1. [2] Considere a função `main` do ficheiro `p1.c`:

a) [1] Defina a macro `SUM_ODD_INDEXS` que guarda na variável indicada no último parâmetro o resultado da soma dos elementos dos índices ímpares do *array* indicado como primeiro parâmetro. O segundo e terceiro parâmetros indicam, respectivamente, o tipo dos elementos e o número de elementos do *array*.

b) [1] Indique o *output* que resulta da execução da função.

p1.c
```
int main() {
  int a[3] = {0x01, 0x0203, 0x04050607}, val;
  SUM_ODD_INDEXS(a, char, 8, val);
  printf("Sum char: val = 0x%x\n", val);
  SUM_ODD_INDEXS(a, short, 5, val);
  printf("Sum short: val = 0x%x\n", val);
  SUM_ODD_INDEXS(a, int, 3, val);
  printf("Sum int: val = 0x%x\n", val);
  return 0;
}
```

2. [2] Considere o ficheiro fonte `p2b.c` e os *outputs* dos utilitários `nm` e `objdump` que apresentam, respectivamente, a tabela de símbolos e o conteúdo da secção `.data` do módulo `p2a.o`.

a) [1] Para cada símbolo correspondente a variáveis, indique uma definição em C que o possa originar.

b) [1] Dado que é possível ligar os módulos compilados, indique o *output* do programa.

p2b.c
```
#include <stdio.h>
int w, y;
static int x = 20;
int f1() {
  static short y = 1000;
  y += 100;
  return w+x+y;
}
int f3() {return y+f1()+f1();}
int main() {
  printf("f3() = %d\n", f3());
  return 0;
}
```

```
> nm p2a.o
00000000 t f1
00000024 T f2
         U f3
00000000 d w
00000004 C x
00000004 D y
00000000 b z
```

C–common (data ou bss)
B–bss D–data
T–text U–undefined
Maiúscula-global
Minúscula-local

```
> objdump -j .data -s p2a.o
Contents of section .data:
Offset   [0]      [1]
0000   01000000 0a000000
```

Apresentação na forma *little-endian*

3. [6] Seja o código apresentado a seguir, que representa um excerto de uma calculadora de expressões.

a) [4] Realize em C uma versão igualmente extensível do código Java.

b) [2] Admita a existência de uma classe `Mult` (análoga à classe `Add`). Escreva uma aplicação de teste que cria e construa os necessários objectos da hierarquia apresentada e invoque os métodos de `Exp` para mostrar na consola `3+4*3=15`

```
interface Exp {  /* Expression */
   int eval();
   void show();
}

public abstract class BinExp implements Exp {
   Exp left, right;
   public BinExp (Exp left, Exp right) {
     this.left = left; this.right=right;
   }
   public abstract char getOper();
   public final void show() {
     left.show(); System.out.print(getOper()); right.show();
   }
}

public class Add extends BinExp {
   public Add(Exp left, Exp right) { super(left,right); }
   public char getOper() { return '+'; }
   public int eval() { return left.eval() + right.eval(); }
}

public class Const implements Exp {
   int val;
   public Const(int val) { this.val=val;  }
   public void show() { System.out.print(val); }
   public int eval() { return val; }
}
```

(continua)

4. [10] Na realização de um programa para gerir horários das turmas dos cursos da área departamental, considere as definições dos tipos e as assinaturas das funções do ficheiro `Schedule.h`. Um horário é composto por um conjunto de células. Cada célula representa um ou dois elementos de hora e meia.

O programa de teste apresentado no ficheiro `ScheduleTest.c` usa as funções e os tipos definidos e produz como resultado indicado.

