Disciplina Compiladores - UFS
Turma T01 2024.1
Profa. Beatriz Trinchão Andrade

Especificação da Linguagem Caju

1. Introdução

Com tantas linguagens de programação em inglês na computação, neste período vamos criar um compilador para uma linguagem de programação em português. A linguagem **Caju** é uma linguagem imperativa, e apresenta as características descritas neste documento.

Caju, 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 “Apresentação do Projeto”) e notificações serão enviadas aos alunos e alunas.

As Seções 2, 3, 4 e 5 deste documento apresentam a especificação da linguagem. A Seção 6 contém informações sobre a avaliação e entregas.

2. Características principais e léxico

Regras para identificadores:

- Pode-se utilizar: números, letras maiúsculas, letras minúsculas e *underscore* ('_').
- O primeiro caractere deve ser sempre uma letra.
- Não são permitidos espaços em branco, acentos (por motivos de portabilidade do projeto) e caracteres especiais (ex.: á, ê, @, $, +, -, ^, % etc.).

Identificadores não podem ser iguais às palavras reservadas ou operadores da linguagem.

Tipos primitivos:

- A linguagem aceita os tipos caractere, booleano, e numero.
- Caracteres são escritos com aspas simples. Exemplo: 'a', '\n'.
- Booleanos podem assumir dois valores: verdadeiro e falso.
- O tipo numero representa valores inteiros e reais. A parte decimal dos números reais deve ser separada por uma vírgula. Os valores inteiros são expressos no sistema numérico decimal.

Vetores:

- Um vetor é composto de uma ou mais variáveis com o mesmo tipo primitivo.
- O tamanho dos vetores é definido durante sua criação.
- Os índices dos vetores vão de 0 a tam-1.
- Existem vetores multidimensionais. Exemplo: **`vetor numero [2][3] nome`**;
- Vetores unidimensionais de caracteres podem ter seus valores definidos por cadeias entre aspas duplas (“ e ”).

Blocos

- Blocos são delimitados pelas palavras **inicio** (sem acento) e **fim**.

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 Caju são similares aos da linguagem C. Exemplo: o que acontece se você compilar /**/*/ em C? Estudem como um compilador C reconhece o fim de um comentário de bloco.

Estruturas de controle (vide funcionamento na Seção Semântico):

- enquanto
- para
- para cada
- se/senao (sem acento)

Subrotinas:

- Funções com parâmetros e retorno de valor. O uso do comando retorne é obrigatório no corpo da função quando o seu retorno for diferente de vazio. Caso seja vazio, pode-se omitir esse comando ou usar “retorne ;”

Operadores:

- Operadores aritméticos: +, -, *, /
- Operadores relacionais: >, <, >=, <=, =
- Operadores booleanos: 'nao', 'e' e 'ou'.
- 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-final).

Palavras reservadas:

- caractere, numero, booleano, vetor, retorne, vazio, inicio, fim, se, senao, enquanto, para, para cada, verdadeiro, falso
- Todos os operadores e divisores da linguagem.

Procedimentos primitivos:

- A linguagem possui dois procedimentos primitivos: **`exibir`** e **`ler`**
- Podem ser usados sem necessidade de definição ou importação.

3. Sintático

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

- Variáveis da gramática são escritas em letras minúsculas sem aspas;
- Tokens são escritos entre aspas simples;
- Símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado;

- O símbolo | indica produções diferentes de uma mesma variável;
- O operador [ ] indica uma estrutura sintática opcional;
- O operador { } indica uma estrutura sintática que é repetida zero ou mais vezes.

```
programa : {dec-variavel} {dec-funcao}
dec-variavel : tipo lista-nomes '.'
lista-nomes : ID { ',' ID }
tipo : tipo-base | 'vetor' tipo-base '[' exp ']' {'[' exp ']'}
tipo-base : 'numero' | 'caractere' | 'booleano'
dec-funcao : ['->'] tipo-retorno ID '(' parametros ')' bloco
tipo-retorno : tipo | 'vazio'
parametros : ε | parametro { '|' parametro }
parametro : tipo ID
bloco : 'inicio' { dec-variavel } { comando } 'fim'
atrib :  var ':=' exp
lista-atrib: atrib {, atrib }
comando :
       'se' '(' exp ')' comando
     | 'se' '(' exp ')' comando 'senao' comando
     | 'enquanto' '(' exp ')' comando
     | 'para' '(' lista-atrib ';' exp ';' lista-atrib  ')' comando
     | 'para cada' '(' tipo ID ':' ID ')' comando
     |  atrib '.'
     |  'retorne' [ exp ] '.'
     |  bloco
     |  chamada '.'
var : ID | var '[' exp ']'
exp : NUMERO | CARACTERE | BOOLEANO
     | var
     | '(' exp ')'
     | chamada
     | exp '+' 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
chamada : ID '(' lista-exp ')'
lista-exp : ε | exp { '|' exp }
```

