Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo 2018/2019
Linguagens e Ambientes Programação – Teste 2 – Parte ❶

11 de junho de 2019 às 14:30

Teste com consulta com 1 hora e 30 minutos de duração

Nome: Num:

Notas: *Este enunciado é constituído por 4 grupos de perguntas. Responda no próprio enunciado, usando a frente e o verso, mas sempre no mesmo caderno em que aparece a pergunta. Normalmente, respostas imperfeitas merecem alguma pontuação. Fraude implica reprovação na cadeira.*

1. **Listas em ANSI-C com processamento dois a dois**. Vamos usar listas idênticas às das aulas de LAP. O tipo `List`, definido abaixo, representa listas ligadas de inteiros sem qualquer ordenação particular. Cada nó de cada lista é constituído por um valor inteiro e um apontador para o nó seguinte. O apontador NULL marca o final da lista e também serve para representar listas vazias. A função **`newNode`** cria nós inicializados.

```
typedef struct Node {
    int value;
    struct Node *next;
} Node, *List;
```

```
List newNode(int value, int count, List next) {
    List n = malloc(sizeof(Node));
    if( n == NULL ) return NULL;
    n->value = value;
    n->next = next;
    return n; }
```

Uma **forma prática de processar uma lista, dois nós em cada iteração,** baseia-se no seguinte esquema:

```
for( ; l != NULL && l->next != NULL ; l = l->next->next ) {
```

Note que, se a lista tiver comprimento ímpar, o último nó da lista poderá ter de ser processado fora do ciclo. Você pode usar esta técnica na resolução de uma ou mais alíneas deste problema, embora não seja obrigatório.

a) [2 valores] Escreva uma função para somar todos os valores com índices pares duma lista, ou seja o valor com índice 0 (o primeiro), mais o valor com índice 2 (o terceiro), mais o valor com índice 4 (o quinto), etc. Se a lista argumento for vazia, o resultado é zero.

```
int sumValuesWithEvenIdx(List l) {
```

b) [2 valores] Escreva uma função para trocar entre si os valores duma lista, dois a dois. Por exemplo, o resultado de processar a lista `[0,1,2,3,4,5,6]` será `[1,0,3,2,5,4,6]`. Esta função não cria nós novos. A função mantém os nós da lista pela mesma ordem e limita-se a efetuar trocas nos valores inteiros guardados nesses nós. Se a lista argumento for vazia, o resultado é também a lista vazia.

```
List swap2By2ChangingValues(List l) {
```

c) [2 valores] Escreva uma função para trocar entre si os nós duma lista, dois a dois. Por exemplo, o resultado de processar a lista `[0,1,2,3,4,5,6]` será `[1,0,3,2,5,4,6]`. Esta função não cria nós novos, nem altera o valor inteiro guardado em cada nó. A função limita-se a mudar a ordem dos nós, alterando os apontadores next. Se a lista argumento for vazia, o resultado é também a lista vazia. **[Resolva nas costas desta mesma folha.]**

```
List swap2By2ChangingNexts(List l) {
```

2. [6 valores] **Vida animal.** O objetivo deste problema é definir em JavaScript um sistema de classes adequado à representação de animais e à representação dos alimentos que eles ingerem.

Precisamos duma classe abstrata Food para representar alimentos em geral, mais uma classe abstrata Animal para representar animais em geral. A classe Animal é subclasse de Food, porque os animais carnívoros alimentam-se de outros animais.

Os atributos comuns às classes Food e Animal precisam de conter, no mínimo: (1) **name** - nome da espécie, (2) **calories** - calorias por grama, (3) **weight** - peso específico (sem contar o conteúdo do estômago, no caso dos animais). Os atributos adicionais da classe Animal precisam de conter, no mínimo: (1) **stomach** - estômago, um array de alimentos que começa vazio; (2) **foods** - tipos de alimentos compatíveis (um array de classes).

O atributo **foods** surge porque queremos escrever código extensível, e uma forma de fazer isso é cada animal saber quais os tipos de alimentos compatíveis para si. Note que, em JavaScript, uma classe é um objeto que pode ser usado como um valor normal, inclusivamente guardado num array, como por exemplo **[Grass, Carrot, Rock]**. Para testar o tipo dum alimento use **instanceof**, até porque podem existir relações de subtipo entre os vários tipos de alimentos (ou seja, não chega testar e verificar se duas classes são a mesma).

Problema

O objetivo deste problema é a definição dum sistema de classes bem fatorizada e extensível, adequado à representação de animais e alimentos. Escreva código compacto, bem fatorizado e extensível. No futuro vamos querer acrescentar novos tipos de animais e novos tipos de alimentos. e o programa tem de estar preparado para isso.

As classes Food e Animal precisam de definir os atributos, o construtor, mais estas funções:

`energy()` – O resultado é inteiro. Calcula as calorias específicas do alimento, levando em conta o seu peso, mais o conteúdo do estômago, se o alimento for um animal.

`validate()` – O resultado é booleano. (0) Um alimento que não seja animal é sempre válido. (1) Um animal **`a`** não pode conter no seu estômago outros animais da sua própria espécie ou subespécies, nem diretamente, nem indiretamente através do estômago de outros animais que tenham sido previamente comidos por **`a`**; (2) quando se valida um animal, todos os animais no seu estômago são também recursivamente validados.

