

**NAME**

bison++ – generate a parser in c or c++.

**SYNOPSIS**

**bison++** [**-dltvyVu**] [**-b** *file-prefix*] [**-p** *name-prefix*] [**-o** *outfile*] [**-h** *headerfile*] [**-S** *skeleton*] [**-H** *header-skeleton*] [**-debug**] [**-defines**] [**-fixed-output-files**] [**-no-lines**] [**-verbose**] [**-version**] [**-yacc**] [**-usage**] [**-help**] [**-file-prefix=prefix**] [**-name-prefix=prefix**] [**-skeleton=skeletonfile**] [**-headerskeleton=headerskeletonfile**] [**-output=outfile**] [**-header-name=header**] *grammar-file*

**DESCRIPTION**

Generate a parser. Based on **bison** version 1.19. See **bison**(1) for details of main functionality. Only changes are reported here.

You now generate a C++ class if you are compiling with a C++ compiler. The generated header is far more rich than before, and is made from a skeleton-header. The code skeleton is also richer, and the generated code is less important compared to the skeletons. It permit you to modify much things only by changing the two skeletons.

In plain C, the **bison++** is compatible with standard **bison**.

**OPTIONS**

**—name-prefix=prefix**

**-p** *prefix*

Set prefix of names of yylex, yyerror. kepted for compatibility, but you should prefer **%define LEX newname**, and similar.

**—skeleton=skeleton**

**-S** *skeleton*

Set filename of code skeleton. Default is **bison.cc**.

**—headerskeleton=header-skeleton**

**-H** *header-skeleton*

Set filename of header skeleton. Default is **bison.h**.

**—header-name=header**

**-h** *header*

Set filename of header skeleton. Default is **y.tab.h**, or *prefix.h* if option **-b** is used or *c\_basename.h* if **-o** is used. **.c**, **.cc**, **.C**, **.cpp**, **.cxx** options for output files are replaced by **.h** for header name.

**DECLARATIONS**

These are new declarations to put in the declaration section :

**%name** *parser\_name*

Declare the name of this parser. User for C++ class name, and to render many names unique. default is **parse**. Must be given before **%union** and **%define**, or never.

**%define** *define\_name content...*

Declare a macro symbol in header and code. The name of the symbol is **YY\_’parser\_name’\_’define\_name’**. The content if given after, as with **#define**. Newline can be escaped as with **#define**. Many symbols are proposed for customisation.

**%union**

as with bison generate a union for semantic type. The difference is that the union is named **yy\_’parser\_name’\_’stype’**.

**%pure\_parser**

As with bison in C. In C++ generate a parser where **yylval**, and **yylloc** (if needed) are passed as parameter to **yylex**, and where some instance variable are local to **yyparse** (like **yydebug...**). Not very useful, since you can create multiple instances for reentering another parser.

**%header{**

Like **%{**, but include this text both in the header, and in the code. End with **%}**. When put in declaration section, the text is added before the definitions. It can be put in the last section so that the text is added after all definition in the header, and in the last section at the current position in the code.

Note that the order of these declaration is important, since they are translated into preprocessor symbols, typedef or code depending on their type. For example use **%name** before any **%define**, since the name is needed to compose the name of the define symbols. Order of **%header** and **%union** is important, since type may be undefined.

**DECLARATION DEFINE SYMBOLS**

These are the symbols you can define with **%define** in declaration section, or that are already defined. Remind that they are replaced by a preprocessor **#define YY\_‘parser\_name’\_‘name’**.

**BISON** defined to **1** in the code. used for conditional code. Don’t redefine it.

**h\_included**

defined in the code, and in the header. used for include anti-reload. Don’t redefine it.

**COMPATIBILITY**

Indicate if obsoleted defines are to be used and produced. If defined to 0, indicate no compatibility needed, else if defined to non-0, generate it. If it is undefined, default is to be compatible if classes are not used.

**USE\_GOTO**

Indicates (if defined as 1) that **goto** are to be used (for backward compatibility) in the parser function. By default **goto** are replaced with a **switch** construction, to avoid problems with some compiler that don’t support **goto** and destructor in the same function block. If **COMPATIBILITY** is 1, and **USE\_GOTO** is not defined, then **USE\_GOTO** is defined to 1, to be compatible with older bison.

**USE\_CONST\_TOKEN**

Indicate (if defined as 1) that **static const int** are to be used in C++, for token IDs. By default an enum is used to define the token IDs instead of const.

**ENUM\_TOKEN**

When **enum** are used instead of **static const int** for token IDs, this symbol define the name of the enum type. Defined to **yy\_‘parser\_name’\_enum\_token** by default.

**PURE** Indicate that **%pure\_parser** is asked... Don’t redefine it.

**LSP\_NEEDED**

if defined indicate that **@** construct is used, so **LLOC** stack is needed. Can be defined to force use of location stack.

**DEBUG**

if defined to non-0 activate debugging code. See **YYDEBUG** in bison.

