COMPILADORES

Prof. Marcus Ramos – Prova 1 – 07 de março de 2012

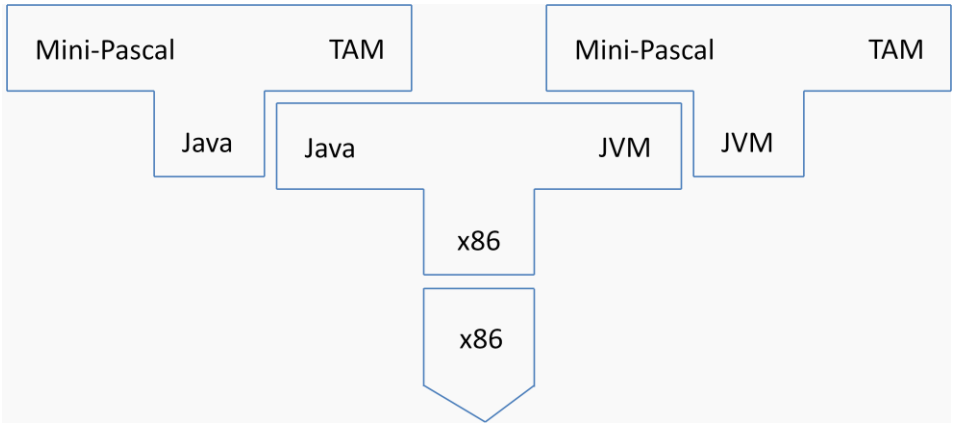
QUESTÃO 1 (0,6 ponto)
Classifique os processadores de linguagens com relação aos tipos das linguagens-fonte e objeto.

Tanto a linguagem-fonte (entrada) quanto a linguagem-objeto (saída) podem ser de alto ou de baixo nível. O quadro abaixo resume as possibilidades:

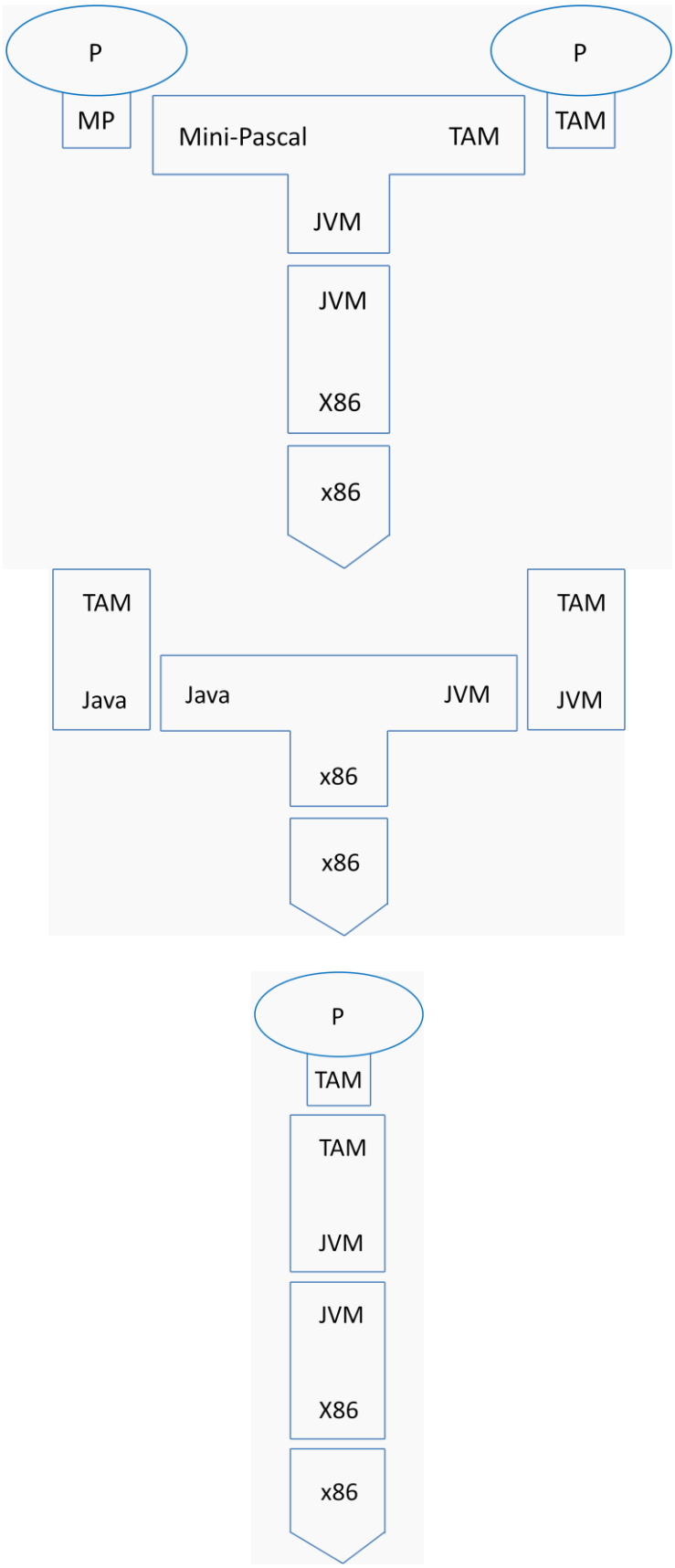
		Linguagem objeto	
		Alto nível	Baixo nível
Linguagem fonte	Alto nível	Tradutor ou filtro	Compilador
	Baixo nível	"Descompilador"	Montador ou tradutor

