NAME

re2c − convert regular expressions to C/C++

SYNOPSIS

re2c [**-esbvh**] [**-o output**] file

DESCRIPTION

re2c is a preprocessor that generates C-based recognizers from regular expressions. The input to **re2c** consists of C/C++ source interleaved with comments of the form `/*!re2c` ... `*/` which contain scanner specifications. In the output these comments are replaced with code that, when executed, will find the next input token and then execute some user-supplied token-specific code.

For example, given the following code

```
#define NULL            ((char*) 0)
char *scan(char *p){
char *q;
#define YYCTYPE         char
#define YYCURSOR        p
#define YYLIMIT         p
#define YYMARKER      q
#define YYFILL(n)
/*!re2c
        [0-9]+          {return YYCURSOR;}
        [\000-\377]     {return NULL;}
*/
}
```

re2c will generate

```
/* Generated by re2c on Sat Apr 16 11:40:58 1994 */
#line 1 "simple.re"
#define NULL          ((char*) 0)
char *scan(char *p){
char *q;
#define YYCTYPE       char
#define YYCURSOR      p
#define YYLIMIT       p
#define YYMARKER      q
#define YYFILL(n)
{
        YYCTYPE yych;
        unsigned int yyaccept;
        goto yy0;
yy1:    ++YYCURSOR;
yy0:
        if((YYLIMIT - YYCURSOR) < 2) YYFILL(2);
        yych = *YYCURSOR;
        if(yych <= '/') goto yy4;
        if(yych >= ':') goto yy4;
yy2:    yych = *++YYCURSOR;
        goto yy7;
yy3:
#line 10
        {return YYCURSOR;}
yy4:    yych = *++YYCURSOR;
yy5:
#line 11
        {return NULL;}
yy6:    ++YYCURSOR;
        if(YYLIMIT == YYCURSOR) YYFILL(1);
        yych = *YYCURSOR;
yy7:    if(yych <= '/') goto yy3;
        if(yych <= '9') goto yy6;
        goto yy3;
}
#line 12

}
```

OPTIONS

re2c provides the following options:

-e Cross-compile from an ASCII platform to an EBCDIC one.

-s Generate nested `if`s for some `switch`es. Many compilers need this assist to generate better code.

-b Implies **-s**. Use bit vectors as well in the attempt to coax better code out of the compiler. Most useful for specifications with more than a few keywords (e.g. for most programming languages).

-h **-?** Invoke a short help.

-v Show version information.

-o output
Specify the output file.

INTERFACE CODE

Unlike other scanner generators, **re2c** does not generate complete scanners: the user must supply some interface code. In particular, the user must define the following macros:

`YYCTYPE`
Type used to hold an input symbol. Usually `char` or `unsigned char`.

`YYCURSOR`
l-expression of type `*YYCTYPE` that points to the current input symbol. The generated code advances `YYCURSOR` as symbols are matched. On entry, `YYCURSOR` is assumed to point to the first character of the current token. On exit, `YYCURSOR` will point to the first character of the following token.

`YLIMIT`
Expression of type `*YYCTYPE` that marks the end of the buffer (`YLIMIT[-1]` is the last character in the buffer). The generated code repeatedly compares `YYCURSOR` to `YLIMIT` to determine when the buffer needs (re)filling.

`YYMARKER`
l-expression of type `*YYCTYPE`. The generated code saves backtracking information in `YYMARKER`.

`YYFILL(`*n*`)`
The generated code "calls" `YYFILL` when the buffer needs (re)filling: at least *n* additional characters should be provided. `YYFILL` should adjust `YYCURSOR`, `YYLIMIT` and `YYMARKER` as needed. Note that for typical programming languages *n* will be the length of the longest keyword plus one.

SCANNER SPECIFICATIONS

Each scanner specification consists of a set of *rules* and name definitions. Rules consist of a regular expression along with a block of C/C++ code that is to be executed when the associated regular expression is matched. Name definitions are of the form “*name* = *regular expression*`;`”.

SUMMARY OF RE2C REGULAR EXPRESSIONS

`"foo"` the literal string `foo`. ANSI-C escape sequences can be used.

`’foo’` the literal string `foo` (characters [a-zA-Z] treated case-insensitive). ANSI-C escape sequences can be used.

`[xyz]` a "character class"; in this case, the regular expression matches either an ’`x`’, a ’`y`’, or a ’`z`’.

`[abj-oZ]`
a "character class" with a range in it; matches an ’`a`’, a ’`b`’, any letter from ’`j`’ through ’`o`’, or a ’`Z`’.

r`\`*s* match any *r* which isn’t an *s*. *r* and *s* must be regular expressions which can be expressed as character classes.

r`*` zero or more *r*’s, where *r* is any regular expression

r`+` one or more *r*’s

r`?` zero or one *r*’s (that is, "an optional *r*")

name the expansion of the "name" definition (see above)

`(`*r*`)` an *r*; parentheses are used to override precedence (see below) *rs* an *r* followed by an *s* ("concatenation")

r|*s* either an *r* or an *s*

r/*s* an *r* but only if it is followed by an *s*. The s is not part of the matched text. This type of regular expression is called "trailing context".

r{*n*} matches *r* exactly *n* times.

r{*n*,} matches *r* at least *n* times.

r{*n*,*m*}
matches *r* at least *n* but not more than *m* times.

