NAME

Yacc − an LALR(1) parser generator

SYNOPSIS

yacc [−BdghilLPrtvVy] [−b *file_prefix* **] [−H** *defines_file* **] [−o** *output_file* **] [−p** *symbol_prefix* **]** *filename*

DESCRIPTION

Yacc reads the grammar specification in the file *filename* and generates an LALR(1) parser for it. The parsers consist of a set of LALR(1) parsing tables and a driver routine written in the C programming language. **Yacc** normally writes the parse tables and the driver routine to the file *y.tab.c*.

The following options are available:

−b *file_prefix*
The **−b** option changes the prefix prepended to the output file names to the string denoted by *file_prefix.* The default prefix is the character *y.*

−B create a backtracking parser (compile-time configuration for **btyacc**).

−d causes the header file **y.tab.h** to be written. It contains #define's for the token identifiers.

−h print a usage message to the standard error.

−H *defines_file*
causes #define's for the token identifiers to be written to the given *defines_file* rather than the **y.tab.h** file used by the **−d** option.

−g The **−g** option causes a graphical description of the generated LALR(1) parser to be written to the file **y.dot** in graphviz format, ready to be processed by **dot**(1).

−i The **−i** option causes a supplementary header file **y.tab.i** to be written. It contains extern declarations and supplementary #define's as needed to map the conventional *yacc* **yy**-prefixed names to whatever the **−p** option may specify. The code file, e.g., **y.tab.c** is modified to #include this file as well as the **y.tab.h** file, enforcing consistent usage of the symbols defined in those files.

The supplementary header file makes it simpler to separate compilation of lex- and yacc-files.

−l If the **−l** option is not specified, **yacc** will insert *#line* directives in the generated code. The *#line* directives let the C compiler relate errors in the generated code to the user's original code. If the **−l** option is specified, **yacc** will not insert the *#line* directives. *#line* directives specified by the user will be retained.

−L enable position processing, e.g., "%locations" (compile-time configuration for **btyacc**).

−o *output_file*
specify the filename for the parser file. If this option is not given, the output filename is the file prefix concatenated with the file suffix, e.g., **y.tab.c**. This overrides the **−b** option.

−p *symbol_prefix*
The **−p** option changes the prefix prepended to yacc-generated symbols to the string denoted by *symbol_prefix.* The default prefix is the string **yy.**

−P create a reentrant parser, e.g., "%pure−parser".

−r The **−r** option causes **yacc** to produce separate files for code and tables. The code file is named *y.code.c*, and the tables file is named *y.tab.c*. The prefix "*y.*" can be overridden using the **−b** option.

−s suppress "**#define**" statements generated for string literals in a "**%token**" statement, to more closely match original **yacc** behavior.

Normally when **yacc** sees a line such as

```
%token OP_ADD "ADD"
```

it notices that the quoted "ADD" is a valid C identifier, and generates a #define not only for OP_ADD, but for ADD as well, e.g.,

```
#define OP_ADD 257
#define ADD 258
```

The original **yacc** does not generate the second “**#define**”. The **–s** option suppresses this “**#define**”.

POSIX (IEEE 1003.1 2004) documents only names and numbers for “**%token**”, though original **yacc** and bison also accept string literals.

–t The **–t** option changes the preprocessor directives generated by **yacc** so that debugging statements will be incorporated in the compiled code.

Yacc sends debugging output to the standard output (compatible with both the original **yacc** and **btyacc**), while **btyacc** writes debugging output to the standard error (like **bison**).

–v The **–v** option causes a human-readable description of the generated parser to be written to the file *y.output.*

–V print the version number to the standard output.

–y **yacc** ignores this option, which bison supports for ostensible POSIX compatibility.

The *filename* parameter is not optional. However, **yacc** accepts a single “–” to read the grammar from the standard input. A double “––” marker denotes the end of options. A single *filename* parameter is expected after a “––” marker.

DIAGNOSTICS

If there are rules that are never reduced, the number of such rules is reported on standard error. If there are any LALR(1) conflicts, the number of conflicts is reported on standard error.

EXTENSIONS

Yacc provides some extensions for compatibility with bison and other implementations of yacc. It accepts several *long options* which have equivalents in yacc. The **%destructor** and **%locations** features are available only if **yacc** has been configured and compiled to support the back-tracking (**btyacc**) functionality. The remaining features are always available:

%code *keyword* { *code* }

Adds the indicated source *code* at a given point in the output file. The optional *keyword* tells **yacc** where to insert the *code*:

top just after the version-definition in the generated code-file.

requires

just after the declaration of public parser variables. If the **–d** option is given, the code is inserted at the beginning of the defines-file.

provides

just after the declaration of private parser variables. If the **–d** option is given, the code is inserted at the end of the defines-file.

If no *keyword* is given, the code is inserted at the beginning of the section of code copied verbatim from the source file. Multiple **%code** directives may be given; **yacc** inserts those into the corresponding code- or defines-file in the order that they appear in the source file.

%debug

This has the same effect as the “–t” command-line option.

%destructor { *code* } *symbol+*

defines code that is invoked when a symbol is automatically discarded during error recovery. This code can be used to reclaim dynamically allocated memory associated with the corresponding semantic value for cases where user actions cannot manage the memory explicitly.

