1. [9] Considere um sistema de registo de coordenadas GPS. Cada entrada do registo é representada por um objecto do tipo `GPS_ENTRY`. Estas entradas são guardadas em `gps_all`, com o inteiro `all_sz` a indicar o número total de entradas, das quais as primeiras `sorted_sz` estão ordenadas de acordo com o critério definido em `cmp_entries`:

gps.h

```
#define DTIME_SIZE 14+1
typedef struct gps_entry {
   int group_id;           // id do grupo
   int lat, lng;           // coord GPS
   char dtime[DTIME_SIZE]; // data/hora'\0'
   char * description;     // descrição
} GPS_ENTRY;

GPS_ENTRY * gps_parse(char * line);
int gps_add(GPS_ENTRY * gps);
int gps_sort();
void gps_removeOneOf(int group_id);
```

gps_a.s

```
   .intel_syntax noprefix
   .global gps_add
gps_add:
   xor  eax, eax
   mov  edx, all_sz
   cmp  edx, 0x50
   je  .L0
   mov  ecx, [esp+0x4]
   mov  [edx*4+gps_all], ecx
   inc  edx
   mov  all_sz, edx
   inc  eax
.L0:
   ret
```

gps.c

```
#define MAX_GPS_ENTRIES   0x50
GPS_ENTRY * gps_all[MAX_GPS_ENTRIES]; /* Colecção de aquisições GPS */
int all_sz;                 /* Número total de elementos em gps_all */
int sorted_sz;          /* Número de elementos ordenados em gps_all */

/* Formato de line:
 *      "group_id|latitude|longitude|yyyymmddhhmmss|description"
 *  ex: "1|163850|-398730|20130118190000|desc1" */
GPS_ENTRY * gps_parse(char * line) { /* ToDo */ }

/* Formato dos dh: "yyyymmddhhmmss" */
int cmp_dtimes(const char *dh1, const char *dh2) { /* Done */ }

/* Compara 2 GPS_ENTRY por grupo e dentro do grupo por data/hora.
 * A ordem do grupo e data/hora é crescente */
int cmp_entries(const void * _v1, const void * _v2) { /* ToDo */ }

/* Ordena gps_all segundo o critério defindo em cmp_entries */
void gps_sort() {
   qsort(gps_all, all_sz, sizeof (void*), cmp_entries);
   sorted_sz = all_sz;
}

/* Remove de gps_all uma das entradas do grupo indicado,
 * libertando os recursos alocados */
void gps_removeOneOf(int group_id) { /* ToDo */ }
```

a) [1] Considere a implementação em *assembly* da função `gps_add` presente em `gps_a.s`. Indique qual o critério de adição de uma coordenada GPS considerando os vários cenários de ocupação de `gps_all`.

b) [2] Implemente em C a função gps_`parse` que cria e retorna uma nova entrada GPS_ENTRY, devidamente iniciada com a informação recebida em `line`. O formato de `line`, juntamente com um exemplo, é mostrado no comentário em `gps.c`.

c) [2] Implemente em *assembly* IA-32 a função `gps_sort`, presente no ficheiro `gps.c`.

d) [2] Implemente em C o critério de ordenação dado pela função `cmp_entries` sabendo que `gps_all` é ordenado por grupos e dentro de cada grupo por data/hora. A ordem de ordenação é crescente. A comparação entre duas data/horas está implementado na função `cmp_dtimes` com o critério de retorno igual ao da função `strcmp`.

e) [2] Implemente em C a função `gps_removeOneOf` que remove uma das entradas do grupo indicado, mantendo a ordem relativa das restantes entradas. Deverão ser libertados os recursos alocados para a entrada removida.
NOTA: Sugere-se a remoção da primeira que for encontrada percorrendo sequencialmente as entradas de `gps_all`.

Declaração de algumas funções da biblioteca C:

```
void qsort(void * base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
void * memcpy(void * dest, const void * src, size_t n);
char * strcpy(char * dest, const char * src);
char * strtok(char * str, const char * delim);
int atoi(const char * nptr);
```

2. [2,5] Considere um programa com dois ficheiros fonte: `f1.c` e `f2.c`. Apresenta-se ao lado o código de `f1.c` e o resultado de invocar o utilitário `nm` sobre `f2.o`, que é o ficheiro objecto resultante da compilação isolada de `f2.c`.

a) [1] Apresente o conteúdo que espera encontrar em `f2.h`. Tenha em conta que `f2err_t` é um nome alternativo para `int`.

