Práctica 2
Al ladrillo con excavadora y grúa

Fecha límite de entrega: 22 de enero de 2017

Una constructora va a realizar una obra. Desea colocar los materiales necesarios para la obra (ladrillos, yeso, etc.) en una fila, en el orden exacto en que los va a necesitar. Desgraciadamente, cada proveedor ha descargado su material que traía donde ha querido dentro de la fila. La constructora tiene que ordenar los materiales en la fila, y para ello cuenta con una excavadora (que permite empujar los materiales horizontalmente, a izquierda o derecha) y una grúa (que permite levantar los materiales de una zona de la fila y dejarlos caer en otro lugar de la fila).

Imaginemos la siguiente situación:

		1					**5**		**7**	**2**		**3**		**4**		**6**			**8**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

Es decir, en la posición **2** de la fila está el primer material a usar, en la posición **7** está el quinto, etc. La constructora da a su excavadora la orden de empujar, desde la posición **10**, dos posiciones hacia la derecha. En su movimiento de dos pasos hacia la derecha, la excavadora primero empuja hacia la derecha el material **2** un paso, y luego empuja el **2** y el **3** conjuntamente (pues ahora se tocan) otro paso hacia la derecha. Queda así:

		1					**5**		**7**			**2**	**3**	**4**		**6**			**8**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

A continuación, la constructora pide a la grúa que levante todo lo que hay entre las posiciones **12** y **19**. Es decir,

2	**3**	**4**		**6**			**8**

y que lo deje caer desde la posición **4**. Así, lo que estaba en la **12** caerá sobre la **4**, lo que estaba sobre la **13** caerá sobre la **5**, …, y lo que estaba sobre la **19** caerá sobre la **11**. Está prohibido que un material caiga sobre otro, pero afortunadamente, los huecos entre los materiales **4** y **6**, y entre los **6** y **8**, evitan la colisión con los materiales **5** y **7** respectivamente, y todo cae sobre huecos libres. Queda así:

		1		**2**	**3**	**4**	**5**	**6**	**7**		**8**								
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

Ahora los materiales, leídos de izquierda a derecha, respetan el orden en que deben ser usados (no importa que haya huecos entre ellos). Con sólo dos órdenes (en este caso, una a la excavadora y otra a la grúa), se ha resuelto el problema y la obra puede comenzar.

Versión 1 del programa. - Vamos a implementar un programa que, al ejecutarse, nos muestre lo siguiente:

```
1.- Cargar fila de fichero
2.- Guarda fila en fichero
3.- Usar grúa (1 elemento)
4.- Usar excavadora (1 posición derecha)
0.- Salir
```

La opción 1 pedirá al usuario que indique el fichero en el que desea cargar la fila. El formato del fichero consistirá en una secuencia de números, donde cada uno indicará qué hay en la posición correspondiente de la fila (0 si no hay nada), y un -1 indicará el final del fichero. Por ejemplo, un fichero que represente la primera fila mostrada en la página anterior podría ser así:

```
0 0 1 0 0 0 0 5 0 7 2 0 3 0 4 0 6 0 0 8 -1
```

Si el fichero indicado por el usuario existe, se cargará y se mostrará en pantalla el contenido de la cinta siguiendo el siguiente formato:

```
|  |  | 1|  |  |  | 5|  | 7| 2|  | 3|  | 4|  | 6|  |  | 8|
-------------------------------------------------------------
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|
```

La opción 2 del menú pedirá al usuario el nombre del fichero en el que desea guardar la fila, y lo guardará en dicho fichero siguiendo el mismo formato. Mostrará en pantalla la fila guardada en el formato habitual.

La opción 3 es una versión simplificada de la grúa, en la que se permite levantar un único material cada vez. En esta opción se preguntará al usuario qué posición de material quiere levantar, así como en qué posición desea dejarlo caer. Si el movimiento no es posible (porque las posiciones de recogida o de caída están fuera del rango de posiciones válidas de la fila, o porque algún material caería encima de algún otro), lo indicará al usuario y no se hará nada.

La opción 4 es una versión simplificada de la excavadora, donde sólo se permite desplazar un único paso, siempre hacia la derecha. En esta opción se preguntará al usuario desde qué posición quiere realizar el desplazamiento. Recordemos nuestro primer ejemplo:

		1					5		7	2		3		4		6			8
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Si pedimos a la excavadora que empuje desde la posición **9**, la configuración en la que debe dejar los materiales es la siguiente:

		1					5			7	2	3		4		6			8
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Si el movimiento no es posible (pues la posición inicial desde la que queremos empujar no existe), lo indicará al usuario y no se hará nada. Cuando la posición inicial se encuentra libre, el movimiento sí se considera es posible: se considera que la excavadora "desplaza" el vacío un hueco, aunque esto no produce ninguna modificación sobre la fila. Imaginaremos que a la izquierda y a la derecha de la fila hay muros infranqueables. Cualquier intento de la excavadora de "aplastar" materiales contra el muro (es decir, desplazarlos hacia el muro sin que haya huecos libres) no producirá ningún efecto.

