1. [9] No desenvolvimento de uma aplicação de cálculo de distâncias quilométricas entre cidades são mantidas, para cada cidade, as distâncias às cidades com ligação directa (sem passar por nenhuma outra), de acordo com as seguintes definições em C:

```
#define MAX_CITIES 32

typedef struct City {
   /* nome da cidade */
   char *name;
   /* array de distâncias a cidades
     adjacentes. A distância à cidade
     i encontra-se na posição i do
     array. As entradas a zero
     indicam que não há ligação
     directa à cidade respectiva */
   int connections[MAX_CITIES];
} City;

typedef struct Cities {
   int nCities;
   City *cities[MAX_CITIES];
} Cities;
```

```
/* mapa de cidades */
static Cities map;

/* devolve a posição da cidade de nome name */
int cityIndex(char *name) {
   int i;
   for(i=0; i < map.nCities; ++i) {
      if (strcmp(name, map.cities[i].name)==0) return i;
   }
   return -1;
}

/* calcula a distância entre duas cidades com ligação directa */
int connectionDistance(int city, int otherCity) {
   return map.cities[city]->connections[otherCity];
}

int addCity(char *city) { … }

void addConnection(char *city, char *adjacentCity, int distance) {
   int index = cityIndex(city);
   int adjacentIndex = cityIndex(adjacentCity);
   map.cities[index]->connections[adjacentIndex] = distance;
}

/* Distância total um dado percurso. O percurso é recebido
na forma de um array de nomes de cidades terminado por NULL*/
int routeDistance(char **route) {
   int d=0;
   char *currCity = *route++, *nextCity;

   int currIndex = cityIndex(currCity);
   while ((nextCity = *route++) != NULL) {
      int nextIndex = cityIndex(currCity);
      d+= connectionDistance(currIndex, nextIndex);
      currIndex = nextIndex;
   }
   return d;
}

int distanceInTwoStepsMost(char *origin, char *dest) { … }
```

a) [1,5] Implemente em C a função `addCity`, que adiciona a `map` a nova cidade, sem ligações. A função retorna -1 caso a cidade já exista ou o mapa esteja cheio. Em caso de sucesso retorna a posição em que a cidade foi inserida.

b) [1,5] Implemente em *assembly* IA-32 a função `connectionDistance`.

c) [1,5] Reimplemente como uma macro a função `cityIndex`. Sugere-se a definição da macro `CITY_INDEX(NAME, IDX)`. Na definição da macro, apenas pode invocar a função `strcmp`.

d) [1,5] Implemente em C a função `distanceInTwoStepsMost`, que retorna a distância entre duas cidades, passando no máximo por uma cidade intermédia. A função retorna -1 se o percurso não existir.

e) [3] Implemente em *assembly* IA-32 a função `routeDistance`, que devolve a distância total do percurso recebido como argumento**.**

2. [2,5] Considere que da compilação de `f1.c` e `f2.c` resultam, respectivamente, `f1.o` e `f2.o`.

f1.c

```
extern int a;
static int b;
extern int c;

void do(int x, int y) {
   y=x; c+=y; b+=1;
}

int main() {
   op(b);
   sh();
   return 0;
}
```

f2.c

```
int a = 2, b = 5, c = 1;

void op(int b) {
   a+=b; do(a, b, c); b+=1;
}

void sh() {
   printf("%d %d %x\n",
      a, b, c
   );
}
```

a) [1] Se ocorrer ligação de `f1.o` com `f2.o`, indique todas as entradas distintas das secções `.data` e `.bss` do executável gerado que têm origem nestes módulos.

b) [1,5] Considere o programa que resulta da ligação de `f1.o` com `f2.o`. Apresente, por ordem de ocorrência, todas as afectações de valores realizadas sobre as variáveis `a`, `b` e `c` de `f2.c`, realizadas durante a execução de tal programa. No final, indique o que é apresentado no *standard output*.

