Disciplina Compiladores - UFS
Turmas T01 e T02 2022.1
Profa. Beatriz Trinchão Andrade

Especificação da Linguagem Sarapatel

1. Introdução

A linguagem **Sarapatel** é uma mistura de linguagens de programação. Haskell, C, Pascal, ADA, Javascript... há um pedacinho de cada aqui.

Apesar da mistura, a linguagem **Sarapatel** é imperativa. Ela apresenta as características descritas neste documento.

Sarapatel, obviamente, é uma linguagem experimental, então esta especificação é passível de adaptações. Em caso de modificações na especificação, versões atualizadas serão postadas via SIGAA (no tópico de aula "Definições do Projeto") e notificações serão enviadas aos alunos.

As Seções 2, 3, 4 e 5 deste documento apresentam a especificação da linguagem. A Seção 6 contém informações sobre a avaliação e entregas, e o arquivo exemplo.srptl (anexo à aula do SIGAA) contém um exemplo de código `.srptl`.

2. Características e léxico

Regras para identificadores:

- Segue as mesmas regras do C, e podem conter números, letras maiúsculas, letras minúsculas e *underscore* ('_'). O primeiro caractere não pode ser um número.
- Não são permitidos espaços em branco e caracteres especiais (ex.: @, $, ç, +, -, ^, % etc.).
- Identificadores não podem ser iguais às palavras reservadas ou operadores da linguagem.

Tipos primitivos:

- A linguagem aceita os tipos inteiro, real, booleano e cadeia.
- Números inteiros podem ser escritos em decimal ou em binário (começando com 0b).
- A parte decimal dos números reais deve ser separada por uma vírgula.
- Booleanos podem assumir dois valores: `verdade` e `falso`.
- Cadeias são escritas entre aspas simples e podem conter quebra de linha. Exemplos:

```
'oi'
'olha a quebra de linha: \n'
```

Matrizes:

- Matrizes são compostas por uma ou mais variáveis com o mesmo tipo.
- O tamanho das matrizes é um valor inteiro, definido durante sua criação.
- Os índices das matrizes vão de 0 a tam-1.
- Exemplos de sintaxe:

```
var: inteiro[3] matriz_1D;
var: inteiro[0b11][0b10] matriz_2D;
```

Blocos:

- Blocos são delimitados pelas palavras `comece` e `termine`.

Comentários:

- A linguagem aceita comentários de linha, indicados pelo símbolo -- no início da linha.
- A linguagem aceita comentários de bloco (possivelmente de múltiplas linhas) delimitados por { e }.
- O funcionamento dos comentários de bloco em Sarapatel são similares aos da linguagem C. Exemplo: o que acontece se você compilar /* */ */ em C? E isso: /* /* */? Estudem como um compilador C reconhece o fim de um comentário de bloco.

Estruturas de controle:

- enquanto
- se
- se/senao

Operadores:

- Operadores aritméticos: +, -, *, /, %
- Operadores relacionais: >, >=, =
- Operadores booleanos: ‘`e`’ ‘`nao`’ ‘`ou`’.
- Operador ternário `se`. Este operador só existe quando é usado como expressão.
- A prioridade dos operadores é igual à de C, e pode ser alterada com o uso de parênteses.
- Atribuição de valores é feita com o operador <<
- Comandos são terminados com ; (ponto e vírgula).

Subrotinas:

- Procedimentos: sem retorno de valor, aceitam zero ou mais argumentos de entrada.
- Funções: com retorno de valor, aceitam zero ou mais argumentos de entrada. Não podem conter declarações de variáveis, e contêm obrigatoriamente a expressão que corresponde ao seu retorno de valor.
- Os parâmetros e argumentos são separados pelo símbolo |.
- Exemplo de procedimento:

```
procedimento: ola_mundo()
comece
     imprima(‘Olá, mundo!’);
termine
```

- Exemplos de função (a palavra reservada não possui acento):

```
funcao: inteiro soma(inteiro a | inteiro b)
comece
  a+b
termine


funcao: booleano eh_par(inteiro numero)
comece
      --operador se ternário
      se (numero%2=0)
            verdade
      senao
            falso
termine
```

  Vide Seção 3 para mais informações sobre a sintaxe dos comandos e sobre as classes de tokens a serem implementadas.

