Compiladores
Prof. Marcus Ramos
Prova 1 – 02/05/2024

1. (2 pontos) Cite três características das linguagens de programação de alto nível que as diferenciam das linguagens de baixo nível. Explique em que consiste cada uma delas.

   Produtividade: o programador consegue executar mais operações por unidade de tempo.
   Legibilidade: o programa é mais fácil de ser interpretado por um humano. pois é mais próximo da linguagem natural.
   Portabilidade: um mesmo programa pode ser executado com facilidade em mais de uma máquina.
   Segurança: mais erros podem ser detectados, tanto em tempo de compilação quanto em tempo de execução.

2. (2 pontos) Você dispõe de um compilador de C para x86 escrito em x86. Mostre os passos que precisam ser realizados para que você possa compilar e posteriormente executar programas JAVA na máquina x86.

   (i) Escrever um compilador de Java para JVM em C.
   (ii) Escrever um interpretador de JVM em C.
   (iii) Compilar (i) no compilador C.
   (iv) Compilar (ii) no compilador C.
   (v) Executar o resultado de (iii) na máquina x86 e compilar um programa Java com ele.
   (vi) Executar o resultado de (iv) na máquina x86 com o resultado de (v).

3. (2 pontos) Por que antigamente era mais comum construir compiladores de um único passo?

   Porque os computadores eram mais lentos e tinham menos memória, o que favorecia a construção de compiladores de um único passo.

4. (2 pontos) A gramática apresentada a seguir é LL(1)? Justifique a sua resposta.

   S → aS | bS | Z#
   Z → cZ | dS | ε

   Sim, pois:
   $first_1(aX) = \{a\}$
   $first_1(bY) = \{b\}$
   $first_1(Z\#) = \{c, d, \#\}$
   Todos os conjuntos são disjuntos. Além disso,
   $first_1(cZ) = \{c\}$
   $first_1(dS) = \{d\}$
   $follow_1(Z) = \{\#\}$
   Todos os conjuntos são disjuntos.

5. (2 pontos) Obtenha o esboço de um analisador sintático usando o método recursivo descendente para a linguagem descrita pela gramática da Questão 4.

```
void parseS () {
switch (currentToken) {
case “a”: acceptIt (); parseS (); break;
case “b”: accepIt (); parseS (); break;
case “c”: case “d”: case”#”: parseZ (); accept (“#”); break;
default: ERRO }
}
void parseZ () {
switch (currentToken) {
case “c”: acceptIt (); parseZ (); break;
case “d”: accepIt (); parseS (); break;
case “#”: break;
default: ERRO }
}
```