Disciplina Compiladores - UFS
Turma T01 2020.2
Profa. Beatriz Trinchão Andrade

Especificação da Linguagem Prefixa

Introdução

A linguagem **Prefixa** é uma linguagem imperativa, e apresenta as características descritas neste documento.

Esta 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.

1. Características e léxico

Regras para identificadores:

- Pode-se utilizar: letras maiúsculas, letras minúsculas e *underscore* ('_').
- O primeiro caractere deve ser sempre uma letra.
- Identificadores não podem ser iguais às palavras reservadas ou operadores da linguagem.

Tipos primitivos:

- A linguagem aceita os tipos caractere, booleano, inteiro.
- A linguagem não implementa o tipo real.
- Caracteres são escritos com aspas simples. Exemplo: **'a'**, **'\n'**.
- Booleanos podem assumir dois valores: **true** e **false**.
- Os valores inteiros podem ser expressos nos sistemas numéricos binário (começando com 0b ou 0B), hexa (começando com 0X ou 0x) e decimal.
- Exemplos: **0B10**, **0x1A**, **8**

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.
- Nesta linguagem, não existem vetores com mais de uma dimensão.
- Vetores de caracteres podem ter seus valores definidos por cadeias entre aspas duplas (**“** e **”**).

Blocos

- Blocos são delimitados pelas palavras **start** e **end**.

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

Estruturas de controle (vide funcionamento na Seção Semântico):

- **while**
- **for each**
- **if-else**

Subrotinas:

- Funções com zero ou mais parâmetros e retorno de valor.
- O uso do comando de retorno (**send back)** é obrigatório no corpo da função. Caso o retorno da função seja vazio (**none**), deve-se usar **send back ;**

Operadores:

Os operadores desta linguagem são **prefixados** (operadores vêm antes dos operandos).

- Operadores aritméticos: +, - , * , / (divisão inteira)
- Operadores relacionais: >, <, >=, <=, =
- Operadores booleanos: **not**, **and** e **or**.
- 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).

2. 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 (similar à ?).
6 - o operador { } indica uma estrutura sintática que é repetida zero ou mais vezes (fecho).

Adicionar print e scan

```
programa : {dec-variavel} {dec-funcao}
dec-variavel : tipo lista-nomes ';'
lista-nomes : ID { ',' ID }
tipo : tipo-base | 'vector' tipo-base '[' exp ']'
tipo-base : 'integer' | 'char' | 'boolean'
dec-funcao : tipo-retorno ID '(' parametros ')' bloco
tipo-retorno : tipo | 'none'
parametros : ε | parametro { '|' parametro }
parametro : tipo ID
bloco : 'start' { dec-variavel } { comando } 'end'
atrib :  var '<<' exp
comando :
        'if' '(' exp ')' comando 'else' comando
      | 'while' '(' exp ')' comando
      | 'for each' '(' tipo ID ':' ID ')' comando
      |  atrib ';'
      |  'send back' [ exp ] ';'
      |  bloco
      |  chamada ';'
var : ID | ID '[' exp ']'
exp : INTEIRO | CARACTERE | BOOLEANO
      | var
      | '(' exp ')'
      | chamada
      | '+' exp exp
      | '-' exp exp
      | '*' exp exp
      | '/' exp exp
      | '=' exp exp
      | '<=' exp exp
      | '>=' exp exp
      | '<' exp exp
      | '>' exp exp
      | 'not' exp
      | 'and' exp exp
      | 'or' exp exp
chamada : ID '(' lista-exp ')'
lista-exp : ε | exp { '|' exp }
```

3. Semântico:

- Escopo das variáveis: global e local;
- A execução de um programa consiste na chamada à função com assinatura: **`none main()`**
- As expressões em if e while devem avaliar um tipo booleano.
- Não existe **if** sem **else**.
- Em qualquer expressão, um caractere não pode ter seu valor automaticamente convertido para inteiro.
- Em qualquer expressão, um valor booleano não pode ter seu valor automaticamente convertido para inteiro.
- O **`for each`** tem funcionamento similar ao comando for each do Java.
  O comando '**`for each ( tipo elemento : vet )`**' faz com que a cada iteração deste laço, **`elemento`** receba um dos valores armazenados no vetor **`vet`**, do índice 0 até o índice tamanho-1.
  `vet` deve obrigatoriamente conter elementos do tipo **`tipo`**
- Nos casos omissos neste documento, a semântica da linguagem segue a semântica de C.

Procedimento primitivo **capture**:

Procedimento para entrada de dados a partir do teclado. Salva os valores lidos em uma ou mais variáveis passadas como argumentos. Podem assumir que o usuário sempre vai digitar alguma coisa do mesmo tipo das variáveis listadas na chamada. Exemplos:

- **`capture (a | b); #captura de forma fictícia dois valores que o usuário digitou, e os salva nas variáveis a e b.`**
- **`capture (1); #errado! O argumento não é uma variável.`**

Procedimento primitivo **show**:

Procedimento para exibição, de forma fictícia, de um ou mais valores resultantes de expressões passadas como argumentos. Exemplos:

- **`show (+ a b); #exibe o resultado da soma entre a e b.`**
- **`show ( a | + 40 2); #exibe o valor armazenado em a e o resultado da soma entre 40 e 2.`**
- **`show ( a ); #exibe o valor armazenado em a.`**

O que checar na análise semântica:

- Se entidades declaradas pelo usuário (variáveis, vetores e funções) são inseridas na tabela de símbolos com os atributos necessários;

- 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 nas:
  - Estruturas de controle
  - Expressões
  - Atribuições
  - Chamadas e retornos de funções

4. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio, dupla ou individualmente. Os grupos devem se cadastrar na planilha:
https://docs.google.com/spreadsheets/d/11Pj62NE8X-_IGjsQwVqpcqWq88dWMQ_6ihDhR0-UPGk/edit?usp=sharing
Prazo de preenchimento da planilha: 15/04/2021.

A submissão das tarefas do projeto deve ser feita via SIGAA, nas datas previstas na Seção 4.3.

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.

4.1. Ferramentas

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

4.2. 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.

4.3. Entregas

O compilador será composto por 4 etapas: Análise Léxica, Análise Sintática, Análise Sintática Abstrata e Análise Semântica. Estas etapas serão desenvolvidas através de 6 tarefas:

1. Análise Léxica - Parte 1
- **Prazo: 19/04/2021**
- **Atividade:** escrever três códigos em Prefixa que, unidos, usem todas as alternativas gramaticais (ou seja, todos os recursos) da linguagem.
- **Formato de entrega:** três códigos, onde cada código deve estar escrito em um arquivo de texto simples, com extensão ".pfx".

2. Análise Léxica - Parte 2
- **Prazo: 28/04/2021**
- **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 `prefixa` (em letras minúsculas).

3. Análise Sintática
- **Prazo: 09/06/2021**

- **Atividade:** implementar analisador sintático em SableCC, com impressão da árvore sintática em caso de sucesso ou impressão dos erros.
- **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 `prefixa` (em letras minúsculas).

4. Sintaxe Abstrata

- **Prazo: 21/06/2021**
- **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 `prefixa` (em letras minúsculas).

5. Análise Semântica – Parte 1

- **Prazo: 30/06/2021**
- **Atividade:** criação e impressão da 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 .pfx 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.

6. Análise Semântica – Parte 2

- **Prazo: 14/07/2021**
- **Atividade:** implementar validação de escopo e de existência 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 .pfx 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.

Entregas após o prazo sofrem penalidade de dois pontos por dia de atraso, aplicada na nota da etapa.

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.

Bom trabalho!