A classe Animal precisa da seguinte função adicional.

`eat(food)` – Não tem resultado. O alimento é adicionado ao estômago apenas no caso de ser um alimento compatível. Se não for compatível, o alimento é ignorado.

Recomendamos que complete as seguintes classes, já com algum código. As classes Food e Animal são abstratas e as restantes são concretas. Complete os construtores e escreva os métodos pedidos. **Acrescente tudo o que for necessário para a solução fazer sentido, por exemplo métodos auxiliares, valores de pesos e de calorias para os alimentos e animais concretos, listas de alimentos compatíveis para os animais concretos.**

```
class Food {
   constructor(name, calories, weight) {



   }
```

```
}
class Grass extends Food {
   constructor(weight) {

   }}
class Carrot extends Food {
   constructor(weight) {

   }}
class Rock extends Food {
   constructor(weight) {

   }}
```

```
class Animal extends Food {
   constructor(name, calories, weight, foods) {



   }
```

```
}

class Rabbit extends Animal {
   constructor(weight) {

   }
}

class Lion extends Animal {
   constructor(weight) {

   }
}

function test() {
   var r1 = new Rabbit(1000)
   var r2 = new JumpingRabbit(1100); // subtipo de Rabbit. Não é para definir.
   r1.eat(new Carrot(50)); r1.eat(new Carrot(50)); r1.eat(new Carrot(60)); r1.eat(new Rock(1))
   r2.eat(new Carrot(100))
   var l = new Lion(100000)
   l.eat(r1); l.eat(r2)
   console.log(l.validate())
}
```

Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo 2018/2019
Linguagens e Ambientes Programação – Teste 2 – Parte ❷

11 de junho de 2019 às 14:30

Teste com consulta com 1 hora e 30 minutos de duração

Nome: Num:

3. [4 valores] Considere o seguinte programa escrito em GCC, uma variante do C que suporta aninhamento de funções:

```
#include <stdio.h>

void X(int *b) {
    void Y(int **c) {
        void Z(int ***d) {
            ++***d;
            X(**d);
        }
        ++**c;
        Z(&c);
    }
    ++*b;
    Y(&b) ;
}

int main(void) {
    int a = 3;
    X(&a);
    return 0;
}
```

Mostre qual o estado da pilha de execução **no momento em que acabou de ser empilhado o 6° registo de ativação**. Não se esqueça dos registos de ativação das funções `start` e `main` (que também contam para o total de 6).

Use as convenções habituais das aulas: Para efeito da criação do registo de activação inicial, imagine que cada programa em GCC está embebido numa função sem argumentos chamada `start`. Depois trate todas as entidades globais do programa como sendo locais à função imaginária `start`. Assuma também que a primeira célula da pilha de execução é identificada como posição 00, a segunda célula como posição 01, etc.

09	19	29
08	18	28
07	17	27
06	16	26
05	15	25
04	14	24
03	13	23
02	12	22
01	11	21
00	10	20

4. [4 valores] Escolha múltipla. As respostas erradas não descontam. Indique as respostas aqui:

A	B	C

A) Até que ponto um sistema de tipos estático, como por exemplo o do OCaml, serve para detetar erros de lógica num programa:

a) O sistema de tipos deteta de forma explícita todos os erros de lógica e assinala-os em tempo de compilação.

b) O sistema de tipos deteta de forma explícita alguns erros de lógica e assinala-os em tempo de compilação.

c) O sistema de tipos ajuda-nos a detetar alguns erros de lógica de forma indireta, concretamente os erros de lógica que acabam por se manifestar através de erros de tipo.

d) O sistema de tipos não ajuda nunca, nem de forma indireta, a detetar erros de lógica.

B) Apresenta-se aqui uma lista de situações, todas elas indesejáveis para quem produz software. Qual destas situações é a mais grave, preocupante e difícil de resolver.

a) O programa não compila.

b) O programa funciona, mas tem erros de execução ocasionais, que forçam a execução parar de forma súbita

c) O programa funciona de forma completamente correta na plataforma em que foi desenvolvido e testado (por exemplo Windows), mas tem erros de execução que já foram deteetados noutras plataformas (por exemplo Linux).

d) O programa funciona sem erros de execução em todas as plataformas. Os resultados parecem estar corretos e toda a gente confia no programa. Contudo, o programa produz resultados errados ocasionalmente. Ainda ninguém detetou esse problema e não se sabe se alguma vez será detetado.

C) Sobre closures (fechos):

a) É necessário usar closures para implementar passagem de funções como argumentos, em qualquer linguagem que suporte passagem de funções como argumento.

b) Nas linguagens com escopo estático, usam-se closures exclusivamente na implementação da passagem de funções como argumento.

c) Nas linguagens com escopo dinâmico, usam-se closures exclusivamente na implementação da passagem de funções como argumento.

d) Nas linguagens com escopo estático, usam-se closures não só na implementação da passagem de funções como argumento, mas também na representação das funções guardadas em estruturas de dados, tais como variáveis (funcionais ou imperativas), arrays ou listas.