Lista de Exercícios 04
Tipos

Exercício 01) Uma diferença significativa entre a definição de tipos primitivos em C++ e JAVA se refere ao intervalo de valores de cada tipo. Enquanto em JAVA os intervalos foram fixados na definição da LP, em C++ é a implementação do compilador que define esses intervalos. Compare estas duas abordagens, justificando a opção de cada uma dessas linguagens.

Exercício 02) Considere as seguintes atribuições em Java.

```
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
float f = 5.67f;
double d = 0.1234;
```

Considere a expressão: `(f * b) + (i / c) - (d * s)`. Mostre qual o tipo resultante em cada uma das etapas intermediárias dessa expressão.

a) `(f * b)`
b) `(i / c)`
c) `(f * b) + (i / c)`
d) `(d * s)`
e) `(f * b) + (i / c) - (d * s)`

Exercício 03) Uma linguagem é considerada segura se ela não permite que operações sejam aplicadas a operandos que não possuam os tipos previstos para essas operações. C e C++ são linguagens inseguras, pois muitas vezes valores armazenados em memória podem ser acessados sem qualquer fiscalização de seus tipos.

a) Escreva um programa em C ou C++ que evidencie o caráter inseguro de uma dessas linguagens.
b) Existem linguagens mais antigas que C ou C++ que são consideradas seguras, portanto, a possibilidade de uso inseguro de tipos não é devido à ignorância sobre os perigos dessa abordagem. C++ foi criada depois de ML, que é considerada segura, e ainda assim C++ não é. Cite um fator que motivou C++ a manter a insegurança de tipos herdada de C.

Exercício 04) Considere o programa a seguir em C que recebe um índice como argumento em linha de comando para acessar um elemento de arranjo de tamanho um (1).

```
01: #include <stdio.h>
02: #include <stdlib.h>
03: void function(int idx) {
04:   int buffer[1] = { 0 };
05:   buffer[idx] += 3;
06: }
```

```
07: int main(int argc, char* argv[]) {
08:   int x = 13;
09:   function(atoi(argv[1]));
10:   printf("%d\n",x);
11:   return 0;
12: }
```

a) Compile esse programa sem otimizações (`-O0`). Qual a saída se ele for executado passando o índice `0` como parâmetro em linha de comando?
b) Existe algum índice que possa ser usado de forma que a saída desse programa seja `16`? Se sim, qual índice seria esse?
c) Esse programa em C ilustra uma vulnerabilidade conhecida como *buffer overrun*. Por que essa vulnerabilidade não ocorre em linguagens fortemente tipadas como Java?

Exercício 05) Uma linguagem é estaticamente tipada quando os tipos são resolvidos em tempo de compilação. Já uma linguagem é dinamicamente tipada quando os tipos são resolvidos em tempo de execução.

a) Dê um exemplo de uma linguagem estaticamente tipada. Como o compilador consegue descobrir o tipo das variáveis, no caso dessa linguagem?
b) Dê um exemplo de uma linguagem dinamicamente tipada. Escreva um programa, muito simples, que evidencie o caráter dinâmico dessa linguagem.
c) Cite uma vantagem da tipagem estática sobre a tipagem dinâmica.
d) Agora, cite uma vantagem da tipagem dinâmica sobre a tipagem estática.

Exercício 06) O autor de ML, Robin Milner, ganhou um prêmio Turing. Uma de suas maiores contribuições à ciência da computação foi um algoritmo para a inferência de tipos. ML é uma linguagem estaticamente tipada, porém desenvolvedores em geral não precisam escrever o tipo durante a declaração de expressões. Os tipos são inferidos automaticamente pelo compilador. O objetivo deste exercício é fazer com que você pense como um compilador, quando esse realiza a inferência de tipos. Tente descobrir o tipo de cada uma das funções abaixo. Assim que achar que tem uma resposta, escreva a função em SML e verifique se a resposta está correta. Escreva o tipo de cada função abaixo em notação de ML.

a) `f(x) = 1`
b) `f(x, y) = 1`
c) `f(x) = x`
d) `f(x, y) = x`
e) `f(g) = g(1)`
f) `f(g, x) = g(x)`
g) `f(g, x, y) = g(x, y)`
h) `f(g, h, x) = g(h(x))`
i) `f(g, x) = g(g(x))`

Exercício 07) Considere a seguinte implementação da função fatorial em PHP.

```
$ php -a
Interactive shell
php > function fact($n) {
php {   if ($n > 1) return $n * fact($n-1);
php {   else return 1;
php { }
php > echo gettype(fact(10));
integer
php > echo gettype(fact(100));
double
```

a) Quando o tipo de `fact(10)` é conhecido? As duas opções são: (i) em tempo de compilação da função `fact`, quando sua declaração passa a ser conhecida, ou (ii) durante a execução do programa, quando a chamada `fact(10)` é feita.
b) Por que o tipo de `fact(10)` é diferente do tipo de `fact(100)`?
c) É possível escrever uma função similar em C que exiba esse mesmo comportamento, ou seja, que o tipo do retorno seja promovido para um inteiro mais largo? Justifique sua resposta.
d) Do ponto de vista de facilidade de programação, qual forma de tipagem torna o programador mais produtivo? Existe alguma desvantagem para os programadores que usam linguagens que suportam tais facilidades?

Exercício 08) Cada um dos programas a seguir executa um comando de atribuição, indicado pela linha sublinhada. Explique o que acontece em cada caso. As respostas possíveis são: (i) o comando de atribuição impede a compilação do programa; (ii) o programa compila, mas ocorre um erro em tempo de execução, (iii) o programa compila e pode ocorrer um erro em tempo de execução; (iv) o programa compila e executa corretamente. Justifique sua resposta citando alguns dentre esses pontos: (a) tipagem forte vs. tipagem fraca; (b) equivalência de estrutura vs. equivalência de nome.

a) Em C++

```
01: struct point { int x, y; };
02: struct triple { int p1, p2, p3; };
03: int main() {
04:       struct point* p = new struct point;
05:       struct triple* t = new struct triple;
06:       t->p1 = t->p2 = t->p3 = 1;
07:       p = (struct point*) t;
08:       printf("%d, %d\n", p->x, p->y);
09:       return 0;
10: }
```

b) Em Java

```
01: class Point { int x, y; }
02: class Triple extends Point { int p1, p2, p3; }
03: class Main {
04:       public static void main(String[] args) {
05:             Point p = new Point();
06:             Triple t = new Triple();
07:             t.p1 = t.p2 = t.p3 = 1;
08:             p = (Point) t;
09:             System.out.printf("%d, %d\n", p.x, p.y);
10:       }
11: }
```

c) Em C

```
01: struct point { int x, y; };
02: struct pair { int x, y; };
03: void main() {
04:       struct point p;
05:       struct pair t = { 1, 1 };
06:       p = t;
07:       printf("%d, %d\n", p.x, p.y);
08: }
```