# File and Directory Manipulation Language (FDL)

Rupayan Basu rb3034@columbia.edu
Pranav Bhalla pb2538@columbia.edu
Cara Borenstein cjb2182@columbia.edu
Daniel Garzon dg2796@columbia.edu
Daniel Newman dln2111@columbia.edu

December 20, 2013

# Motivation


Safari File Edit View History Bookmarks Window Help
Mon 2:19 PM

# FDL

- File and Directory Manipulation Language (FDL, pronounced “fiddle”)  provides a simple and intuitive syntax for managing file systems.
- Simple to code, simple to understand

# Language Overview

- “path” datatype allows users to create variables using the relative/absolute paths of files/directories
- “path” has built-in attributes: kind, name, type
- Special operators to copy/move files from one directory to another
- Users can iterate through files/subdirectories in a directory with a unique for loop
- The built-in list data structure allows users to conveniently store and access groups of files/directories.

# Introduction to FDL

The following program copies a file from one specified location to a destination directory:

```
def int main()
        path src = "./sample_dir/sample_file.pdf"
        path dest =  "./test"
        dest <- src
        return 0
end
```

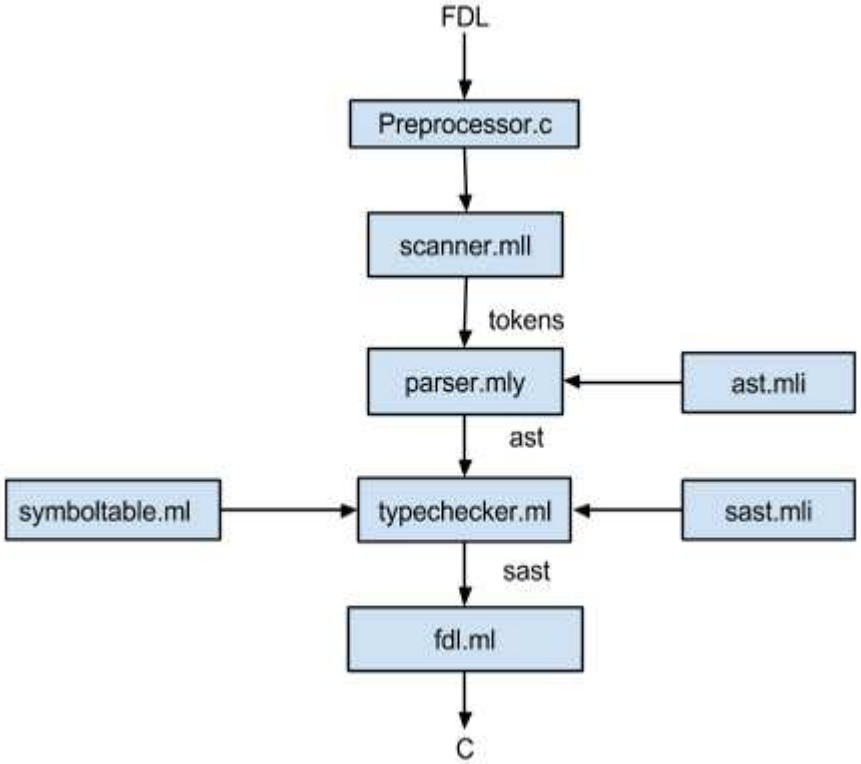
Within the main method, the path variable, ‘src’, is initialized to the file path of a file that we wish to copy. The file path of the directory into which we wish to copy ‘src’ is stored in the path variable ‘dest’. The copy operator, ‘<-’ is then called, and a copy of the src file will now exist in both the src location of the file system, as well as in the dest location.

