Compiladores
Aula 14
YACC – Bison: Gerador de *Parser*

Prof. Dr. Luiz Eduardo G. Martins

UNIFESP

YACC – Gerador de *Parser*

- *Parsers* podem ser gerados automaticamente, a partir da especificação da GLC
- YACC
  - *Yet Another Compiler-Compiler*
- Existem várias implementações de YACC
  - O GCC disponibiliza a versão *Bison*
  - *Bison* gera uma rotina chamada *yyparse()*
    - *yyparse()* faz a análise sintática de um arquivo de entrada (programa fonte)
      - Retorna 0 (não identificou erro)
      - Retorna 1 (identificou erro)

YACC – Gerador de *Parser*

- YACC - *Bison*
  - *yyparse()*
    - É uma função gerada em linguagem C
    - Implementa um *parser LALR(1)*
      - *LA: Look Ahead* (verificação à frente para se obter o próximo token)
      - *L: Left* (entrada processada da esquerda para a direita)
      - *R: Right* (derivação à direita)
      - *(1):* verificação de um símbolo à frente
    - *LALR(1)* é um caso especial de *LR(1)*
    - O *parser LR* é uma analisador ascendente
    - *LR* é considerado o método de análise sintática mais geral que pode ser aplicado a linguagens e gramáticas passíveis de análise determinística

YACC – Gerador de *Parser*

- Visão Geral da Análise Sintática Ascendente
  - Um analisador sintático ascendente usa uma pilha para efetuar a análise
  - A pilha conterá símbolos terminais e não-terminais
  - A pilha está vazia no início da análise e conterá o símbolo inicial ao término de uma análise bem sucedida
  - Um AFD define os estados de um analisador sintático ascendente, é utilizado para acompanhar os estados do analisador sintático

YACC – Gerador de *Parser*

- Visão Geral da Análise Sintática Ascendente
  - O analisador ascendente tem duas ações possíveis (além da ação "aceita" ao término da análise bem sucedida):
    - **Carrega (ou desloca)** um terminal da cadeia de entrada para o topo da pilha;
    - **Reduz** uma cadeia α do topo da pilha para um símbolo não-terminal *A*, dada a escolha BNF $A \rightarrow \alpha$
  - Uma característica adicional dos analisadores ascendentes é a gramática aumentada
    - Necessária para garantir um único estado inicial do autômato finito (com pilha)

YACC – Gerador de *Parser*

- Visão Geral da Análise Sintática Ascendente
  - Como exemplo, considere a gramática aumentada para operações de adição:

    $E' \rightarrow E$

    $E \rightarrow E + n \mid n$

    Para a entrada n + n, as ações do analisador ascendente são:

lookahead

lookahead

Pilha	Entrada	Ação
$	n + n $	carrega
$n	+ n $	reduz E → n
$E	+ n $	carrega
$E +	n $	carrega
$E + n	$	reduz E → E + n
$E	$	Reduz E' → E
$E'	$	aceita

YACC – Gerador de *Parser*

- Visão Geral da Análise Sintática Ascendente
  - Utiliza-se o conceito de item para indicar um passo intermediário no reconhecimento do lado direito de uma escolha específica de regra gramatical
  - Item de uma GLC é uma escolha de produção com uma posição identificada em seu lado direito
  - Essa posição identificada será indicada por um ponto

  Exemplo:

  $E' \rightarrow .E$

  $E' \rightarrow E.$

  $E \rightarrow .E + n$

  $E \rightarrow E. + n$

  $E \rightarrow E +. n$

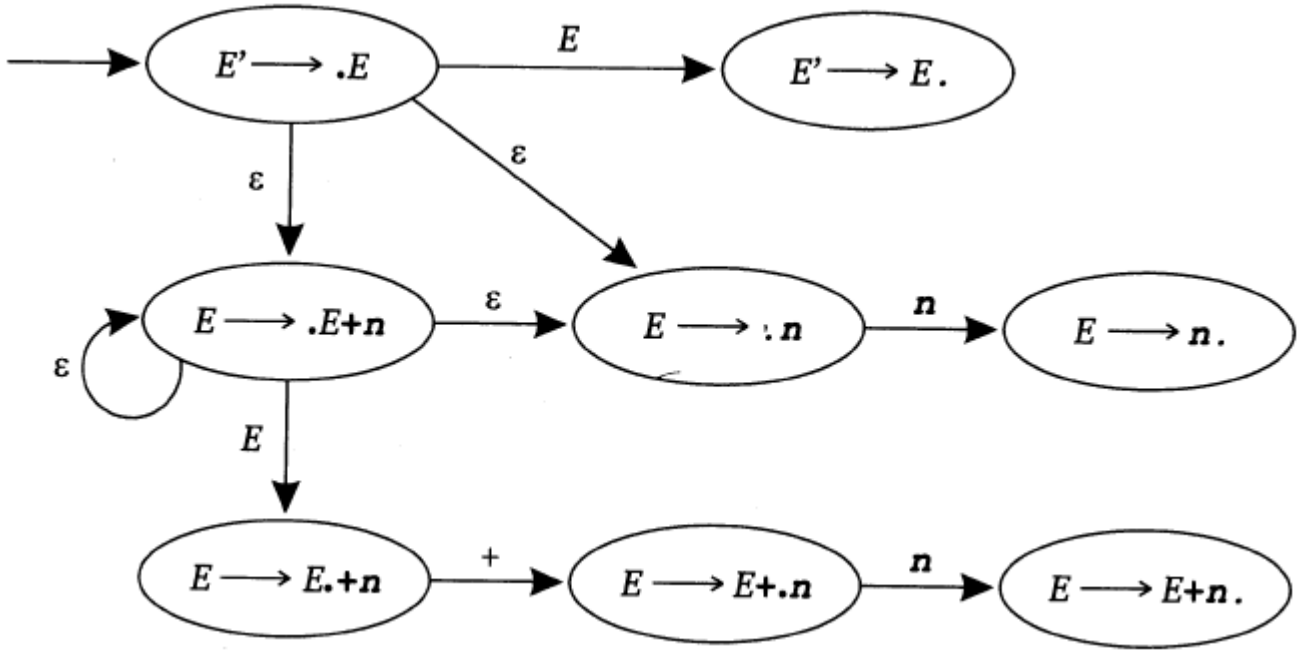
  $E \rightarrow E + n.$

  $E \rightarrow .n$

  $E \rightarrow n.$

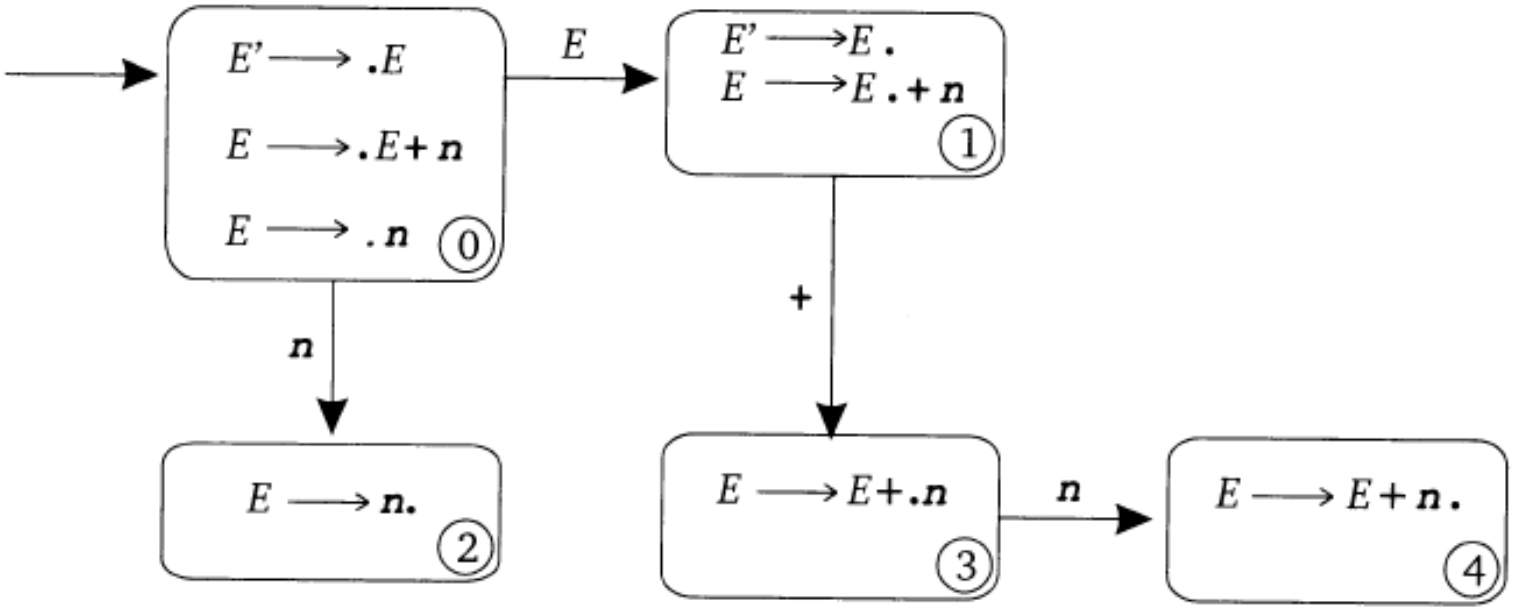
YACC – Gerador de *Parser*

- Visão Geral da Análise Sintática Ascendente