Schedule.h

```
typedef unsigned char PackedTime;
/*  bit7-5 [0..5] --> Dia da Semana (Seg, Ter, Qua, Qui, Sex, Sab) */
/*  bit4-1 [0..9] --> Slot da hora inicial: (8:00,9:30,...,21:30)  */
/*  bit0   [0..1] --> Duração (1:30, 3:00)                         */
/*  Exemplo:  011 0010 1  --> Quinta-feira das 11:00 às 14:00      */

typedef struct _Cell {   /* Célula de 1 ou 2 slots de hora e meia  */
  char uc[4];         /* Nome da unidade curricular. Exemplo: "PSC" */
  PackedTime time;    /* Dia da semana, hora de inicio e duração.   */
} Cell;

typedef struct _Schedule { /* Horário de uma turma */
  unsigned int len;  /* Número de células no horário.  */
  char className[6]; /* Nome da turma. Exemplo: "LI31D"*/
  Cell cell[0];      /* Células (alojamento dinâmico)  */
} Schedule;

typedef enum _WeekDay {MON,TUE,WED,THU,FRI,SAT} WeekDay;

PackedTime tm(WeekDay wd, int slot, int nSlots);

Schedule * createSchedule(const char *name, int maxCells);
void addCell(Schedule *s, const char * uc, PackedTime tm);
void sortSchedule(Schedule *s);
void printSchedule(Schedule *s);
void destroySchedule(Schedule *s);
```

ScheduleTest.c

```
#include "Schedule.h"

int main() {
  Schedule *sh;
  sh = createSchedule("LI31D",4);
  addCell(sh,"PSC",tm(WED, 2 ,2));
  addCell(sh,"PSC",tm(FRI, 2 ,1));
  addCell(sh,"AED",tm(TUE, 1 ,2));
  addCell(sh,"AED",tm(FRI, 1 ,1));
  sortSchedule(sh);
  printSchedule(sh);
  destroySchedule(sh);
  return 0;
}
```

```
Turma LI31D:
AED -> Ter 9:30..12:30
PSC -> Qua 11:00..14:00
AED -> Sex 9:30..11:00
PSC -> Sex 11:00..12:30
```

c) [1] Implemente em C a função `tm` que já está implementada em IA-32 no ficheiro `tm.s`.

d) [3] Implemente em IA-32 a função `printSchedule`, cuja implementação em C consta no ficheiro `printSchedule.c`, continuando a chamar as funções `printCell` e `printTitle` implementadas em C.

tm.s

```
  .intel_syntax noprefix
.globl tm
tm:
  mov     dl, [esp+8]
  shl     dl, 1
  cmp     byte ptr [esp+12], 1
  setg    al
  or      dl, al
  mov     eax, [esp+4]
  shl     al, 5
  or      al, dl
  ret
```

printSchedule.c

```
#include "Schedule.h"
static char *wdName[] = {"Seg","Ter","Qua","Qui","Sex","Sab"};
static char *hTxt[] = {"8:00","9:30","11:00","12:30",…,"23:00"};

void printCell(Cell *c) {
  int tm = (c->time >> 1) & 0xF;
  printf("%s -> %s %5s..%5s\n", c->uc, wdName[c->time >> 5], hTxt[tm], hTxt[tm+1 + (c->time & 1)]);
}
void printTitle(Schedule *s) { printf("Turma %s:\n",s->className); }
void printSchedule(Schedule *s) {
  int n;  Cell *c = s->cell;
  printTitle(s);
  for(n=s->len ; n ; --n)
    printCell(c++);
}
```

e) [2] Implemente em IA-32 a função `cmpTime` implementada em C em `fxSchedule.c`.

fxSchedule.c

```
#include "Schedule.h"
Schedule * createSchedule(const char *name, int maxCells) {...}
void addCell(Schedule *s, const char * uc, PackedTime tm) {...}
void destroySchedule(Schedule *s) { free(s); }

int cmpTime(const void *a, const void *b) {
  return ((const Cell*)a)->time - ((const Cell*)b)->time;
}
void sortSchedule(Schedule *s) {
  qsort(s->cell,s->len,sizeof(Cell),cmpTime);
}
```

f) [2] Implemente em C a função `createSchedule`. Esta função deve alojar dinamicamente uma estrutura Schedule para o número máximo de células indicadas e iniciar os respectivos campos para um horário sem células.

g) [2] Implemente em C a função `addCell`. Esta função acrescenta uma célula ao horário para a unidade curricular. Não é realizada qualquer verificação sobre a validade da célula.

Duração: 2 horas e 30 minutos

Bom teste!