On encountering a parse error, the generated parser discards symbols on the stack and input tokens until it reaches a state that will allow parsing to continue. This error recovery approach results in a memory leak if the **YYSTYPE** value is, or contains, pointers to dynamically allocated memory.

The bracketed *code* is invoked whenever the parser discards one of the symbols. Within *code*, "**$$**" or "**$<***tag***>$**" designates the semantic value associated with the discarded symbol, and "**@$**" designates its location (see **%locations** directive).

A per-symbol destructor is defined by listing a grammar symbol in *symbol*+. A per-type destructor is defined by listing a semantic type tag (e.g., "<some_tag>") in *symbol*+; in this case, the parser will invoke *code* whenever it discards any grammar symbol that has that semantic type tag, unless that symbol has its own per-symbol destructor.

Two categories of default destructor are supported that are invoked when discarding any grammar symbol that has no per-symbol and no per-type destructor:

- the code for "**<*>**" is used for grammar symbols that have an explicitly declared semantic type tag (via "**%type**");
- the code for "**<>**" is used for grammar symbols that have no declared semantic type tag.

%empty
ignored by **yacc**.

%expect *number*
tells **yacc** the expected number of shift/reduce conflicts. That makes it only report the number if it differs.

%expect−rr *number*
tell **yacc** the expected number of reduce/reduce conflicts. That makes it only report the number if it differs. This is (unlike bison) allowable in LALR parsers.

%locations
tells **yacc** to enable management of position information associated with each token, provided by the lexer in the global variable **yylloc**, similar to management of semantic value information provided in **yylval**.

As for semantic values, locations can be referenced within actions using **@$** to refer to the location of the left hand side symbol, and @*N* (*N* an integer) to refer to the location of one of the right hand side symbols. Also as for semantic values, when a rule is matched, a default action is used the compute the location represented by **@$** as the beginning of the first symbol and the end of the last symbol in the right hand side of the rule. This default computation can be overridden by explicit assignment to **@$** in a rule action.

The type of **yylloc** is **YYLTYPE**, which is defined by default as:

```
typedef struct YYLTYPE {
    int first_line;
    int first_column;
    int last_line;
    int last_column;
} YYLTYPE;
```

YYLTYPE can be redefined by the user (**YYLTYPE_IS_DEFINED** must be defined, to inhibit the default) in the declarations section of the specification file. As in bison, the macro **YYLLOC_DEFAULT** is invoked each time a rule is matched to calculate a position for the left hand side of the rule, before the associated action is executed; this macro can be redefined by the user.

This directive adds a **YYLTYPE** parameter to **yyerror**(). If the **%pure−parser** directive is present, a **YYLTYPE** parameter is added to **yylex**() calls.

%lex−param { *argument-declaration* }
By default, the lexer accepts no parameters, e.g., **yylex**(). Use this directive to add parameter declarations for your customized lexer.

%parse−param { *argument-declaration* }
By default, the parser accepts no parameters, e.g., **yyparse()**. Use this directive to add parameter declarations for your customized parser.

%pure−parser
Most variables (other than **yydebug** and **yynerrs**) are allocated on the stack within **yyparse**, making the parser reasonably reentrant.

%token−table
Make the parser's names for tokens available in the **yytname** array. However, **yacc** does not predefine "$end", "$error" or "$undefined" in this array.

PORTABILITY

According to Robert Corbett,

```
Berkeley Yacc is an LALR(1) parser generator.  Berkeley Yacc
has been made as compatible as possible with AT&T Yacc.
Berkeley Yacc can accept any input specification that
conforms to the AT&T Yacc documentation.  Specifications
that take advantage of undocumented features of AT&T Yacc
will probably be rejected.
```

The rationale in

```
http://pubs.opengroup.org/onlinepubs/9699919799/utilities/yacc.html
```

documents some features of AT&T yacc which are no longer required for POSIX compliance.

That said, you may be interested in reusing grammar files with some other implementation which is not strictly compatible with AT&T yacc. For instance, there is bison. Here are a few differences:

- **Yacc** accepts an equals mark preceding the left curly brace of an action (as in the original grammar file **ftp.y**):

```
|   STAT CRLF
    = {
            statcmd();
    }
```

- **Yacc** and bison emit code in different order, and in particular bison makes forward reference to common functions such as yylex, yyparse and yyerror without providing prototypes.
- Bison's support for "%expect" is broken in more than one release. For best results using bison, delete that directive.
- Bison has no equivalent for some of **yacc**'s command-line options, relying on directives embedded in the grammar file.
- Bison's "**−y**" option does not affect bison's lack of support for features of AT&T yacc which were deemed obsolescent.
- **Yacc** accepts multiple parameters with **%lex−param** and **%parse−param** in two forms

```
{type1 name1} {type2 name2} ...
{type1 name1,  type2 name2 ...}
```

  Bison accepts the latter (though undocumented), but depending on the release may generate bad code.
- Like bison, **yacc** will add parameters specified via **%parse−param** to **yyparse**, **yyerror** and (if configured for back-tracking) to the destructor declared using **%destructor**. Bison puts the additional parameters *first* for **yyparse** and **yyerror** but *last* for destructors. **Yacc** matches this behavior.

SEE ALSO

bison(1), **btyacc**(1), **lex**(1), **flex**(1), **yacc**(1)