Compiler Construction

Coupling Flex and Bison

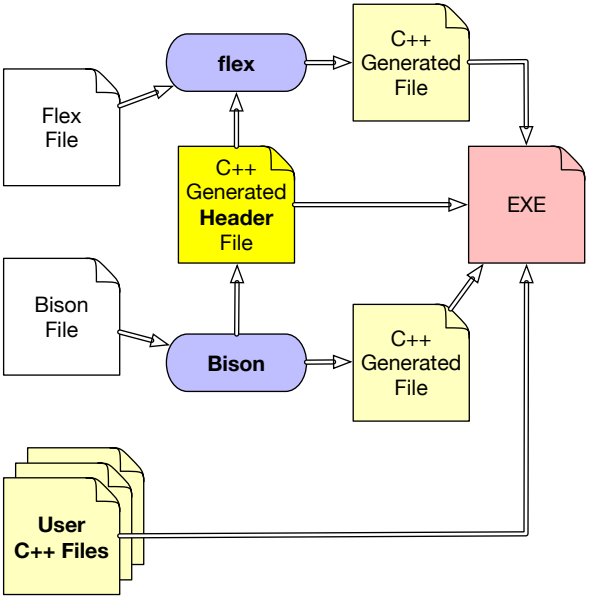
Goal

How to produce a stream of tokens in the scanner that will be analyzed by the parser?

Problems

1. Both Flex and Bison need the definition of **tokens**!
2. How to forward the stream of token from Flex to Bison?
3. How to share locations between Flex and Bison.

Overview


flex
C++
Generated
File
Flex
File
C++
Generated
Header
File
EXE
Bison
File
Bison
C++
Generated
File
User
C++ Files

Locations tracking

locations help to provide better errors messages

Locations are tracked in the scanner but

- they can be used by the parser (*Syntax errors*)
- they must be forwarded to the rest of the compiler (*Semantic errors*)

Detailed example: Bison file (1/5)

```
%language "C++"
%require "3.2"
// Don't use unions, but variants
%define api.value.type variant

// Require for make_FOO type-safe functions
%define api.token.constructor

// Prefix all the token with TOK_
%define api.token.prefix {TOK_}

%define parse.error verbose

%define parse.trace // Enable yydebug.
```

Detailed example: Bison file (2/5)

```
// Enable location tracking.
%locations
// Parser & scanner exchange  the number  of errors.
%param { int& num_errors }

%code provides
{
// Declaration, for the scanner.
#define YY_DECL \
  yy::parser::symbol_type \
  yylex(int& num_errors)

// Declaration for the parser.
YY_DECL;
}
```

Detailed example: Bison file (3/5)

```
%code requires
{
#include <iostream>
}

%code
{
 int res = 0;
}

// Our tokens.
%token<int> NUMBER "number"
%token      PLUS   "+"
%token      EOF 0  "end-of-file"
%nterm <int> exp // Our (only) typed non terminals.
```

Detailed example: Bison file (4/5)

```
%printer { yyo << $$; } <int>; // Better debug traces.
%left "+" // Precedence/associativity.

%%
%start unit;
unit:
   exp EOF {res = $1;}
;

exp:
   exp "+" exp { $$ = $1 + $3;}
 | NUMBER      { $$ = $1;}
;
%%
```

Detailed example: Bison file (5/5)

```
// Epilogue.
void yy::parser::error(const location_type& loc,
                       const std::string& s)
{
  std::cerr << loc << ": " << s << '\n';
}

int main(int argc, char** argv)
{

  auto num_errors = 0;
  yy::parser parser(num_errors);
  auto status = parser.parse();
  std::cout << res << std::endl;
}
```

Detailed example: Flex file (1/2)

```
%option noyywrap
blank [ \t]
int   [0-9]+

%{
#include "parser.hh"
yy::parser::location_type loc;

#define YY_USER_ACTION                          \
  do {                                          \
    loc.columns(yyleng);                        \
  } while (false);
%}
%%
  loc.step();
```

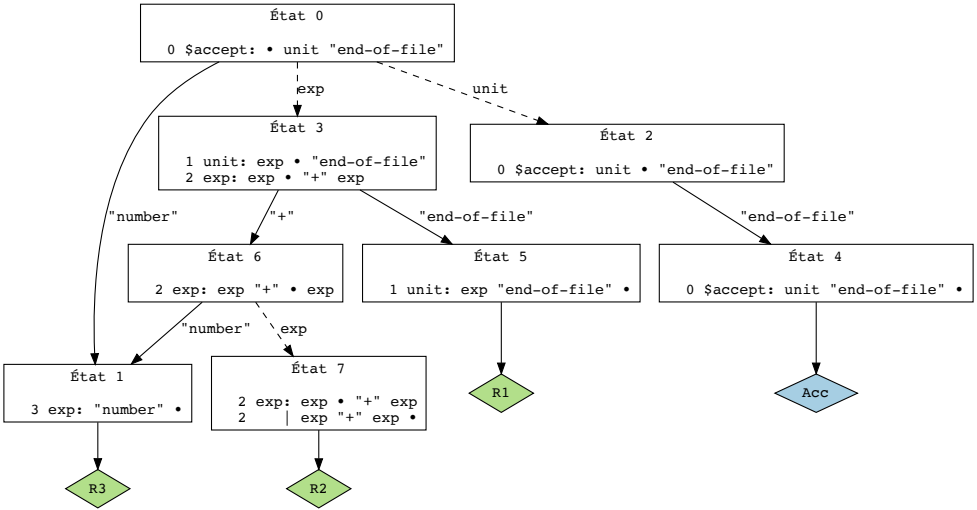
Detailed example: Flex file (2/2)

```
 // Rules.
"+"      return yy::parser::make_PLUS(loc);
[0-9]+   return yy::parser::make_NUMBER( \
                      strtol(yytext, nullptr, 0), loc);
{blank}+   loc.step ();
[\n]+      loc.lines (yyleng); loc.step ();
.          {
  std::cerr << "unexpected character: " << yytext << '\n';
  num_errors += 1;
}
<<EOF>>  return yy::parser::make_EOF(loc);
%%
// Epilogue.
```

Altogether

```
$ bison -o parser.cc  -d --graph  tmp.yy
$ flex -o lexer.cc  tmp.ll
$ g++ -std=c++20 lexer.cc parser.cc
$ echo "1+2+3+98" | ./a.out
104
$ echo "1+2+3++98" | ./a.out
1.7: syntax error, unexpected +, expecting number
0
```

A Final word on Bison



Summary