Analisador sintáctico Bison/Yacc

Introdução

- Yacc - Yet another compiler-compiler
  - Stephen Johnson em 1975
  - Copyright Bell Labs/AT&T
- Reimplementações populares:
  - Berkeley yacc (byacc) (Robert Corbett, 1989)
    - Re-implementação com algoritmo de construção mais eficiente
    - Licença dominio público
  - Bison (Robert Corbett, 1985)
    - Extensão compatível com YACC (Richard Stallman).
    - Licença GNU GPL-3

Analisador sintáctico Bison/Yacc

Bison:
Dada a especificação de uma gramática, gera código capaz de organizar os tokens da entrada numa árvore sintáctica de acordo com a gramática.
Bison é compatible com Yacc.

- Gramática especificada em Backus-Naur Form (BNF)
  - cada regra está associada a uma acção semântica
  - As acções semânticas são executadas quando cada nó é **reduzido** (i.e., quando todo o corpo foi visto)
- Parser gerado é do tipo LALR(1) (Look-Ahead LR)
  - Para além de LALR(1), e a diferença do Yacc, o Bison é capaz de gerar outros parsers (ex: canonical LR(1)).

Analisador sintáctico Bison/Yacc

Estrutura da especificação

```
%{
código de preparação
%}
definições
%%
regras e acções semânticas
%%
código
```

Três secções separadas por uma linha, apenas com os caracteres %%

- Código de preparação é adicionado ao topo do `file.tab.c`
- Definições e Regras vão definir a função `yyparse()` do `file.tab.c`
- Código é adicionado ao fim do `file.tab.c`

Analisador sintáctico Bison/Yacc

Estrutura - Código de preparação

O código de preparação pode conter:

- includes (e.g. `#include <iostream>`)
- declaração de variáveis globais
- definição de funções auxiliares
- macros
- ...

Analisador sintáctico Bison/Yacc

Estrutura - Definições

Definições podem incluir:

- Definição de símbolos terminais (usados Flex & YACC)
  `%token tWHILE tIF tPRINT tREAD tBEGIN tEND`

Analisador sintáctico Bison/Yacc

Estrutura - Definições

Definições podem incluir:

- Definição de símbolos terminais (usados Flex & YACC)
  `%token tWHILE tIF tPRINT tREAD tBEGIN tEND`
- Tipos disponíveis para os símbolos (terminais ou não terminais)
  `%union { ... };` (slide seguinte)

Analisador sintáctico Bison/Yacc

Estrutura - Definições

Definições podem incluir:

- Definição de símbolos terminais (usados Flex & YACC)
  `%token tWHILE tIF tPRINT tREAD tBEGIN tEND`
- Tipos disponíveis para os símbolos (terminais ou não terminais)
  `%union { ... };` (slide seguinte)
- Tipificação de símbolos terminais
  `%token<s> tIDENTIFIER tSTRING`

Analisador sintáctico Bison/Yacc

Estrutura - Definições

Definições podem incluir:

- Definição de símbolos terminais (usados Flex & YACC)
  `%token tWHILE tIF tPRINT tREAD tBEGIN tEND`
- Tipos disponíveis para os símbolos (terminais ou não terminais)
  `%union { ... };` (slide seguinte)
- Tipificação de símbolos terminais
  `%token<s> tIDENTIFIER tSTRING`
- Tipificação de símbolos não terminais
  `%type<lvalue> lval`

