Lista de Exercícios 02
Sistemas de Linguagens

Exercício 01) O programa `gcc` presente em sistemas Unix para invocar o compilador de C desenvolvido pela *GNU* é na verdade um script que invoca vários outros programas. É possível descobrir a cadeia de programas invocados por `gcc` adicionando o parâmetro `-v` à linha de comando. Cada um dos itens a seguir corresponde a um dos passos adotados por `gcc` para produzir um arquivo binário a partir de um programa fonte. Descreva o que cada linha faz.

a) `gcc -E hello.c -o hello_p.c`
b) `gcc -S hello_p.c -o hello_p.s`
c) `as hello_p.s -o hello.o`
d) `ld hello.o -o a.out`

Exercício 02) Para cada uma das variações da sequência clássica indique duas vantagens e duas desvantagens. Liste uma linguagem que usa essa variação.

a) Interpretação
b) Máquinas virtuais
c) Ligação tardia
d) Instrumentação de código (*profiling*)
e) Compilação dinâmica (*JIT*: *Just in Time*)

Exercício 03) Um compilador *Just in Time* (*JIT*) compila os programas enquanto os mesmos ainda estão sendo interpretados. O programa a seguir, escrito em C, representa um compilador *JIT* muito rudimentar.

```
01: #include <stdio.h>
02: #include <stdlib.h>
03: #include <sys/mman.h>
04:
05: void main(void) {
06:   char* program;
07:   int (*fnptr)(void);
08:   int a;
09:   program = mmap(NULL, 1000, PROT_EXEC | PROT_READ |
10:     PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
11:   program[0] = 0xB8;
12:   program[1] = 0x34;
13:   program[2] = 0x12;
14:   program[3] = 0;
15:   program[4] = 0;
16:   program[5] = 0xC3;
17:   fnptr = (int (*)(void)) program;
18:   a = fnptr();
19:   printf("Result = %X\n", a);
20: }
```

a) Compile esse programa e diga o que será impresso por ele.
b) Qual é o "programa" produzido por esse compilador JIT? Onde esse "programa" é armazenado?
c) Esse programa funcionaria em uma arquitetura diferente de x86? Por quê?

Exercício 04) Compiladores modernos são muito bons em otimizar código. Contudo, algumas linguagens de programação permitem que o desenvolvedor indique como esses programas podem ser otimizados.

a) Considere a palavra-chave `register` da linguagem C. Para que ela serve e de que forma ela permite que o desenvolvedor "guie" o otimizador de código? Ilustre sua resposta com um trecho de código.
b) Responda as mesmas perguntas da questão anterior, porém dessa vez com a palavra-chave `inline` em C++.

Exercício 05) Existem diversos depuradores de código. Dois famosos, usados para C/C++, são `gdb` e `valgrind`. Esses depuradores têm propósitos muito diferentes conforme será visto a seguir.

a) A ferramenta `gdb` possibilita uma execução *interativa* de um programa. Para que isso funcione bem, é preciso compilar o programa com o parâmetro `-g`. Quando isso acontece, nota-se que o código binário produzido cresce. Por que acontece tal crescimento? A título de exemplo.

```
$ g++ -o prog-nodbg prog.cpp
$ g++ -g -o prog-dbg prog.cpp
$ ls -l prog-*
-rwxr-xr-x  1 rimsa  staff  69752 Oct  4 22:05 prog-dbg
-rwxr-xr-x  1 rimsa  staff  59952 Oct  4 22:05 prog-nodbg
```

b) A ferramenta `valgrind` permite depurar problemas de memória como vazamentos de espaço alocado, popularmente conhecidos como *memory leaks*. Qual a saída produzida para o `valgrind` para o programa abaixo (execute como `valgrind ./a.out`)? E se fosse removido o comentário sobre o comando `free`?

```
01: #include <stdio.h>
02: #include <stdlib.h>
03:
04: void problem() {
05:     int* i = (int*) malloc(sizeof(int));
06:     *i = 3;
07:     printf("%d\n", *i);
08:     // free (i);
09: }
10: void main(void) {
11:     problem();
12: }
```

Exercício 06) Considere o trecho de código a seguir. Indique pelo menos cinco diferentes tipos de amarrações que acontecem nesse código.

```
float j = 3.2;
j = j - 1.7;
```

Exercício 07) Para cada uma das atribuições a seguir, identifique em que momento é feita a amarração do tipo à variável. Justifique.

a) PHP: `$x = 5;`
b) C#: `int x = 5;`

Exercício 08) Para os seguintes construtos em Java, identifique em que tempo de amarração é feito.

a) A localização em memória de uma variável local em um método.
b) A localização em memória de um campo não estático de uma classe.
c) O significado da palavra **`while`**.
d) O tamanho em bits de um valor do tipo inteiro
e) Os *bytecodes* que formam uma classe
f) A definição do método `m`, quando uma chamada `a.m()` é executada.
g) O tipo de uma variável local em um método.
h) Os valores atribuídos a uma variável.
i) O tamanho de uma referência.