Ejercicio 1 (3,25 puntos): Disponemos de la clase *sparse_vector_t* de gestión de vectores dispersos que consta de:

- Un método público *int get_nz(void)* const que devuelve el número de elementos no cero dentro del vector disperso.
- Un método público int get_n(void) const que devuelve el tamaño del vector original.
- La sobrecarga del operador corchete que permite acceder a un elemento const pair_t operator[] (const int) const.

donde la clase **pair_t** es para la gestión de pares (double, int) que almacenará los valores respectivamente los elementos distintos de cero del vector disperso y contiene:

- Un método público double get_val(void) const que devuelve el valor del double del par.
- Un método público int get_inx(void) const que devuelve el valor del int del par.

Usando sólo los anteriores elementos, se pide implementar en C++ el operador * que devuelva el producto escalar (un double) de dos objetos de tipo **sparse_vector_t**. Recuerda que el producto escalar de dos vectores a y b es el siguiente:

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \dots + A_n B_n$$

El operador puede implementarse como método de la clase o como función externa, pero siempre deberá comprobar que el tamaño original de ambos vectores será el mismo.

```
double operator*(const sparse_vector_t& a, const sparse_vector_t& b) {
   assert(a.get_n() == b.get_n());
   int aux1{0}, aux2{0};
   double r;
   while (aux1 < a.get_nz() && aux2 < b.get_nz()) {
     if (a[aux1].get_inx() == b[aux2].get_inx()) {
        r += a[aux1++].get_val() * b[aux2++].get_val();
     } else if (a[aux1].get_inx() < b[aux2].get_inx()) {
        ++aux1;
     } else ++aux2;
   }
   return r;
}</pre>
```

Ejercicio 2 (3,5 puntos): disponemos de las siguientes clases para gestionar listas enlazadas simples:

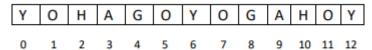
- template<class T> sll_t con el atributo privado sll_node_t* head_ y los métodos públicos bool empty(void) const, void push_front(sll_node_t*) y void insert_after(sll_node_t* prev, sll_node_t*).
- template<class T> sll_node_t con los atributos privados T data y sll_node_t* next y
 los métodos públicos sll_node_t* get_next() const, void set_next(sll_node_t*) y const
 T& get_data() const.

Se pide implementar en C++ un método **push_sorted** para la clase **sll_t<T>** que inserte un nuevo nodo **sll_node_t<T>* n**, dado por parámetro de forma ordenada: todos los valores de los nodos anteriores a **n** deberán tener un dato menor al dato de **n**, y los datos de los siguientes nodos a **n**, deberán tener datos superiores al dato de **n**, ordenados todos ellos de menor a mayor, incluido **n**. Por ejemplo, de tener una lista (1, 3, 5, 7) e introducir el valor 4 en la lista, el método **push_sorted** resultaría en la lista (1, 3, 4, 5 7).

Deberá tenerse en cuenta los casos en los que el dato de *n* sea el menor o el mayor dato de la lista, introduciéndose así en la cabeza o en la cola directamente. Para resolver este ejercicio pueden utilizarse variables artificiales y cualquier método de las clases *sll_t* y *sll_node_t* siempre respetando los modificadores de acceso (private, protected, public).

```
template <class T> void sll_t<T>::push_sorted(sll_node_t<T>* n) {
    sll_node_t<T>* aux = head_;
    sll_node_t<T>* prev = head_;
    if (aux->get_data() >= n->get_data()) {
        push_front(n);
        return;
    }
    aux = aux->get_next();
    while (aux != NULL) {
        if (aux->get_data() >= n->get_data()) {
            insert_after(prev, n);
            return;
        }
        prev = aux;
        aux = aux->get_next();
    }
    insert_after(prev,n);
}
```

Ejercicio 3 (3,25 puntos): Se define como palíndromo aquella palabra o frase que puede ser leída de igual manera de izquierda a derecha o de derecha a izquierda. Por ejemplo, "Yo hago yoga hoy" es un palíndromo. Suponemos que la frase se guarda en un objeto de la clase **vector_t<char>**, a mayúsculas, sin acentos y sin espacios: "YOHAGOYOGAHOY":



Se pide desarrollar un algoritmo *recursivo* que retorne *true* si una frase dentro de un *vector_t<char>* es palíndromo, o *false* si no lo es, y con la siguiente cabecera:

bool is_pal(const vector_t& string, int left, int right)

Ejemplo invocación de la función:

```
vector_t<char> s(13);
a.read() // Introducimos "YOHAGOYOGAHOY"
cout << is_pal(s, 0, s.size() - 1) << endl // resultado 1</pre>
```

NOTA: se valorará la eficiencia del método recursivo desarrollado.

```
template<class T> bool vector_t<T>::is_pal(const vector_t& string, int
left, int right) {
  if (right - left < 1) return true;
  else return string[left] == string[right] ? is_pal(string, left + 1,
  right - 1) : false;
}</pre>
```