**ERROR\_VERBOSE**

if defined activate dump parser stack when error append.

**STYPE** the type of the semantic value of token. defined by **%union**. default is **int**. See **YYSTYPE** in bison. Don’t redefine it, if you use a **%union**.

**LTYPE**

The token location type. If needed default is **yytype**. See **YYLTYPE** in bison. default **yytype** is a typedef and struct defined as in old bison.

**LLOC** The token location variable name. If needed, default is **yyloc**. See **yyloc** in bison.

**LVAL** The token semantic value variable name. Default **yyval**. See **yyval** in bison.

**CHAR** The lookahead token value variable name. Default **yychar**. See **yychar** in bison.

**LEX** The scanner function name. Default **yylex**. See **yylex** in bison.

**PARSE** The parser function name. Default **yyparse**. See **yyparse** in bison.

#### **PARSE\_PARAM**

The parser function parameters declaration. Default **void** in C++ or ANSIC, nothing if old C. In ANSIC and C++ contain the prototype. In old-C contain just the list of parameters name. Don't allow default value.

#### **PARSE\_PARAM\_DEF**

The parser function parameters definition, for old style C. Default nothing. For example to use an **int** parameter called **x**, **PARSE\_PARAM** is **x**, and **PARSE\_PARAM\_DEF** is **int x**. In ANSIC or C++ it is useless and ignored.

#### **ERROR**

The error function name. Default **yyerror**. See **yyerror** in bison.

#### **NERRS**

The error count name. Default **ynerrs**. See **ynerrs** in bison.

#### **DEBUG\_FLAG**

The runtime debug flag. Default **yydebug**. See **yydebug** in bison.

These are only used if class is generated.

**CLASS** The class name. default is the parser name.

#### **INHERIT**

The inheritance list. Don't forget the **:** before, if not empty list.

#### **MEMBERS**

List of members to add to the class definition, before ending it.

#### **LEX\_BODY**

The scanner member function body. May be defined to **=0** for pure function, or to an inline body.

#### **ERROR\_BODY**

The error member function body. May be defined to **=0** for pure function, or to an inline body.

#### **CONSTRUCTOR\_PARAM**

List of parameters of the constructor. Don't allow default value.

#### **CONSTRUCTOR\_INIT**

List of initialisation before constructor call. If not empty don't forget the **:** before list of initialisation.

#### **CONSTRUCTOR\_CODE**

Code added after internal initialisation in constructor.

#### **OBSOLETE PREPROCESSOR SYMBOLS**

if you use new features, the following symbols should not be used, though they are proposed. The symbol **COMPATIBILITY** control their disponibility. Incoherence may arise if they are defined simultaneously with the new symbol.

#### **YYLTYPE**

prefer **%define LTYPE**.

#### **YYSTYPE**

prefer **%define STYPE**.

#### **YYDEBUG**

prefer **%define DEBUG**.

#### **YYERROR\_VERBOSE**

prefer **%define ERROR\_VERBOSE**.

**YYLSP\_NEEDED**

prefer **%define LSP\_NEEDED**.

**yystype** Now a preprocessor symbol instead of a typedef. prefer **yy\_'parser\_name'\_stype**.

**CONSERVED PREPROCESSOR SYMBOLS**

These symbols are kept, and cannot be defined elsewhere, since they control private parameters of the generated parser, or are actually unused. You can **#define** them to the value you need, or indirectly to the name of a **%define** generated symbol if you want to be clean.

**YYINITDEPTH**

initial stack depth.

**YYMAXDEPTH**

stack overflow limit depth.

**yyoverflow**

instead of expand with `alloca`, `realloc` manually or raise error.

**OTHER ADDED PREPROCESSOR SYMBOLS****YY\_USE\_CLASS**

indicate that class will be produced. Default if C++.

**C++ CLASS GENERATED**

To simplify the notation, we note **%SYMBOLNAME** the preprocessor symbol generated with a **%define** of this name. In fact see the use of **%define** for it's real name.

Note that there is sometime symbols that differ from only an underscore `_`, like **yywrap** and **yy\_wrap**. They are much different. In this case **yy\_wrap()** is a virtual member function, and **yywrap()** is a macro.

