,
      NULLOP, LPAREN, RPAREN, SEMICOLON, COMMA, ASSIGNOP, PLUSOP, MINUSOP, MULTOP,DIVOP, NOTOP, LESSOP,
      LESSEQUALOP, GREATEROP, GREATEREQUALOP, EQUALOP, NOTEQUALOP, SCANEOF, ERROR
21 };
22
31 const char * Token_GetName(int id);
32
38 #pragma region token_catch
39 struct TokenCatch{
40     /* A type corresponding to the TOKEN enum.  */
41     short token;
42     /* The character that was found.  */
43     char * raw;
44     /* The line number it was found on.  */
45     int line_no;
46     /* The column where it started.  */
47     int col_no;
48
49 };
50
59 struct TokenCatch* Token_Catch(short tokenType, char* raw_text_found, int line_found_at, int
      col_found_at);
60
66 char * Token_GetOpRaw(short tokenType);
67
76 struct TokenCatch* Token_CatchOp(short tokenType, int line_found_at, int col_found_at);
77
```

```
86 struct TokenCatch* Token_CatchError(char badChar, int line_found_at, int col_found_at);
87
92 void Token_Destroy(struct TokenCatch* token);
93
94 #pragma endregion token_catch
95
96 #endif
```

Index