1. Linguagens de programação baseadas em pilha

A pilha é uma estrutura de dados presente em linguagens de programação. Isso porque as linguagens de programação naturalmente possuem elementos com características *last-in-first-out*, como, por exemplo, laços aninhados: o laço externo fica pendente quando o laço interno entre em execução e só é retomado quando este tiver se encerrado. Outro exemplo relevante são as funções (sub-rotinas), que podem gerar um encadeamento de chamadas: a função 1 chama a função 2, que chama a função 3, logo 1 fica pendente aguardando 2, que fica pendente aguardando 3; quando 3 termina, 2 é retomada, e quando 2 termina, 1 é retomada.

As linguagens em programação em geral usam pilhas internamente. É assim que elas fazem funcionar seus elementos *last-in-first-out*. No entanto, existem linguagens de programação que exigem do programador que ele explicitamente elabore seus programas manipulando uma pilha.

O exemplo mais popular de mecanismos computacionais explicitamente baseados em pilha são as calculadoras científicas com RPN (*Reverse Polish Notation*). Geralmente elas exibem no visor o valor no topo da pilha. Nas calculadoras mais sofisticadas, é possível visualizar os primeiros valores mais próximos do topo da pilha. Realizar operações em calculadoras baseadas em pilha é equivalente a escrever expressões aritméticas em notação pós-fixa (RPN), que dispensam o uso de parênteses para resolver ambiguidades. Por exemplo, a expressão, em notação em-fixa, (4 + 3) * (5 - 7) seria 4 3 + 5 7 - * em notação pós-fixa. Há modelos de calculadoras científicas baseadas em pilha que, além de operações aritméticas, apresentam operações de manipulação de pilha e programação, como é o caso das linhas de calculadoras científicas da marca HP.

A programação baseada em pilha é das mais simples de se implementar, pois parte do processamento realizado por um interpretador de uma linguagem de programação em geral fica a cargo do próprio programador no caso das linguagens baseadas em pilha. Em outras palavras, na comparação com programas em linguagens genéricas, é como se um programa em linguagem baseada em pilha já tivesse passado por uma etapa de processamento, que foi realizada no pensamento do programador, o que simplifica o trabalho do interpretador. Não é à toa que esse tipo de linguagem de programação é o mais empregado em dispositivos limitados. Além das calculadoras científicas, outro exemplo emblemático era um aplicativo de programação para os antigos palmtops que adotava a linguagem Forth, uma das mais antigas linguagens de programação baseadas em pilha, e que é usada até hoje em certos nichos.

Todas arquiteturas de computadores modernas contam com um mecanismo de pilha. Isso, historicamente, foi uma demanda que se originou na implementação de linguagens de programação de alto nível. A pilha do hardware é um componente interno fundamental em qualquer ambiente de execução de linguagens de programação.

A linguagem de programação baseada em pilha mais curiosa que já existiu é PostScript, da empresa Adobe. O PDF (*Portable Documento Format*), criado pela mesma empresa, ocupa hoje o lugar que um dia foi do PostScript. (O PDF é uma evolução do PostScript como linguagem de descrição de gráficos, mas não é tão notadamente uma linguagem de programação como PostScript é). Existiram até as impressoras a laser baseadas em PostScript: enviava-se pela porta serial os caracteres do programa PostScript e a impressora soltava o papel com o desenho descrito pelo programa. Impressoras são dispositivos limitados para processar linguagens de programação, portanto, são mais um exemplo da conveniência do modelo baseado em pilha em termos de facilidade de implementação. PostScript foi reconhecidamente inspirada em Forth.

A linguagem de máquina da JVM (máquina virtual Java), conhecida como bytecode, é baseada em pilha (Ver Seção 3). É curioso que a Dalvik (máquina virtual Android) seja baseada em registradores. Embora o Android adote a linguagem Java em alto nível, os ambientes de execução são diferentes. Os bytecodes precisam ser transformados para rodar em um dispositivo Android.