3. Sintático

A gramática da linguagem, descrita a seguir, utiliza as seguintes convenções:

- Variáveis da gramática são escritas em letras minúsculas sem aspas.
- Lexemas que correspondem diretamente a tokens são escritos entre aspas simples.
- Símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado.
- O símbolo | indica alternativa.
- O operador * indica uma estrutura sintática que é repetida zero ou mais vezes.
- O operador + indica uma estrutura sintática que é repetida uma ou mais vezes.
- O operador ? indica uma estrutura sintática opcional.

```
programa : ( dec_var | dec_cons)*  ( dec_procedimento | dec_funcao )+
dec_var : 'var' ':' tipo (ID | id_atribuição) ';'
dec_cons : 'const' ':' tipo id_atribuição ';'

tipo : tipo_base | tipo '[' exp ']'
tipo_base : 'inteiro' | 'booleano' | 'real' | 'cadeia'

id_atribuição : ID '<<' exp
array_comp : '[' ID '|' exp ']'

dec_procedimento: ('>>')? 'procedimento' ':' ID '(' parametros ')' comando
dec_funcao : 'funcao' ':' tipo ID '(' parametros ')' exp

parametros : parametro ( '|' parametro )*
           | Ɛ
parametro : tipo_parametro ID
tipo_parametro :  tipo_base | tipo_parametro '[' ']'

comando : 'se' '(' exp ')' comando
        | 'se' '(' exp ')' comando 'senao' comando
        | 'enquanto' '(' exp ')' comando
        |  id_atribuição ';'
        |  chamada_procedimento ';'
        |  bloco_comandos

bloco_comandos : 'comece' ( dec_var | dec_cons)* ( comando )* 'termine'

valor : ID | valor '[' exp ']'

exp : INTEIRO | REAL | CADEIA | BOOLEANO
     | valor
     | '(' exp ')'
     | chamada_funcao
     | array_comp
     | '-' exp
     | 'se' '(' exp ')' exp 'senao' exp
     | exp '+' exp
     | exp '-' exp
     | exp '*' exp
     | exp '/' exp
     | exp '%' exp
     | exp '=' exp
     | exp '>=' exp
     | exp '>' exp
     | 'nao' exp
     | exp 'e' exp
     | exp 'ou' exp
     | bloco_expressoes
bloco_expressoes : 'comece' ( dec_cons )* exp 'termine'
chamada_funcao : ID '(' lista_exp ')'
chamada_procedimento : ID '(' lista_exp ')'
lista_exp : Ɛ | exp ( '|' exp )*
```

4. Semântico:

- Variáveis são identificadas com o prefixo **`var :`** , e podem ser inicializadas durante a declaração ou depois dela, com uma atribuição. Variáveis só podem ser usadas se forem inicializadas.

- Constantes são identificadas com o prefixo **`const :`** , e podem ser inicializadas apenas durante a declaração.

- Escopo das variáveis e constantes: global e local.

- A execução de um programa consiste na execução de um procedimento de partida, precedido pelo símbolo **`>>`**. A inexistência de um procedimento de partida não impede a compilação, mas gera um *warning*.

- A prioridade dos operadores é igual à de C.

- Em operações entre os tipos inteiro e real, os valores inteiros devem ser convertidos para reais.

- Por padrão, parâmetros são passados por cópia em funções e procedimentos.

- Vetores podem ser inicializados usando uma *comprehension*, seguindo a sintaxe **`array_comp`**.

Exemplos de inicialização com *comprehension*:

```
var: inteiro [10] a << [ i | i+1 ]
```

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

```
var: inteiro [10][5] a << [ i | [ j | i+j ] ]
```

onde cada elemento `a[i][j]` é inicializado com `i+j`, com `i` variando, por padrão, de `0` a `9, e j de 0 a 4`.

-Existem três subrotinas pré-definidas:

- **`imprima()`**: procedimento que recebe como argumento uma expressão e a imprime em tela.
- **`leia()`**: função que retorna o valor inserido pelo usuário no teclado. Este valor precisa ser de um tipo primitivo da linguagem (inteiro, real, booleano ou cadeia).
- **`size()`**: tamanho(): função que recebe um vetor (matriz unidimensional) como argumento e retorna o número total de elementos deste vetor.

- O que checar na análise semântica:

- Se entidades definidas pelo usuário (variáveis, vetores e funções) são inseridas na tabela de símbolos - com os atributos necessários - quando são declaradas;
- 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 quando isso se fizer necessário;
- Checar a compatibilidade dos tipos de dados envolvidos nos **comandos**, **expressões**, **atribuições** e **chamadas de função**.

5. Geração de Código

- O código alvo do compilador é C ou comandos compatíveis com o gerador de código do LLVM.

6. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio (preferencialmente), dupla ou individualmente. Os grupos devem se cadastrar na planilha correspondente à turma dos seus componentes:

TURMA 01:
https://docs.google.com/spreadsheets/d/1p3eJSZoVXlFRJY6AQ-DRol4AQ7qfgKI6vQ53Llh8U58/edit?usp=sharing

TURMA 02
https://docs.google.com/spreadsheets/d/1l0UGM1dhSjeg4RrKxO35JXd30VN1aJLfaIMHGrsNnbs/edit?usp=sharing

Serão abertos fóruns 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

- Implementação com SableCC, linguagem Java.
- IDE Java (recomendação: Eclipse).
- Submissão das etapas do projeto via SIGAA. Foi criada uma tarefa para cada etapa.

6.2. Avaliação

- A avaliação será feita com base nas etapas entregues e em entrevistas feitas com os grupos durante as aulas de acompanhamento do projeto.
- O valor de cada etapa está definido no plano de curso da disciplina.
- O cumprimento das requisições de formato também será avaliado na nota de cada etapa.

6.3. Etapas

Etapa 1. Códigos Sarapatel
- **Prazo: 25/07/2022**
- **Atividade:** escrever três códigos em Sarapatel que, unidos, usem todas as alternativas gramaticais (ou seja, todos os recursos) da linguagem.
- **Formato de entrega:** arquivo comprimido contendo três códigos, onde cada código deve estar escrito em um arquivo de texto simples, com extensão “.srptl”.

Etapa 2. Análise Léxica
- **Prazo: 10/08/2022**
- **Atividade:** implementar analisador léxico da linguagem com o SableCC, fazendo a impressão dos lexemas e tokens reconhecidos ou imprimindo erro quando o token não for reconhecido.

- **Formato de entrega:** apenas o arquivo .sable deve ser enviado. O nome do arquivo deve ser `grupo_X.sable`, onde `X` é o número do grupo (vide planilha de cadastro de grupos).
  O nome do pacote a ser gerado pelo sablecc deve se chamar `sarapatel` (em letras minúsculas).

Etapa 3. Análise Sintática
- **Prazo: 03/10/2022**
- **Atividade:** implementar analisador sintático da linguagem com o SableCC, fazendo impressão da árvore sintática em caso de sucesso ou impressão dos erros caso contrário.
- **Formato de entrega:** apenas o arquivo .sable deve ser enviado. O nome do arquivo deve ser `grupo_X.sable`, onde `X` é o número do grupo (vide planilha de cadastro de grupos).

O nome do pacote a ser gerado pelo sablecc deve se chamar `sarapatel` (em letras minúsculas).

Etapa 4. Análise Sintática Abstrata
- **Prazo: 17/10/2022**
- **Atividade:** implementar analisador sintático abstrato em SableCC, com impressão da árvore sintática.
- **Formato de entrega:** apenas o arquivo .sable deve ser enviado. O nome do arquivo deve ser `grupo_X.sable`, onde `X` é o número do grupo (vide planilha de cadastro de grupos).
  O nome do pacote a ser gerado pelo sablecc deve se chamar `sarapatel` (em letras minúsculas).

Etapa 5. Tabela de Símbolos
- **Prazo: 31/10/2022**
- **Atividade:** implementação da tabela de símbolos (a tabela deve ser impressa no console, durante a execução do projeto, sempre que é atualizada).
- **Formato de entrega:** projeto completo, incluindo obrigatoriamente: o arquivo .sable; todas as classes java escritas pelo grupo ou geradas automaticamente; e arquivos .srptl que demonstrem o que foi feito nesta tarefa.
  Também é obrigatória a entrega de um pdf contendo uma breve explicação sobre o que foi implementado nesta etapa e como.

Etapa 6. Análise Semântica
- **Prazo: 16/11/2022**
- **Atividade:** validar escopo, declaração e definição de identificadores. Implementar verificação de tipos.
- **Formato de entrega:** projeto completo, incluindo obrigatoriamente: o arquivo .sable; todas as classes java escritas pelo grupo ou geradas automaticamente; e arquivos .srptl que demonstrem o que foi feito nesta tarefa.
  Também é obrigatória a entrega de um pdf contendo uma breve explicação sobre o que foi implementado nesta etapa e como.

Etapa 7. Geração de código (extra: 2 pontos na segunda unidade):
- **Prazo: o mesmo da Etapa 6**
- Compilação de código Sarapatel com geração de código em linguagem alvo (C ou LLVM)

Entregas após o prazo sofrem penalidade de metade da nota da etapa por dia de atraso.

- Trabalhos entregues com atraso devem ser submetidos na Tarefa ‘Entrega após prazo’, no SIGAA, que ficará aberta durante todo o período. Arquivos enviados por e-mail não serão considerados.

Bom trabalho!