2ª Prova de Linguagens de Programação

Prof.:Carlos Camarão

7 de Julho de 2008

1. (4 pontos) Considere o seguinte programa em C++:

```
#include <iostream>
using namespace std;

class A {
  public:
    void p1() { cout << "pai1 "; };
    virtual void p2() { cout << "pai2 "; };
};
class B : public A {
  public:
    void p1() { cout << "filho1 "; };
    void p2() { cout << "filho2 "; };
};
main() {
  A a; B b; A* ptra;
  a.p1(); a.p2(); b.p1(); b.p2();
  ptra = &a; ptra -> p1(); ptra -> p2();
  ptra = &b; ptra -> p1(); ptra -> p2();
}
```

Indique e explique o que será impresso pelo programa acima.

2. (3 pontos) Considere as seguinte funções, escritas em ML:

```
exception E of int;
fun twice (f,x) = f(f x) handle E(x) => x;
fun pred  (x)   = if x=0 then raise E(x) else x-1;
fun dumb  (x)   = raise E(x);
fun smart (x)   = 1 + pred(x) handle E(x) => 1;
```

Indique qual é o resultado da avaliação de cada uma das seguintes expressões, em cada caso incluindo explicitamente qual exceção foi causada e como resultado da avaliação de qual expressão.

1. `twice(pred,1)`
2. `twice(dumb,1)`
3. `twice(smart,1)`

3. (3 pontos) Explique a regra usada na redefinição de métodos virtuais em `C++`, em relação aos tipos dos argumentos e do resultado do método, e explique porque essa regra é utilizada.

4. (3 pontos) Explique o mecanismo de chamada a funções virtuais em `C++` e porque ele permite que uma chamada a uma tal função em `C++` seja mais eficiente do que no caso de linguagens que não provêem suporte a tipagem estática (como por exemplo Common Lisp e Smalltalk).

5. (3 pontos) Explique como é feito o suporte a herança múltipla em `C++`, de modo a continuar com um suporte relativamente eficiente a chamadas de funções virtuais.

6. (2 pontos) Considere o seguinte programa em `C++`:

```
class C {
  public:
    int x;
    virtual void f();
}
class D : public C {
  public:
    int y;
    virtual void f();
} main() {
  C c; C* pc; D d; D* pd = new D();
  c = d;
  pc = pd;
  pc->f();
  c.f();
}
```

Explique o que ocorre (quais valores são copiados) nas atribuições `c = d;` e `pc = pd;`, e porque isso é razoável, com base no que ocorre nas chamadas `pc->f()` e `c.f()`.

7. (2 pontos) Indique F ou V (Falso ou Verdadeiro):

1. O mecanismo de herança em linguagens orientadas por objetos usuais faz com que deva existir um novo tipo de interface para classes, se subclasses de uma classe puderem ter acesso a variáveis e métodos não públicos.

2. Uma das razões para o fato de a eficiência, em termos de tempo de execução, do código da biblioteca STL de `C++` ser um pouco menor, em geral, do que códigos escritos por programadores, é relativa ao fato de que o código da biblioteca usa o paradigma orientado por objetos, e portanto envolve uma pequena sobrecarga devida ao uso de funções virtuais.

3. Uma das razões para o fato de a eficiência, em termos de espaço gasto em memória, do código da biblioteca STL de `C++` ser um pouco menor, em geral, do que códigos escritos por programadores, é relativa ao fato de que o código da biblioteca faz uso extensivo de *templates* da linguagem `C++`, os quais são expandidos em tempo de ligação, sendo um código distinto gerado pelo ligador para cada tipo usado na instanciação de um *template*.

4. O polimorfismo paramétrico usado em `C++` é menos flexível do que o de linguagens com suporte a inferência de tipos como `ML`, por exemplo, por requerer que o programador indique explicitamente os tipos usados, e o código gerado é geralmente maior, mas mais rápido, do que o que seria gerado usando `ML`.

8. (2 pts extras) Para o seguinte programa, escrito em uma linguagem fictícia do tipo Algol, indique o que será impresso pelo programa, considerando cada um dos mecanismos de passagens de parâmetros a seguir: i) passagem por valor; ii) passagem por referência; iii) passagem por valor-resultado.

A passagem por valor-resultado é tal que os argumentos devem ser variáveis, o valor armazenado em cada variável é copiado para o parâmetro correspondente antes da execução do corpo da "função"e copiado do parâmetro para essa variável depois da execução do corpo da função, na ordem em que as variáveis aparecem textualmente na chamada da "função".

```
begin
  integer i;
  procedure p (x, y) {
    integer x,y;
    begin
      x := x + 1;
      y := x + 1;
      x := y;
      i := i + 1;
    end;
  i := 1;
  p(i,i);
  print i
end.
```