Mestrado Integrado em Engenharia Informática (FCT/UNL)
Ano Lectivo 2013/2014
Linguagens e Ambientes Programação – Teste 2

02 de Junho de 2014 às 18:00

Teste com consulta com 1 hora e 40 minutos de duração + 15 minutos de tolerância

Nome: Num:

Notas: *Este enunciado é constituído por 4 grupos de perguntas. Responda no próprio enunciado, usando a frente e o verso.*
Pode definir funções auxiliares sempre que quiser.
Normalmente, respostas imperfeitas merecem alguma pontuação.
Fraude implica reprovação na cadeira.

1. [4 valores] Escolha múltipla. As respostas erradas não descontam. Indique as respostas mais correctas aqui:

A	B	C

```
#include <stdio.h>

int global(int x) {
    int local_A(int a) { return a * x; }
    int local_B(int x) { return local_A(x) ; }
    return local_B(3);
}

int main(void) {
    printf("%d\n", global(10));
    return 0;
}
```

A) Considere programa GCC que se encontra atrás. O que escreve o programa e porquê?

a) 30, porque o GCC usa escopo dinâmico.

b) 30, porque o GCC usa escopo estático.

c) 9, porque o GCC usa escopo dinâmico.

d) 9, porque o GCC usa escopo estático.

B) Considere programa GCC que se encontra atrás. Observe a função **global** e o seu argumento **x**. Quais os *momentos de ligação* destes dois nomes.

a) Ambos são ligados em tempo de compilação.

b) Ambos são ligados em tempo de ligação.

c) Ambos são ligados em tempo de carregamento.

d) O nome **global** é ligado em tempo de ligação; o nome **x** é ligado em tempo de execução.

C) Sobre polimorfismo.

a) O Java suporta polimorfismo de inclusão, paramétrico, overloading e coerção.

b) O C suporta polimorfismo de inclusão, paramétrico, overloading e coerção.

c) O C não suporta polimorfismo baseado em coerções.

d) O OCaml suporta polimorfismo de inclusão, paramétrico, overloading e coerção.

2. [4 valores] Considere o seguinte programa escrito em GCC, uma variante do C que suporta aninhamento de funções:

```
#include <stdio.h>

void f1(int a[], int b[], int n) {

    void f3(int k) {
       b[k] = a[k];
       Ψ
    }
    void f2(int j) {
       a[j] += b[j];
       f3(j+1);
    }

    f2(0);
}

int main(void) {
    int a[3] = {20, 21, 22};
    int b[3] = {30, 31, 32};
    f1(a,b,3);
    return 0 ;
}
```

Mostre qual o estado da pilha de execução no momento em que a execução atinge o ponto marcado com o símbolo Ψ. Não se esqueça de registar nas células da pilha as alterações de valor causadas pelas atribuições que ocorrem durante a execução do programa. Repare também que cada um dos arrays locais à função `main` ocupa três posições de memória.

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.

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

3. Considere, em ANSI-C, o seguinte tipo que permite definir listas ligadas de inteiros. Cada nó da lista contém um valor inteiro e um apontador para o nó que se segue. O apontador NULL marca o final da lista.

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

Importante: Nos problemas que se seguem, pretendem-se soluções iterativas, portanto, sem uso de recursão. Em todos os problemas, pretende-se ainda que a lista-argumento seja percorrida apenas uma vez.

a) [2 valores] Escreva em C uma função para eliminar todos os nós duma lista, usando a função de biblioteca **free**. Portanto o objetivo é "apagar" toda a lista. Tenha cuidado com o seguinte: quando se faz o free dum nó, o conteúdo desse nó fica indefinido, incluindo o campo **next**.

```
void delete(List l) {
```

b) [2 valores] Escreva em C uma função que permita obter um apontador para o nó que se situa a meio duma lista. No caso da lista ter comprimento par, existem dois candidatos naturais e você poderá retornar um apontador para qualquer um deles. Se a lista de entrada for vazia, então o resultado é também a lista vazia. [Ideia: Percorra a lista apenas uma vez e, para saber quando chegou ao meio, use um apontador auxiliar que navega na lista saltando de dois em dois nós.]

```
List middle(List l) {
```

c) [2 valores] Escreva em C uma função para reorganizar os nós duma lista para que os nós com valor impar fiquem todos antes dos nós com valor par. Para além disto, as posições relativas dos nós com valor impar não devem mudar, e as posições relativas dos nós com valor par também não devem mudar. Se a lista de entrada for vazia, então o resultado é também a lista vazia. Resolva nas costas desta mesma página.

```
List oddEven(List l) {
```