  - NFA de itens para a gramática exemplificada



YACC – Gerador de *Parser*

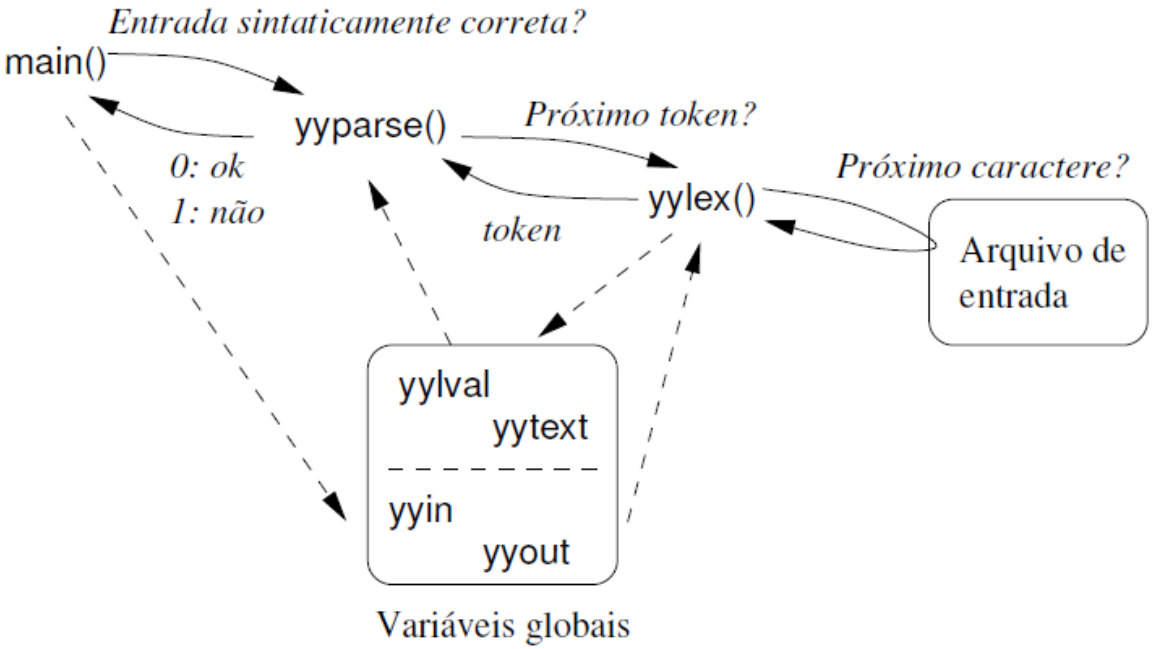
- Visão Geral da Análise Sintática Ascendente
  - Os itens podem ser utilizados como estados de um autômato finito, que mantém as informações sobre a pilha de análise sintática e o progresso de uma análise carrega-reduz
  - DFA de itens para a gramática exemplificada