O modelo de linguagens de programação baseadas em pilha nunca deixará de despertar interesse no estudo das linguagens de programação. Turbak e Gifford (2008), especialistas em semântica de linguagens de programação, criaram sua própria linguagem baseada em pilha, o PostFix[1], como ambiente de execução alvo para seus compiladores de linguagens. Sestoft (2012) também cria uma máquina virtual baseada em pilha como alvo de compilação (Capítulo 8).

Uma forma muito eficaz de explicitar a semântica (o significado) de um comando de uma linguagem baseada em pilha é exibir a pilha antes e depois da execução do comando:

ADD $p, i_1, i_2 \Rightarrow p, (i_1 + i_2)$
DUP $p, v \Rightarrow p, v, v$
EXCH $p, v_1, v_2 \Rightarrow p, v_2, v_1$

[1] PostFix é baseada em Forth, PostScript e outras.

2. Roteiro de atividades sobre linguagens de programação baseadas em pilha

(2.1) Calculadoras. Encontre um emulador de calculadora baseada em pilha e experimente realizar operações aritméticas e de manipulação de piha com ele.

(2.2) Forth. Estude a linguagem Forth e experimente-a no intepretador `gforth`. Confira o livro *Starting Forth* [BRODIE] até o Capítulo 2.[2]

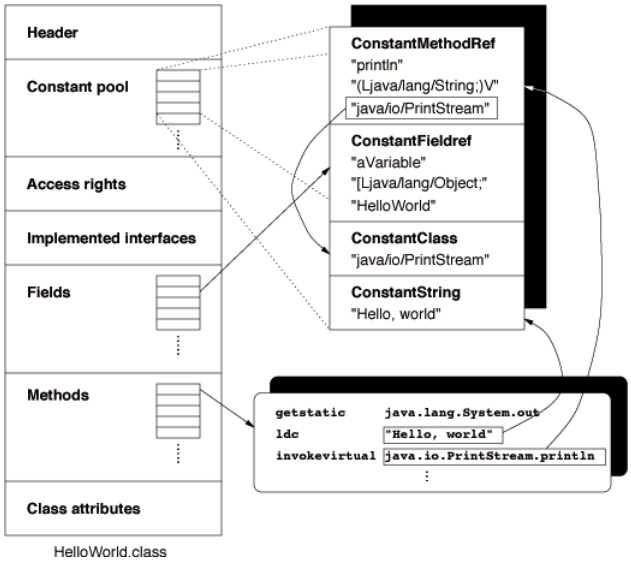
(2.3) PostScript. Estude a linguagem PostScript e experimente-a no interpretador `ghostscript`. Confira o livro *PostScript Language : Tutorial and Cookbook* [ADOBE SYSTEMS, 1985] até o Capítulo 7.

(2.4) Lógica de programação com linguagens baseadas em pilha. Escolha a linguagem de programação baseada em pilha de sua preferência.
(a) Coloque na pilha um valor representando uma temperatura em graus celsius. Converta a temperatura para graus Farenheit com operações aritméticas da linguagem.
(b) Sejam a, b e c os coeficientes de um polinômio de segundo grau $ax^2 + bx + c$. Escolha valores para a, b e c de forma que o polinômio tenha duas raízes reais. Ou seja, escolha valores para a, b e c tais que $b^2 - 4ac > 0$. Execute comandos para obter as duas raízes reais do polinômio de coeficientes a, b e c.
(c) [opcional] Resolva os exercícios 1.1, 1.2, 1.3 e 1.4 de Turbak e Gifford (2008) para a linguagem PostFix.

3. Java Virtual Machine

O código objeto executado pela máquina virtual Java é uma linguagem baseada em pilha. A definição oficial da JVM versão 7 é Lindholm, Yellin, Bracha e Buckley (2011). Sestoft (2012) traz uma análise da JVM no Capítulo 9, e Scott (2009), no Capítulo 15.

O arquivo `.class` gerado pelo compilador Java contém o código objeto a ser executado pela JVM. O conteúdo do arquivo pode ser examinado com a ferramenta `javap` distribuída com o JDK (*Java Developer Kit*).



