C++ include directories

Distribution: **ACCESSIBLE BY THE PUBLIC**

Problem

Authors of C++ code compiled by Bazel frequently want their headers to be included at paths that are not the same as their repository-relative paths. Worse yet, headers in external repositories now need to be included using an external/ prefix.

The current solutions to this problem (`includes=`, `cc_inc_library`) are ad-hoc and not very principled.

Interface

We add two attributes to `cc_library` which control the path under which the includes of said library is accessible exclusively (that is, it won't be accessible at its repository-relative path):

- `strip_include_prefix`: if set, the value of this attribute is stripped from the include path of the headers of this library. All headers must be under this directory. Uplevel references (`..` path segments) are not allowed. If a relative path, it's implicitly prefixed with the package path, if an absolute path, it's not.
- `include_prefix`: if set, header files of this library will be accessible at a path that is the value of this attribute concatenated with the package-relative path of the headers or the path of the headers remaining after removing the prefix specified in `strip_include_prefix` . Uplevel references are not allowed.

For example, given this rule in package foo/bar:

```
cc_library(
  name=”qux”,
  hdrs=[“x/a.h”, “x/y/b.h”],
  include_prefix=$A,
  strip_include_prefix=$B)
```

The headers will be available at the following locations for different values of these attributes:

include_prefix	**strip_include_prefix**	**a.h**	**b.h**
`<unset>`	`<unset>`	`foo/bar/x/a.h`	`foo/bar/x/y/b.h`
`"zyc"`	`<unset>`	`zyc/x/a.h`	`zyc/x/y/b.h`

<unset>	"x"	a.h	y/b.h
"zyc"	"x"	zyc/a.h	zyc/y/b.h
<unset>	"/foo"	bar/x/a.h	bar/x/y/b.h
"zyc"	"/foo"	zyc/bar/x/a.h	zyc/bar/x/y/b.h

When a particular include statement is ambiguous due to the same include file being supplied at the same source path by two different libraries in the transitive closure of a rule, the include will be available through two -I entries on the command line, in unspecified order. (the order being unspecified is a bug, not a feature, but that's the status quo and it is out of scope for this document to fix it)

Implementation

The first iteration will mirror that of `cc_inc_library`: each `cc_library` with at least one of these attributes set will create a symlink tree for its headers (but not the headers in its transitive closure), the root of which will be added as an include path entry for each rule that transitively depends on it.

If this becomes unwieldy (either because the symlink trees yield a confusing output directory structure or because include paths end up being too slow), we can consider creating a symlink tree for each compile action instead. When using a sandbox, this could be implemented for free.

Future work

- Authors of some libraries may want to control whether their code is accessible using includes with quotes or includes with angle brackets. We may end up adding another attribute to cc_library or some other means to control this.
- If we eventually want to give Bazel the ability to install `cc_library` rules, this is a good way to specify where the headers should go under the include root (e.g. `/usr/include`)