1. [8] Na realização de um programa para gerir uma agenda de tarefas, considere as seguintes definições em C:

agenda.h
```
typedef struct Entry {
   int hour, min;
   char *description;
   struct Entry *next;
} Entry;
typedef struct Agenda {
   int year;
   Entry **entries;
} Agenda;

int leap_year(int year);
```

ly.s
```
.intel_syntax noprefix
.global leap_year

leap_year:
   mov  ecx, [esp + 4]
   xor  eax, eax
   and  ecx, 3
   jnz  lY10
   inc  eax
lY10:
   ret
```

agenda.c
```
/* número de dias de cada mês (ano não bisssexto) */
static int month_days[] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
/* cria e constrói uma nova agenda para o ano indicado */
Agenda *agenda_new(int year) { … }

/* obtém o índice da agenda correspondente ao dia ("d", "m") do ano */
static int get_index(int d, int m, int b) {
   int i=0;
   int pos=0;
   for (i=0; i < m; ++i)  pos += month_days[i];
   if (m > 2) pos += b;
   return pos + d- 1;
}
/* cria uma nova entrada de agenda */
Entry *entry_new(int h, int m, char *description, Entry *next) {
   Entry *e = (Entry *) malloc(sizeof(Entry));
   e->hour = h; e->min = m; e->next = next;
   e->description = strdup(description);
   return e;
}

/* adiciona uma nova entrada na agenda */
void agenda_add_entry(Agenda *a, int day, int month, int h, int m, char *d) {
   int pos = get_index(day, month, leap_year(a->year));
   a->entries[pos] = entry_new(hour, minute, description, a->entries[pos]);
}
```

a) [1] Considere a implementação (simplificada), em *assembly*, da função `leap_year` presente em `ly.s`. Implemente a função em C.

b) [2] Implemente a função `agenda_new` que cria e retorna uma nova agenda do ano indicado, devidamente iniciada. A agenda contém um *array* de ponteiros para estruturas do tipo `Entry`, com tantas entradas quantas o número de dias do ano. Cada ponteiro do *array* representa a cabeça da lista de tarefas do dia correspondente.

c) [2] Implemente em assembly IA-32 a função `get_index`, presente no ficheiro `agenda.c`.

d) [1,5] Realize em C a função `void agenda_remove_entries_for_day(Agenda *a, int dia, int mes)` que remove todas as entradas da agenda presentes no dia do ano passado por argumento.

e) [1,5] Implemente em assembly IA-32 a função `agenda_add_entry`, presente no ficheiro `agenda.c`.

2. [3] Considere o excerto da tabela de símbolos do executável `app` e os ficheiros fonte `a.c`, `b.c` e `d.c`.

a) [1] Escreva um possível `Makefile` que gere a aplicação `app` de acordo com a tabela de símbolos apresentada.

a.c
```
extern int func2(int);
extern int func3(int);
int var1 = 100;

int func1(int v) {
  return func2(v) + func3(v) + var1;
}
```

b.c
```
extern int var1;
static int var2 = 10;
int func2(int v)
{ return v + var1 + var2; }
```

d.c
```
static int func4(int v)
{ return v * 1024; }
int func3(int v) {
  return v + func4(v);
}
```

```
> nm -n app
         U func3
080484d4 T main
08048518 T func1
08048548 T func2
0804a018 D var1
0804a01c d var2
B-bss     D-data
T-text    U-undefined
```

b) [2] Actualize, justificando os cálculos, os valores finais dos símbolos indefinidos presentes nos ficheiros objecto recolocáveis `a.o`, `b.o` e `d.o`, considerando o valor `0xB8092300` para o símbolo `func3`. Considere também que a instrução `CALL` é codificada com um endereço relativo ao PC e que o acesso a uma variável é codificada com um endereço absoluto. Note que apenas estão apresentadas as linhas de código a necessitar de actualização. Indique, justificando, a fase no processo de geração da aplicação onde cada símbolo poderá ser resolvido.

```
> objdump -d -M intel a.o b.o d.o
```

a.o
```
00000000 <func1>:
0d: e8 __ __ __ __ call  <func2>
1a: e8 __ __ __ __ call  <func3>
22: a1 __ __ __ __ mov   eax,<var1>
2f: c3             ret
```

b.o
```
00000000 <func2>:
03:  a1 __ __ __ __ mov   eax,<var1>
0d:  a1 __ __ __ __ mov   eax,<var2>
0c:  c3           ret
```

d.o
```
00000000 <func4>:
0a:  c3            ret
0000000d <func3>:
17: e8 __ __ __ __ call  <func4>
20: c3             ret
```

3. [2] Seja o seguinte programa em C:

a) [1] Indique o *output* apresentado na execução do programa.

b) [1] Escreva o excerto de código que, utilizando a macro XS, inverta os *bytes* de um inteiro na arquitectura IA-32.