YACC – Gerador de *Parser*

- YACC
  - *Bison* gera a rotina *yyparse()*
  - *Flex* gera a rotina *yylex()*

Integração lex-yacc



YACC – Gerador de *Parser*

- YACC

Tabela 5.12 Nomes internos e mecanismos de definição Yacc

Nome interno Yaac	Significado/Uso
`y.tab.c`	Nome do arquivo de saída Yacc
`y.tab.h`	Arquivo de cabeçalho gerado pelo Yacc que contém as definições de marcas
`yyparse`	Rotina de análise sintática Yacc
`yylval`	Valor da marca corrente na pilha
`yyerror`	Impressora de mensagens de erro definida pelo usuário e utilizada pelo Yacc
`error`	Pseudomarca de erro Yacc
`yyerrok`	Procedimento que reinicia o analisador depois de um erro
`yychar`	Contém a marca de verificação à frente que provocou um erro
`YYSTYPE`	Símbolo de pré-processador que define o tipo de valor da pilha de análise sintática
`yydebug`	Variável que, se ajustada pelo usuário para 1, leva à geração de informação em tempo de execução sobre as ações do analisador sintático
Mecanismo de definição Yacc	**Significado/Uso**
`%token`	Define os símbolos pré-processadores de marcas
`%start`	Define o símbolo não-terminal inicial
`%union`	Define uma união `YYSTYPE`, com a permissão de valores de tipos distintos na pilha do analisador sintático
`%type`	Define o tipo diferenciado de união retornado por um símbolo
`%left %right %nonassoc`	Define a associatividade e precedência (por posição) dos operadores

YACC – Gerador de *Parser*

Especificação do analisador

Formato yacc

Como arquivo lex, três seções separadas por %%:

1. Definições e declarações
2. Regras da gramática e ações associadas
3. Código complementar

YACC – Gerador de *Parser*

Especificação da gramática

Notação estilo BNF:

```
símbolo   :   expansão   ;
```

símbolo lado esquerdo da produção, um símbolo não-terminal

expansão lado direito da produção, a expansão de símbolo em termos de outros símbolos, sejam eles terminais ou não-terminais (definidos em outra produção da gramática)

Produção = Regra Gramatical

YACC – Gerador de *Parser*

Produções

Símbolo sentencial = Símbolo inicial

- Símbolo sentencial: declaração `%start`
  - Na ausência da declaração, lado esquerdo da primeira produção é considerado o símbolo sentencial
- Se produção é recursiva, recursão à esquerda é preferível
- Expansões alternativas podem ser separadas com o símbolo |

YACC – Gerador de *Parser*

Tokens

Os símbolos terminais das produções podem ser representados de duas formas:

- nomes simbólicos, declarados com o comando `%token`, ou
- representação direta do caractere, se for um único caractere

YACC – Gerador de *Parser*

Tokens

Operadores

Símbolos terminais para operadores podem usar, ao invés de `token`, declaração com associatividade explícita:

%left	operador com associatividade à esquerda
%right	operador com associatividade à direita
%nonassoc	operador não-associativo

Estratégia para eliminar ambigüidade

- Operadores declarados na mesma linha têm mesma precedência
- Operadores declarados na última linha têm maior precedência

YACC – Gerador de *Parser*

- Exemplo

A gramática de expressões (soma ou multiplicação) é usada neste exemplo como a semente para uma calculadora:

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E \times E \\
E &\rightarrow (E) \\
E &\rightarrow v
\end{aligned}
$$

```
%start   entrada
%token   VALOR FIMLIN
%left    SOMA
%left    MULT
%token   ABRPAR FECPAR
```

YACC – Gerador de *Parser*

- Exemplo

A gramática de expressões (soma ou multiplicação) é usada neste exemplo como a semente para uma calculadora:

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E \times E \\
E &\rightarrow (E) \\
E &\rightarrow v
\end{aligned}
$$

%left SOMA Exemplo: A - B - C é interpretada como (A - B) - C
%right SOMA Exemplo: A - B - C é interpretada como A - (B - C)
%nonassoc SOMA Exemplo: A - B - C é interpretada como incorreta

(operador definido como não associativo, não permite encadeamento em uma expressão)

YACC – Gerador de *Parser*

Manipulação das sentenças reconhecidas

Ação semântica

- Código C cuja execução está associada à aplicação da produção
- Especificado entre chaves após a expansão da produção
- Pode referenciar o valor semântico de um não-terminal da expansão
  - Notação posicional: $1 é o primeiro símbolo, $2 o segundo...
  - Para tokens, analisador léxico deve definir esse valor por meio da variável `yylval`
  - Um valor semântico pode ser associado ao lado esquerdo ($$), para uso em outras expansões
  - Tipo padrão do valor semântico é `int`, mas pode ser alterado com a redefinição da *string* `YYSTYPE`

