Universidade Federal de Roraima
Departamento de Ciência da Computação
Linguagens de Programação

DISCIPLINA: Linguagens de Programação – DCC206

1ª Lista

Prazo de Entrega: 06/05/2017

ALUNO: Antonio de Oliveira Viana

1) **Conceitue e descreva as diferenças entre (i) linguagem de alto-nível e (ii) linguagem de baixo-nível. Descreva o papel e a importância do compilador no processo de criação de programas de computador.**

   (i) São as linguagens de programação que possuem uma estrutura e palavras-chave que são mais próximas da linguagem humana. Tornando os programas mais fáceis de serem lidos e escritos. Esta é a sua principal vantagem sobre as linguagens de nível mais baixo. Os programas escritos nessas linguagens são convertidos para a linguagem de baixo nível através de um programa denominado compilador ou de um interpretador.

   (ii) A linguagem é muito mais voltada ao dispositivo (processador, microcontrolador, etc.). Normalmente envolve números e letras que nada mais são que instruções diretas ao dispositivo. O programador tem que entender não só da linguagem em si, mas de toda a arquitetura do dispositivo que ele irá trabalhar. Por ser uma linguagem mais próxima ao dispositivo, são necessárias menos conversões dessa linguagem para a "linguagem de máquina" do dispositivo. Quando se escreve numa linguagem de alto nível, muitas conversões são necessárias para alcançar a "linguagem de máquina", como essa geração é feita de forma automática, o código resultante acaba sendo maior do que se fosse escrito diretamente por um programador usando a linguagem de baixo nível.

   As linguagens de programação podem ser implementadas por um de três métodos gerais. Em um extremo, os programas podem ser traduzidos para linguagem de máquina, a qual pode ser executada diretamente no computador. Esse método é chamado de implementação baseada em compilação, com a vantagem de ter uma execução de programas muito rápida, uma vez que o processo de tradução estiver completo. A maioria das implementações de produção das linguagens, como C, COBOL e Ada, é feita por meio de compiladores.
   A linguagem que um compilador traduz é chamada de linguagem fonte. O processo de compilação e a execução do programa ocorrem em fases diferentes, cujas mais importantes são: O analisador léxico, o analisador sintático, o gerador de código intermediário, a otimização, o gerador de código e a tabela de símbolos.

2) Faça um programa escrito na linguagem C, C++, Python e Perl que aceite como entrada um número inteiro *n* maior ou igual a 1 e retorne como saída o valor da série:

$$\frac{1^1}{2^2} + \frac{3^3}{4^4} * \frac{5^5}{6^6} + \frac{7^7}{8^8} * \ldots ? \frac{(2n-1)^{(2n-1)}}{(2n)^{(2n)}}$$

$$? = + \text{ se } n \text{ é par}$$
$$? = * \text{ se } n \text{ é ímpar}$$

```
#include <stdio.h>
int main (){
    int n;
    printf("Informe um numero inteiro:\n");
    scanf("%d",&n);
    if (n >= 1){
                    if (n%2==0){
                            printf("  %d^%d\n",2*n-1,2*n-1);
                            printf("+ ----\n");
                            printf("  %d^%d\n",2*n,2*n);
                    }else{
                            printf("  %d^%d\n",2*n-1,2*n-1);
                            printf("* ----\n");
                            printf("  %d^%d\n",2*n,2*n);
                    }
    }else{
                    printf("O numero deve ser maior ou igual a 1");
    }
}
```

O programa foi criado e executado no programa *Geany* Windows: clicando em *Construir* e em seguida *Executar* é exibido no prompt uma mensagem *"Informe um numero inteiro"*, assim que informado, o programa verifica se o valor é maior ou igual a 1, não sendo é mostrado na tela: *"O numero deve ser maior ou igual a 1"* e finaliza, se o valor for maior ou igual a 1 atingi o critério e vai para a etapa de verificação se é *par* ou *ímpar*, sendo par: o programa imprime o sinal de + juntamente com a fração equivalente da fórmula. Sendo ímpar o programa imprime o sinal de * juntamente com a fração equivalente da fórmula.

5) Descreva as seguintes categorias de linguagens de programação e apresente o nome de duas linguagens de programação com seus respectivos exemplos.

(A) Imperativas Variáveis, atribuição e iteração. Inclui programação orientada a objeto, scripting e visuais – C e Java.

(B) Funcionais Aplicação de funções sob parâmetros – LISP e Haskell.

(C) Lógicas Baseada em fatos e regras – Prolog.

(D) Marcação/Híbrida Linguagens de marcação estendida para suportar alguma programação – JSTL e XSLT.

6) Escreva uma gramática no formato BNF e não ambígua para expressões envolvendo as variáveis A, B, C e os operadores * (multiplicação) e ^ (exponenciação). A precedência deve seguir as regras usuais da matemática.

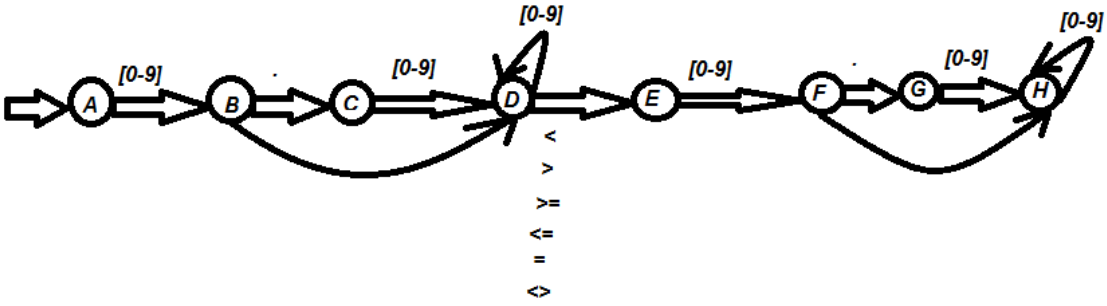
<programa> -> **begin** <stmt_list> **end**
<stmt_list> -> <stmt>
| <stmt> ; <stmt_list>
<stmt> -> <var> = <expressão>
<var> -> A | B | C
<expressão> -> <var> ^ <var>
| <var> * <var>
| <var>

