Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo 2016/2017
Linguagens e Ambientes Programação – Teste 2 – Parte ❶

16 de junho de 2016 às 09:00

Teste com consulta com 1 hora e 40 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. Programação em ANSI-C. O tipo `List` permite representar listas ligadas de inteiros. Cada nó duma lista contém um valor inteiro, e um apontador para o nó que se segue. 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, List next)
{
    List n = malloc(sizeof(Node));
    if( n == NULL ) return NULL;
    n->value = value;
    n->next = next;
    return n; }
```

As nossas listas não estão sujeitas a qualquer ordenação particular e podem ter elementos repetidos.

Programe soluções iterativas, portanto sem usar recursão. A complexidade das soluções deve ser linear. Sempre que possível, percorra o argumento principal apenas uma vez.

a) [2 valores] Escreva uma função para criar uma lista a partir dum array de inteiros com comprimento dado. A ordem dos elementos deve ser respeitada. No caso desse comprimento ser zero, produza a lista vazia. Há muitas formas de escrever esta função - quanto mais simples, melhor.

```
List arrayToList(int a[], int n) {
```

b) [2 valores] Escreva uma função para preencher um array de inteiros com os valores duma lista ligada, respeitando a ordem dos elementos. Um dos argumentos da função indica a capacidade do array. Se a lista for demasiado longa, os últimos elementos que não couberem no array são ignorados. A função retorna o número de valores que foram efetivamente colocados no array.

```
int listToArray(List l, int a[], int n)
```

c) [2 valores] Numa lista, é normal ocorrerem valores repetidos e até mesmo valores repetidos consecutivos. Escreva uma função de limpeza que garanta que não permanecem na lista valores repetidos consecutivos. O número de nós a remover deve ser mínimo e a ordem dos elementos que ficam não é alterada. Aplique a função `free` a todos os nós removidos.

```
List clean(List l) {
```

2. [4 valores] Escolha múltipla. As respostas erradas não descontam. Indique as respostas aqui:

A	B	C

A) Escopo estátivo e escopo dinâmico. Qual das sequintes alternativas é a mais incorreta:

a) Com escopo estático, o âmbito dum identificador é determinado pela localização da sua definição no código, e ainda pela localização das definições de outros identificadores com o mesmo nome (porque podem interferir).

b) Com escopo dinâmico, o âmbito dum identificador é determinado pela sequência de chamadas que conduziram ao uso do identificador.

c) Com escopo estático, a resolução dos nomes não locais dentro duma função é feito no ambiente existente no ponto da chamada da função e não no ambiente da definição da função.

d) Tratam-se de duas regras de resolução de nomes.

B) "Todas as linguagens com tipificação dinâmica suportam polimorfismo de forma inerente." Esta frase é verdadeira para que espécie de polimorfismo? (Repare que, nestas linguagens, as funções aceitam argumentos de qualquer tipo.)

a) Polimorfismo paramétrico.

b) Polimorfismo de inclusão

c) Overloading.

d) Coerção.

C) Qual destas descrições da tecnologia AJAX é a mais correta:

a) Permite enviar e receber dados do servidor sem interferir no conteúdo e comportamento da página Web visível.

b) Suporta os formulários HTML, introduzidos em 1993, que permitem fazer queries dinâmicas ao servidor.

c) Consiste na possibilidade de usar scripts para criar páginas Web altamente interativas.

d) Refere-se ao modelo de eventos usado do lado do cliente.

Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Letivo 2016/2017
Linguagens e Ambientes Programação – Teste 2 – Parte ❷

16 de junho de 2017 às 09:00

Teste com consulta com 1 hora e 40 minutos de duração

Nome: Num:

3. [6 valores] **Observer pattern.** Trata-se dum padrão de desenho de software no qual um objeto particular, designado de ***subject***, regista internamente um conjunto de objetos dependentes que são chamados de ***observers***. O objetivo deste esquema é os *observers* serem notificados pelo *subject* sempre que este muda de estado. O *subject* fornece um método **subscribe** que os *observers* usam para se registarem. Os *observers* fornecem um método **notify** que o *subject* usa para os notificar. No *subject*, há um método **publish** que é chamado sempre que o *subject* muda de estado para enviar notificações.

Este padrão de desenho é usado em muitas situações: Um site de notícias é um *subject* e os leitores do site são *observers* que se podem registar para serem notificados das últimas notícias, por exemplo via email. Um site de leilões é um *subject* e os licitantes são *observers* que se registam para serem notificados das últimas ofertas. Na interface gráfica dum browser Web, um botão é um *subject* e os observers (*listeners*) são blocos de código que se registam usando o atributo ONCLICK para serem notificados de que o botão foi clicado.

Problema

Escreva em JavaScript duas classes abstratas, **Subject** e **Observer**, para implementar este padrão de desenho de forma abstrata. O seu código deverá estar bem fatorizado. As classes abstratas **Subject** e **Observer** devem poder ser usadas na implementação do padrão do observador, nas mais variadas concretizações.

Depois escreva duas subclasses concretas, **Sensor** e **Display**, para obter um programa funcional. Quando tudo estiver definido, a seguinte função

