C99 Parser Hacker's Guide

rough and incomplete

Matt Wette

1 The Introduction

This is a manual for ...

1.0.0.1 CPP If-Then-Else Logic Block (ITLB) Processing

The parser needs to have a "CPI" (CPP processing info) stack to deal with types (re)defined in multiple branches of a #if...#endif statement chain. If we are in "code" mode then we may be skipping code so need to track when to shift and when not to.

The state is contained in a stack `ppxs` States are

ode¿0 ode=

cskip-done
skip code until `#endif`, passed true

cskip-look
skipping code, but still looking for true at this level

ckeep
keep code

cskip1-pop
skip one token and pop skip-stack

Also, if we want to pass on all the sections of an ITLB to the parser we need to remove typedef names because a typedef may appear multiple times, as in

```
#ifdef SIXTYFOURBIT
typedef short int32_t;
#else
typedef long int32_t;
#endif
```

To achieve this we keep a stack of valid typedefs. On `#if` we push, on `#elif` we shift (i.e., pop, then push) and on `#endif` we pop. ;; The grammar looks like

```
(code
 ("if" cond code "endif")
 ("if" cond code "else" code "endif")
 ("if" cond code elif-list "endif")
 ("if" cond code elif-list "else" code "endif")
 (other))
(elif-list
 ("elif" cond code)
 (elif-list "elif" cond code))
```

1.0.0.2 CPP Macro Expansion

Within C code the lexer will call `expand-cpp-macro-ref`.

And the if/then processing will call `expand-cpp-cond-text`.

1.1 Thoughts

```
Alternatives:
@itemize
@item include: in-place as-tree ignore
@item defdict: keep ignore
@item parsdef: yes no
@item error: eval parse ignore
@item execflo: yes no
@item pragma:
@item expand-id: yes no
@item mode: file (parse cpp lines) ; code (eval cpp lines)
@item eval-but-nodef: fail
@end itemize

Options:
@itemize
@item Option 1 (intended file mode):
@itemize
@item include: parse-tree (but switch exec-cflow?
@item defines: ignore
@item error:
@end itemize

Use a special token for ‘‘could be anything’’ in the inc-helpers:
@code{C99_ANY}.

Note: @code{xtxt} in the lexer is used to denote if text has already been
macro expanded.  This is a bit of a kludge.  Alternatives, are
@enumerate
@item keep as is
@item macro expander returns token list (yuck)
@end enumerate


Controls:
@itemize
@item tddict: enable includes to be skipped, w/ added typedefs
@end itemize
@verbatim
alt: use ftn to return typenames and defs
      '("limits.h" "ayx_t" "ABC=123")
 need include-entry->typenames and include-entry->defs
 defined but not well-defined (i.e., limits.h )
xdef?: enable how idents are expanded
need 64bit typical, 32bit typical
```

`execflo implies need parsdef`

Notes on file mode:

- We need to avoid repeated includes. So need to add `#defined` symbols to the def's list.
- Idea: If file mode and PP-exec-stack (aka `ppxs`) level is non-zero, then add to the define's dict with value `C99_ANY`.

stuff to watch out for:

- typename aliases

```
in incl file: #define SFLOAT static float
in code file: SFLOAT x;
=> in file mode, check #defines for typenames
```

1.2 Todos

I think I have these

`#define #undef #include #if #ifdef #ifndef #else #endif #elif defined #-operator ##-operator #pragma #error`

I still have these to go

```
#line
_Pragma()
```

1.3 The Free Documentation License

The Free Documentation License is included in the Guile Reference Manual. It is included with the NYACC source as COPYING.DOC.