Next: Glossary, Previous: FAQ, Up: Top [Contents][Index]

Appendix A Bison Symbols

Variable: @$

In an action, the location of the left-hand side of the rule. See Tracking Locations.

Variable: @*n*

Symbol: @*n*

In an action, the location of the *n*-th symbol of the right-hand side of the rule. See Tracking Locations.

In a grammar, the Bison-generated nonterminal symbol for a midrule action with a semantic value. See Midrule Action Translation.

Variable: @*name*

Variable: @[*name*]

In an action, the location of a symbol addressed by *name*. See Tracking Locations.

Symbol: $@*n*

In a grammar, the Bison-generated nonterminal symbol for a midrule action with no semantics value. See Midrule Action Translation.

Variable: $$

In an action, the semantic value of the left-hand side of the rule. See Actions.

Variable: $*n*

In an action, the semantic value of the *n*-th symbol of the right-hand side of the rule. See Actions.

Variable: $*name*

Variable: $[*name*]

In an action, the semantic value of a symbol addressed by *name*. See Actions.

Delimiter: %%

Delimiter used to separate the grammar rule section from the Bison declarations section or the epilogue. See The Overall Layout of a Bison Grammar.

Delimiter: %{*code*%}

All code listed between '`%{`' and '`%}`' is copied verbatim to the parser implementation file. Such code forms the prologue of the grammar file. See Outline of a Bison Grammar.

Directive: %?{*expression*}

Predicate actions. This is a type of action clause that may appear in rules. The expression is evaluated, and if false, causes a syntax error. In GLR parsers during nondeterministic operation, this silently causes an alternative parse to die. During deterministic operation, it is the same as the effect of YYERROR. See Semantic Predicates.

Construct: /* … */

Construct: // …

Comments, as in C/C++.

Delimiter: :

Separates a rule's result from its components. See Syntax of Grammar Rules.

Delimiter: ;

Terminates a rule. See Syntax of Grammar Rules.

Delimiter: |

Separates alternate rules for the same result nonterminal. See Syntax of Grammar Rules.

Directive: <*>

Used to define a default tagged `%destructor` or default tagged `%printer`.

See Freeing Discarded Symbols.

Directive: <>

Used to define a default tagless `%destructor` or default tagless `%printer`.

See Freeing Discarded Symbols.

Symbol: $accept

The predefined nonterminal whose only rule is '`$accept: start $end`', where *start* is the start symbol. See The Start-Symbol. It cannot be used in the grammar.

Directive: %code *{code}*

Directive: %code *qualifier {code}*

Insert *code* verbatim into the output parser source at the default location or at the location specified by *qualifier*. See %code Summary.

Directive: %debug

Equip the parser for debugging. See Decl Summary.

Directive: %define *variable*

Directive: %define *variable value*

Directive: %define *variable {value}*

Directive: %define *variable "value"*

Define a variable to adjust Bison's behavior. See %define Summary.

Directive: %defines

Bison declaration to create a parser header file, which is usually meant for the scanner. See Decl Summary.

Directive: %defines *defines-file*

Same as above, but save in the file *defines-file*. See Decl Summary.

Directive: %destructor

Specify how the parser should reclaim the memory associated to discarded symbols. See Freeing Discarded Symbols.

Directive: %dprec

Bison declaration to assign a precedence to a rule that is used at parse time to resolve reduce/reduce conflicts. See Writing GLR Parsers.

Directive: %empty

Bison declaration to declare make explicit that a rule has an empty right-hand side. See Empty Rules.

Symbol: $end

The predefined token marking the end of the token stream. It cannot be used in the grammar.

Symbol: error

A token name reserved for error recovery. This token may be used in grammar rules so as to allow the Bison parser to recognize an error in the grammar without halting the process. In effect, a sentence containing an error may be recognized as valid. On a syntax error, the token `error` becomes the current lookahead token. Actions corresponding to `error` are then executed, and the lookahead token is reset to the token that originally caused the violation. See Error Recovery.

Directive: %error-verbose

An obsolete directive standing for '`%define parse.error verbose`' (see The Error Reporting Function `yyerror`).

Directive: %file-prefix "*prefix*"

Bison declaration to set the prefix of the output files. See Decl Summary.

Directive: %glr-parser

Bison declaration to produce a GLR parser. See Writing GLR Parsers.

Directive: %initial-action

Run user code before parsing. See Performing Actions before Parsing.

Directive: %language

Specify the programming language for the generated parser. See Decl Summary.