4. Semântico:

Geral

- Nos casos omissos neste documento, a semântica da linguagem segue a semântica de C.
- A execução de um programa consiste na execução de uma função marcada com `->`
- Nos casos omissos neste documento, a semântica da linguagem segue a semântica de C.

Blocos

- Contêm comandos e declarações
- Podem ser aninhados
- Escopo: o tempo de vida dos identificadores declarados em um bloco é igual ao tempo de vida deste bloco

Espaços de memória

- Não há constantes na linguagem, apenas variáveis. Variáveis podem ser inicializadas apenas em comandos posteriores à sua declaração, com uma atribuição (operador :=) ou através do comando ler. Variáveis só podem ser usadas se forem inicializadas.

Estruturas de controle

- O **`para cada`** tem funcionamento similar ao comando for each do Java.
  Como exemplo, o comando '**`para cada ( tipo elemento : vet )`**' faz com que a cada iteração deste laço, **`elemento`** receba um dos valores armazenados em **`vet`**, do índice 0 até o índice tamanho-1.
  `vet` deve obrigatoriamente conter elementos do tipo **`tipo`**
- As outras estruturas são similares às suas equivalentes em C, lembrando que somente podem ser usados operadores desta linguagem (exemplo: não temos o operador unário ++ para as estruturas para e para-cada).

Operadores**:**

- A prioridade dos operadores é igual à de C, e pode ser alterada com o uso de parênteses.

Procedimentos primitivos:

- **`ler:`** procedimento fictício para entrada de dados a partir do teclado. Salva os valores lidos nas variáveis que foram passadas como argumentos.
- **`exibir:`** procedimento para exibição de um ou mais valores resultantes de expressões passadas como argumentos.

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**.

6. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em grupos de 4 alunos (preferencialmente), trios, duplas ou individualmente. Os grupos devem se cadastrar na planilha:

https://docs.google.com/spreadsheets/d/1rzUuA-kYhkis7ux3Yon4iJJXB8DgUEQ8S-psUmnyGR8/edit?usp=sharing

Prazo de preenchimento da planilha: 26/07/2024.

A submissão das tarefas do projeto deve ser feita via SIGAA, nas datas previstas na Seção 6.3. 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

- Implementação com SableCC, linguagem Java.
- IDE Java (recomendação: Eclipse).
- Submissão via SIGAA.

6.2. Avaliação

- A avaliação será feita com base nas etapas entregues e em arguições feitas com os grupos.
- 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. Entregas

Etapa 1. Códigos em Caju

- **Prazo: 29/07/2024**
- **Atividade:** escrever três códigos em **Caju** 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 "**.cj**".

Etapa 2. Análise Léxica

- **Prazo: 12/08/2024**
- **Atividade:** implementar analisador léxico em 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 **`caju`** (em letras minúsculas e sem hífen).

Etapa 3. Análise Sintática

- **Prazo: 16/09/2024**
- **Atividade:** implementar analisador sintático em 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 **`caju`**.

Etapa 4. Sintaxe Abstrata

- **Prazo: 30/09/2024**
- **Atividade:** implementar analisador sintático abstrato em SableCC, com impressão da árvore sintática resultante.
- **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 **`caju`**.

Etapa 5. Tabela de Símbolos e Análise Semântica

- **Prazo: 14/10/2024**
- **Atividade:** validar escopo, declaração e definição de identificadores. Implementar verificação de tipos. Implementar e usar tabela de símbolos.
- **Formato de entrega:** projeto completo, incluindo obrigatoriamente: o arquivo .sable; todas as classes java escritas pelo grupo ou geradas automaticamente; e arquivos **.cj** 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. Geração de código (extra: 2.0 pontos):

- **Prazo: o mesmo da Etapa 6**
- Compilação de código **Caju** com geração de código em linguagem alvo (C)

Observações importantes:

- 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!