Disciplina Compiladores - UFS
Turma T01 2019.1
Profa. Beatriz Trinchão Andrade

Especificação da Linguagem Frank

1. Introdução

A linguagem **Frank** (alusão ao Monstro[1] do Dr. Frankenstein, personagem da escritora Mary Shelley) é um remendo de linguagens de programação. C, Pascal, Javascript, Haskell... há algo de cada aqui.

Ela apresenta as seguintes características principais:

Tipos: int, real, bool
Tipos compostos: arrays (arrays de arrays são permitidos);
Estruturas de controle: while, if, if-else;
Escopo das variáveis: global e local;
Subrotinas: procedure e function;
Comentários: linha ou bloco (estilo C).

Frank, 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 similares aos do C, delimitados por /* e */
- Comentários de linha devem ser iniciados com um //
- Identificadores seguem as mesmas regras de formação da linguagem C
- Números inteiros podem ser escritos em decimal ou em hexadecimal (começando com 0x)
- 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*10^-3 = 0.005

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

1 O Frankenstein na verdade é o cientista que criou o monstro.

Um pensamento: o livro Frankenstein é um clássico, recomendo pra quem gostar do gênero. No entanto, o Dr. Victor Frankenstein não é um bom exemplo de cientista. O título de doutor pesa ainda mais no estereótipo. Há muitos personagens assim na ficção; já se perguntaram por quê?

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 : { dec_var | dec_cons}  { dec_procedimento | dec_funcao }
dec_var : 'var' tipo (ID | inicializacao) { ',' (ID | inicializacao) } ';'
dec_cons : 'cons' tipo inicializacao { ',' inicializacao } ';'
inicializacao : ID := ( exp | array_ext | array_comp )
array_ext : '[' exp { ',' exp } ']'
array_comp : '[' ID '|' (array_comp | exp) ']'

tipo : tipo_base | tipo '[' exp ']'
tipo_base : 'int' | 'bool' | 'real'
dec_procedimento: 'procedure' ID '(' parametros ')' comando
dec_funcao : 'function' tipo ID '(' parametros ')' exp
parametros : Ɛ | parametro { ',' parametro }
parametro : [ 'var' ] tipo_parametro ID
tipo_parametro :  tipo_base | tipo_parametro '[' ']'
comando : 'if' '(' exp ')' comando
        | 'if' '(' exp ')' comando 'else' comando
        | 'while' '(' exp ')' comando
        |  var ':=' exp ';'
        |  chamada ';'
        |  bloco
bloco : '{' { dec_var | dec_cons} comando } '}'
var : ID | var '[' exp ']'
exp : NUMERAL | true | false
      | var
      | '(' exp ')'
      | chamada
      | '-' exp
      | 'if' '(' exp ')' 'then' exp 'else' 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 : '{' { dec_cons } exp '}'
chamada : ID '(' lista_exp ')'
lista_exp : Ɛ | exp { ',' exp }
```

4. Semântico:

- Variáveis podem ser inicializadas durante a declaração ou depois dela, com uma atribuição.

- A execução de um programa consiste na execução de um procedimento com nome main.

- 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.

- Por padrão, parâmetros são passados por cópia em funções e procedimentos. Para fazer a passagem de um parâmetro por referência, é necessário colocar a palavra reservada 'var' antes do parâmetro.

- Arrays estáticos podem ser inicializados por extenso, seguindo a sintaxe **`array_ext`**, ou usando uma *comprehension*, seguindo a sintaxe **`array_comp`**.
Como exemplo de inicialização com *comprehension* temos:

```
var int[10] a = [ i | i+1 ]
```

onde cada elemento `a[i]` é inicializado com `i+1`, com `i` variando de `0` a `9`.

- Existe uma função pré-definida `size` que recebe um array como argumento e retorna o número total de elementos deste array.

- Existe um procedimento pré-definido `print`, que recebe como argumento uma expressão e a imprime em tela.

- Existe um procedimento pré-definido `read` com um parâmetro passado por referência. O parâmetro pode ser do tipo int, real ou bool.

- Escopo das variáveis e constantes: global e local.

O que deve ser verificado na análise semântica:

- Se o nome do programa e as entidades criadas pelo usuário são inseridos na tabela de símbolos - com os atributos necessários - quando são declarados;
- Se uma entidade foi declarada e está em um escopo válido no momento em que ela é utilizada (regras de escopo são iguais às de C);
- Se entidades foram definidas (inicializadas) quando isso se fizer necessário;
- Checar a compatibilidade dos tipos de dados envolvidos nos **comandos**, **expressões e atribuições**.

5. Geração de Código

- O código alvo do compilador é C

6. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio, 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 (PasCript | Frank) 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
- 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!