UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO ACADÊMICO DE CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE GRADUAÇÃO DE CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO - COMPILADORES

Luis P. A. Afonso RA:77429
Vítor A. C. Silva RA:104409

MARINGÁ
2023

1. Introdução

Este relatório discute elementos da linguagem LPVC (criada pelos autores) e detalhes da implementação do compilador para essa linguagem, que foi feito até parte da análise semântica.

2. A linguagem

A linguagem que o compilador implementa é chamada de *"Luis Paulo & Vitor Cecilio" programming language,* as construções/elementos da linguagem estão na língua portuguesa e alguns elementos da sintaxe da linguagem foram baseados nas linguagens de programação C e Python.

2.1. Constructos da linguagem

O compilador reconhece constructos básicos para linguagens de programação. Todos os *statements* da linguagem devem terminar com um ';' (Ponto e vírgula) para sinalizar que aquele *statement* terminou. Caso um ';' não seja fornecido, será retornado um erro de sintaxe.

Exemplo de *Tokens* sobre tipos que são reconhecidos pelo compilador:

- Token *VAR_NUMERO:* Token para o tipo numérico da linguagem, é reconhecido pelo reconhecimento da palavra reservada 'numero';
- Token *VAR_TEXTO*: Token para o tipo textual (string/char) da linguagem, é reconhecido pelo reconhecimento da palavra reservada 'texto';
- Token *VAR_BOOLEANO:* Token para o tipo booleano, é reconhecido pela reconhecimento da palavra reservada 'booleano';

Exemplo de *Tokens* sobre valores literais que são reconhecidos pelo compilador:

- Token *NUMERO:* é reconhecido pela expressão regular `r'\d+(\.\d+)?'`;
- Token *TEXTO:* é reconhecido pela expressão regular `r'("[^"]*")|(\'[^\']*\')'`;
- Token *VERDADEIRO*: é reconhecido pelo reconhecimento da palavra reservada 'verdadeiro';
- Token *FALSO:* é reconhecido pelo reconhecimento da palavra reservada 'falso';

Exemplo de *Tokens* sobre operadores que são reconhecidos pelo compilador:

- Token operador MAIORIGUAL (>=): reconhecido pela expressão regular `r'\>\='`;
- Token operador MAIOR (>): reconhecido pela expressão regular `r'\>'`;
- Token operador MENORIGUAL (<=): reconhecido pela expressão regular `r'\<\='`;
- Token operador MENOR (<): reconhecido pela expressão regular `r'\<'`;
- Token operador IGUAL (==): reconhecido pela expressão regular `r'\=\='`;
- Token operador DIFER (<>): reconhecido pela expressão regular `r'\<\>'`;
- Token operador MAIS (+): reconhecido pela expressão regular `r'\+'`;
- Token operador MENOS (-): reconhecido pela expressão regular `r'\-'`;
- Token operador VEZES (*): reconhecido pela expressão regular `r'\*'`;
- Token operador DIVIDE (/): reconhecido pela expressão regular `r'\/'`;
- Token operador RECEBE (=): reconhecido pela expressão regular `r'\='`;

Exemplo de *Tokens* sobre pontuações (com significados especiais na linguagem) que são reconhecidos pelo compilador:

- Token pontuação LPAREN ‘(‘ : com expressão regular reconhecedora `r'\('`;
- Token pontuação RPAREN ‘)’ : com expressão regular reconhecedora `r'\)'` ;
- Token pontuação LCHAV ‘{‘ : com expressão regular reconhecedora `r'\{'`;
- Token pontuação RCHAV ‘}’ : com expressão regular reconhecedora `r'\}'`;
- Token pontuação PONTO_VIRGULA ‘;’ : com expressão regular reconhecedora `r'\;'`;
- Token pontuação VIRGULA ‘,’ : com expressão regular reconhecedora `r'\,'`;
- Token pontuação DOIS_PONTOS ‘:’ : com expressão regular reconhecedora `r'\:'`;

Quanto a parte da sintaxe (construções que são válidas em relação a combinação dos Tokens), temos os seguintes exemplos de *Statements* reconhecidos pelo compilador:

- Declaração de variáveis (tipo_variavel ID RECEBE expressao):
  - *numero primeiro = 3;*
  - *texto nome = ‘luis’;*
  - *booleano situacao = falso;*

- Redeclaração de variáveis (ID RECEBE expressao):
    - primeiro = 5;
    - nome = ‘vitor’;
    - situacao = verdadeiro;

- Declaração para procedimentos (PROCEDIMENTO ID LPAREN argumentos RPAREN DOIS_DOIS PONTOS tipo_variavel LCHAV outro_statement RETORNE expressao PONTO_VIRGULA RCHAV):

```
procedimento aniversario(numero dia, numero mes) : texto {
    numero primeiro = 1;
    texto segundo = 'segundo';
    texto terceiro = 'terceiro';

    retorne segundo + terceiro;
};
```

- Chamada de função Built-in imprima (IMPRIMA LPAREN expressao RPAREN):
    - imprima(3+1);
    - imprima(nome);

- Chamada de função Built-in leia (LEIA LPAREN variavel RPAREN):
    - leia(nome);
    - leia(primeiro);

- Laço de controle se/senão (SE condicao LCHAV outro_statement RCHAV SENAO LCHAV outro_statement RCHAV)

```
se 3>1 {
    numero data = 13;
} senao {
    procedimento outro_dia(numero minuto) : numero {
        retorne 8;
    };
};
```

- Laços de repetição:
    - while (ENQUANTO condicao LCHAV outro_statement RCHAV):

```
enquanto (3 > 1) {
    numero segundo = 2;

    imprima(primeiro);
    imprima(segundo);

    enquanto 3 > 2 {
        numero terceiro = 3;
        imprima(primeiro);
        imprima(segundo);
        imprima(terceiro);
    };
    #imprima(terceiro);
};
```

    - for (PARA expressao ATE expressao EM expressao LCHAV outro_statement RCHAV):

```
para 3 ate 10 em 1 {

    imprima('oi');

};
```

- Chamada de expressões:
    - 3+1;
    - primeiro + 20;
    - ‘Luis ‘ + ‘Paulo’; (concateção de strings)
    - 3 > 1;

3. Dificuldades e limitações

As principais dificuldades que surgiram ao desenvolvimento do compilador se deram principalmente na parte da análise semântica. Contudo, uma dificuldade que o grupo encontrou na parte da sintaxe foi em como elaborar a árvore sintática abstrata. A documentação do PLY (biblioteca utilizada) deixa claro que a geração da árvore é de responsabilidade dos programadores, assim, decidiu-se utilizar tuplas para a representação da

árvore. De forma que cada tupla que representa um nó tem o primeiro campo para o nome do nó e demais campos com base nos valores que tal nó deve ter.

Na parte da análise semântica, a maior dificuldade foi em resolver o problema de identificar o tipo correto para se avaliar em uma expressão. Outro ponto de dificuldade foi em projetar os espaços de endereço local e global. Também houve dificuldade para verificar semanticamente uma chamada de procedimento (de forma que ela possa ser usada como expressao). As dificuldades aqui discutidas foram resolvidas.

Uma limitação é mesmo que na análise semântica os tipos, operações e afins são verificados corretamente, os valores de expressões não são resolvidos/calculados (essa parte imagina-se que aconteceria durante a execução no RUN-TIME) uma variável ‘numero primeiro = 3+1;’ é armazenada com o valor ‘3+1’ e não como ‘4’. A parte de geração de código intermediária não foi feita.