The regular expressions listed above are grouped according to precedence, from highest precedence at the top to lowest at the bottom. Those grouped together have equal precedence.

A LARGER EXAMPLE

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

#define ADDEQ   257
#define ANDAND  258
#define ANDEQ   259
#define ARRAY   260
#define ASM     261
#define AUTO    262
#define BREAK   263
#define CASE    264
#define CHAR    265
#define CONST   266
#define CONTINUE        267
#define DECR    268
#define DEFAULT 269
#define DEREF   270
#define DIVEQ   271
#define DO      272
#define DOUBLE  273
#define ELLIPSIS        274
#define ELSE    275
#define ENUM    276
#define EQL     277
#define EXTERN  278
#define FCON    279
#define FLOAT   280
#define FOR     281
#define FUNCTION        282
#define GEQ     283
#define GOTO    284
#define ICON    285
#define ID      286
#define IF      287
#define INCR    288
#define INT     289
#define LEQ     290
#define LONG    291
#define LSHIFT  292
#define LSHIFTEQ        293
#define MODEQ   294
```

```
#define MULEQ   295
#define NEQ     296
#define OREQ    297
#define OROR    298
#define POINTER 299
#define REGISTER        300
#define RETURN  301
#define RSHIFT  302
#define RSHIFTEQ        303
#define SCON    304
#define SHORT   305
#define SIGNED  306
#define SIZEOF  307
#define STATIC  308
#define STRUCT  309
#define SUBEQ   310
#define SWITCH  311
#define TYPEDEF 312
#define UNION   313
#define UNSIGNED        314
#define VOID    315
#define VOLATILE        316
#define WHILE   317
#define XOREQ   318
#define EOI     319

typedef unsigned int uint;
typedef unsigned char uchar;

#define BSIZE   8192

#define YYCTYPE         uchar
#define YYCURSOR        cursor
#define YYLIMIT         s->lim
#define YYMARKER        s->ptr
#define YYFILL(n)       {cursor = fill(s, cursor);}

#define RET(i)  {s->cur = cursor; return i;}

typedef struct Scanner {
    int                 fd;
    uchar               *bot, *tok, *ptr, *cur, *pos, *lim, *top, *eof;
    uint                line;
} Scanner;

uchar *fill(Scanner *s, uchar *cursor){
    if(!s->eof){
        uint cnt = s->tok - s->bot;
        if(cnt){
            memcpy(s->bot, s->tok, s->lim - s->tok);
            s->tok = s->bot;
            s->ptr -= cnt;
            cursor -= cnt;
            s->pos -= cnt;
            s->lim -= cnt;
        }
        if((s->top - s->lim) < BSIZE){
            uchar *buf = (uchar*)
                malloc(((s->lim - s->bot) + BSIZE)*sizeof(uchar));
            memcpy(buf, s->tok, s->lim - s->tok);
            s->tok = buf;
            s->ptr = &buf[s->ptr - s->bot];
            cursor = &buf[cursor - s->bot];
            s->pos = &buf[s->pos - s->bot];
            s->lim = &buf[s->lim - s->bot];
            s->top = &s->lim[BSIZE];
            free(s->bot);
            s->bot = buf;
        }
        if((cnt = read(s->fd, (char*) s->lim, BSIZE)) != BSIZE){
            s->eof = &s->lim[cnt]; *(s->eof)++ = ’\n’;
        }
        s->lim += cnt;
    }
    s->cur = cursor;
    return cursor;
}

int scan(Scanner *s){
        uchar *cursor = s->cur;
std:
        s->tok = cursor;
/*!re2c
any     = [\000-\377];
O       = [0-7];
D       = [0-9];
L       = [a-zA-Z_];
H       = [a-fA-F0-9];
E       = [Ee] [+-]? D+;
FS      = [fFlL];
IS      = [uUlL]*;
ESC     = [\\] ([abfnrtv?’"\\] | "x" H+ | O+);
*/

/*!re2c
        "/*"                    { goto comment; }

        "auto"                  { RET(AUTO); }
        "break"                 { RET(BREAK); }
        "case"                  { RET(CASE); }
        "char"                  { RET(CHAR); }
        "const"                 { RET(CONST); }
        "continue"              { RET(CONTINUE); }
        "default"               { RET(DEFAULT); }
        "do"                    { RET(DO); }
        "double"                { RET(DOUBLE); }
        "else"                  { RET(ELSE); }
        "enum"                  { RET(ENUM); }
        "extern"                { RET(EXTERN); }
        "float"                 { RET(FLOAT); }
        "for"                   { RET(FOR); }
        "goto"                  { RET(GOTO); }
        "if"                    { RET(IF); }
        "int"                   { RET(INT); }
        "long"                  { RET(LONG); }
        "register"              { RET(REGISTER); }
        "return"                { RET(RETURN); }
        "short"                 { RET(SHORT); }
```