Directive: %left

Bison declaration to assign precedence and left associativity to token(s). See Operator

Precedence.

Directive: %lex-param *{argument-declaration} …*

Bison declaration to specifying additional arguments that `yylex` should accept. See Calling Conventions for Pure Parsers.

Directive: %merge

Bison declaration to assign a merging function to a rule. If there is a reduce/reduce conflict with a rule having the same merging function, the function is applied to the two semantic values to get a single result. See Writing GLR Parsers.

Directive: %name-prefix "*prefix*"

Obsoleted by the `%define` variable `api.prefix` (see Multiple Parsers in the Same Program).

Rename the external symbols (variables and functions) used in the parser so that they start with *prefix* instead of '`yy`'. Contrary to `api.prefix`, do no rename types and macros.

The precise list of symbols renamed in C parsers is `yyparse`, `yylex`, `yyerror`, `yynerrs`, `yylval`, `yychar`, `yydebug`, and (if locations are used) `yylloc`. If you use a push parser, `yypush_parse`, `yypull_parse`, `yypstate`, `yypstate_new` and `yypstate_delete` will also be renamed. For example, if you use '`%name-prefix "c_"`', the names become `c_parse`, `c_lex`, and so on. For C++ parsers, see the `%define api.namespace` documentation in this section.

Directive: %no-lines

Bison declaration to avoid generating `#line` directives in the parser implementation file. See Decl Summary.

Directive: %nonassoc

Bison declaration to assign precedence and nonassociativity to token(s). See Operator Precedence.

Directive: %nterm

Bison declaration to declare nonterminals. See Nonterminal Symbols.

Directive: %output "*file*"

Bison declaration to set the name of the parser implementation file. See Decl Summary.

Directive: %param *{argument-declaration}* …

Bison declaration to specify additional arguments that both `yylex` and `yyparse` should accept. See The Parser Function `yyparse`.

Directive: %parse-param *{argument-declaration}* …

Bison declaration to specify additional arguments that `yyparse` should accept. See The Parser Function `yyparse`.

Directive: %prec

Bison declaration to assign a precedence to a specific rule. See Context-Dependent Precedence.

Directive: %precedence

Bison declaration to assign precedence to token(s), but no associativity See Operator Precedence.

Directive: %pure-parser

Deprecated version of '`%define api.pure`' (see api.pure), for which Bison is more careful to warn about unreasonable usage.

Directive: %require "*version*"

Require version *version* or higher of Bison. See Require a Version of Bison.

Directive: %right

Bison declaration to assign precedence and right associativity to token(s). See Operator Precedence.

Directive: %skeleton

Specify the skeleton to use; usually for development. See Decl Summary.

Directive: %start

Bison declaration to specify the start symbol. See The Start-Symbol.

Directive: %token

Bison declaration to declare token(s) without specifying precedence. See Token Type Names.

Directive: %token-table

Bison declaration to include a token name table in the parser implementation file. See Decl Summary.

Directive: %type

Bison declaration to declare symbol value types. See Nonterminal Symbols.

Symbol: $undefined

The predefined token onto which all undefined values returned by `yylex` are mapped. It cannot be used in the grammar, rather, use `error`.

Directive: %union

Bison declaration to specify several possible data types for semantic values. See The Union Declaration.

Macro: YYABORT

Macro to pretend that an unrecoverable syntax error has occurred, by making `yyparse` return 1 immediately. The error reporting function `yyerror` is not called. See The Parser Function `yyparse`.

For Java parsers, this functionality is invoked using `return YYABORT;` instead.

Macro: YYACCEPT

Macro to pretend that a complete utterance of the language has been read, by making `yyparse` return 0 immediately. See The Parser Function `yyparse`.

For Java parsers, this functionality is invoked using `return YYACCEPT;` instead.

Macro: YYBACKUP

Macro to discard a value from the parser stack and fake a lookahead token. See Special Features for Use in Actions.

Variable: yychar

External integer variable that contains the integer value of the lookahead token. (In a pure parser, it is a local variable within `yyparse`.) Error-recovery rule actions may examine this variable. See Special Features for Use in Actions.

Variable: yyclearin

Macro used in error-recovery rule actions. It clears the previous lookahead token. See Error Recovery.

Macro: YYDEBUG

Macro to define to equip the parser with tracing code. See Tracing Your Parser.

Variable: yydebug

External integer variable set to zero by default. If `yydebug` is given a nonzero value, the parser will output information on input symbols and parser action. See Tracing Your Parser.

