Relatório sobre a Expansão da Gramática e Implementação do Analisador Sintático para a Linguagem Provol-One

Nome: Thiago Pereira Camerato

Matrícula: 2212580

Introdução:

Este relatório detalha as mudanças e expansões realizadas na gramática original da linguagem Provol-One, bem como a implementação correspondente no analisador sintático usando a ferramenta Yacc/Bison. O objetivo foi melhorar a expressividade e funcionalidades da linguagem, permitindo uma maior variedade de construções e garantindo uma boa geração de código em **Python**.

Expansões na Gramática:

A gramática original foi expandida para incluir novas construções e suportar operações com números. Abaixo estão as principais alterações e adições:

- **Inclusão de Operações com Números:**
  - Adição de **exp** (expressões) para possibilitar operações aritméticas e lógicas.
  - A inclusão dos tokens **<numval> para** representar números inteiros na linguagem.

exp : exp MAIS exp | exp MENOS exp | exp MULT exp | exp DIVID exp | ...

- **Comandos de Controle de Fluxo Aprimorados:**
  - Adição de **SE ... ENTAO ... (SENAO ...)? FIML** para permitir construções condicionais mais expressivas.
  - Introdução de **FACA cmds VEZES id FIML** para laços de repetição **for**.

- **Diferença entre FIML e FIMP:**
  - **FIML**: Marca o fim de um bloco de controle de fluxo (**ENQUANTO**, **SE**, **FACA**) e é utilizado para manter a estrutura indentada do código.
  - **FIMP**: Indica o fim do programa (**F IM**) e é utilizado para finalizar a execução do código.

Implementação:

A implementação seguiu as diretrizes padrão do Yacc/Bison, com adição de ações semânticas para construir a árvore de análise durante o parsing. A manipulação de diferentes tipos de valores foi realizada através do uso de **%union**.

- **Uso de %left e %nonassoc:**
- **%left**: Define a associatividade de operadores binários, como adição (**MAIS**) e subtração (**MENOS**).
- **%nonassoc**: Indica a não associatividade de operadores relacionais e de igualdade, garantindo a correta prioridade nas expressões.

Uso de NUM:

- Para garantir a correta manipulação de números, foi implementada uma ação semântica específica, alocando espaço suficiente e convertendo o valor numérico para string.

```
| NUM {
    int num_digits = snprintf(NULL, 0, "%d", $1);//calcula a qtd de digitos
    char *result = malloc(num_digits + 1);  // +1 para o caractere nulo de terminação
    snprintf(result, num_digits + 1, "%d", $1);//guarda a formatacao do num em string
    $$ = result;
}
```

Execução:

- Gerar o arquivo C a partir do código Yacc: **bison -d provol-one_yacc.y**.
- Compilar os arquivos C gerados juntamente com o analisador léxico (Flex/Bison): **gcc -o provol_one_parser lex.yy.c provol-one_yacc.tab.c -lfl**.
- Executar o programa gerado: **./provol_one_parser**.

Alguns exemplos de execução:

```
ENTRADA X,Y
SAIDA X
Y = 10
X = 0
ENQUANTO X<Y FACA
 INC(X)
FIML
FIMP
Código Fonte Resultante:
Y=10
X=0
while X<Y:
X+=1
```

```
ENTRADA X,Y
SAIDA Z
X = 0
Y =100
ENQUANTO X < Y FACA
 SE X == 0 ENTAO
    Z = 10
  FIML
 FIML
FIMP
Código Fonte Resultante:
X=0
Y=100
while X<Y:
if X==0:
Z=10
```

```
ENTRADA X,Y
SAIDA Z
X = 0
Y = 100
Z = Y/2
FACA INC(X)
VEZES Z FIML
SE X==Z ENTAO
 ZERA(Y)
FIML
FIMP
Código Fonte Resultante:
X=0
Y=100
Z=Y/2
for Z in range(Z):
X+=1
if X==Z:
Y=0
```

```
ENTRADA X,Y
SAIDA Z
X = 10
Y = X*9
 SE X < 10 ENTAO
   Z = X
SENAO
   Z = Y/12
FIML
FIMP
Código Fonte Resultante:
X=10
Y=X*9
if X<10:
Z=X
else:
Z=Y/12:
```

Conclusão:

A expansão da gramática e implementação do analisador sintático permitiram uma linguagem Provol-One mais expressiva, com suporte a operações numéricas, estruturas de controle de fluxo mais complexas.

Links utilizados para a realização do trabalho:

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiH2-egmfyCAxXTrpUCHSr6BLYQFnoECBEQAQ&url=https%3A%2F%2Fweb.iitd.ac.in%2F~sumeet%2Fflex__bison.pdf&usg=AOvVaw3Mz0lvK6H3Y-pG2-Wwuyqq&opi=89978449