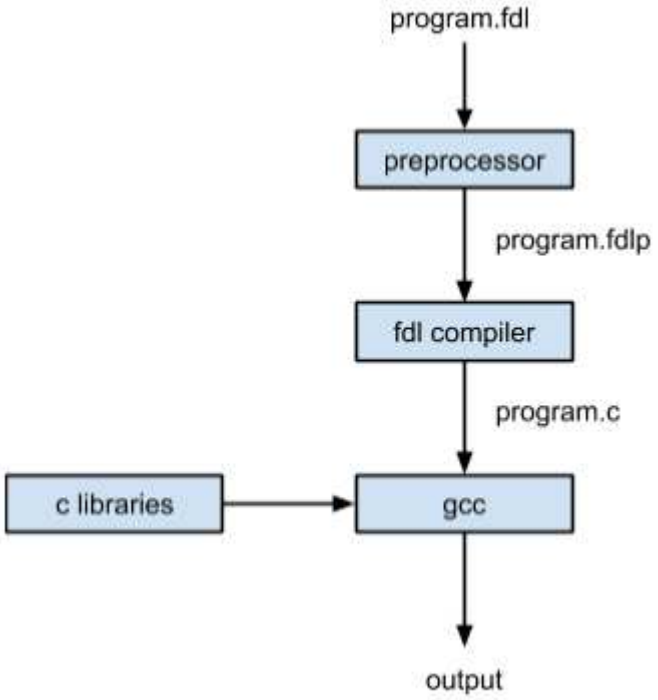
# Example

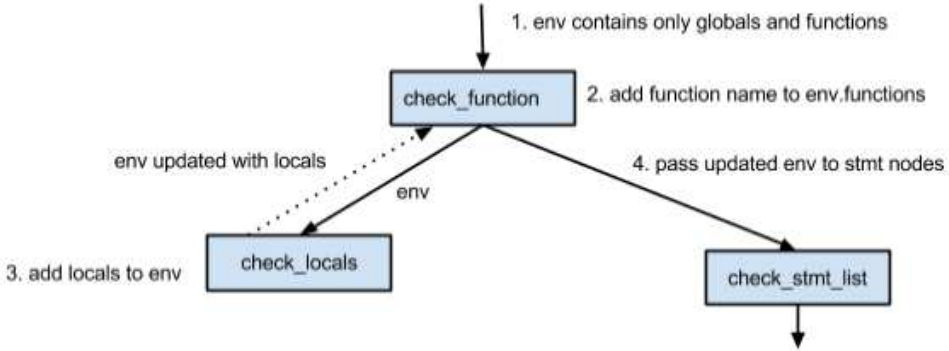
If we wish to do more than copy just one file, we can place the copy operation into a loop that iterates through a full directory, moving all files in the source directory to a target directory, as follows:

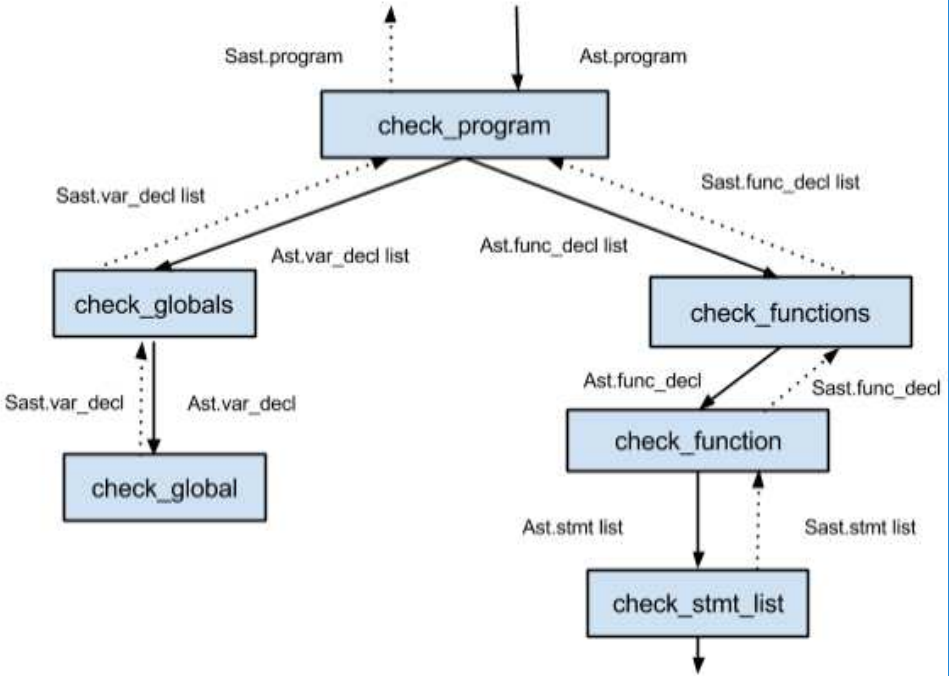
```
def int main()
        path src = "./sample_dir"
        path dest = "./test"
        path f
        for ( f in dir )
    print "file path "
    print f
    if (f.kind == 0) then
      print f
      dest <- f
    end
        end
        return 1
end
```

# Architecture




1. env contains only globals and functions
check_function
2. add function name to env.functions
env updated with locals
env
4. pass updated env to stmt nodes
3. add locals to env
check_locals
check_stmt_list


Sast.program
Ast.program
check_program
Sast.var_decl list
Ast.var_decl list
Ast.func_decl list
Sast.func_decl list
check_globals
check_functions
Sast.var_decl
Ast.var_decl
Ast.func_decl
Sast.func_decl
check_global
check_function
Ast.stmt list
Sast.stmt list
check_stmt_list

# Implementation

- Language features discussed and finalized (Lists, Paths, For loops, No punctuation ;{)
- Basic Skeleton first, Scanner, Parser, AST and Code Generator for “Hello World”
- Divided up features, kept code integrated by using Git branches
- Kept adding test cases as we added features

# Lessons Learned

- Learning Curve
- Typechecking
- TESTING!
  - testing the work of others
  - capturing all possible situations (copy, preprocessor..)
- Version Control (git), Coding with a Team
- Consistency system-system
- Design Decisions

# Thank you