File and Glob iterator proposal

Tim Zakian and Brad Chamberlain

August 4, 2014

The goal of this proposal is to describe the interface of an iterator that we are planning to implement as part of Chapel's standard libraries, and to solicit comments on it. The intention of the iterator is to yield the names of files and/or directories reachable from a given start directory (defaulting to the current working directory). This iterator is meant to model (to an extent) the capabilities of things like `glob`, `wordexp`, `listdir`, `ls [-R]`, `find`, and the like. The main design questions tend to be a struggle between capability and simplicity.

1 User-facing interface

The proposed user interface for this iterator is as follows:

```
iter glob(pattern   : string = "*",
          startdir  : string = "",
          recursive : bool   = false,
          files     : bool   = true,
          dirs      : bool   = false,
          dotfiles  : bool   = false,
          sorted    : bool   = false,
          expand    : bool   = false
         ) : string
```

where

- `pattern` is a valid `glob` or `wordexp` glob pattern used to filter the filenames and/or directory names yielded by the iterator. Note that while the `pattern` affects what is yielded, we do not expect it to filter which subdirectories are traversed in a recursive crawl. More information on what exactly a valid pattern is can be found on the `man(3)` page for `glob` and `wordexp`.

- `startdir` is the directory from which to start the search. Filenames and directory names yielded by the iterator will be relative to this string and include the string as a prefix. The empty string `""` starts from the current working directory just like `"."` but has the effect of yielding names without `"./"` prefixed to them.

- `recursive` indicates whether the iterator should recursively consider subdirectories or not. There is an open issue about whether the iterator should take a `depth` argument in addition to (or in place of) `recursive` to limit the recursion at a particular subdirectory depth (see Section 2).
- `files` says whether the iterator should yield filenames or not.
- `dirs` says whether the iterator should yield directory names or not.
- `dotfiles` tells whether the iterator should yield dotfiles or not. It also indicates whether the iterator should recursively descend into dot directories.
- `sorted` tells whether the iterator should yield its output in lexicographically sorted order.
- `expand` says whether the iterator should expand environment variables, or run shell commands in `pattern` and `startdir` (a la `wordexp`). For more information on this capability, see the `man(3)` page for `wordexp`.

Additional flags have been discussed as possibilities. For a discussion of these, see Section 2.

Note that while this is an argument-rich iterator, we expect that typical uses will only need to supply a few arguments due to the heavy use of default argument values and Chapel's support for keyword-based argument passing. For example, here are some sample use cases:

- "give me all files in $PWD and its descendents" → `glob(recursive=true)`
- "give me all files and directories in $PWD" → `glob(dirs=true)`
- "give me Python glob" → `glob(pattern=...)`
- "give me C glob" → `glob(pattern=..., sorted=true)`
- "give me C wordexp" → `glob(pattern=..., expand=true)`

We anticipate having a parallel version of this iterator that supports invocation with `forall` loops. It is currently expected that `sorted` will not be a valid option when using the parallel version.

2 Open Issues

The following are open issues that we are wrestling with, and for which we are seeking opinions (though opinions on anything in this proposal are fair game).

- Is there a better name for this iterator than `glob`? On one hand, `glob` is short, sweet, and unambiguous; on the other hand, this iterator is also a lot like `ls [-R]`, and is not particularly glob-like if no argument is provided.

- Should `sorted` be true by default? This is arguably a productivity vs. performance issue (the implementation would have to sort a directory's local contents before yielding them or doing any recursion; though these are arguably going to be small/quick sorts). Ironically, it seems that C defaults to sorting while Python does not.

- Should we add a `depth` argument to limit the depth of the recursion? Or alternatively, replace `recursive` with a `depth` argument? The main downside to the first proposal is that it uses two flags to indicate one logical thing; the main downside to the second proposal is that it makes the very common case of unbounded recursion uglier (e.g., requiring `max(int)` or a sentinel value like `-1` to be passed in rather than simply `recursive=true`).

- Should we support an argument to control dropping the final slash when yielding directory names? (i.e., we expect directory names to be yielded in the form `"subdir/"` by default).