```
"signed"            { RET(SIGNED); }
"sizeof"            { RET(SIZEOF); }
"static"            { RET(STATIC); }
"struct"            { RET(STRUCT); }
"switch"            { RET(SWITCH); }
"typedef"           { RET(TYPEDEF); }
"union"             { RET(UNION); }
"unsigned"          { RET(UNSIGNED); }
"void"              { RET(VOID); }
"volatile"          { RET(VOLATILE); }
"while"             { RET(WHILE); }

L (L|D)*            { RET(ID); }

("0" [xX] H+ IS?) | ("0" D+ IS?) | (D+ IS?) |
([’] (ESC|any\[\n\\’])* [’])
                    { RET(ICON); }

(D+ E FS?) | (D* "." D+ E? FS?) | (D+ "." D* E? FS?)
                    { RET(FCON); }

(["] (ESC|any\[\n\\"])* ["])
                    { RET(SCON); }

"..."               { RET(ELLIPSIS); }
">>="               { RET(RSHIFTEQ); }
"<<="               { RET(LSHIFTEQ); }
"+="                { RET(ADDEQ); }
"-="                { RET(SUBEQ); }
"*="                { RET(MULEQ); }
"/="                { RET(DIVEQ); }
"%="                { RET(MODEQ); }
"&="                { RET(ANDEQ); }
"^="                { RET(XOREQ); }
"|="                { RET(OREQ); }
">>"                { RET(RSHIFT); }
"<<"                { RET(LSHIFT); }
"++"                { RET(INCR); }
"--"                { RET(DECR); }
"->"                { RET(DEREF); }
"&&"                { RET(ANDAND); }
"||"                { RET(OROR); }
"<="                { RET(LEQ); }
">="                { RET(GEQ); }
"=="                { RET(EQL); }
"!="                { RET(NEQ); }
";"                 { RET(’;’); }
"{"                 { RET(’{’); }
"}"                 { RET(’}’); }
","                 { RET(’,’); }
":"                 { RET(’:’); }
"="                 { RET(’=’); }
"("                 { RET(’(’); }
")"                 { RET(’)’); }
"["                 { RET(’[’); }
"]"                 { RET(’]’); }
"."                 { RET(’.’); }
"&"                 { RET(’&’); }
"!"                 { RET(’!’); }
```

```
        "~"                     { RET(’~’); }
        "-"                     { RET(’-’); }
        "+"                     { RET(’+’); }
        "*"                     { RET(’*’); }
        "/"                     { RET(’/’); }
        "%"                     { RET(’%’); }
        "<"                     { RET(’<’); }
        ">"                     { RET(’>’); }
        "^"                     { RET(’^’); }
        "|"                     { RET(’|’); }
        "?"                     { RET(’?’); }


        [ \t\v\f]+              { goto std; }

        "\n"
            {
                if(cursor == s->eof) RET(EOI);
                s->pos = cursor; s->line++;
                goto std;
            }

        any
            {
                printf("unexpected character: %c\n", *s->tok);
                goto std;
            }
*/

comment:
/*!re2c
        "*/"                    { goto std; }
        "\n"
            {
                if(cursor == s->eof) RET(EOI);
                s->tok = s->pos = cursor; s->line++;
                goto comment;
            }
        any                     { goto comment; }
*/
}

main(){
    Scanner in;
    int t;
    memset((char*) &in, 0, sizeof(in));
    in.fd = 0;
    while((t = scan(&in)) != EOI){
/*
        printf("%d\t%.*s\n", t, in.cur - in.tok, in.tok);
        printf("%d\n", t);
*/
    }
    close(in.fd);
}
```

FEATURES

re2c does not provide a default action: the generated code assumes that the input will consist of a sequence of tokens. Typically this can be dealt with by adding a rule such as the one for unexpected characters in the example above.

The user must arrange for a sentinel token to appear at the end of input (and provide a rule for matching it): **re2c** does not provide an `<<EOF>>` expression. If the source is from a null-byte terminated string, a rule matching a null character will suffice. If the source is from a file then the approach taken in the example can be used: pad the input with a newline (or some other character that can't appear within another token); upon recognizing such a character check to see if it is the sentinel and act accordingly.

re2c does not provide start conditions: use a separate scanner specification for each start condition (as illustrated in the above example).

No [ˆx]. Use difference instead.

BUGS

Only fixed length trailing context can be handled.

The maximum value appearing as a parameter *n* to `YYFILL` is not provided to the generated code (this value is needed for constructing the interface code). Note that this value is usually relatively small: for typical programming languages *n* will be the length of the longest keyword plus one.

Difference only works for character sets.

The **re2c** internal algorithms need documentation.

SEE ALSO

flex(1), lex(1).

More information on **re2c** can be found here:
http://sourceforge.net/projects/re2c/

AUTHORS

Peter Bumbulis <peter@csg.uwaterloo.ca>
Brian Young <bayoung@acm.org>
Dan Nuffer <nuffer@users.sourceforge.net>
Marcus Boerger <helly@users.sourceforge.net>
Hartmut Kaiser <hkaiser@users.sourceforge.net>

VERSION INFORMATION

This manpage describes **re2c**, version 0.9.4.