

Yacc (yet another compiler-compiler) [Steve Johnson, 1972]

A parser is a syntax analyzer.

yacc is a parser generator.

Input: a grammatical specification of a language

Output: a parser

Bison is a general-purpose parser generator that converts a grammar description for an LALR(1) context-free grammar into a C program to parse that grammar.

Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change.

Bison was written primarily by Robert Corbett; Richard Stallman made it Yacc-compatible. Wilfred Hansen of Carnegie Mellon University added multi-character string literals and other features.

Stages in using Bison:

1. Write a precise grammar for a language:
2. Add an action to each rule or production of the grammar
3. Write a lexical analyzer to process source program of the language
4. Add a controlling routine to call the parser produced by Bison

In the translation process, a source program P is treated as **a stream of strings** separated by white spaces defined by a language.

Program P is translated into **a stream of tokens** by a lexical analyzer (the `yylex()` function in Bison).

A stream of tokens is recognized or parsed into **syntactic structures (parse trees)** by a syntax analyzer (parser) based on grammatical rules defined in a language (the `yyparse()` function in Bison).

Syntactic structures are translated into **Intermediate code** (I-code, which are instructions of a virtual machine) by an Intermediate Code Generator.

The I-code can be optimized optionally.

Code generators are used to translate I-codes into real machine codes.

Bison source file format:

Bison source file is named with .y extension.

The following is the typical layout of a bison source file:

```
/* the first section includes optional C definitions */
/* the section is delimited by %{ and %} */

%{
.....
#define YYSTYPE double /* data type of bison stack */
.....
%}

/* the second section includes Bison Declarations */
/* token names and types and operator symbols and precedence are defined here */

%token TOKEN_NAME
%left '+' '-'
%left '*' '/'

/*the third section includes grammar rules and actions */
/* the section is delimited by double percent signs */

%%
.....
.....
%%

/* the fourth section includes optional C statements */

/* the fifth section includes controlling routines */

main() { ...; yyparse(); ... }

yylex() { ..... }

other functions
```

Notes:

Nonterminal symbols are identifiers in lower case.

Terminal symbols are identifiers in upper case.

Grammar rules are terminated by semicolons.

Meta characters in rules include : | ;

e.g.:

```
lrs1:   rhs1 {action 1}
        | rhs2 {action 2}
        | rhs3 {action 3}
;
lrs2: rhs1
;
```

Semantic Values:

Each token in a Bison grammar has a **token type**, which is a terminal symbol defined in the grammar and a **semantic value**, which is the meaning (value) of the token.

Token types must be declared in the Bison declaration section by using one of

```
%token name
%left
%right
%nonassoc
```

Semantic values of tokens must be stored into the global variable `yylval`. This is done in the `yylex()` function.

If multiple data types are used, we need to use the dot-operator to select a member from the `%union` declaration declared in the Bison declaration section. E.g.: given the following declaration

```
%union {
    int intval;
    double val;
    symrec *tpr;
}
```

Then, in `yylex()`, for an INT token with integer value, we have

```
yylval.intval = whatever_value;
```