Analisador sintáctico Bison/Yacc

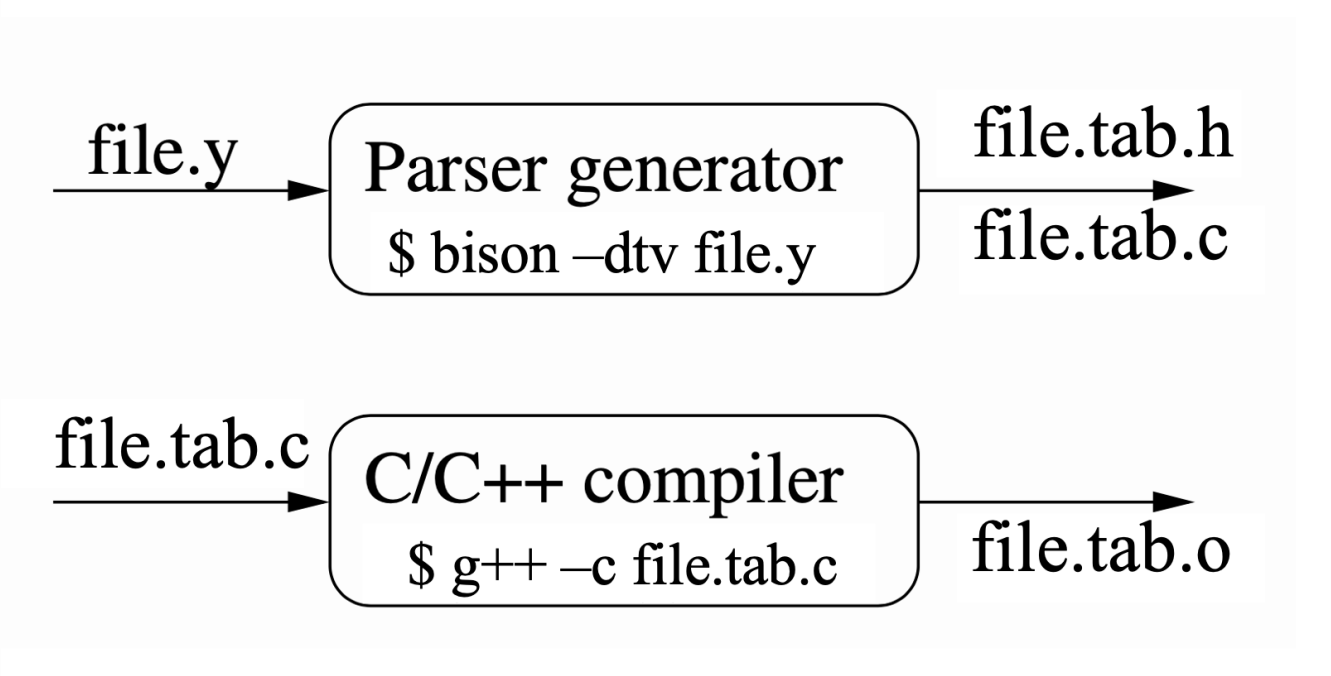
Estrutura - %union

Tipos disponíveis para os símbolos (terminais ou não terminais)

```
%union {
  int                    i;          /* XPL example */
  std::string           *s;          /* symbol name */
  cdk::basic_node       *node;       /* node pointer */
  cdk::sequence_node    *sequence;   /* sequence node */
  cdk::expression_node  *expression; /* expression node */
  cdk::lvalue_node      *lvalue;     /* lvalue node */
};
```

- União de todos os tipos dado que cada *token* corresponde apenas a um dos casos
- Cada novo tipo de *token* (símbolo terminal) ou nó da árvore (símbolo não terminal), tem de ser declarado na %union

Analisador sintáctico Bison/Yacc



- ficheiro `file.tab.c` com o parser gerado
- **-t** inclusão de instruções de debug no código compilado
- **-v** ficheiro `file.output` com descrição do parser gerado
- **-d** ficheiro `file.tab.h` com identificação de tokens e os tipos

Analisador sintáctico Bison/Yacc

simple_parser.tab.h

```
/* Token kinds.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  enum yytokentype
  {
    YYEMPTY = -2,
    YYEOF = 0,                     /* "end of file"  */
    YYerror = 256,                 /* error  */
    YYUNDEF = 257,                 /* "invalid token"  */
    tINTEGER = 258,                /* tINTEGER  */
    tIDENTIFIER = 259,             /* tIDENTIFIER  */
    tSTRING = 260,                 /* tSTRING  */
    tWHILE = 261,                  /* tWHILE  */
    tIF = 262,                     /* tIF  */
    tPRINT = 263,                  /* tPRINT  */
    tREAD = 264,                   /* tREAD  */
    tBEGIN = 265,                  /* tBEGIN  */
    tEND = 266,                    /* tEND  */
    tIFX = 267,                    /* tIFX  */
    tELSE = 268,                   /* tELSE  */
    tGE = 269,                     /* tGE  */
    tLE = 270,                     /* tLE  */
    tEQ = 271,                     /* tEQ  */
    tNE = 272,                     /* tNE  */
    tUNARY = 273                   /* tUNARY  */
  };
  typedef enum yytokentype yytoken_kind_t;
#endif
```

Analisador sintáctico Bison/Yacc

simple_parser.tab.h

```
union YYSTYPE
{
#line 17 "simple_parser.y"

  //--- don't change *any* of these: if you do, you'll break the compiler.
  YYSTYPE() : type(cdk::primitive_type::create(0, cdk::TYPE_VOID)) {}
  ~YYSTYPE() {}
  YYSTYPE(const YYSTYPE &other) { *this = other; }
  YYSTYPE& operator=(const YYSTYPE &other) { type = other.type; return *this; }

  std::shared_ptr<cdk::basic_type> type;        /* expression type */
  //--- don't change *any* of these --- END!

  int                   i;      /* integer value */
  std::string          *s;      /* symbol name or string literal */
  cdk::basic_node      *node;   /* node pointer */
  cdk::sequence_node   *sequence;
  cdk::expression_node *expression; /* expression nodes */
  cdk::lvalue_node     *lvalue;

#line 100 "simple_parser.tab.h"

};
typedef union YYSTYPE YYSTYPE;
```

