Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Teste Global de 2ª Época, Verão de 2014/2015

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

1. [1.5] Escreva a função `extsign`, que toma o *bit* na posição `n` de `val` como *bit* de sinal e estende-o até ao *bit* de maior peso do tipo `int`. Note que o *bit* de menor peso corresponde à posição 0.
   Por exemplo: `extsign(1,0)==-1; extsign(11,2)==3; extsign(0x0B6C391F,21)==0xFFEC391F`

   ```
   int extsign(int val, size_t n);
   ```

2. [1.5] Implemente a função `strdelc`, que elimina o carácter presente na posição `n` da *string* `str`, desde que `str` tenha pelo menos `n` caracteres. Por exemplo, se `char str[] = "directo";` a instrução `strdelc(str, 4);` deixa `"direto"` em `str`, enquanto `strdelc(str, 8)` não tem efeito.

   ```
   void strdelc(char * str, size_t n);
   ```

3. [2] Considerando as definições abaixo, apresente uma versão da função `checkInfoItem` em *assembly* IA-32.

   ```
   typedef struct info { int id; unsigned count; unsigned char *value; } Info;

   int checkInfoItem(Info *ref, Info *items[], unsigned idx, unsigned vidx) {
     return ref[idx].id == items[idx]->id && ref[idx]->val[vidx] == items[idx]->val[vidx];
   }
   ```

4. [3.5] Desenvolva, em *assembly* IA-32, a função `isort` cuja definição se apresenta a seguir.

   ```
   void isort(void *data[], size_t nelems, int (*compare)(const void *, const void *)) {
     size_t i, j;
     for (i = 1; i < nelems; ++i)
       for (j = i; j > 0 && (*compare)(data[j - 1], data[j]) > 0; j--) {
         void *tmp = data[j];
         data[j] = data[j - 1];
         data[j - 1] = tmp;
       }
   }
   ```

5. [2.5] Considere os ficheiros fonte `f1.c` , `f1.c`, , `f1.c`.

   ```
   /* f1.c */
   #define N 3
   const char NL[] = "nl";
   int main() {
      puts(NL);
      prt(0, N-1);
      return endl();
   }
   ```

   ```
   /* f2.c */
   const char NL = '\n';
   void showc(char c) {
      putchar(c);
   }
   void endl() {
      puts(BAR);
      putchar(NL);
   }
   ```

   ```
   /* f3.c */
   char msg[N] = "LEIC";
   const char * BAR = "|";
   void prt(int b, int e) {
      int i = b;
      for (; i < e; ++i) {
         showch(msg[i]);
      }
   }
   ```

   a) [1] Apresente as tabelas de símbolos dos ficheiros objecto resultantes da compilação dos ficheiros fonte.

   b) [1.5] Enumere os erros. Para cada caso, explicite se provoca falha de compilação ou de ligação e sugira uma correcção, que nunca pode envolver a modificação do ficheiro fonte `f1.c`.

6. [1] Considere uma *cache* com organização *4-way set associative*, cuja capacidade permite armazenar exactamente um *array* com 4096 instâncias de `Info`, quando o endereço do *array* é alinhado a 64 *bytes* (i.e., é múltiplo de 64). Indique, apresentando os cálculos necessários, quantos *sets* tem a *cache*, e quantas instâncias de `info` são armazenadas numa linha.

```
typedef struct { unsigned int id; char key[4]; double factors[3]; } Info;
```

7. [3.5] Considere o conjunto de blocos de dados opacos, armazenados numa lista simplesmente ligada, com nós do tipo `LNode`, e representada pelo ponteiro para o primeiro nó da lista. Cada bloco de dados é armazenado numa instância do tipo `Info`. Implemente a função `listOfInfoToArray` que, a partir dos dados armazenados numa lista, constrói um *array* cujos elementos são do tipo ponteiro para `Info`, que definem o endereço da respectiva instância de `Info`. Toda a memória usada para armazenar o *array* deverá ser alocada dinamicamente; a função devolve, por valor, o ponteiro para o primeiro elemento do *array*, sendo o seu número de elementos devolvido através do parâmetro `elemsPtr`. Implemente também a função `freeInfoArray`, cuja responsabilidade é libertar toda a memória alocada pela função `listOfInfoToArray` aquando da construção do *array*.

```
typedef struct info Info;
typedef struct list_node LNode;
struct info { const char *tag; unsigned count; unsigned data[32]; };
struct list_node {LNode *next; Info *info;};

Info **listOfInfoToArray(LNode *list, unsigned *elemsPtr);
void freeInfoArray(Info **array, unsigned nelems);
```

8. [4.5] O código abaixo implementa, em *Java*, uma parte de uma hierarquia de tipos para processamento de *arrays* de *bytes*. Apresente uma versão em C das classes `Scrambler` e `VScrambler` que também permita integrar novos tipos na hierarquia sem alterar o código base. As chamadas aos métodos de instância de `Scrambler` requerem despacho dinâmico através de uma tabela de métodos virtuais, excepto `scramble`, cujas chamadas são sempre directas.

```
public abstract class Scrambler {
  public void prepare() {}
  public abstract byte process(byte b);
  public final void scramble(byte[] data) {
    prepare();
    for (int i = 0; i < data.length; ++i) {
      data[i] = process(data[i]);
    }
  }
  public static void main(String[] args) {
    byte[] data1 = new byte[] {0,1,2,3,4,5};
    byte[] data2 = new byte[] {9,1,-3,12,0};
    Scrambler cs = new CScrambler((byte)3);
    Scrambler vs = new VScrambler(new byte[] {3,5,2});
    cs.scramble(data1);
    vs.scramble(data2);
  }
}
```

```
public class CScrambler extends Scrambler {
  private byte k;
  public CScrambler(byte k) { this.k = k; }
  public byte process(byte b) { return (byte)(b + k); }
}

public class VScrambler extends Scrambler {
  private byte[] k;
  private int c;
  public VScrambler(byte[] k) { this.k = k; }
  public void prepare() { c = 0; }
  public byte process(byte b) {
    byte r = (byte)(b + k[c]);
    c = (c + 1) % k.length;
    return r;
  }
}
```

Duração: 2 horas e 30 minutos
ISEL, 24 de Julho de 2015