1. Copyright.

2. *c_literal* **Thread.**
It recognizes single quoted c++ literal strings with embeded escape sequences. The single quoted sequence accepts 2 single ' within the embeded character sequence without any conversion — the 2nd ' is not dropped. Escape sequences are just concatenated to the string. No conversion is done at present; just recognize the embeded character sequence. See comments in *angled_string* thread regarding badly terminated strings and escape sequences.

Returned data is the chacacter string without the bounding single quotes.

Errors:
Err_bad_esc and *Err_bad_eos*.

Returned T: *T_literal*

3. Fsm Cc_literal class.

4. Cc_literal constructor directive.

```
⟨ Cc_literal constructor directive 4 ⟩ ≡
  ddd_idx_ = 0;
  ddd_[ddd_idx_] = 0;
```

5. Cc_literal op directive.

```
⟨ Cc_literal op directive 5 ⟩ ≡
  ddd_idx_ = 0;
  ddd_[ddd_idx_] = 0;
```

6. Cc_literal user-declaration directive.

```
⟨ Cc_literal user-declaration directive 6 ⟩ ≡
public: char ddd_[1024 * 32];
  int ddd_idx_;
  void copy_str_into_buffer(std::string * Str);
  void copy_kstr_into_buffer(const char *Str);
```

7. Cc_literal user-implementation directive.

```
⟨ Cc_literal user-implementation directive 7 ⟩ ≡
  void Cc_literal::copy_str_into_buffer(std::string * Str)
  {
    const char *y = Str→c_str( );
    int x(0);
    for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
    ddd_[ddd_idx_] = 0;
  }
```

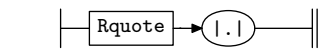
8. *copy_kstr_into_buffer*.

```
⟨ More code 8 ⟩ ≡
  void Cc_literal::copy_kstr_into_buffer(const char *Str)
  {
    const char *y = Str;
    int x(0);
    for ( ; y[x] ≠ 0; ++x, ++ddd_idx_) ddd_[ddd_idx_] = y[x];
    ddd_[ddd_idx_] = 0;
  }
```

9. Cc_literal user-prefix-declaration directive.

```
⟨ Cc_literal user-prefix-declaration directive 9 ⟩ ≡
#include "esc_seq.h"
```

10. *Rc_literal* **rule.**

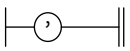


11. Rc_literal op directive.

```
⟨ Rc_literal op directive 11 ⟩ ≡
  Cc_literal * fsm = ( Cc_literal * ) rule_info__.parser__→fsm_tbl__;
  CAbs_lr1_sym * sym = new T_c_literal((const char *) &fsm→ddd_);
  sym→set_rc(*rule_info__.parser__→start_token__, __FILE__, __LINE__);
  RSVP(sym);
```

12. *Rquote* **rule.**

```
Rquote
```



```
⟨ Rquote subrule 1 op directive 12 ⟩ ≡
  Cc_literal * fsm = ( Cc_literal * ) rule_info__.parser__→fsm_tbl__;
loop:
  switch (rule_info__.parser__→current_token( )→enumerated_id__) {
  case T_Enum :: T_raw_lf_: goto overrun;
  case T_Enum :: T_raw_cr_: goto overrun;
  case T_Enum :: T_LR1_eog_: goto overrun;
  case T_Enum :: T_raw_right_quote_: goto rtquote;
  case T_Enum :: T_raw_back_slash_: goto escseq;
  default: goto other;
  }
rtquote:
  {     /* end of literal? */
    rule_info__.parser__→get_next_token( );
    if (rule_info__.parser__→current_token( )→enumerated_id__ ≡ T_Enum :: T_raw_right_quote_) {
      fsm→copy_kstr_into_buffer("'");
      rule_info__.parser__→get_next_token( );
      goto loop;
    }
    return;     /* end of literal */
  }
overrun:
  {
    CAbs_lr1_sym * sym = new Err_bad_eos;
    sym→set_rc(*rule_info__.parser__→start_token__, __FILE__, __LINE__);
    RSVP(sym);
    rule_info__.parser__→set_stop_parse(true);
    return;
  }
  escseq: {     /* what type of escape */
  using namespace NS_esc_seq;
  Parser :: parse_result result = rule_info__.parser__→start_manually_parallel_parsing(ITH_esc_seq.thd_id__);
  if (result ≡ Parser :: erred) {     /* in this case, it will not happen: here for education */
    rule_info__.parser__→set_abort_parse(true);
    return;
  }     /* process returned token */
  Caccept_parse & accept_parm = *rule_info__.parser__→arbitrated_token__;
  CAbs_lr1_sym * rtn_tok = accept_parm.accept_token__;
  int id = rtn_tok→enumerated_id__;
  accept_parm.accept_token__ = 0;
  if (id ≠ T_Enum :: T_T_esc_seq_) {
    RSVP(rtn_tok);
    return;
  }
  T_esc_seq * finc = ( T_esc_seq * ) (rtn_tok);
  fsm→copy_str_into_buffer(finc→esc_data( ));
  rule_info__.parser__→override_current_token(*accept_parm.la_token__, accept_parm.la_token_pos__);
```

```
  delete finc;
  goto loop; } ;
other:
  {
    fsm→copy_kstr_into_buffer(rule_info__.parser__→current_token( )→id__);
    rule_info__.parser__→get_next_token( );
    goto loop;
  }
```

13. First Set Language for O_2^{linker}.

```
/*
 File: c_literal.fsc
 Date and Time: Fri Jan  2 15:33:28 2015
*/
transitive   n
grammar-name "c_literal"
name-space   "NS_c_literal"
thread-name  "TH_c_literal"
monolithic   n
file-name    "c_literal.fsc"
no-of-T      569
list-of-native-first-set-terminals 1
   raw_right_quote
end-list-of-native-first-set-terminals
list-of-transitive-threads 0
end-list-of-transitive-threads
list-of-used-threads 0
end-list-of-used-threads
fsm-comments
"C literal lexer."
```

14. Lr1 State Network.

⇒ State: 1 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
c	Rquote			2	1	1	'			1	2	2	
c	Rc_literal			1	1	1	Rquote |.|			1	3	4	

⇒′ State: 2 state type: ^{r}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rquote			2	1	2				1	0	2	1

⇒^{Rquote} State: 3 state type: ^{s}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rc_literal			1	1	2	|.|			1	4	4	

⇒\|.\| State: 4 state type: ^{r}

	←	rule	→	R#	sr#	Po	←	subrule element	→	Brn	Gto	Red	LA
t	Rc_literal			1	1	3				1	0	4	2

15. Index.

c_literal Grammar

Date: January 2, 2015 at 15:34

File: c_literal.lex Ns: NS_c_literal

Version: 1.0 Debug: false

Grammar Comments: Type: Thread

C literal lexer.

1 element(s) in Lookahead Expression below

eolr