Analisador sintáctico Bison/Yacc

Interligação com o Flex

```
%{
#include "simple_parser.tab.h"
%}
%x X_STRING
%%

"--".*                  ; /* ignore comments */
">="                    return tGE;
"<="                    return tLE;
"=="                    return tEQ;
"!="                    return tNE;
"while"                 return tWHILE;
"if"                    return tIF;
"else"                  return tELSE;
"print"                 return tPRINT;
"read"                  return tREAD;
"begin"                 return tBEGIN;
"end"                   return tEND;
[A-Za-z][A-Za-z0-9_]*   yylval.s = new std::string(yytext); return tIDENTIFIER;
\'                      yy_push_state(X_STRING); yylval.s = new std::string("");
<X_STRING>\'            yy_pop_state(); return tSTRING;
<X_STRING>\\\'          *yylval.s += yytext + 1;
<X_STRING>.             *yylval.s += yytext;
<X_STRING>\n            yyerror("newline in string");
[0-9]+                  yylval.i = strtol(yytext, nullptr, 10); return tINTEGER;
[-()<>=+*/%;{}.]        return *yytext;
[ \t\n]+                ; /* ignore whitespace */
.                       yyerror("Unknown character");

%%
```

Analisador sintáctico Bison/Yacc

Estrutura - Regras/Acções

Uma especificação Bison/Yacc tem associado pares:

Regra { Acção semântica }

- o corpo da Regra é constituído por zero ou mais símbolos terminais e não terminais
  - $A \rightarrow \epsilon$ é representado pelo corpo vazio
- a Acção (código C/C++ arbitrário)

Analisador sintáctico Bison/Yacc

Estrutura - Regras/Acções

Uma especificação Bison/Yacc tem associado pares:

Regra { Acção semântica }

- o corpo da Regra é constituído por zero ou mais símbolos terminais e não terminais
  - $A \rightarrow \epsilon$ é representado pelo corpo vazio
- a Acção (código C/C++ arbitrário)

A pilha do Bison:

- Contém todos os símbolos do corpo
- Quando a avaliação do corpo chega ao fim
  - é feito pop de todos os símbolos do corpo
  - é feito push do símbolo da cabeça da regra
  - é avaliado o atributo da cabeça da regra

Analisador sintáctico Bison/Yacc

Estrutura - Regras/Acções

Comunicação entre as acções e o parser, é feita através do símbolo "$"

- $1, $2, ..., $n refere-se ao 1º, 2º, ..., nº símbolo do corpo
- $$ refere-se ao valor do símbolo não terminal na cabeça da regra
- Por omissão, se a acção semântica for vazia, o valor atribuído ao símbolo na cabeça da regra é o valor do 1º símbolo do corpo ($$ = $1)

Analisador sintáctico Bison/Yacc

Estrutura - Regras/Acções

```
list : stmt        { $$ = new cdk::sequence_node(LINE, $1); }
     | list stmt   { $$ = new cdk::sequence_node(LINE, $2, $1); }
     ;
stmt : expr ';'                          { $$ = new simple::evaluation_node(LINE, $1); }
     | tPRINT expr ';'                   { $$ = new simple::print_node(LINE, $2); }
     | tREAD lval ';'                    { $$ = new simple::read_node(LINE, $2); }
     | tWHILE '(' expr ')' stmt          { $$ = new simple::while_node(LINE, $3, $5); }
     | tIF '(' expr ')' stmt %prec tIFX  { $$ = new simple::if_node(LINE, $3, $5); }
     | tIF '(' expr ')' stmt tELSE stmt  { $$ = new simple::if_else_node(LINE, $3, $5, $7); }
     | '{' list '}'                      { $$ = $2; }
     ;
expr : tINTEGER                { $$ = new cdk::integer_node(LINE, $1); }
     | tSTRING                 { $$ = new cdk::string_node(LINE, $1); }
     | '-' expr %prec tUNARY   { $$ = new cdk::neg_node(LINE, $2); }
     | expr '+' expr           { $$ = new cdk::add_node(LINE, $1, $3); }
     | expr '-' expr           { $$ = new cdk::sub_node(LINE, $1, $3); }
     | expr '*' expr           { $$ = new cdk::mul_node(LINE, $1, $3); }
     | expr '/' expr           { $$ = new cdk::div_node(LINE, $1, $3); }
     | expr '%' expr           { $$ = new cdk::mod_node(LINE, $1, $3); }
     | expr '<' expr           { $$ = new cdk::lt_node(LINE, $1, $3); }
     | expr '>' expr           { $$ = new cdk::gt_node(LINE, $1, $3); }
     | expr tGE expr           { $$ = new cdk::ge_node(LINE, $1, $3); }
     | expr tLE expr           { $$ = new cdk::le_node(LINE, $1, $3); }
     | expr tNE expr           { $$ = new cdk::ne_node(LINE, $1, $3); }
     | expr tEQ expr           { $$ = new cdk::eq_node(LINE, $1, $3); }
     | '(' expr ')'            { $$ = $2; }
     | lval                    { $$ = new cdk::rvalue_node(LINE, $1); }
     | lval '=' expr           { $$ = new cdk::assignment_node(LINE, $1, $3); }
     ;
lval : tIDENTIFIER             { $$ = new cdk::variable_node(LINE, $1); }
     ;
```