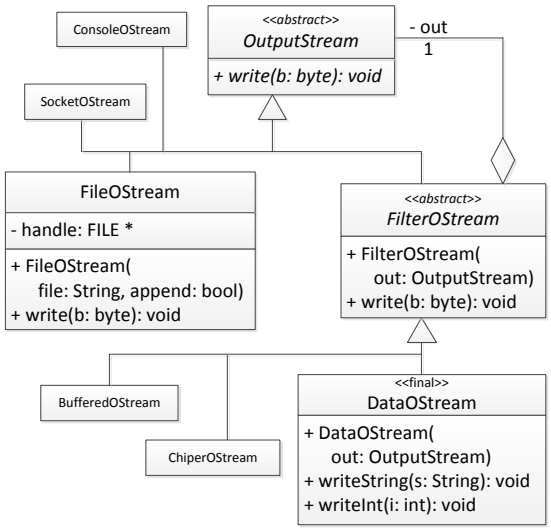
```
#define XS(T,V,I,SZ) \
  { int ni = ((I)+1) % (SZ); \
    T *v = (T*) (V);  \
    T a= v[I]; v[I] = v[ni]; v[ni] = a; }

int main() {
  int i;
  double values[] = { 0.0, 0.2, 0.1, 0.3 };
  XS(double, values, 1, 4);

  for (i=0; i < 4; ++i)
    printf("%lf ", values[i]);
  return 0;
}
```

4. [2] Considerando um sistema com dimensões de tipos iguais às do ambiente de desenvolvimento utilizado em PSC e com uma *cache* de dados de *8-way set-associative* e 2048 sets, e assumindo que uma instância da estrutura `DataBlock` pode ocupar exactamente 1 linha de cache, qual a dimensão da *cache*?

```
struct DataBlock {
  int   id;
  char  names[10][4];
  double values[2];
  int age;
};
```

5. [5] Considere o diagrama de classes que representa um excerto da hierarquia de *streams* da linguagem Java. As classes `FileOStream`, `SocketOStream` e `ConsoleOStream` implementam acessos concretos a *streams*. As classes derivadas de `FilterOStream` acrescentam funcionalidades adicionais à *stream* recebida na construção. Por exemplo, a classe `DataOStream` decora o `OutputStream` associado com as funcionalidades `writeString` e `writeInt`. Deste modo é possível criar uma cadeia de *streams* com funcionalidades adicionais sobre uma *stream* concreta.

Analise o código abaixo. Para efeitos de legibilidade do código foram retiradas directivas de pré-processamento.



filter_ostream.c

```
typedef struct filter_ostream {
   ...
   OutputStream *out;
} FilterOStream;

typedef OStreamVtbl FilterOStreamVtbl;

FilterOStreamVtbl vtbl = {
   ...
};
void filter_ostream_init(
  FilterOStream * this, OStream * out) {
   ...
   this->out = out;
}
void filter_ostream_write(
  FilterOStream * this, byte b) {
   /* Escrita polimórfica do byte b em out */
```

data_ostream.c

```
typedef FilterOStream DataOStream;

DataOStream * data_ostream_new(OStream * out) {
   DataOStream * this =
     (DataOStream *)malloc(sizeof(DataOStream));
   filter_ostream_init((FilterOStream*)this, out);
   return this;
}
void data_ostream_write_int(DataOStream * this, int v)
{
   /* NÃO IMPLEMENTE */
   /* Escrita de v em ASCII no out de this. */
}
void data_ostream_write_string(DataOStream * this,
  const char * s) {
   /* NÃO IMPLEMENTE */
   /* Escrita de s no out de this. */
}
```

a) [3] Escreva na linguagem C uma definição equivalente dos tipos `OutputStream` e `FileOStream`. Tome em atenção a necessidade de libertar recursos alocados nas *streams*.

b) [2] Tirando partido dos tipos definidos, escreva um programa cliente que crie o ficheiro de texto `aluno_id.txt` com o seu número de aluno e, na segunda linha, o seu nome.

Duração: 2 horas e 30 minutos

Bom teste!