Normas para la realización del examen: **Duración: 2.5 horas**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

◁ Ejercicio 1 ▷ Sumatoria [3 puntos]

Dado un vector de enteros `v` y dado un número `T`, se quiere ver si hay una serie consecutiva de elementos del vector que sume `T`. Por ejemplo, si `v = [4,1,3,9,2]`:

- Si `T = 6`, no hay ninguna secuencia.
- Si `T = 12`, sí hay. Sería la secuencia `[3,9]`
- Si `T = 14`, sí hay. Sería la secuencia `[3,9,2]`

Suponga que ya tiene declarado un vector `v` en el `main` y ha leído sus valores, así como el valor `T`. Escriba el código que compruebe lo especificado anteriormente e imprima en pantalla el índice en el que comienza la secuencia (-1 si no existe dicha secuencia).

◁ Ejercicio 2 ▷ Contiguos [3.5 puntos]

Suponga que tiene ya implementado el código de una clase `SecuenciaCaracteres` que representa un conjunto de caracteres consecutivos.

SecuenciaCaracteres
Datos miembros privados:

`static const int TAMANIO = 100`
`char vector_privado[TAMANIO]`
`int total_utilizados`

Métodos públicos que NO hay que implementar y se pueden usar:

`SecuenciaCaracteres()`
`int TotalUtilizados()`
`int Capacidad()`
`void Aniade(char nuevo)`
`char Elemento(int indice)`

Sobre la clase `SecuenciaCaracteres`, se pide construir un método que compruebe si después de cada aparición de un carácter determinado `izda` siempre viene una secuencia de caracteres determinada `dcha`. Cada `izda` debe estar *asociada* a una secuencia `dcha` diferente.
Ejemplos en los que el método devuelve `true`:

```
secuencia principal = a _ T T U _ a T T U     izda = a    dcha = T T U
secuencia principal = a _ a _ T T U T T U     izda = a    dcha = T T U
```

Ejemplos en los que el método devuelve `false`:

```
secuencia principal = a _ T T U _ _ T T U a     izda = a    dcha = T T U
secuencia principal = a _ a _ T T U T T _       izda = a    dcha = T T U
```

Tenga en cuenta lo siguiente:

- En el caso de que la secuencia `dcha` contenga al carácter `izda`, el método devolverá `false`
- No puede utilizar el tipo de dato `string` en ningún sitio.
- Puede definir los métodos auxiliares que estime oportuno.

◁ Ejercicio 3 ▷ Secuenciación Genética [3.5 puntos]

Una **secuencia de ADN** es una secuencia de nucleótidos que pueden tomar uno entre cuatro valores: **A** (Adenina), **C** (Citosina), **G** (Guanina) y **T** (Timina). Para gestionar esta información, una **secuencia de ADN** individual se codificará según la Clase `SecuenciaADN`. Un ejemplo de secuencia de ADN sería el siguiente:

`T C G G G G A T T T T C C`

Nuestro problema a resolver trata sobre un laboratorio que realiza lecturas de varias secuencias de ADN para diferentes células que comparten información genética. De esta manera, se utiliza una clase `TablaSecuenciasADN` para almacenar todas las secuencias de ADN recopiladas (no más de 2000).
Tanto la `SecuenciaADN` como `TablaSecuenciasADN` siguen un diseño estándar, tal como aparece a continuación:

SecuenciaADN
Datos miembros privados:

`static const int MAXLONGITUD = 100`
`char secuencia[MAXLONGITUD]`
`int longitud`

Métodos públicos que NO hay que implementar y se pueden usar:

`SecuenciaADN()`
`int Longitud()`
`int MaxLongitud()`
`void Aniade(char nucleotido)`
`char Nucleotido(int indice)`

TablaSecuenciasADN
Datos miembros privados:

`static const int CAPACIDAD = 2000`
`SecuenciaADN tabla[CAPACIDAD]`
`int total_secuencias`

Métodos públicos que NO hay que implementar y se pueden usar:

`TablaSecuenciasADN(int longitud_secuencia)`
`int TotalSecuencias()`
`int Capacidad()`
`void Aniade(SecuenciaADN secADN)`
`SecuenciaADN SecADN(int indice)`

Aunque las células comparten información genética, las secuencias correspondientes sufren mutaciones, de forma que, por ejemplo una **A** podría cambiar a una **T**. En la siguiente tabla se muestra un ejemplo con un conjunto de secuencias. En mayúsculas aparecen destacados los nucleótidos originales y en minúsculas, las posibles mutaciones (observe que realmente los nucleótidos de las secuencias siempre se representan con una mayúscula. Se han usado minúsculas en la tabla del ejemplo para resaltar la mutación)

T	C	G	G	G	G	g	T	T	T	t	t
c	C	G	G	t	G	A	c	T	T	a	C
a	C	G	G	G	G	A	T	T	T	t	C
T	t	G	G	G	G	A	c	T	T	t	t
a	a	G	G	G	G	A	c	T	T	C	C
T	t	G	G	G	G	A	c	T	T	C	C
T	C	G	G	G	G	A	T	T	c	a	t
T	C	G	G	G	G	A	T	T	c	C	t
T	a	G	G	G	G	A	a	c	T	a	C
T	C	G	G	G	t	A	T	a	a	C	C

De acuerdo a la descripción anterior, implemente los siguientes métodos **indicando en qué clase** estarían definidos (además, puede definir todos los métodos auxiliares adiciones que estime oportuno):

1. `SecuenciaConsenso` que devuelve la secuencia de ADN más probable a partir de la información almacenada en una `TablaSecuenciasADN`. Dicha secuencia es un objeto de la clase `SecuenciaADN` y se forma asignando a su i-ésima posición el nucleótido (A, C, G, T) que más veces se repite en la posición i de todas las secuencias (es decir, en la columna i de la tabla) En caso de empate, se elige cualquiera de los repetidos. En el ejemplo anterior, la secuencia de consenso generada sería: `T C G G G G A T T T T C C`

2. `MismoConsenso` que comprueba si dos objetos de la clase `TablaSecuenciasADN` contienen la misma secuencia de consenso.

3. `SecuenciaInconexa` que calculará y devolverá la secuencia que más se "aleja" de la secuencia de consenso. Para ello utilizaremos la `Distancia de Hamming` que suma el valor 1 si los valores en la misma posición de dos secuencias son distintos, y 0 si son iguales. En caso de igual distancia, queda a elección del programador la secuencia de ADN devuelta. Por ejemplo, la distancia de Hamming entre `T T G C A` y `T T T A A` sería 2.