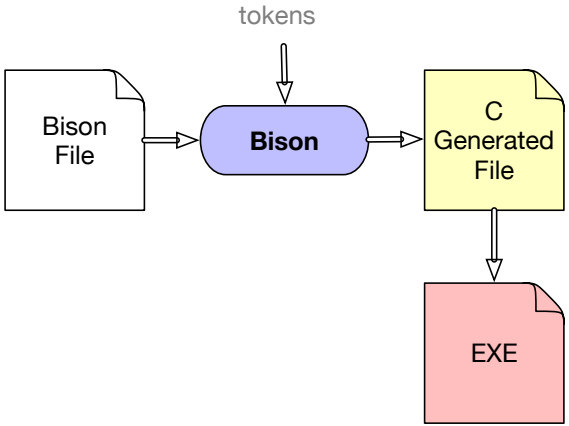
Compiler Construction

Bison

Bison

- **Bison**: replacement of YACC (***Y**et **A**nother **C**ompiler **C**ompiler*)
  Pun between Yak and Bison
- **Initial release**: 1985
- Written in C by Robert Corbett
- Generates syntactic analyzers (parsers)
- Generates LALR, LR, IELR, GLR, [...] parsers
- Used by GCC (until 2004/2006) and Go compiler (until 2015)

Overview



Typical Bison file

```
%{
  [pre-code C (nec. def.)]
%}

[definitions and options]

%%

[production rules]

%%

[post-code C (subprograms)]
```

Production rules

```
non-terminal:
    seq. of symb. { /*C*/ }
  | another seq.  { /*C*/ }
  | ... ;
```

Remark

C-code is optional and is only executed when the rule is **reduced**

Rules Reductions: LALR vs. GLR

LALR-1 – Default for bison

- Default behavior when a conflict occurs:
  - reduce/reduce: reduce to the first rule in conflict
  - shift/reduce: performs the shift
- During a shift/reduce conflict the parser may be *miles away from the ball*

Rules Reductions: LALR vs. GLR

LALR-1 – Default for bison

- Default behavior when a conflict occurs:
  - reduce/reduce: reduce to the first rule in conflict
  - shift/reduce: performs the shift
- During a shift/reduce conflict the parser may be *miles away from the ball*

GLR

1. During a conflict the parser walks the two branches hoping that one of the two will win.
2. Maintains multiple parse stacks
3. Allows ambigous grammars

Example

```
%%
exp:
     "if" exp "then" exp
   | "if" exp "then" exp "else" exp
   | "exp";
%%
```

Example

```
%%
exp:
     "if" exp "then" exp
   | "if" exp "then" exp "else" exp
   | "exp";
%%
```

Problem: **Dangling Else**

”else” should rattach to which ”if”? Inner one or outer one? **if ”exp” then if ”exp” then ”exp” else ”exp”**

Ambiguous grammar: solution

```
%expect 0
%right "else" "then"
%%
exp:
    "if" exp "then" exp
  | "if" exp "then" exp "else" exp
  | "exp" ;
%%
```

Ambiguous grammar: solution

```
%expect 0
%right "else" "then"
%%
exp:
     "if" exp "then" exp
   | "if" exp "then" exp "else" exp
   | "exp" ;
%%
```

- %right: choose shift
- %left: choose reduce
- %expect: the number of expected conflicts

Another solution would be to add "fi".

Bison – associativity

Let us consider *x op y op z*

- **left associativity** (%left)
  will group *((x op y) op z)*
- **right associativity** (%right)
  will group *(x op (y op z))*
- **No-associativity** (%nonassoc) means that *x op y op z* is considered as a syntax error

Bison – associativity (2/2)

Important

- %precedence gives only precedence to the symbols, and defines no associativity at all.
- tokens declared in a single precedence declaration have equal precedence
- When two tokens declared in different precedence declarations associate, the one declared later has the higher precedence and is grouped first

Bison – details

- **yyparse**:
  - consumes the input stream (sequence) of token
  - checks if the sequence can be reduce to the initial rule(%start)
  - executes C-code associated to production rules used to reduce the input
  - may raise errors (yyerror)
  - return 0 or 1
- **yyerror**:
  - to be provided by the user
  - may be used for error recovery

Summary