Analisador sintáctico Bison/Yacc

Estrutura - Precedências

Precedências/associatividades:

- Associados a *tokens* na secção Declarações
- Usados na resolução de conflictos/ambiguidades

Especificado com linhas iniciadas com `%left`, `%right` ou `%nonassoc`

- Todos os *tokens* na mesma linha têm o mesmo nível de precedência/associatividade
- Linhas subsequentes têm maior precedência/associatividade
- `%left` - define *tokens* associativos à esquerda
- `%right` - define *tokens* associativos à direita
- `%nonassoc` - define *tokens* que não se podem associar com eles próprios

Analisador sintáctico Bison/Yacc

Estrutura - Precedências

Exemplo:

```
%nonassoc tIFX
%nonassoc tELSE

%right '='
%left tGE tLE tEQ tNE '>' '<'
%left '+' '-'
%left '*' '/' '%'
%nonassoc tUNARY
```

Input: $a = b = c * d - e - f * g$

lido como: $a = (b = (((c * d) - e) - (f * g)))$

Analisador sintáctico Bison/Yacc

Estrutura - Precedências

`%prec` - muda o nível de precedência associado a uma regra

- Aparece imediatamente depois do corpo da regra
- Seguido de um *token*
- Faz com que a regra fique com a mesma precedência do token

Analisador sintáctico Bison/Yacc

Estrutura - Precedências

`%prec` - muda o nível de precedência associado a uma regra

- Aparece imediatamente depois do corpo da regra
- Seguido de um *token*
- Faz com que a regra fique com a mesma precedência do token

```
%right '='
%left tGE tLE tEQ tNE '>' '<'
%left '+' '-'
%left '*' '/' '%'
%nonassoc tUNARY
%%

expr : tINTEGER                 { $$ = new cdk::integer_node(LINE, $1); }
     | tSTRING                  { $$ = new cdk::string_node(LINE, $1);  }
     | '-' expr %prec tUNARY    { $$ = new cdk::neg_node(LINE, $2);     }
     | expr '+' expr            { $$ = new cdk::add_node(LINE, $1, $3); }
     | expr '-' expr            { $$ = new cdk::sub_node(LINE, $1, $3); }
     | expr '*' expr            { $$ = new cdk::mul_node(LINE, $1, $3); }
 ...
```

Analisador sintáctico Bison/Yacc

Debugging

- O despiste de problemas em especificações Flex e Bison pode ser realizado acrescentando ao ficheiro de especificação Flex (`.l`):
  - A opção debug no início: `%option debug`
  - A seguente ação antes da primeira regra:

```
...
%%
                                {yydebug=1;}
expr : tINTEGER                 { $$ = new cdk::integer_node(LINE, $1); }
...
```

- O desenvolvimento da gramática nos compiladores simples abordados deve ser realizado de forma incremental:
  - Maior facilidade de deteção de possíveis conflitos.
  - A opção `-Wcounterexamples` do Bison permite gerar exemplos dos conflitos.

Analisador sintáctico Bison/Yacc

Exemplo calculadora simples

Uma calculadora simples tem um número não especificado de variáveis inteiras e os operadores inteiros binários comuns (ou seja, adição, subtração, multiplicação, divisão e módulo) e operadores inteiros unários (`+` e `-`).
A linguagem contêm os seguintes conceitos (tokens): `VAR` (uma variável: o atributo `s` correspondente conterá seu nome); `INT` (um inteiro: o atributo `i` correspondente contém seu valor); e os operadores. Múltiplas operações são separadas por `,` ou `:`, neste caso, mostrando o resultado pela saída estândar.