1. [1,5] Considere o seguinte ficheiro fonte de um programa em C:

```
#define PRT(a,b) printf("%c=%d\n",*a,(b))
int main() { int d=31; PRT("%s",(d/3)); return 0; }
```

a) [1] Apresente o código resultante da pré-compilação deste ficheiro.

b) [0,5] Descreva os *warnings* apresentados na geração do programa e qual o resultado da sua execução.

2. [3] Um determinado ficheiro executável resulta da ligação de dois ficheiros objecto, `fa.o` e `fb.o`. A seguir apresenta-se um excerto da informação relevante sobre os símbolos, obtida com o utilitário `nm`.

`nm fa.o`

```
         U _a
00000010 b _b
00000000 T _c
00000000 b _d
00000000 d _e
00000010 T _f
00000010 t _g
00000004 D _h
00000008 D _i
```

`nm fb.o`

```
00000000 D _a
00000000 T _d
00000004 d _e
         U _f
00000008 D _i
0000000d T _main
```

```
D – data
T – text
B – bss (data não iniciada)
U – undefined

Maiúscula - global
minúscula – local
```

a) [2] Para cada símbolo correspondente a variáveis, indique uma definição em C que o possa originar.

b) [1] Não é possível realizar a ligação destes dois módulos. Porquê?

3. [10] Para realizar um programa que contabiliza as ocorrências de valores inteiros positivos, considere os ficheiros `VC.h` e `VC.c` seguintes:

```
#include <stdio.h>

typedef struct _VC {
  unsigned value;   /*Valor*/
  unsigned counter; /*Contador de ocorrências*/
} VC;

void toString(VC *r, char *str);
void callAll(VC *v, int dim, void (*fx)(VC*));
void addValue(VC *v, int *dim, unsigned value);
void print(VC *r);
```

```
#include "VC.h"

void toString(VC *r, char *str)
{ sprintf(str,"%u|%u",r->value,r->counter); }

void callAll(VC *v, int dim, void (*fx)(VC*))
{ for( ; dim ; --dim,++v ) fx(v); }

void addValue(VC *v, int *dim, unsigned val) {
  int n = *dim;
  for( ; n ; ++v, --n )
    if (v->value==val) { v->counter++; return;}
  v->value = val;  v->counter = 1; ++*dim;
}
```

a) [2] Apresente uma implementação em IA-32 da função `callAll`.

Considere também o ficheiro `srepl.s` com a implementação em IA-32 da função `sreplace`:

```
sreplace:
  push ebp
  mov  ebp, esp
  mov  edx, [ebp+8]
  mov  cl,  [ebp+12]
  mov  ch,  [ebp+16]
  jmp  .L1
.Loop:
  inc  edx
.L1:
  mov  al, [edx]
  test al, al
  je   .End
  cmp  al, cl
  jne  .Loop
  mov  [edx],ch
  jmp  .Loop
.End:
  pop  ebp
  ret
```

b) [2] Implemente uma função em C equivalente à função `sreplace`.

c) [2] Faça um módulo `print.s` com a implementação em IA-32 da função `void print(VC *r)` que, usando apenas a função `puts` da biblioteca *standard* e as funções `toString` e `sreplace`, escreva no *standard output* os dois campos da estrutura apontada, mas separados por ':' em vez de '|'. Lembre-se que um valor inteiro a 32 bits não tem mais que 10 dígitos na base decimal.

d) [3] Usando as funções anteriores faça em C um módulo `prog.c` com a função `main` de um programa que, usando um *array* de estruturas `VC`, apresenta o número de ocorrências dos diferentes valores inteiros positivos lidos do *standard input*. A dimensão do *array* é indicada como argumento na linha de comando. O programa termina quando for lido um valor negativo.

e) [1] Faça o *makefile* que gera o programa da alínea anterior, compilando separadamente cada um dos módulos.

4. [1,5] As bibliotecas de ligação dinâmica (também conhecidas, em ambiente Unix, como *shared objects*) podem ser carregadas e ligadas com o programa principal logo após o carregamento do ficheiro executável (mas antes de iniciar a execução) ou mais tarde, durante a execução do programa. No entanto, só um destes modelos serve para estender dinamicamente (via *plugins*) a funcionalidade de um programa. Qual e porquê?

5. [4] Considere as seguintes definições de tipos e funções correspondentes a dois esquemas de cifra `Scrambler0` e `Scrambler1`.

```
typedef unsigned char byte;

typedef struct scrambler0 { byte k; } Scrambler0;

void initScrambler0(Scrambler0 * p, byte k) { p->k = k; }
void processScrambler0(Scrambler0 * p, byte * pb) { *pb += p->k; }

typedef struct scrambler1 { byte k[8]; size_t c; } Scrambler1;

void initScrambler1(Scrambler1 * p, byte * k) { memcpy(p->k, k, 8); }
void prepareScrambler1(Scrambler1 * p) { p->c = 0; }
void processScrambler1(Scrambler1 * p, byte * pb) { *pb ^= p->k[p->c]; p->c = (p->c + 1) % 8; }
```

A função `process` altera o conteúdo da sequência de bytes especificada pelo par (`data`, `len`) de acordo com o esquema indicado por `op` e `ctx`. Os dados apontados por `ctx` foram previamente iniciados com a função `initScramble0` ou `initScramble1`, dependendo do valor de `op`.

```
void process(byte * data, size_t len, int op, void * ctx) {
  int i;
  if (op == 1) prepareScrambler1((Scrambler1*)ctx);
  for (i = 0; i < len; ++i)
         switch (op) {
                case 0: processScrambler0((Scrambler0*)ctx, &data[i]);
                case 1: processScrambler1((Scrambler1*)ctx, &data[i]);
                case /* ... */
         }
}
```

Infelizmente, a função `process` tem de ser alterada sempre que é definido um novo esquema.

Apresente as alterações necessárias para que a função `process` utilize chamadas virtuais da seguinte forma:

```
void process(byte * data, size_t len, Scrambler * op) {
  int i;
  op->vptr->prepare(op);   /* chamar método prepare de op  */
  for (i = 0; i < len; ++i)
      op->vptr->process(op,data+i); /* chamar método process de op */
}
```

Duração: 2 horas e 30 minutos

Bom teste!