Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

**Programação de Sistemas Computacionais**

Teste Global de 1ª Época, Verão de 2020/2021

---

Nas questões em que não se indiquem explicitamente outras condições, considere as características do ambiente de referência usado na unidade curricular neste semestre.

1. [2,5] Implemente, em linguagem C, a função **`check_for_pattern`** que retorna um valor booleano (0 `false`, 1 `true`) que indica se existe em **value**, alguma sequência de bits igual à sequência entre o *bit* mais significativo com o valor um e o bit menos significativo com o valor um de **`pattern`**. (Exemplos: pattern = 5 e value = 0xa retorna true; pattern = 0xc e value = 3 retorna true; pattern = 0xb e value = 0x2d retorna true )

   ```
   int check_for_pattern(int value, int pattern);
   ```

2. [2,5] Implemente, em linguagem C, a função **`strtrim`** que suprime do início e do fim da *string* passada no parâmetro **`str`**, os caracteres que façam parte da string passada no parâmetro **`delim`**. Exemplo: se for passada a *string* `"  abcd. "` em `str` e `" ,.:"` em **`delim`**, a função **`strtrim`** retorna a *string* com o conteúdo `"abcd"`. A *string* retornada é uma *substring* de **`str`**. Valorizam-se soluções cujo algoritmo execute uma única passagem pelos elementos da *string* **str**.

   ```
   char * strtrim(char *str, const char *delim);
   ```

3. [2,5] Implemente em *assembly* x86-64 a função **`find_first`**, cuja definição em linguagem C se apresenta a seguir.

   ```
   typedef struct item { int key; char label[16]; } Item;
   typedef struct item_coll { size_t length; Item *items[]; } ItemColl;
   char *find_first(ItemColl *ic, int key) {
       for (size_t i = 0; i < ic->length; i++)
              if (ic->items[i]->key == key)
                     return ic->items[i]->label;
       return NULL;
    }
   ```

4. [5] Considere a função **`for_each`**, cuja definição em linguagem C se apresenta a seguir.

   ```
   void for_each(void *array, size_t size, size_t elem_size,
              void(*do_it)(void *, void *), void *context) {
       for (size_t i = 0; i < size; ++i)
        do_it((char *)array + elem_size * i, context);
   }
   ```

   a. [2,5] Implemente a função **`for_each`** em *assembly* x86-64.

   b. [2,5] Para efeitos de teste, escreva em linguagem C, um programa que, utilizando a função **`for_each`**, determina qual é a palavra com mais ocorrências. Considere que as palavras e o respetivo contador de ocorrências se encontram num *array* de *structs* do tipo **`struct word { int counter, char *word }`**.

      Defina um *array* com pelo menos três palavras, a função de processamento específico a passar como argumento em **`do_it`** e a função **`main`**, contendo a invocação de **`for_each`** e a apresentação do resultado.

5. [2,5] Considere o conteúdo dos ficheiros fonte `f1.c` e `f2.c`.

```c
/* f1.c */
int a;
static int b;
extern int c;

void y(int);
void d(int);

void x(int x, int y) {
    a += x;
    b += y;
    d(c);
}

int main() {
    y(a);
}
```

```c
/* f2.c */
#define    D    3

int a = 2;
int b = 1;
int d = D;
int c;

void x(int, int, int);

void y(int c) {
    a+=b;
    x(a, b, c);
    b+=D;
}
```

   a. [1,5] Indique o conteúdo das tabelas de símbolos dos ficheiros objecto relocáveis resultantes da compilação de `f1.c e f2.c`. Para cada símbolo, indique o nome, a secção e o respectivo âmbito (local ou global). Pode usar as convenções do utilitário **`nm`**.

   b. [1] Note que a declaração de **d** é diferente da sua definição. É gerado algum erro na ligação dos dois ficheiros objeto resultantes da compilação de **f1.c** e **f2.c**? Justifique.

6. [1] Qual a dimensão de uma *cache* com 256 conjuntos (*sets*), 8 vias (ways) e 64 *bytes* de bloco? Justifique.

7. [4] O tipo **Data** permite armazenar um número arbitrário de valores do tipo **uint32_t** identificados por uma chave. A chave é definida pelo campo **`key`**, o número de valores é definido pelo campo **`length`** e os valores propriamente ditos são armazenados no campo **`data`**. O tipo **`NodeData`** é usado para organizar instâncias do tipo **`Data`** em listas simplesmente ligadas. A função **`array_to_list_by_key`** devolve uma lista ligada onde cada nó referencia uma réplica dos dados contidos no *array* **`data`** que possuam a chave indicada por **`key`**. Toda a memória utilizada pela função `array_to_list_by_key` para armazenar a lista deverá ser alocada dinamicamente; toda esta memória deverá ser completamente libertada pela função **`free_data_list`**. Implemente as duas funções em linguagem C.

   Nota: Na alocação de memória para armazenar instâncias do tipo `Data` deve ter em consideração que o valor do operador `sizeof (Data)` tem apenas em consideração a dimensão dos dois primeiros campos da estrutura.

```c
typedef struct data { int key; size_t length; uint32_t data[]; } Data;

typedef struct data_node { struct data_node *next; Data *data; } NodeData;

NodeData *array_to_list_by_key(Data *data[], size_t length, int key);

void free_data_list(NodeData *data_list);
```

Duração: 2 horas e 30 minutos

ISEL, 6 de julho de 2021