O formato do arquivo Class prescreve um *pool* de constantes como strings e valores numéricos. Com referência às strings, além de todas as strings criadas pelo usuário, há as strings relativas aos nomes das classes, campos e métodos. A link-edição é feita pela JVM em tempo de execução com base em strings. É útil conhecer os códigos de tipos embutidos nas strings, inclusive nas assinaturas de métodos.

Cada método possui, além dos argumentos recebidos e das variáveis locais, uma pilha para execução das instruções internas. O compilador deve calcular o tamanho máximo a ser alcançado por esta pilha, e reservar um espaço de memória com esse tamanho.

[2] *Off-topic*: em suas ilustrações, Brodie usa alegorias para os componentes das linguagens de programação: um garçom seria o interpretador, um padre, o compilador, e um carrasco, o ambiente de execução. Comente a adequação das alegorias à teoria estudada em aulas anteriores.

4. Roteiros de atividades sobre a JVM

(4.1) Digite o seguinte programa em um arquivo chamado `Aritmetica.java`.

```
public class Aritmetica {

	public static void main(String[] args) {
		int a = 8;
		int b = 693;
		int res = (a + a) * (52245 - b/27);
		System.out.println(res);
	}
}
```

Em seguida, compile o programa na linha de comando, digitando `javac Aritmetica.java`
Execute o programa na linha de comando, digitando `java Aritmetica`
Examine os bytecodes em linguagem de máquina com um visualizador de arquivos binários. Digite `hexdump –C Aritmetica.class`
Examine os bytecodes em linguagem de montagem digitando

```
javap -c -v Aritmetica.class
```

Observe que a expressão aritmética que define a variável `res` foi transformada numa sequência de operações de pilha. Não se preocupe com o significado preciso das instruções. Algumas delas são autoexplicativas.

(4.2) Crie uma arquivo em Java com um repertório diversificado de itens da linguagem, como diversos métodos com tipos variados (`int`, `long`, `float`, `double`, `void`, classes, etc.), números de parâmetros variados, métodos `static` e não `static`, constantes grandes e pequenas, etc. Examine os bytecodes. Observe a presença dos símbolos `I`, `D`, `V`, `L`, `;` , `()` , etc. É possível deduzir seu significado.

(4.3) O debugger JDB. Se a ferramenta `javap` (*disassembler*) permite visualizar os programas objeto contidos em arquivos `.class`, os *debuggers* permitem acompanhar a execução do programa. Siga o tutorial https://www.packtpub.com/books/content/debugging-java-programs-using-jdb
Comandos do debugger JDB:

`stop in <pto>` – cria um breakpoint em <pto> (ex.: `stop in Pessoa.main`)
`clear` – lista breakpoints
`clear <pto>` – apaga um breakpoint

`run` – inicia a execução do programa
`cont` – (de *continue*) retoma a execução do programa parado

`next` – executa uma instrução de uma só vez (executa um método sem entrar nele)
`step` – executa uma instrução (se for método, entra nele)
`step up` – executa de uma vez o método atual e retorna para o método que o invocou
`stepi` – executa uma instrução de bytecode

`list` – mostra o código fonte em Java do ponto de execução atual e vizinhança
`where` – mostra a pilha de chamadas
`wherei`
`print <obj>` – mostra dados
`dump <obj>` – mostra dados
`locals` – mostra variáveis locais
`trace methods`
`untrace methods`
`classes`
`methods <class>`
`fields <class>`

5. Referências

ADOBE SYSTEMS. **PostScript Language :** Tutorial and Cookbook. Addison-Wesley, 1985

BRODIE, L. **Starting Forth**. Disponível em: http://www.forth.com/starting-forth/sf1/sf1.html

LINDHOLM, T; YELLIN, F; BRACHA, G; BUCKLEY, A. **The Java Virtual Machine Specification** : Java SE 7 Edition. Oracle, 2011.

SCOTT, M. L. **Programming language pragmatics.** Morgan Kaufmann, 2009.

SESTOFT, P. **Programming language concepts**. Londres: Springer, 2012.

TURBAK, F. ; GIFFORD, D. **Design concepts in programming languages**. MIT Press, 2008.