- Should we add a `symlinks` argument to say whether or not we should follow symbolic links? The main problems that we anticipate are (1) it adds another flag, and (2) without care, the iterator could get into a possible infinite loop if a symlink points to a directory that contains the directory we are currently in.

- For recursive mode, should we support an argument indicating a subdirectory name to avoid recursively descending into? The idea behind this being that if we wanted to avoid certain directory names (*e.g.* `.svn` or `.git`), we could set this flag to the string (or strings?) we wanted to skip. Alternatively, we could have a boolean indicating whether or not to skip over commmon SCM-based directory names. On one hand, this feels like a reasonably fiddly option to support; on the other hand, Brad is always happy when tools bother to support it rather than trying to build the equivalent himself with more general features.

- Should we support an argument for enabling warnings or a verbose mode? The idea behind this flag would be to warn you about certain underlying problems. For instance, if we were using `glob` with `expand = true` and `pattern = "$CHPL_HOME/*"` and we had not set `$CHPL_HOME`, we would generate a warning that we were using an undefined environment variable in expansion rather than just expanding it to the empty string.

- Is it reasonable for the parallel version of the iterator to not support sorting? Implementing it would either require tricky synchronization within the iterator, or else gathering all the results and then sorting them (which is expensive and can trivially be expressed outside of the iterator). Note that, in general, Chapel iterators that generate sorted results in their serial form (like `for i in 1..n`) are not necessarily sorted in their `forall` forms.

- Should we support a multi-locale version of the parallel version? If so, what scheduling policy should be used? Should one be able to select between single- and multi-locale implementations via an argument? What should the default be?
- What should the iterator do by default in the event of special files like block special files, character special files, named pipes, or sockets? Would we want to support additional arguments (or a bit vector) to control this behavior?

3 Examples

3.1 Example 1

```
// Print all files in the current directory
// think: ls -F | grep -v /  (though potentially unsorted)
for fl in glob() do
  writeln(fl);
```

3.2 Example 2

```
// Print all files in the current directory in sorted order.
// think: ls -F | grep -v /
for fl in glob(sorted=true) do
  writeln(fl);
```

3.3 Example 3

```
// Print all files and directories in the current directory
// think: ls
for fl in glob(dirs=true) do
  writeln(fl);
```

3.4 Example 4

```
// Print all subdirectories of the current directory.
// Won't recurse (so we only get the children of of the current dir)
// think: ls -F | grep /
for dir in glob(dirs=true, files=false) do
  writeln(dir);
```

3.5 Example 5

```
// Print all subdirectories of the current directory, recursively.
for dir in glob(dirs=true, files=false, recursive=true) do
  writeln(dir);
```

3.6 Example 6

```
// Print all .c files in the current directory
// think: ls *.c
for fl in glob(pattern="*.c") do
  writeln(fl);
```

3.7 Example 7

```
// Print all .c files in this directory or any of its subdirectories
// think: ls -R *.c
for fl in glob(pattern="*.c", recursive=true) do
  writeln(fl);
```

3.8 Example 8

```
// Print all [a-p] directories and files from this directory down
for fl in glob(pattern="[a-p]", dirs=true, recursive=true) do
  writeln(fl);
```

3.9 Example 9

```
// Print all [a-p] directories and files from this directory down.
// Print them in sorted order.
for fl in glob(pattern="[a-p]", dirs=true, recursive=true, sorted=true) do
  writeln(fl);
```

3.10 Example 10

```
// Print all [a-p] directories and files from the $CHPL_HOME directory down.
// Print them in sorted order.
for fl in glob(pattern="[a-p]", startdir="$CHPL_HOME", dirs=true,
               recursive=true, sorted=true, expand=true) do
  writeln(fl);
```

3.11 Example 11 (uses open issue)

```
// Print all [a-p] directories and files from $CHPL_HOME down.
// Ignore all .git directories and files. Print them in sorted order.
for fl in glob(pattern="[a-p]", dirs=true, startdir="$CHPL_HOME"
               recursive=true, sorted=true, expand=true, skipdirs=".git") do
  writeln(fl);
```

3.12 Example 12 (uses open issue)

```
// Print all subdirectories of the current directory
// In this case we'd get all subdirectories up to depth 10
for dir in glob(dirs=true, files=false, depth=10) do
  writeln(dir);
```