YACC – Gerador de *Parser*

Tratamento de erros

error	símbolo não-terminal pré-definido que está associado a sentenças não reconhecidas pela gramática especificada
yyerrok	macro definida em C que limpa a entrada de caracteres para que a análise possa prosseguir
yyerror()	função definida pelo usuário, invocada pelo analisador para apresentação de mensagens de erro

YACC – Gerador de *Parser*

Desenvolvimento de uma aplicação

A gramática de expressões (soma ou multiplicação) é usada neste exemplo como a semente para uma calculadora:

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E \times E \\
E &\rightarrow (E) \\
E &\rightarrow v
\end{aligned}
$$

YACC – Gerador de *Parser*

Exemplo

Primeira seção: declarações C/C++ (1)

```
%{
#include <iostream>
using namespace std;
    extern "C"
    {
            int yyparse(void);
            int yylex(void);
            int yywrap()  {
                    return 1;
            }

    }
void yyerror(char *);
%}
```

YACC – Gerador de *Parser*

Exemplo

Primeira seção: declarações yacc (2)

```
%start  entrada
%token  VALOR FIMLIN
%left   SOMA
%left   MULT
%token  ABRPAR FECPAR
%%
```

YACC – Gerador de *Parser*

Exemplo

Segunda seção: produções e ações (3)

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E \times E \\
E &\rightarrow (E) \\
E &\rightarrow v
\end{aligned}
$$

```
expr :   expr SOMA expr      {  $$ = $1 + $3; }
     |   expr MULT expr      {  $$ = $1 * $3; }
     |   ABRPAR expr FECPAR     {  $$ = $2;  }
     |   VALOR
     ;
```

YACC – Gerador de *Parser*

Exemplo

Segunda seção: produções e ações (4)

```
entrada :  /* vazia */
        |  entrada result
        ;
result :  FIMLIN
       |  expr FIMLIN
          { cout << "Resposta: " << $1 << endl; }
       |  error FIMLIN  {  yyerrok;  }
       ;
```

YACC – Gerador de *Parser*

Exemplo

Terceira seção: código C (5)

```
%%
void yyerror(char* msg) {
    extern char* yytext;
    cout << msg << ": " << yytext << endl;
}
```

YACC – Gerador de *Parser*

Integração com lex

- Arquivo de cabeçalho produzido com opção `-d`
- Definições de valores para os nomes simbólicos associados aos tokens, declarados na primeira seção do arquivo de especificação de yacc (`acumuly.y`)
- Execução com `bison` (implementação Gnu de yacc)

```
> bison -d acumuly.y
```

> bison –d –v -g acumuly.y (opção –v gera o arquivo acumuly.output, com informações sobre o *parser;* opção –g gera acumuly.dot, DFA que pode ser visualizado com KGraphEditor)

YACC – Gerador de *Parser*

Integração com lex

Exemplo: arquivo acumuly.l (6)

```
%{
#include "acumuly.tab.h"
extern YYSTYPE yylval;
%}
%%
[0-9]+ {  yylval = atoi(yytext);
          return VALOR;
        }
\+      { return SOMA; }
\*      { return MULT; }
\(      { return ABRPAR; }
\)      { return FECPAR; }
\n      { return FIMLIN; }
%%
```

YACC – Gerador de *Parser*

Geração da aplicação

- Analisador léxico é um programa C, produzido com `flex`

```
> flex acumuly.l
> gcc -c lex.yy.c
```

- Compilação do analisador sintático

```
> g++ -o acc lex.yy.o acumuly.tab.c -ly -lfl
```

Em algumas versões do Linux, usar somente a biblioteca -lfl

YACC – Gerador de *Parser*

Exemplo

Execução (7)

```
> ./acc
1+2+4*5
Resposta: 23
2+2
Resposta: 4
1=2+3+4
=syntax error: 2
1+2
Resposta: 3
[^d]
>
```

YACC – Gerador de *Parser*

Sugestões de leitura (Web)

- Bison: Gnu parser generator
  `http://www.gnu.org/software/bison/`

YACC – Gerador de *Parser*

- Bibliografia consultada

LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004. (Cap. 5)

RICARTE, I. **Introdução à Compilação.** Rio de Janeiro: Editora Campus/Elsevier, 2008. (Cap. 4)