Livro-texto: Compiladores: princípios, técnicas e ferramentas.

Exemplos: 1.1, 1.2 e 1.3

EXEMPLO 1.1: Os processadores da linguagem Java combinam compilação e interpretação, como mostrado na Figura 1.4. Um programa fonte em Java pode ser primeiro compilado para uma forma intermediária, chamada *bytecodes*. Os bytecodes (ou códigos de bytes) são então interpretados por uma máquina virtual. Como um benefício dessa combinação, os bytecodes compilados em uma máquina podem ser interpretados em outra máquina, talvez por meio de uma rede.

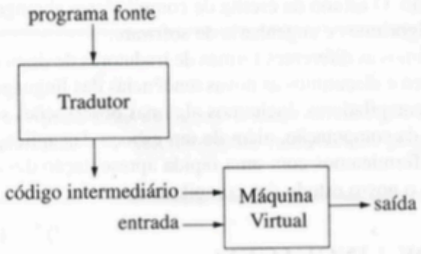

FIGURA 1.4 Um compilador híbrido.

A fim de conseguir um processamento mais rápido das entradas para as saídas, alguns compiladores Java, chamados compiladores *just-in-time*, traduzem os bytecodes para uma dada linguagem de máquina imediatamente antes de executarem o programa intermediário para processar a entrada.

EXEMPLO 1.2: A palavra-chave ***register*** da linguagem de programação C é um velho exemplo da interação entre a tecnologia de compiladores e a evolução da linguagem. Quando a linguagem C foi criada em meados da década de 1970, considerou-se importante permitir o controle pelo programador de quais variáveis do programa residiam nos registradores. Esse controle tornou-se desnecessário quando foram desenvolvidas técnicas eficazes de alocação de registradores, e a maioria dos programas modernos não usa mais esse recurso da linguagem.

Na verdade, os programas que usam a palavra-chave **register** podem perder a eficiência, pois os programadores normalmente não são os melhores juízes em questões de muito baixo nível, como a alocação de registradores. A escolha de uma boa estratégia para a alocação de registradores depende muito de detalhes específicos de uma arquitetura de máquina. Tomar decisões sobre o gerenciamento de recursos de baixo nível, como a alocação de registradores, pode de fato prejudicar o desempenho, especialmente se o programa for executado em máquinas diferentes daquela para a qual ele foi escrito.

EXEMPLO 1.3: Como outro exemplo da distinção entre estático e dinâmico, considere o uso do termo *static* aplicado aos dados em uma declaração de classe Java. Em Java, uma variável é um nome que designa uma localização de memória usada para armazenar o valor de um dado. Neste contexto, *static* refere-se não ao escopo da variável, mas sim à capacidade de o compilador determinar a localização na memória onde a variável declarada pode ser encontrada. Uma declaração como

```
public static int x;
```

torna *x* uma *variável de classe* e diz que existe apenas uma única cópia de *x*, não importa quantos objetos dessa classe sejam criados. Além disso, o compilador pode determinar uma localização na memória onde esse inteiro *x* será mantido. Ao contrário, se "static" fosse omitido dessa declaração, cada objeto da classe teria sua própria localização onde *x* seria mantido, e o compilador não poderia determinar todos esses lugares antes da execução do programa.