Tras realizar la opción 1, la opción 2, la opción 3 o la opción 4, el programa mostrará cómo queda la fila siguiendo el formato habitual, y entonces se volverá a mostrar el menú. La opción 0 terminará el programa.

Antes de cargar alguna fila desde fichero, la fila estará vacía.

Detalles de implementación de la Versión 1.- Declara un tipo de arrays `tArray` de `MAX_PROD` enteros para representar las filas de elementos (ponlo a 25). Declara una constante `PROD_NULO` para representar la ausencia de elemento en una posición (ponla a `0`), y una constante `CENTINELA` para representar el centinela de los archivos (`-1`). Aunque se declare el array con tamaño `MAX_PROD`, el número de casillas reales de la fila que representamos será en general diferente, así que representaremos la fila como un array y una variable contadora que nos diga cuántas posiciones del array son válidas

(independientemente de que contengan algún material o estén vacías). Por ejemplo, podríamos haber usado un array de 25 enteros para almacenar el primer ejemplo, pero la variable contadora recordaría que la fila que realmente queremos representar sólo tiene 20 posiciones. Nada puede ir más a la izquierda de la posición 0 ni más a la derecha de la posición 19 del correspondiente array.

Tu solución debe incluir la implementación de, *al menos*, los siguientes subprogramas. En todas las funciones siguientes, los parámetros `fila` y `tam` denotarán al array de elementos y su variable contadora.

Métodos de manejo de filas:

- ✓ `void mostrarFila(const tArray fila, int tam)`: Dada una fila, la muestra en pantalla siguiendo el formato del final de la segunda página.
- ✓ `bool esPosValida(int tam, int pos)`: Devuelve si la posición `pos` es válida en la fila de tamaño `tam`.
- ✓ `bool estaVacia(const tArray fila, int pos)`: Devuelve si la posición `pos` está vacía.

Métodos de manejo de ficheros:

- ✓ `void leerFilaFich(istream &fich, tArray fila, int &tam)`: Dado el fichero de lectura abierto `fich`, carga su contenido en la fila.
- ✓ `void escribirFilaFich(ostream &fich, const tArray fila, int tam)`: Dado el fichero de escritura abierto `fich`, guarda en dicho fichero el contenido de la `fila`.

Métodos de grúa:

- ✓ `bool esPosibleGrua1Entre(const tArray fila, int tam, int posIni, int posSoltar)`: Devuelve cierto si `posIni` y `posSoltar` son posiciones válidas de la fila y además es posible dejar caer el contenido de la fila situado en `posIni` en su posición `posSoltar`.
- ✓ `bool grua1Elemento(tArray fila, int tam, int posIni, int posSoltar)`: Si es posible, realiza el movimiento de la grúa entre las posiciones `posIni` y `posSoltar`. Devuelve si el movimiento se ha realizado.

Métodos de excavadora:

- ✓ `int posHuecoLibreDerecha(const tArray fila, int tam, int posIni)`: Devuelve la posición del primer hueco libre de la fila si, comenzando por `posIni`, vamos avanzando paso a paso hacia la derecha. Si no hay ninguno, devolverá el valor `tam`.

- ✓ `bool excavadora1Derecha(tArray fila, int tam, int posIni)`: Desplaza la pala de la excavadora desde `posIni` una posición hacia la derecha, empujando desde su posición todos los elementos adyacentes (es decir, no separados por huecos) hacia la derecha, y así, eliminando el primer hueco libre que haya en tal dirección. Recuerda que los materiales empujados no atraviesan los muros del final de la fila. Devuelve un booleano indicando si el desplazamiento ha podido ser realizado. Se considera así si la posición inicial es correcta, aunque no realice ninguna modificación visible sobre la fila.

Métodos del menú:

- ✓ `int menu()`: Muestra el menú de opciones, solicita al usuario su respuesta, y devuelve la opción escogida cuando el usuario haya respondido una opción válida.
- ✓ `void ejecutarLeerFichero(tArray fila, int &tam)`: Pide al usuario el nombre del fichero del que cargará su fila. Si existe, carga la fila, y si no, la fila se inicializa como vacía. Debe informar al usuario sobre si ha logrado realizar la carga, así como mostrar la fila leída.
- ✓ `void ejecutarGuardarFichero(const tArray fila, int tam)`: Pide al usuario el nombre del fichero en el que se guardará la fila. Si logra abrirlo, guarda en el fichero la fila, usando el formato visto antes. Debe informar al usuario sobre si ha logrado realizar el guardado.
- ✓ `void ejecutarGrua(tArray fila, int tam)`: En esta versión, pregunta al usuario qué posición quiere levantar y en qué posición quiere dejarlo caer. Si es posible, realizará el movimiento. Si el movimiento no fue posible, informará al usuario de ello. Se mostrará cómo queda la fila tras el movimiento.
- ✓ `void ejecutarExcavadora(tArray fila, int tam)`: Pregunta al usuario desde qué posición empujará la excavadora. Entonces lo hace, o bien indica que no lo ha hecho (porque la posición desde la que pidió empujar es inválida). Muestra cómo queda la fila tras el movimiento.
- ✓ `void ejecutarOpc(int opc, tArray fila, int &tam)`: Dada la opción escogida por el usuario en el menú, la ejecuta.

