Ejercicio 1 (3.3 puntos): Disponemos de la clase ***template<class T> matrix_t*** de gestión de matrices que contiene:

- Un método público ***int get_m() const*** que devuelve el número de filas de la matriz.
- Un método público ***int get_n() const*** que devuelve el número de columnas de la matriz.
- La sobrecarga del operador paréntesis que permite acceder a un elemento para su lectura ***const T& operator() (const int, const int) const.***

usando los anteriores elementos, se pide implementar en C++ un nuevo método de la clase ***template<class T> matrix_t*** que detecte si la matriz invocante es simétrica. Recuerde que una matriz M es simétrica si y solo si $\boldsymbol{M = M^t}$. Donde $\boldsymbol{M^t}$ es la matriz traspuesta, que se obtiene a partir de ***M***, cambiando las filas por las columnas.

En el caso de que a matriz invocante no tenga igual número de filas que de columnas se considerará que no es simétrica.

```
template<class T> bool matrix_t<T>::symmetric(void) {
  for(int i{1}; i <= get_m() - 1; ++i) {
    for(int j{i+1}; j <= get_n(); ++j) {
      if (at(i,j) != at(j,i)) return false;
    }
  }
  return true;
}
```

Ejercicio 2 (3.4 puntos): Disponemos de las siguientes clases para gestionar listas listas enlazadas simples:

- ***template<class T> sll_t***, con el atributo privado ***sll_node_t<T>* head_*** y los métodos públicos ***bool empty(void) const***, ***sll_node_t<T>* pop_front(void)*** y ***void push_sorted(sll_node_t<T>* n)***, este último método que no es necesario implementar en este ejercicio, introduce el nodo n en la lista de forma que, si estaba ordenada, permanece ordenada.
- ***template<class T> sll_node_t*** que contiene los atributos privados ***T data_*** y ***sll_node_t<T>* next_*** y los métodos públicos ***sll_node_t<T>* get_next() const***, ***void set_next(sll_node_t<T>*)*** y ***const T& get_data() const***.

Se pide implementar en C++ un método ***sort*** para la clase ***sll_t<T>*** que ordene el contenido del objeto invocante. Este método no recibe parámetros ni devuelve nada. Para resolver este ejercicio puede utilizarse variables auxiliares y cualquier atributo o método de las clases sll_t y sll_node_t, siempre que se respeten los modificadores de acceso (public, private, protected) pero se recomienda usar sólo los que se muestran en este enunciado.

```
template<class T> void sll_t<T>::sort(void) {
  sll_node_t<T>* aux = head_;
  sll_node_t<T>* node;
  if(empty()) return;
  while(aux->get_next() != NULL) {
    if (aux->get_next()->get_data() < aux->get_data()) {
      node = aux->get_next();
      aux->set_next(aux->get_next()->get_next());
      push_sorted(node);
      aux = node;
    } else aux = aux->get_next();
  }
}
```

Ejercicio 3 (3.3 puntos): Se dispone de una clase de C++ ***bitset_c*** de gestión de conjuntos de caracteres en la que la existencia en memoria de elementos se codicará como un bit a 1 o a 0 respectivamente en un atributo privado block_ de tipo entero. El conjunto universal I es ***('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F')*** y la codificación interna del conjunto

Bit	15	14	13	12	11	10		9	8	7		2	1	0
Carácter	'F'	'E'	'D'	'C'	'B'	'A'		'9'	'8'	'7'		'2'	'1'	'0'
Valor ASCII	70	69	68	67	66	65		57	56	55		50	49	48

ha de ser:

a) Asumiendo que los tipos de datos enteros ***char***, ***short***, ***int*** y ***long*** tienen 1, 2, 4 y 8 bytes respectivamente, ¿cuál es el tipo de dato más pequeño que podría tener el atributo interno ***block_***? (0.8 puntos)

RESPUESTA: short (2 bytes -> 16 bits)

b) Implemente un método de la clase que introduzca un carácter en el conjunto atendiendo a la codificación interna antes descrita. El método debe abortar ***(assert)*** si se escribe un carácter no pertenezca al conjunto universal. (2.5 puntos)

```
class bitset_c {
    private:
        block_t block; // Siendo block_t la respuesta al ejercicio (a)
    public: bitset_c(): block_(0) {};
        void insert(const char); // Ejercicio (b)
};
```

```
void bitset_t::insert(const char c) {
  assert((47 < c && c < 58) || (64 < c && c < 71));
  block_t uno = 0x1;
  block_ |= (uno << (c - (c < 58 ? 48 : 55)));
}
```