```
function test() {
    var s1 = NEW(Sensor,"sensor1",0.0);
    var s2 = NEW(Sensor,"sensor2",0.0);
    var d1 = NEW(Display,"display1");
    var d2 = NEW(Display,"display2");
    s1.subscribe(d1); s1.subscribe(d2);
    s2.subscribe(d1);
    s1.changeValue(2.0); s2.changeValue(3.0);
}
```

deverá produzir o seguinte texto:

```
display1: sensor1 has value 2.0
display2: sensor1 has value 2.0
display1: sensor2 has value 3.0
```

A função **test** indica indiretamente quais os métodos que é preciso definir e quais os seus argumentos. Um detalhe: repare que se indica um nome sempre que se cria um sensor ou um display - consideramos que tanto um subject como um observer possuem um nome, ou seja os dois conceitos possuem em comum o atributo **name**.

Ajuda

No caso da classe **Subject** é possível escrever bastante código: definir um array para guardar os observadores; implementar os métodos **subscribe** e **publish**; não esquecer o construtor. No caso da classe **Observer**, é preciso criar um método **notify**, mas não é possível determinar abstratamente o que faz esse método. Por isso, o corpo do método pode ser simplesmente **throw "please redefine me"**. Não esquecer o construtor.

A classe **Sensor** representa sensores de temperatura. Possui o atributo **value** para representar a temperatura corrente e o método **changeValue** para implementar mudanças de temperatura. A classe **Display** representa ecrãs com capacidade de apresentar texto.

Como sabe, em JavaScript, para acrescentar um elemento a um array escreve-se **array.push(elem)**.

Solução

A primeira linha das várias classes já foi escrita. Se tiver falta de espaço, aproveite as costas da folha.

```
var Named = EXTENDS(JSRoot, {
    name: "",
    INIT: function(name) {
        this.name = name;
    }

});
```

```
var Subject = EXTENDS(Named, {
```

```
var Observer= EXTENDS(Named, {
```

```
var Sensor = EXTENDS(Subject, {
```

```
var Display = EXTENDS(Observer, {
```

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

```
#include <stdio.h>

void A(int n) {
    void B(int n) {
      void C(int n) {
        if(n%2==0) A(n+1); else B(n);
      }
      if(n%2==0) C(n+1); else A(n);
    }
    B(n+1);
}

int main(void) {
    A(1);
    return 0;
}
```

Mostre qual o estado da pilha de execução **no momento em que acabou de ser empilhado o 7° registo de ativação**. Não se esqueça dos registos de ativação das funções `start` e `main`.

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.

15	31	47
14	30	46
13	29	45
12	28	44
11	27	43
10	26	42
09	25	41
08	24	40
07	23	39
06	22	38
05	21	37
04	20	36
03	19	35
02	18	34
01	17	33
00	16	32