Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Teste Global de 2ª Época, Inverno de 2016/2017

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 `is_binary_palindrome` que determina se a sequência de 32 *bits* passada como argumento é um palíndromo binário. Consideram-se palíndromos as sequências de símbolos cuja leitura é a mesma quando são lidos da esquerda para a direita ou vice-versa. (Por exemplo, a sequência de 32 *bits* `0xa0000005` é um palíndromo binário e a sequência `0xa000000a` é um palíndromo hexadecimal, mas não é um palíndromo binário.)

   ```
   int is_binary_palindrome(unsigned int bits);
   ```

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

   ```
   typedef struct item { int code; int *prices; } Item;
   int get_price(Item items[], size_t nitems, int code, int quantity) {
     for (size_t i = 0; i < nitems; ++i)
       if (items[i].code == code)
         return items[i].prices[quantity];
     return -1;
   }
   ```

3. [4,5] Considere a função `list_search`, cuja definição em linguagem C se apresenta a seguir. Esta função procura numa lista simplesmente ligada, formada por nós do tipo `ListNode`, o primeiro elemento que verifica a condição programada na função passada através do parâmetro `compar`.

   ```
   typedef struct list_node { struct list_node *next; void *data; } ListNode;
   ListNode *list_search(const ListNode *list, int (*compar)(const void *, const void *),
                         const void *context) {
     for (ListNode *p = list->next; p != NULL; p = p->next)
       if ((*compar)(p->data, context) == 0)
         return p;
     return NULL;
   }
   ```

   a) [3] Implemente a função `list_search` em *assembly* x86-64.

   b) [1,5] Escreva, em linguagem C, um programa de teste que utilize a função `list_search` para procurar numa lista, cujo elementos de dados são *strings* C, a primeira ocorrência da *string* que for passada com o argumento `context` daquela função. No programa deve explicitar a iniciação de uma lista com pelo menos três elementos, a definição da função de comparação, a chamada à função `list_search`, assim como a apresentação do resultado.

4. [2,5] Considere o conteúdo dos ficheiro fonte `f1.c` e `f2.c`.

   ```
   /* f1.c */
   #include <stdio.h>

   char *increment(char *p, size_t off);
   int proc(void *);
   extern int BIAS;

   static unsigned short data[] =
   { 3, 0, 0, 0, 10, 0, 20, 0, 30, 0 };
   ```

   ```
   /* f2.c */
   #include <stdio.h>

   #define      BIAS   1000
   unsigned int data = BIAS;
   int main();

   struct vector { size_t len; int data[]; };
   ```

```
int main() {
  printf("%d\n", proc(data));
  return 0;
}
```

```
int proc(struct vector *pv) {
  int r;
  for (r = 0; pv->len > 0; pv->len--)
    r += pv->data[pv->len - 1];
  return r + data;
}
```

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 (e.g., local ou global).

b) [1] Se considerar que os ficheiros objecto `f1.o` e `f2.o` podem ser combinados pelo *linker* com sucesso gerando um ficheiro executável, diga o que é mostrado na consola quando o mesmo executa. Se, pelo contrário, considerar que o *linker* falhará a combinação, diga qual a razão ou razões pelas quais isso acontece.

5. [1,5] Considere uma *cache* com organização *16-way set associative*, cuja capacidade permite armazenar exactamente um *array* com 16384 ponteiros, quando o endereço do *array* é múltiplo de 32 *bytes*. Tendo em consideração que são usados 48 *bits* para exprimir os endereços físicos, indique, apresentando os cálculos necessários, quais os índices dos *sets* da *cache* onde serão colocados os blocos de memória cujos endereços são `0x123456789abcf540` e `0x7f3712344d39efe0`.

6. [3,5] Implemente em linguagem C as funções `doc_to_lines` e `lines_to_doc`. A função `doc_to_lines` converte um documento (zero ou mais linhas de texto passadas através da *string* C `doc`) nas respectivas linhas de texto. A função devolve por valor o endereço do *array* com o conteúdo das linhas; o número de elementos do *array* é devolvida através do parâmetro de saída `out_length`. A função `lines_to_doc` converte um *array* de linhas de texto (`lines`) no respectivo documento (*string* C). Uma linha de texto é constituída pela sua dimensão e pelos respectivos caracteres de acordo com a definição do tipo `Line`. Ambas as funções usam os recursos de memória estritamente necessários para representar os dados a retornar e ambas devem libertar a memória usada pela estrutura de dados de entrada.

```
typedef struct line { size_t len; char *line; } Line;
Line *doc_to_lines(char *doc, size_t *out_length);
char *lines_to_doc(Line lines[], size_t length);
```

7. [2,5] O seguinte código *Java* implementa parte de uma hierarquia de tipos utilizados para representar expressões aritméticas. Apresente uma versão em linguagem C das classes `BinExp` e `Add` que permita integrar novos tipos na hierarquia sem alterar o código base. Indique, justificando, qual a *string* mostrada na consola pela execução do método `Exp.main`. (Tenha em consideração que na implementação do método `BinExp.literal` deve usar despacho dinâmico para invocar os métodos `literal` e `operator`).

```
public abstract class Exp {
  public abstract int eval();
  public abstract String literal();
  public static void main(String[] args) {
    Add add = new Add(new Const(40),
                      new Const(2));
   System.out.println(add.literal() + "="
                      + add.eval());
  }
}
class Const extends Exp {
  private int value;
  public Const(int v) { value = v; }
  public String literal() {
    return Integer.toString(value);
  }
  public int eval() { return value; }
}
```

```
abstract class BinExp extends Exp {
  protected Exp left, right;
  public BinExp(Exp l, Exp r) {
    left = l; right = r;
  }
  public abstract char operator();
  public final String literal() {
    return left.literal() + operator() +
           right.literal();
  }
}
class Add extends BinExp {
  public Add(Exp l, Exp r) { super(l, r); }
  public char operator() { return '+'; }
  public int eval() {
    return left.eval() + right.eval();
  }
}
```