

## **Especificação da Linguagem Quase**

### **1. Introdução**

A linguagem **Quase** é uma linguagem quase orientada a objetos com aspectos quase funcionais e imperativos.

Ela apresenta as seguintes características principais:

Tipos: int, real, bool

Classes: suporte a polimorfismo e herança

Atributos das classes: tipos primitivos e outras classes. Tempo de vida é o tempo de vida da classe

Métodos das classes: procedure e function

Estruturas de controle: while, if, if-else

Escopo das variáveis e constantes: local nos métodos

Comentários: bloco (estilo C).

**Quase**, obviamente, é uma linguagem experimental, então esta especificação é passível de adaptações. Em caso de modificações na especificação, estas serão notificadas via SIGAA.

### **2. Léxico**

- Comentários de bloco são delimitados por { e } e seguem a mesma regra do C (ou seja, não há aninhamento)
- Não há comentários de linha
- Identificadores de variáveis e constantes devem começar com letras, e conter apenas letras e underline (ID).
- Identificadores de classe devem começar com underline, e conter apenas letras e underline (CID).
- Números inteiros podem ser escritos em decimal ou binário. Em binário, começam com 0b.  
Exemplo: 0b1010 é o número decimal 10
- Números reais podem ser escritos em decimal ou notação científica (com E ou e). Os valores em decimal devem ser separados por ponto.  
Exemplo:  $5e-3 = 5 \cdot 10^{-3} = 0.005$

Vide Seção 3 para mais informações sobre as classes de tokens a serem implementadas nesta etapa.

### 3. Sintático

A gramática abaixo foi escrita em uma versão de E-BNF seguindo as seguintes convenções:

- 1 - variáveis da gramática são escritas em letras minúsculas sem aspas.
- 2 – lexemas que correspondem diretamente a tokens são escritos entre aspas simples
- 3 - símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado.
- 4 - o símbolo | indica produções diferentes de uma mesma variável.
- 5 - o operador [ ] indica uma estrutura sintática opcional.
- 6 - o operador { } indica uma estrutura sintática que é repetida zero ou mais vezes.

```
programa : familia def_classe {def_classe}

familia: relação {'&' relação} ';' | ε
relação: 'classe' CID 'filha da classe' CID

def_classe : 'classe' CID 'começa' atributos metodos 'termina'

atributos : { dec_obj | dec_var | dec_cons }
métodos: { dec_procedimento | dec_funcao }

dec_obj : 'objeto' CID ID { ',' ID } ';'
dec_var : 'var' tipo ID { ',' ID } ';'
dec_cons : 'cons' tipo_primitivo inicializacao { ',' inicializacao } ';'
inicializacao : ID ':=' exp

tipo : tipo_classe | tipo_primitivo
tipo_primitivo : 'int' | 'bool' | 'real'
tipo_classe: CID

dec_procedimento: ['=>'] 'procedimento' ID '(' parametros ')' comando
dec_funcao : 'função' tipo ID '(' parametros ')' exp
parametros : ε | parametro { ',' parametro }
parametro : tipo ID
comando : 'se' '(' exp ')' comando
          | 'se' '(' exp ')' comando 'senão' comando
          | 'enquanto' '(' exp ')' comando
          | ID '=' exp ';'
          | ID '.' chamada ';'
          | bloco
bloco : 'começa' { dec_var | dec_cons } { comando } 'termina'
exp : NUMERAL | 'true' | 'false'
    | ID
    | ID '.' chamada ';'
    | ID '.' atributo ';'
    | '(' exp ')'
    | '-' exp
    | 'se' '(' exp ')' 'então' exp 'senão' exp
    | exp '+' exp
    | exp '-' exp
    | exp '*' exp
    | exp '/' exp
    | exp '%' exp
    | exp '==' exp
```

```
| exp '<' exp
| '!' exp
| exp 'and' exp
| exp 'or' exp
| bloco_exp
bloco_exp : 'começa' { dec_cons } exp 'termina'
chamada : ID '(' lista_exp ')'
lista_exp :  $\epsilon$  | exp { ',' exp }
```

#### 4. Semântico:

- O ponto de entrada do programa é o procedimento precedido pelos caracteres ‘=>’. No máximo um procedimento deve ter essa marcação em um programa. A execução fictícia de um programa em **Quase** é iniciada com a criação de uma instância da classe onde o procedimento marcado está declarado e a chamada a este procedimento. Um programa sem esta marcação gera uma biblioteca após a compilação.

- Não há modificadores de acesso nas classes. Por padrão, todos os métodos são públicos e todos os atributos são privados.

- Classes que não são declaradas como filhas de nenhuma outra são implicitamente filhas da classe `_Ancestral`. A classe `_Ancestral` não pode ser instanciada.

- Não há variáveis compostas homogêneas (arrays).

- A prioridade dos operadores é igual à de C.

- Em operações entre os tipos `int` e `float`, os valores `int` devem ser convertidos para `float`.

- Variáveis só podem ser usadas se forem inicializadas.

- Parâmetros sempre são passados por cópia (valor) em funções e procedimentos.

- Existe uma classe `IO` com dois métodos pré-definidos:

- `print`: procedimento que recebe como argumento uma expressão e a imprime em tela.

- `read()`: função que **retorna** o valor inserido pelo usuário no teclado

Obs: Assumam que os dois métodos são compatíveis com os tipos primitivos (`int`, `real` ou `boolean`)

**A definir na próxima versão:** o que deve ser verificado na análise semântica.

#### 5. Geração de Código

- O código alvo do compilador é C

#### 6. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio (preferencialmente), dupla ou individualmente. Foi aberto um fórum no SIGAA para a discussão sobre as etapas. Em caso de dúvida, verifique inicialmente no fórum se ela já foi resolvida. Se ela persiste, consulte a professora.

##### 6.1. Ferramentas

- Submissão das etapas do projeto via SIGAA. Será criada uma ou mais tarefas para cada etapa.

- Implementação com SableCC e Java.

## 6.2. Avaliação

- A avaliação será feita com base nas etapas entregues e em entrevistas feitas com os grupos.
- A aula que define o prazo de entrega de cada etapa está especificada no plano de curso da disciplina.

## 6.3. Etapas

Análise Léxica (valor: 2.5)

- Tarefa 1: três códigos em **Quase** que, unidos, usem todas as alternativas gramaticais (ou seja, todos os recursos) da linguagem.
- Tarefa 2: Analisador léxico em SableCC, com impressão dos lexemas e tokens reconhecidos ou impressão dos erros

Análise Sintática (valor: 2.5):

- Tarefa 1: analisador sintático em SableCC, com impressão da árvore sintática em caso de sucesso ou impressão dos erros

Sintaxe Abstrata (valor: 2.5):

- Tarefa 1: analisador sintático abstrato em SableCC, com impressão da árvore sintática

Análise Semântica (valor: 2.5):

- Tarefa 1: criação e impressão da tabela de símbolos e da árvore de relacionamento entre classes
- Tarefa 2: validação de escopo e de existência de identificadores. Verificação de tipos

Geração de código (extra: 2.0):

- Código em linguagem alvo

A critério da docente o valor das etapas pode ser modificado, desde que este novo cálculo produza uma nota não menor que a produzida pelo cálculo original. Entregas após o prazo sofrem penalidade de meio ponto por dia de atraso.

**Bom trabalho!**