# FileSizes – A Sample Design Document

Project Specification:

1. Description:  
   FileSizes searches a directory tree for all files matching a set of patterns and returns their file sizes.
2. Program inputs:  
   Inputs are provided on the command line and take the form:  
    FileSizes /P . /p \*.h,\*.cpp
3. Program outputs:  
   Output starts with a path, followed by a list of file names with their sizes.
4. Program queries:  
   Before termination the program supports command line queries of the form:  
    L 10 or S 5  
   L 10 specifies that the largest 10 files are displayed in order sorted by size. S 5 specifies that the smallest 5 files are displayed in sorted order.

Design Description:

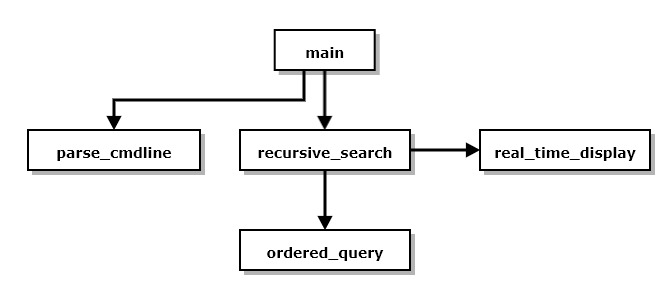
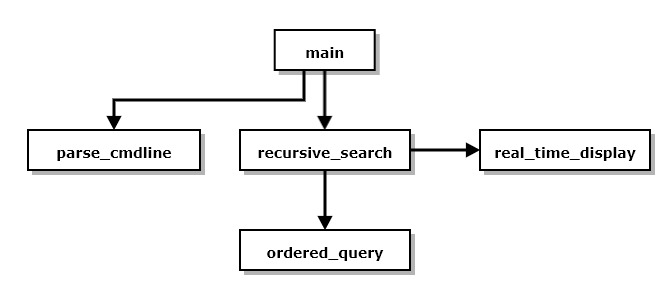
1. Use:  
   Assuming the FileSizes executable is on the user’s PATH, the user navigates to the root directory in a terminal, then issues the FileSizes command. If no command line arguments are supplied, the program finds all files in the directory tree rooted at the current path. Both /P and /p are optional with default being the current working directory and all files.  
     
   The program emits filenames and sizes as they are encountered. The program accepts an optional argument indicating that real-time file display is to be suppressed, e.g., M for mute.  
   When traversal completes the program waits for queries of the form “L n” or “S m” or “Q” for quit.
2. Program tasks:
   1. Accept arguments from the command line and parse them. The path will be stored in a path variable and patterns in a collection of strings.
   2. Display an output header consisting of program name and date, followed by the path and patterns.
   3. Traverse the directory tree finding files and their sizes.
   4. Display, as the directory search progresses, each discovered file with its size.
   5. Request an output query, L m or S n or Q.
   6. Emit the query results and repeat the last step, exiting on Q.
3. Program structure:  
   The diagram below shows a calling tree for FileSizes processing. The individual functions are derived from the program tasks, as follows:

Figure 1. – FileSize Structure

* 1. **Main** is responsible for displaying the output header, calling parse\_cmdline with its command line arguments, and saving the results in a path variable and a vector of pattern strings. Each pattern is an extension identifying a file type to process.
  2. **Recursive\_search** accepts path and patterns and searches the directory tree for matches. When a match is found, it calls real\_time\_display to display the path and file names with their sizes. While navigating through the directory tree it builds a data structure to hold the file, size, path information it encounters, and passes a reference to that to ordered\_query.
  3. **Real\_time\_display** is responsible for displaying discovered path, file, and size in real-time. The first time a new path is encountered it displays that on the console. Then, for each file in that path it displays on a separate line the file name and size.
  4. **Ordered\_query** prompts for a query, and sorts the file collection ordered by file size. Any collisions are resolved by order of the filename, using stable sorts. It displays the results with one line for each entry, holding file size, file name, and path in that order.

1. **Processing:**FileSize processing is concentrated in recursive\_search and ordered\_query. Recursive\_search uses a recursive navigation process: