# 1er Parcial

**1.** Una lista ordenada de enteros es una lista cuyos elementos son enteros, y además cumplen el invariante de estar ordenadas: el elemento $i$-ésimo es menor o igual que el elemento $j$-ésimo si $i \leq j$.
Por ejemplo, las siguientes son listas ordenadas:

- $\langle -1, 3, 45 \rangle$
- $\langle 2, 2, 3, 5 \rangle$

En este ejercicio hay que generar una manera de almacenar estas listas en memoria, con este fin deberá inspirarse en las listas enlazadas (`SList`) vistas en clase.[1]

**a)** Definir un tipo `SOList` correspondiente a los nodos, pensaremos una lista como un puntero a estos nodos.

**b)** Definir una operación `SOList *solist_add(int, SOList *)` que dado un entero y una lista, agregue el entero a la lista en la posición correspondiente (según el orden).

**c)** Definir una función `void solist_destroy(SOList *)` que destruya una lista ordenada creada con la función anterior.

**d)** Definir una función `void solist_print(SOList *)` que imprima una lista ordenada pasada por argumento.

**e)** Definir una función de nombre `print_ordered` que dado un arreglo de enteros y su longitud, imprima en pantalla los elementos en orden (para esto, utilizar las funciones anteriores).

**2.** Dado el siguiente código:

```
typedef struct _foos {
  int x, y;
} foos;

void foof(foos s) {
  s.x = 42;
}

void arrf(int a[]) {
  a[0] = 42;
}

void f(void) {
  foos s = {1, 2};
  int a[] = {1, 2};
  foof(s);
  arrf(a);
  printf("%d %d", s.x, a[0]);
}
```

Indicar la salida en pantalla que se produce al llamar a `f`. Explicar las razones.

[1] Tener en cuenta que no es necesario implementar algo especial en la estructura para cumplir el invariante, este invariante se puede asegurar escribiendo correctamente las operaciones.