FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

PROJETO

COMPILADORES

LICENCIATURA EM ENGENHARIA INFORMÁTICA

2022/2023

Ana Rita Martins Oliveira – 2020213684 – anaoliveira@student.dei.uc.pt

Hugo Sobral de Barros – 2020234334 – hugobarros@student.dei.uc.pt

Índice

Ana Rita Martins Oliveira - 2020213684
Hugo Sobral de Barros - 2020234332

Gramática Reescrita

A meta 2 consistia em fazer um analisador sintático, programado em C, utilizando as ferramentas lex e yacc, para a linguagem Juc, juntamente com uma gramática inicial ambígua escrita em notação EBNF.

Na notação EBNF, quando é apresentado {…}, significa que, existem 0 ou mais repetições do seu conteúdo, para reescrever esta gramática de modo que esta seja interpretada pelo yacc, como por exemplo:

FieldDecl → PUBLIC STATIC Type ID { COMMA ID } SEMICOLON

Traduzimos para:

```
FieldDecl:          PUBLIC STATIC Type ID FieldDecl1 SEMICOLON       {};

FieldDecl1:          /*Blank*/                                       {}
          |          COMMA ID FieldDecl1                             {};
```

Quando encontramos {…} separamos em 2 produções, A e B, em que B contém o que está dentro de {…} terminando em B, fazendo com que haja um "loop" e adicionamos o caso de não acontecer.

No caso de […], ou seja, o seu conteúdo é opcional, por exemplo:

Expr → ID [DOTLENGTH]

Traduzimos para:

```
Expr:         ID                                                    {}
    |         ID DOTLENGTH                                         {};
```

Quando encontramos um […] reproduzimos o mesmo efeito apenas fazendo duas produções, uma sem o que está dentro de […] e outra com o que está dentro de […].

Nesta situação:

Expr → INTLIT | REALLIT | BOOLLIT

É claro o objetivo, Expr é definido por INTLIT, ou REALLIT, ou BOOLLIT, por isso traduzimos para:

```
Expr:         INTLIT                                                {}
    |         REALLIT                                              {}
    |         BOOLLIT                                              {};
```

Algoritmos e estruturas de dados da AST e da tabela de símbolos

A Árvore de Sintaxe Abstrata é contruída por uma lista ligada, com a seguinte estrutura:

```
typedef struct no* node;
typedef struct no {
    node brother;
    node child;
    node parent;

    char* info;
    char* type;

    int n_children;

    char* anot;
    int print;
    int call;
} no;
```

A lista contém um "irmão" que será o nó que se encontra na linha abaixo na mesma coluna, um "filho" que será o nó que se encontra na linha abaixo e na coluna seguinte e um "pai" que será o nó que se encontra acima na coluna anterior.

Cada nó da lista tem um tipo e uma informação associada a esse tipo, por exemplo o nó "Id(factorial)" tem o tipo "Id" e a informação "factorial", porém o nó "Int" apenas possui o tipo "Int" e a sua informação fica vazia (""). Cada nó guarda o número de "filhos" que tem para o cálculo dos pontos antes de cada nó (……Id(factorial)).

Para anotar a AST é utilizado o campo "anot" para guardar o que vai ser anotado e posteriormente impresso, para este processo percorremos a AST.

Existem também duas variáveis para auxiliar o código: "print" e "call". Se a variável "print" estiver a 1, na função de imprimir a árvore anotada, será impresso também o conteúdo da variável "anot". Se a variável "call" estiver a 1, significa que o "pai" do nó é "Call".

Ana Rita Martins Oliveira - 2020213684
Hugo Sobral de Barros - 2020234332

As tabelas de símbolos são feitas através de listas ligadas. Optamos por utilizar as seguintes estruturas em C:

```
typedef struct no_tab* node_tab;
typedef struct no_tab{

    char* name;
    char* type;
    int is_method;
    int is_params;
    node_tab brother;
    node_tab vars;
    node_tab method;

}no_tab;

typedef struct table* tab;
typedef struct table{

    char* name;
    node_tab symbols;
    tab next;

}table;
```

Foram criadas variáveis para auxiliar o código, como “is_method” e “is_param”. Estas são utilizadas para diferenciar “Field Declarations” de “Method Declarations”, e para diferenciar argumentos de variáveis, respetivamente. Se o is_method for igual a 1, uma tabela de “Method” será criada com os valores do nó e se is_param for 1, os parâmetros do método serão impressos na linha do método.

Para a construção das tabelas, percorremos a AST, e sempre que é encontrado um nó que seja necessário adicionar à tabela, é adicionado com os atributos corretos.