<programa> => **begin** <stmt_list> **end**
⇨ **begin** <stmt> ; <stmt_list> **end**
⇨ **begin** <var> = <expressão>; <stmt_list> **end**
⇨ **begin** A = <expressão> ; <stmt_list> **end**
⇨ **begin** A = <var> ^ <var> ; <stmt_list> **end**
⇨ **begin** A = B ^ <var> ; <stmt_list> **end**
⇨ **begin** A = B ^ C ; <stmt_list> **end**
⇨ **begin** A = B ^ C ; <stmt> **end**
⇨ **begin** A = B ^ C ; <var> = <expressão> **end**
⇨ **begin** A = B ^ C ; B = <expressão> **end**
⇨ **begin** A = B ^ C ; B = <var> * <var> **end**
⇨ **begin** A = B ^ C ; B = C * <var> **end**
⇨ **begin** A = B ^ C ; B = C * A **end**

7) Descreva o que é um paradigma de programação.

Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns. A classificação de linguagens em paradigmas é uma consequência de decisões de projeto que tem impacto na forma segundo a qual uma aplicação real é modelada do ponto de vista computacional.

8) Apresente um autômato para reconhecer uma comparação entre números reais. A comparação deve ser feita utilizando os seguintes símbolos: < (menor), > (maior), >= (maior igual), <= (menor igual), = (igual), <> (diferente). Exemplo: 1.2 <= 59.13



9) Defina análise semântica e descreva: Semântica Operacional; Semântica Axiomática; e Semântica Denotacional.

A semântica trata da análise do significado das expressões, das instruções e das unidades de programa. A semântica é importante para que os programadores saibam precisamente o que as instruções de uma linguagem fazem. Descreve o significado das expressões, das instruções e das unidades de programas. As razões para descrever a semântica: saber precisamente o que as instruções de uma linguagem fazem e as provas de exatidão do programa recorrem a descrição formal da semântica da linguagem. Não existe uma notação ou um formalismo amplamente aceito para descrever semântica.

Semântica Operacional é descrever o significado de uma sentença ou programa pela especificação dos efeitos de rodá-lo em uma máquina. Os efeitos na máquina são vistos como sequências de mudanças em seu estado, onde o estado da máquina é a coleção de valores em seu armazenamento. Uma descrição de semântica operacional óbvia é dada pela execução de uma versão compilada do programa em um computador.

Semântica Axiomática chamada assim porque é baseada em lógica matemática, é a abordagem mais abstrata para a especificação de semântica. Em vez de especificar diretamente o significado de um programa, a semântica axiomática especifica o que pode ser aprovado sobre o programa. Um dos usos possíveis de especificação semântica é a prova de corretude de programas. Na semântica axiomática, não existe um modelo do estado de uma máquina ou programa, nem um modelo de mudanças de estado que ocorrem quando o programa é executado. A semântica axiomática tem duas aplicações distintas: a verificação de programas e a especificação de semântica de programas.

Semântica Denotacional é o método mais rigoroso e mais conhecido para a descrição do significado de programas. Ela é solidamente baseada na teoria de funções recursivas. O método é chamado denotacional porque os objetos matemáticos denotam o significado de suas entidades sintáticas correspondentes. Na semântica denotacional, o domínio é chamado de domínio sintático, porque são mapeadas estruturas sintáticas. A imagem é chamada de domínio semântico. A semântica denotacional está relacionada à semântica operacional, na qual as construções de linguagem de programação são traduzidas para construções mais

simples, o que se torna a base para o significado da construção. Na semântica denotacional, as construções de linguagem de programação são mapeadas para objetos matemáticos – conjuntos ou funções. Entretanto, diferentemente da semântica operacional, a semântica denotacional não modela o processamento computacional passo a passo dos programas.

10) Utilizando a semântica axiomática apresente a pré-condição para os seguintes programas.

<table>
<tr>
<td>PROGRAMA – A
x = 122 * y – 144
{x > 144}

122 * y – 144 > 144
122 * y > 144 + 144
y > 288 / 122
{y > 2,360}</td>
<td>PROGRAMA – B
y = 5 * x – 5
x = y + 5
{x < 45}

y + 5 < 45
y < 45 – 5
{y < 40}

40 > 5 * x – 5
5 * x < 40 + 5
x < 45 / 5
{x < 9}</td>
</tr>
<tr>
<td>PROGRAMA – C
if(x < 200)
  y = y + 2;
else
  y = y – 2;
{y > 2}

y = y + 2 {y > 2}
y + 2 > 2
{y > 0}

y = y – 2 {y > 2}
y – 2 > 2
{y > 4}

{y > 0} => {y > 4}</td>
<td>PROGRAMA – D
while i < N do
  i = i + 1
end
{i == N}

{i = N}

wp(i = i + 1, {i = N})
{i + 1 = N}
{i = N - 1}

wp(i = i + 1,{i = N - 1})
{i + 1 = N - 1}
{i = N - 2}
wp(i = i + 1, {i = N - 2})
{i + 1 = N - 2}
{i = N - 3}</td>
</tr>
<tr>
<td>PROGRAMA – E
i = 1; sn = 0; n = 32767;
while (i <= n){
  sn = Sn + a;
  i = i + 1;
}
{sn == n * a}</td>
<td></td>
</tr>
</table>