b) [1] Indique que símbolos existirão no ficheiro objecto resultante da compilação de `f1.c`, indicando, para os definidos no módulo, as secções a que pertencem.

c) [0,5] Apresente a definição de `err` que deverá existir em `f2.c`.

f1.c

```
#include "f2.h"
#define F1_ERR 31
int arg = 3;
int main() {
    int res;
    f2err_t e;
    res = f2_oper(arg);
    e = f2_err();
    return (e == F2_OK) ? res : F1_ERR;
}
```

nm f2.o

```
00000004 D F2_ERR
00000000 D F2_OK
00000000 b err
00000000 T f2_err
0000000a T f2_oper
```

(continua)

3. [2] No programa apresentado ao lado existem dois erros. A constante definida como `CODE` deveria ser `123`, mas foi inserido um `W` por lapso entre o `2` e o `3`. Já em `main`, invoca-se `printhex` com `h` minúsculo, quando deveria ser `printHex` com `H` maiúsculo. Responda às seguintes questões, justificando devidamente as suas respostas:

a) [1] A compilação do código apresentado não terá sucesso. Em que linha ou linhas serão indicados erros?

b) [1] Se a constante `CODE` for corrigida para `123`, mas mantendo-se o erro em `printhex`, que erro ou erros ocorrem durante a geração do executável a partir do código fonte?

```
1  #include <stdio.h>
2
3  #define CODE 12W3
4  #define printHex(VAL) printf("%X", VAL)
5
6  int main() {
7    printhex(CODE);
8    return 0;
9  }
```

4. [1] Uma *cache direct-mapped* pode ser vista como um caso particular de *cache N-way set-associative*, com N=1. Já uma *cache fully-associative* pode ser vista como outro caso particular, em que só existe um *set*. Qual a vantagem das *caches N-way set-associative*, com N baixo mas maior do que 1, em relação aos outros dois tipos?

5. [5,5] Considere o código em C apresentado a seguir que define a interface `InputStream` e a utilização de duas concretizações: `FileInputStream`, que tem como fonte de dados um ficheiro, e `StringInputStream`, cuja fonte é uma *string*.

```
#include <stdio.h>

struct InputStreamMethods;
typedef struct InputStream {
  InputStreamMethods * vptr;
} InputStream;

typedef int (*IS_getc_t)(InputStream *);
typedef int (*IS_cleanup_t)(InputStream *);

typedef struct InputStreamMethods {
  /* leitura de um char da stream
   *   retorna -1 se não há mais caracteres
   */
  IS_getc_t getc;
  /* liberta os recursos internos da stream */
  IS_cleanup_t cleanup;
} InputStreamMethods;

/* Elimina InputStream alocado dinamicamente */
void IS_delete(InputStream * is) {
  is->vptr->cleanup(is); free(is);
}

/* Implementação parcial do construtor
 * de FileInputStream
 */
void FIS_init(FileInputStream *this,
   const char *filename) {
  /* To complete */
  this->file = fopen(filename, "r");
}

/* Implementação de "getc" em FileInputStream */
int FIS_getc(FileInputStream * this)  {
  return fgetc(this->file);
}
```

```
/* Construtor de StringInputStream */
void SIS_init(StringInputStream * this,
              const char * str);

/* Alocação dinâmica dos vários tipos de InputStream */
StringInputStream * newSIS(const char * str);
FileInputStream * newFIS(const char * filename);

void dumpAll(InputStream * strs[], int nstrs) {
  int i, c;
  for (i = 0; i < nstrs; ++i)
    while ((c = strs[i]->vptr->read(strs[i])) != EOF)
      putchar(c);
}

void deleteAll(InputStream * strs[], int nstrs) {
  int i;
  for (i = 0; i < nstrs; ++i) {
    IS_delete(strs[i]);
  }
}

int main() {
   InputStream * streams[] = {
      (InputStream *) newSIS("Primeira linha.\n"),
      (InputStream *) newFIS("teste.txt"),
      (InputStream *) newSIS("Ultima linha.\n")
   };
   const int nstreams= sizeof (streams) /
                       Sizeof (InputStream *);

   dumpAll(streams, nstreams);
   deleteAll(streams, nstreams);
   return 0;
}
```

```
OUTPUT:
Primeira linha.
Conteúdo do ficheiro teste
Ultima linha.
```

a) [1] Implemente as funções `newSIS e newFIS.`

b) [4,5] Complete o construtor de `FileInputStream` e defina os tipos, variáveis e funções que permitem a implementação completa de `FileInputSteam` e `StringInputStream`.

Duração: 2 horas e 30 minutos

Bom teste!