QUESTÃO 2 (1,4 pontos)
Represente, na forma de diagramas-T, os passos que deverão ser executados durante o desenvolvimento de um compilador para a linguagem Mini-Pascal, supondo que ele gere código da máquina virtual TAM. Considere disponíveis um compilador Java para JVM que é executável em x86, um interpretador JVM escrito em x86 e um interpretador TAM escrito em Java. Em particular, mostre:

a) Os passos necessários para a obtenção de uma versão executável (direta ou indiretamente) do compilador Mini-Pascal na máquina x86;



b) Os passos necessários para a execução de um programa escrito em Mini-Pascal na máquina x86.



QUESTÃO 3 (2 pontos)
Conceitue e explique:

a) Fase de compilação;

Etapa bem caracterizada do processo de compilação. Exemplos: análise sintática, análise de contexto e geração de código.

b) Passo de compilação;
Leitura completa do programa fonte ou de alguma representação do mesmo, como por exemplo uma árvore de sintaxe.

c) Compilação em um passo;
Esquema de organização em que todas as fases de compilação são executadas num único passo.

d) Compilação em vários passos;
Esquema de organização em que as fases de compilação são executadas ao longo de dois ou mais passos; cada fase pode demandar um ou mais passos de compilação.

e) Front-end de um compilador;
Nome que se dá para a combinação da fase de análise sintática com a fase de análise de contexto; geralmente reúne todas as fases relacionadas apenas com o processamento da linguagem fonte;

f) Back-end.
Nome que se dá para o agrupamento das fases de compilação relacionadas principalmente com a tradução de uma representação intermediária da linguagem fonte para a linguagem objeto.

QUESTÃO 4 (2 pontos)
A gramática abaixo apresenta a sintaxe das sentenças de uma lógica proposicional sobre o alfabeto {a,b,c} com os conectivos básicos e parênteses.

$< P >::= < P > \leftrightarrow < Q > \mid < Q >$
$< Q >::= < Q > \rightarrow < R > \mid < R >$
$< R >::= < R > \vee < S > \mid < S >$
$< S >::= < S > \wedge < T > \mid < T >$
$< T >::= \neg < T > \mid a \mid b \mid c \mid "(" < P > ")"$

Considere a sentença $(a \rightarrow b) \leftrightarrow (\neg a \vee b)$.

a) Apresente a seqüência de movimentos executados por um analisador descendente no reconhecimento da mesma;
$< P >$
$\Rightarrow < P > \leftrightarrow < Q >$
$\Rightarrow < Q > \leftrightarrow < Q >$
$\Rightarrow < R > \leftrightarrow < Q >$
$\Rightarrow < S > \leftrightarrow < Q >$
$\Rightarrow (< P >) \leftrightarrow < Q >$
$\Rightarrow (< Q >) \leftrightarrow < Q >$
$\Rightarrow (< Q > \rightarrow < R >) \leftrightarrow < Q >$
$\Rightarrow (< R > \rightarrow < R >) \leftrightarrow < Q >$
$\Rightarrow (< S > \rightarrow < R >) \leftrightarrow < Q >$
$\Rightarrow (< T > \rightarrow < R >) \leftrightarrow < Q >$
$\Rightarrow (a \rightarrow < R >) \leftrightarrow < Q >$
$\Rightarrow (a \rightarrow < S >) \leftrightarrow < Q >$
$\Rightarrow (a \rightarrow < T >) \leftrightarrow < Q >$

$\Rightarrow (a \rightarrow b) \leftrightarrow < Q >$
$\Rightarrow (a \rightarrow b) \leftrightarrow < R >$
$\Rightarrow (a \rightarrow b) \leftrightarrow < S >$
$\Rightarrow (a \rightarrow b) \leftrightarrow < T >$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< P >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< Q >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< R >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< R > \vee < S >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< S > \vee < S >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (< T > \vee < S >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (\neg < T > \vee < S >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (\neg < a > \vee < S >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (\neg < a > \vee < T >)$
$\Rightarrow (a \rightarrow b) \leftrightarrow (\neg < a > \vee b)$

b) O que significam as letras e o número em "LL(1)"?
LL(1): Leitura da esquerda para a direita ("L" de "Left-to-right"), uso de derivações mais à esquerda ("L" de "Leftmost derivations") com lookahead de apenas um símbolo.

c) O que significam as letras e o número em "LR(2)"?
LR(2): Leitura da esquerda para a direita ("L" de "Left-to-right"), uso de reduções mais à esquerda ("R" de "Rightmost derivations", porém na ordem inversa) com lookahead de no máximo dois símbolos.

d) O que significa dizer que uma gramática é LL(k)?
Significa que essa gramática gera uma linguagem que pode ser reconhecida de forma descendente, através de derivações mais à esquerda, e com lookahead de no máximo k símbolos;

e) O que significa dizer que uma linguagem é LL(k)?
Significa que existe pelo menos uma gramática LL(k) que gera essa linguagem.