Macro: yyerrok

Macro to cause parser to recover immediately to its normal mode after a syntax error. See Error Recovery.

Macro: YYERROR

Cause an immediate syntax error. This statement initiates error recovery just as if the parser itself had detected an error; however, it does not call `yyerror`, and does not print any message. If you want to print an error message, call `yyerror` explicitly before the '`YYERROR;`' statement. See Error Recovery.

For Java parsers, this functionality is invoked using `return YYERROR;` instead.

Function: yyerror

User-supplied function to be called by `yyparse` on error. See The Error Reporting Function `yyerror`.

Macro: YYERROR_VERBOSE

An obsolete macro used in the `yacc.c` skeleton, that you define with `#define` in the prologue to request verbose, specific error message strings when `yyerror` is called. It doesn't matter what definition you use for `YYERROR_VERBOSE`, just whether you define it.

Using '`%define parse.error verbose`' is preferred (see The Error Reporting Function `yyerror`).

Macro: YYFPRINTF

Macro used to output run-time traces. See Enabling Traces.

Macro: YYINITDEPTH

Macro for specifying the initial size of the parser stack. See Memory Management.

Function: yylex

User-supplied lexical analyzer function, called with no arguments to get the next token. See The Lexical Analyzer Function `yylex`.

Variable: yylloc

External variable in which `yylex` should place the line and column numbers associated with a token. (In a pure parser, it is a local variable within `yyparse`, and its address is passed to `yylex`.) You can ignore this variable if you don't use the '`@`' feature in the grammar actions. See Textual Locations of Tokens. In semantic actions, it stores the location of the lookahead token. See Actions and Locations.

Type: YYLTYPE

Data type of `yylloc`; by default, a structure with four members. See Data Types of Locations.

Variable: yylval

External variable in which `yylex` should place the semantic value associated with a token. (In a pure parser, it is a local variable within `yyparse`, and its address is passed to `yylex`.) See Semantic Values of Tokens. In semantic actions, it stores the semantic value of the lookahead token. See Actions.

Macro: YYMAXDEPTH

Macro for specifying the maximum size of the parser stack. See Memory Management.

Variable: yynerrs

Global variable which Bison increments each time it reports a syntax error. (In a pure

parser, it is a local variable within `yyparse`. In a pure push parser, it is a member of `yypstate`.) See The Error Reporting Function `yyerror`.

Function: yyparse

The parser function produced by Bison; call this function to start parsing. See The Parser Function `yyparse`.

Macro: YYPRINT

Macro used to output token semantic values. For `yacc.c` only. Deprecated, use `%printer` instead (see Printing Semantic Values). See The `YYPRINT` Macro.

Function: yypstate_delete

The function to delete a parser instance, produced by Bison in push mode; call this function to delete the memory associated with a parser. See The Parser Delete Function `yypstate_delete`. Does nothing when called with a null pointer.

Function: yypstate_new

The function to create a parser instance, produced by Bison in push mode; call this function to create a new parser. See The Parser Create Function `yypstate_new`.

Function: yypull_parse

The parser function produced by Bison in push mode; call this function to parse the rest of the input stream. See The Pull Parser Function `yypull_parse`.

Function: yypush_parse

The parser function produced by Bison in push mode; call this function to parse a single token. See The Push Parser Function `yypush_parse`.

Macro: YYRECOVERING

The expression `YYRECOVERING ()` yields 1 when the parser is recovering from a syntax error, and 0 otherwise. See Special Features for Use in Actions.

Macro: YYSTACK_USE_ALLOCA

Macro used to control the use of `alloca` when the deterministic parser in C needs to extend its stacks. If defined to 0, the parser will use `malloc` to extend its stacks and

memory exhaustion occurs if `malloc` fails (see Memory Management). If defined to 1, the parser will use `alloca`. Values other than 0 and 1 are reserved for future Bison extensions. If not defined, `YYSTACK_USE_ALLOCA` defaults to 0.

In the all-too-common case where your code may run on a host with a limited stack and with unreliable stack-overflow checking, you should set `YYMAXDEPTH` to a value that cannot possibly result in unchecked stack overflow on any of your target hosts when `alloca` is called. You can inspect the code that Bison generates in order to determine the proper numeric values. This will require some expertise in low-level implementation details.

Type: YYSTYPE

Deprecated in favor of the `%define` variable `api.value.type`. Data type of semantic values; `int` by default. See Data Types of Semantic Values.

Next: Glossary, Previous: FAQ, Up: Top [Contents][Index]