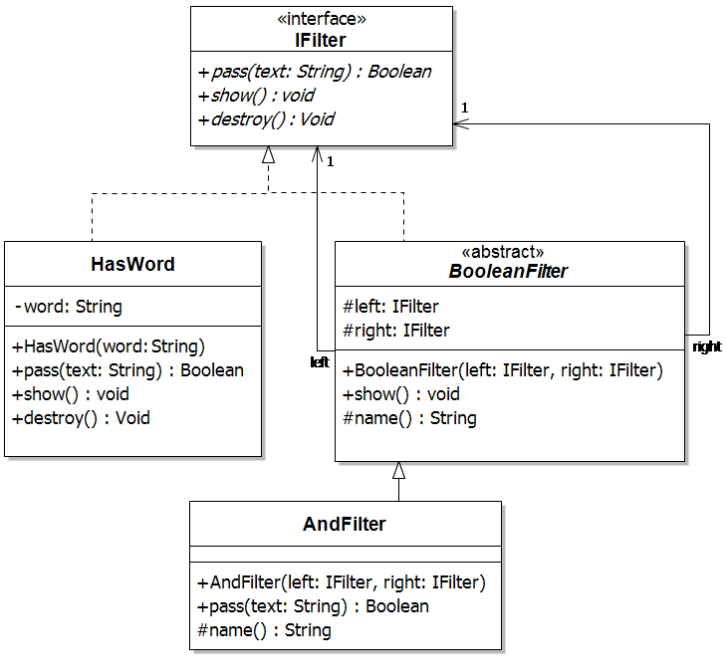
(continua)

3. [2,5] É possível estender um programa Java com código nativo através de bibliotecas de ligação dinâmica, em que as interacções se realizam de acordo com o especificado na Java Native Interface (JNI), abaixo designadas módulos JNI.

a) [1,25] Os módulos JNI têm sempre de ser, eles próprios, ligados com a biblioteca que implementa a API da JNI? Se sim, apresente, para o ambiente de referência de PSC, o comando de invocação do *linker* para produzir o módulo JNI `myext.so` a partir do único ficheiro objecto `myext.o`. Se não, como é que no módulo se chega às funções da JNI?

b) [1,25] Um módulo JNI é provavelmente usado apenas no contexto de um programa Java, pelo não será frequente obter ganhos de ocupação de espaço em disco ou em memória. Tendo em conta que a máquina virtual Java usada no ambiente de referência de PSC é construída em código nativo (maioritariamente em C/C++), por que outro(s) motivo(s) se utilizam bibliotecas de ligação dinâmica para os módulos de extensão via JNI.

4. [6] O diagrama UML apresenta a hierarquia de classes Java usada para definir filtros sobre *strings*. O método `pass` determina se a *string* recebida satisfaz o critério do filtro. O método `show` apresenta uma descrição do filtro e o método `destroy` liberta recursos eventualmente alocados no filtro. Para cada classe, apenas são apresentados os métodos definidos pela primeira vez ou redefinidos nessa classe. O filtro `HasWord` é satisfeito pelas *strings* que contenham a palavra especificada na construção do mesmo. O resultado do filtro `AndFilter` corresponde à operação `and` entre os dois filtros recebidos na sua construção.




```
int main() {
   char *names[] = {"Bom Petisco",
                    "Boa Peça",
                    "bom garfo"};
   int i;
   IFilter *f;
   /* criar filtros */
   f->vptr->show(f);
   putchar('\n');
   for (int i=0; i < 3; ++i)
      if (f->vptr->pass(f, names[i]))
         printf("%s\n", names[i]);
   /* destruir filtros */
}
```

```
HasWord(bom) And IsCapitalized
Bom Petisco
```

Output resultante da execução do programa acima.

a) [1,5] Considere a existência do filtro adicional `IsCapitalized`, que testa se todas as palavras do texto são iniciadas com maiúscula. Acrescente à função `main` acima o código de construção e destruição de filtros de modo a que o *output* do programa corresponda ao indicado.

b) [5] Escreva código em C com:

i. [2] Implementação equivalente e completa de `IFilter` e `HasWord`.

ii. [1] Definição da estrutura `BooleanFilter` e do tipo que representa a respectiva tabela de métodos virtuais.

iii. [1] Implementação do método `show` de `BooleanFilter`, válida para todos os filtros de classes derivadas de `BooleanFilter`.

iv. [1] Implementação do método `name` de `AndFilter` e da tabela de métodos virtuais associada às suas instâncias.

Duração: 2 horas e 30 minutos

Bom teste!