QUESTÃO 5 (2 pontos)
Calcule todos os $first$ e $follow$ da gramática abaixo. Ela é LL(1)? Justifique a sua resposta.

$S ::= aS \mid YZYW$
$Y ::= bY \mid c \mid \varepsilon$
$Z ::= dZ \mid eW \mid W \mid \varepsilon$
$W ::= aW \mid a$

S:
$first(aS) = \{a\}$
$first(YZYW) = \{b, c, d, e, a\}$
Intersecção não vazia.

Y:
$first(bY) = \{b\}$
$first(c) = \{c\}$
$follow(Y) = \{d, e, b, c, a\}$

Intersecção não vazia.

Z:
$first(dZ) = \{d\}$
$first(eW) = \{e\}$
$first(W) = \{a\}$
$follow(Z) =) = \{b, c, a\}$
Intersecção não vazia.

W:
$first(aW) = \{a\}$
$first(a) = \{a\}$
Intersecção não vazia.

Logo, por vários motivos, a gramática não é LL(1).

QUESTÃO 6 (2 pontos)
Obtenha o esboço de um reconhecedor recursivo descendente para a linguagem gerada pela gramática da questão 4. Não se esqueça de:

a) Verificar se a gramática acima é LL(1);
b) Fazer a conversão se necessário;
c) Provar que a gramática que será usada como base é LL(1), calculando todos os conjuntos first e follow;
d) Codificar os métodos de forma correspondente.

Primeiro passo: eliminação das recursões à esquerda.

$< P >:: = < Q > \ (\leftrightarrow < Q >) *$
$< Q >:: = < R > \ (\rightarrow < R >) *$
$< R >:: = < S > \ (\vee < S >) *$
$< S >:: = < T > \ (\wedge < T >) *$
$< T >:: = \ \neg < T > \ | \ a \ | \ b \ | \ c \ |$ "(" $< P >$ ")"

Segundo passo: verificação da condição LL(1).

Em $< P >$:
$first(< Q > \ (\leftrightarrow < Q >) *) = \{\neg, a, b, c, (\}$
$first(\leftrightarrow < Q >) = \{\leftrightarrow\}$
$follow((\leftrightarrow < Q >) *) = \{\vdash\}$
$\{\leftrightarrow\} \cap \{\vdash\} = \emptyset$

Em $< Q >$:
$first(< R > \ (\rightarrow < R >) *) = \{\neg, a, b, c, (\}$
$first(\rightarrow < R >) = \{\rightarrow\}$
$follow((\rightarrow < R >) *) = \{\leftrightarrow, \vdash\}$
$\{\rightarrow\} \cap \{\leftrightarrow, \vdash\} = \emptyset$

Em $< R >$:

$first(< S > \ (\vee < S >) *) = \{\neg, a, b, c, (\}$
$first(\vee < S >) = \{\vee\}$
$follow((\vee < S >) *) = \{\rightarrow, \leftrightarrow, \vdash\}$
$\{\vee\} \cap \{\rightarrow, \leftrightarrow, \vdash\} = \emptyset$

Em $< S >$:
$first(< T > \ (\wedge < T >) *) = \{\neg, a, b, c, (\}$
$first(\wedge < S >) = \{\wedge\}$
$follow(\wedge < T >) *) = \{\vee, \rightarrow, \leftrightarrow, \vdash\}$
$\{\wedge\} \cap \{\vee, \rightarrow, \leftrightarrow, \vdash\} = \emptyset$

Em $< T >$:
$first(\neg < T >) = \{\neg\}$
$first(a) = \{a\}$
$first(a) = \{b\}$
$first(a) = \{c\}$
$first("(" < P > ")") = \{"("\}$
$\{\neg\} \cap \{a\} \cap \{b\} \cap \{c\} \cap \{"("\} = \emptyset$

Terceiro passo: codificação dos métodos.

```
parseP() {
  parseQ();
  while (sc=='↔') do {acceptIt(); parseQ()};
  }
parseQ() {
  parseR();
  while (sc=='→') do {acceptIt(); parseR()};
  }
parseR() {
  parseS();
  while (sc=='∨') do {acceptIt(); parseS()};
  }
parseS() {
  parseT();
  while (sc=='∧') do {acceptIt(); parseT()};
  }
parseT() {
  switch (sc) {
    case '¬': acceptIt();
              parseT();
              break;
    case 'a':
    case 'b':
    case 'c': acceptIt();
              break;
    case '(': acceptIt();
              parseP();
              accept(')');
    }
```

```
}
```