4. [6 valores] **Termómetros.** O objectivo deste problema é a representação em JavaScript de diversos tipos de termómetros.

Thermometer – Este é o tipo dos termómetros mais básicos. Um **termómetro básico** armazena internamente a temperatura corrente (um número real), e ainda dois valores reais especiais **bottom** e **top** - o termómetro só funciona corretamente dentro destes limites, que nunca poderão ser ultrapassados.

Um termómetro básico tem 4 funções públicas: **init**, **get**, **set**, **reset**.

A função **init(bottom, top)** inicializa um termómetro usando os dois argumentos indicados. Além da inicialização dos dois limites, a temperatura registada internamente fica a ser a média entre **bottom** e **top**.

A função **get()** serve para perguntar a um termómetro qual a temperatura registada internamente.

A função **set(value)** serve para indicar ao termómetro qual é a nova temperatura que deve ser registada internamente. No caso do argumento estar fora dos limites permitidos, o valor interno não é alterado.

A função **reset()** volta a colocar a temperatura interna na média entre **bottom** e **top**.

MinThermometer – Um **termómetro com mínimo** é parecido com um termómetro básico, diferindo no facto de memorizar a temperatura mais baixa observada até ao momento. Também fornece uma função específica **min()**, que serve para perguntar qual é esse valor mínimo. Note que a gestão do valor mínimo obriga a repensar a maior parte das funções herdadas dos termómetros básicos.

MultiThermometer – Um **termómetro múltiplo** difere dum termómetro básico no facto de possuir múltiplos sensores, representados por vários termómetros básicos, todos semelhantes entre si. A inicialização **init(n, top, bottom)** dum termómetro múltiplo requer três argumentos: **n** é o número de termómetros básicos pretendidos, **top** e **bottom** são os valores usados para inicializar todos os termómetros básicos. Assuma (sem testar) que o valor de **n** é sempre positivo, o que significa que existirá sempre, pelo menos, um termómetro básico.

Num termómetro múltiplo existe a noção de *termómetro básico selecionado*. Quando o termómetro múltiplo é criado, o termómetro básico selecionado fica a ser o primeiro. Existe ainda uma função específica **select(i)**, que permite selecionar qual é o termómetro corrente.

Num termómetro múltiplo, as funções **get** e **set** trabalham apenas sobre o termómetro selecionado. Já a função **reset** efetua o reset de todos os termómetros básicos.

Thermograph – Um **termografo** difere dum termómetro básico no facto de registar internamente uma sequência de temperaturas. Por isso a função **set** tem de ser reescrita. Nas funções **init** e **reset**, a sequência de temperaturas deve ser inicializada com a temperatura inicial, ficando assim a sequência com comprimento 1 inicialmente.

Os termografos possuem uma função específica **average()**, que permite obter a média das temperaturas armazenadas.

Problema

O objetivo deste problema é a definição dum sistema de protótipos bem fatorizado e extensível, adequado à representação de termómetros através de objetos. Escreva código compacto e **bem fatorizado**. Pense bem nos detalhes da fatorização, pois há alguns pormenores relevantes. Em muitas situações, existe uma fatorização que se pode considerar ser a ideal de forma objetiva.

Sugestão: Se quiser, pode usar os seguintes quatro protótipos, já definidos. Mas, se preferir, pode ignorá-los.

```
var Thermometer = { // concrete
```

```
var MinThermometer = extend(Thermometer, {  // concrete
```

```
var MultiThermometer = extend(Thermometer, {  // concrete
```

```
var Thermograph = extend(Thermometer, {  // concrete
```