**General Class declaration**

```
class %CLASS %INHERIT
{
public:
#if %USE_CONST_TOKEN != 0
static const TOKEN_NEXT;
static const AND_SO_ON;
// ...
#else
enum %ENUM_TOKEN { %NULL_TOKEN
    ,TOKEN_FIRST=256
    ,TOKEN_NEXT=257
    ,AND_SO_ON=258
};
// ...
#endif
public:
int %PARSE (%PARSE_PARAM);
virtual void %ERROR(char *msg) %ERROR_BODY;
```

```

#ifdef %PURE
    // if %PURE , we must pass the value and (eventually) the location explicitly
    #ifdef %LSP_NEEDED
        // if and only if %LSP_NEEDED , we must pass the location explicitly
        virtual int %LEX (%STYPE %%LVAL,%LTYPE %%LLOC) %LEX_BODY;
    #else
        virtual int %LEX (%STYPE %%LVAL) %LEX_BODY;
    #endif
#else
    // if not %PURE , we must declare member to store the value and (eventually) the location
    // explicitly
    // if not %PURE ,%NERRS and %CHAR are not local variable to %PARSE, so must be
    // member
    virtual int %LEX() %LEX_BODY;
    %STYPE %LVAL;
    #ifdef %LSP_NEEDED
        %LTYPE %LLOC;
    #endif
    int %NERRS;
    int %CHAR;
#endif

#ifdef %DEBUG != 0
    int %DEBUG_FLAG; /* nonzero means print parse trace */
#endif

public:
    %CLASS(%CONSTRUCTOR_PARAM);
public:
    %MEMBERS
};

// here are defined the token constants
// for example:
#ifdef %USE_CONST_TOKEN != 0
    const %CLASS::TOKEN_FIRST=1;
#endif

// here is the construcor
%CLASS::%CLASS(%CONSTRUCTOR_PARAM) %CONSTRUCTOR_INIT
{

```

```

#if %DEBUG != 0
%DEBUG_FLAG=0;
#endif
%CONSTRUCTOR_CODE;
};

```

#### Default Class declaration

```

// Here is the default declaration made in the header when you %define nothing
// typical yyltype
typedef struct yytype
{
int timestamp;
int first_line;
int first_column;
int last_line;
int last_column;
char *text;
} yytype;
// class definition
class parser
{
public:
enum yy_parser_enum_token { YY_parser_NULL_TOKEN
    ,TOKEN_FIRST=256
    ,TOKEN_NEXT=257
    ,AND_SO_ON=258
} ;
// ...
public:
int yyparse (yyparse_PARAM);
virtual void yyerror(char *msg) ;
#ifdef YY_parser_PURE
    #ifdef YY_parser_LSP_NEEDED
        virtual int yylex (int *yylval,yytype *yylloc) ;
    #else
        virtual int yylex (int *yylval) ;
    #endif
#else
    virtual int yylex() %LEX_BODY;
    int yyval;

```

```

#ifdef YY_parser_LSP_NEEDED
    yytype yyloc;
#endif
int yynerrs;
int yychar;

#endif
#if YY_parser_DEBUG != 0
int yydebug;
#endif
public:
parser();
public:
};
// here is the constructor code
parser::parser()
{
#if YY_parser_DEBUG != 0
yydebug=0;
#endif
};

```

**USAGE**

Should replace **bison**, because it generate a far more customisable parser, still beeing compatible.

You should always use the header facility.

Use it with **flex++** (same author).

**EXAMPLES**

This man page has been produced through a parser made in C++ with this version of **bison** and our version of **flex++** (same author).

**FILES****bison.cc**

main skeleton.

**bison.h** header skeleton.

**bison.hairy**

old main skeleton for semantic parser. Not adapted to this version. Kept for future works.

**ENVIRONNEMENT****DIAGNOSTICS****SEE ALSO**

**bison**(1), **bison.info** (use texinfo), **flex++**(1).

**DOCUMENTATION****BUGS**

Tell us more !

The **%semantic\_parser** is no more supported. If you want to use it, adapt the skeletons, and maybe **bison++** generator itself. The reason is that it seems unused, unuseful, not documented, and too complex for us to support. tell us if you use, need, or understand it.

Header is not included in the parser code. Change made in the generated header are not used in the parser code, even if you include it voluntarily, since it is guarded against re-include. So don't modify it.

For the same reasons, if you modify the header skeleton, or the code skeleton, report the changes in the other skeleton if applicable. If not done, incoherent declarations may lead to unpredictable result.

Use of defines for **YYLTYPE**, **YYSTYPE**, **YYDEBUG** is supported for backward compatibility in C, but should not be used with new features, as **%defines** or C++ classes. You can define them, and use them as with old **bison** in C only.

Parameters are richer than before, and nothing is removed. POSIX compliance can be enforced by not using extensions. If you want to forbid them, there is a good job !

#### **FUTUR WORKS**

tell us !

Support semantic parser. Is it really used ?

POSIX compliance. is'nt it good now ?

Use lex and yacc (flex/bison) to generate the scanner/parser. It would be comfortable for futur works, though very complicated. Who feel it good ?

**iostream** : this is a great demand. this work will be done as soon as possible. The virtual members permit such work still easily.

#### **INSTALLATION**

With this install the executable is named bison++. rename it bison if you want, because it could replace **bison**.

#### **TESTS**

#### **AUTHORS**

Alain Coëtmeur (coetmeur@icdc.fr), R&D department (RDT) , Informatique-CDC, France.

#### **RESTRICTIONS**

The words 'author', and 'us' mean the author and colleagues, not GNU. We don't have contacted GNU about this, nowadays. If you're in GNU, we are ready to propose it to you, and you may tell us what you think about.

Based on GNU version 1.21 of bison. Modified by the author.