Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Teste Global de 2ª Época, Verão de 2015/2016

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] Escreva, em linguagem C, a função `strrstr` que pesquisa a *string* `big` procurando ocorrências da *string* `little`. A função devolve o endereço em `big` do início da última ocorrência de `little` ou `NULL` se não detectar nenhuma ocorrência. Não pode utilizar nenhuma das funções da biblioteca *standard* da linguagem C.

   ```
   char *strrstr(const char *big,  const char *little);
   ```

2. [1] Qual o valor do operador `sizeof` em cada uma das seguintes situações? Justifique.

   a) `sizeof(char *)`

   b) `sizeof(struct {int i; char c;})`

   c) `sizeof(int *[4])`

   d) `sizeof(int (*)[4])`

3. [3] Considerando as definições abaixo, apresente uma versão em *assembly* IA-32 da função `find_dblock`.

   ```
   typedef struct dblock { size_t length; double *values; const char *label; } DBlock;

   const char *find_dblock_(DBlock *dblocks, size_t length, double value) {
     for (size_t i = 0; i < length; i++) {
       DBlock *db = &dblocks[i];
       for (int j = 0; j < db->length; j++)
         if (db->values[j] == value)
           return db->label;
     }
     return NULL;
   }
   ```

4. [4] Considere a função `bubble_sort` cuja definição em linguagem C se apresenta a seguir.

   ```
   void bubble_sort(void **items, size_t nitems, int (*cmp)(const void *, const void *)) {
     int i, swapped;
     do {
       for (swapped = 0, i = 1; i < nitems; i++)
         if ((*cmp)(items[i - 1],  items[i]) > 0) {
           void *tmp = items[i - 1];
           items[i - 1] = items[i];
           items[i] = tmp;
           swapped = 1;
         }
     } while (swapped != 0);
   }
   ```

   a) [3] Escreva uma versão da função `bubble_sort` em *assembly* IA-32.

   b) [1] Escreva, em linguagem C, um programa que utiliza a função `bubble_sort` para ordenar, por ordem decrescente, um *array* de ponteiros para valores do tipo `float`. Deverá declarar o *array* de ponteiros e a respectiva iniciação, definir a função de comparação e indicar os argumentos da chamada à função `bubble_sort`.

5. [2] Considere os seguintes conteúdos dos ficheiros fonte `m1.c` e `m2.c`

```c
/* m1.c */
#include <stdio.h>

int a = 10, b = 20;
static int c[] = {30, 40};
int func(int, int *, int *);

int main() {
    printf("%d\n", func(b, c, &c[2]));
    return 0;
}
```

```c
/* m2.c */
extern int a;
int b;

int func(char x, char *y, char *z) {
    return a + b + (z - y);
}
```

a) [1] Indique o conteúdo das tabelas de símbolos dos módulos objecto resultantes da compilação de `m1.c` e de `m2.c`. Para cada símbolo, indique o nome, a secção e o âmbito (i.e., global ou interno).

c) [1] Se considerar que os ficheiros objecto `m1.o` e `m2.o` podem ser combinados com sucesso pelo *linker* para gerar uma imagem executável, diga o que é mostrado na consola quando essa imagem é executada. Se, pelo contrário, considerar que o *linker* falha a combinação, diga qual a razão ou razões pelas quais isso acontece.

6. [1,5] Considere uma *cache* com dimensão 128 *KiB* e organização em mapeamento direto (E = 1). Admitindo que a dimensão do bloco é de 32 *byte*, indique, justificando, dois endereços que colidam na mesma linha da *cache*.

7. [3] O tipo `Chunk` representa uma bloco de dados identificados por uma chave e composto por um número arbitrário de valores do tipo `long`; os valores são armazenados num *array*, com dimensão `length`, cujo endereço é armazenado no campo `values`. O tipo `ChunkNode` é um tipo auxiliar usado para organizar grupos de blocos de dados em lista ligada. A função `merge_chunks_by_key` recebe uma lista de blocos de dados, representado por instâncias do tipo `ChunkNode`, e devolve o endereço de uma instância do tipo `Chunk`, completamente armazenada em memória alocada dinâmicamente, com os dados associados a todos os nós da lista cuja chave seja igual ao argumento `key`, ou `NULL` se não existir nenhum nó na lista com a chave especificada. A função `free_chunk` liberta a memória alocada dinamicamente pela função `merge_chunks_by_key` para armazenar a instância de `Chunk` retornada. Implemente estas duas funções em linguagem C.

```c
typedef struct chunk { char *key; size_t length; long *values; } Chunk;
typedef struct chunk_node { struct chunk_node *next; Chunk chunk; } ChunkNode;

Chunk *merge_chunks_by_key(ChunkNode *clist, const char *key);

void free_chunk(Chunk *chunk);
```

8. [3] O código abaixo implementa, em *Java*, parte de uma hierarquia de tipos usados na validação de representações textuais de números representados em diversas bases de numeração. Apresente a implementação, em linguagem C, da classe base `Validator` de modo a que seja possível integrar novos tipos na hierarquia sem alterar o código desta classe. Apresente também a implementação em linguagem C da classe `HexValidator`.

```
class Validator {
  public abstract boolean isDigitOk(char c);
  public final boolean isValid(String s) {
    for (int i = 0; i < s.length(); ++i)
      if (!isDigitOk(s.charAt(i)))
        return false;
    return true;
  }
}
```

```
class HexValidator extends Validator {
  public boolean isDigitOk(char c) {
    return '0' <= c && c <= '9' ||
           'A' <= c && c <= 'F';
  }
}

class OctalValidator extends Validator {
  public boolean isDigitOk(char c) {
    return '0' <= c && c <= '7';
  }
}
```

Duração: 2 horas e 30 minutos
ISEL, 20 de Julho de 2016