Para preparar tu código para la siguiente versión, al terminar la versión 1 es recomendable que añadas también el movimiento de la excavadora hacia la izquierda. Crea las dos funciones correspondientes para el avance a la izquierda, observa que ambas son muy parecidas a las correspondientes

funciones hacia la derecha, y después reemplaza tus versiones de izquierda y de derecha de cada función por una única función común que logre una cosa u otra simplemente en función de un nuevo parámetro entero `direccion` (que significará *derecha* si vale 1 e *izquierda* si vale -1). Generaliza y aprovecha lo que hay en común, es decir, evita que tu código se limite a escoger entre el código que desplaza a la izquierda o el que desplaza a la derecha con un `if`, evita que tu programa tenga dos fragmentos de código casi idénticos (uno por dirección). Tus funciones generalizadas para ambas direcciones serán las siguientes:

```
int posHuecoLibreDir(const tArray fila, int tam, int posIni, int direccion)

bool excavadora1Dir(tArray fila, int tam, int posIni, int direccion)
```

Versión 2 del programa. - Vamos a extender las funcionalidades de las opciones 3 y 4 del menú, de forma que se puedan realizar varios desplazamientos a la vez con la excavadora o levantar varios materiales a la vez con la grúa. En el menú, sustituye las antiguas opciones 3 y 4 por las siguientes:

```
3.- Usar grúa
4.- Usar excavadora
```

El nuevo movimiento de la grúa (opción 3) deberá preguntar tanto la posición inicial y final de los materiales a levantar (extremos izquierdo y derecho del segmento a levantar), como la posición de inicio sobre la que soltarlos. Si el movimiento no es posible (porque las posiciones de recogida o de caída están fuera del rango de posiciones válidas de la fila, porque el extremo izquierdo del segmento a levantar es mayor que su extremo derecho, o porque algún material caería encima de algún otro), lo indicará al usuario y no se hará nada.

Del mismo modo, la ejecución de la excavadora (opción 4) deberá preguntar la posición inicial desde la que realizar el desplazamiento, así como la cantidad de desplazamientos a realizar y en qué dirección realizarlos. Si el movimiento no es posible (pues la posición inicial desde la que queremos empujar no existe), lo indicará al usuario y no se hará nada. Además, recuerda que los muros son infranqueables: si en la dirección del movimiento hay menos huecos libres que el número de posiciones que se pide desplazar, se realizarán sólo los desplazamientos posibles.

Detalles de implementación de la Versión 2.- Tu implementación deberá contar, al menos, con las siguientes nuevas funciones:

Métodos de grúa:

- ✓ `bool sonPosicionesPosiblesGrua(int tam, int posIni, int posFin, int posSoltar)`: Devuelve si el segmento a levantar no tiene su extremo derecho antes que el izquierdo y si todas las posiciones involucradas son

válidas en la fila de tamaño `tam`. Fíjate en que, para comprobar lo segundo, te basta con comprobarlo para los cuatro extremos involucrados.

- ✓ `bool esPosibleGrua(const tArray fila, int tam, int posIni, int posFin, int posSoltar)`: Devuelve si es posible realizar el movimiento de la grúa, es decir, si es posible dejar caer todos los materiales de la fila desde `posIni` hasta `posFin` a partir de la posición `posSoltar`. Recuerda que cada material no puede caer encima de otro material, sino que debe caer sobre un hueco libre. Pero fíjate también en que el segmento a levantar no solapa consigo mismo: si tomamos la segunda fila que se muestra en la página 1 y dejamos caer los materiales **2**, **3** y **4** una posición a la derecha de donde están, el **2** no solapará con el **3** al caer, pues el **3** ya no estará allí.
- ✓ `bool grua(tArray fila, int tam, int posIni, int posFin, int posSoltar)`: Primero se comprueba si el movimiento es posible tal y como descubre la función anterior. Si es así, realiza el movimiento de la grúa de la siguiente manera: se ponen en un array auxiliar los elementos a levantar, se ponen huecos en sus correspondientes posiciones de la fila, y después se sueltan los elementos del array auxiliar en la fila, a partir de la posición a soltar. Recuerda que los materiales anteriores que ya estuvieran en la zona de caída (no levantados ahora) se respetan.

Métodos de excavadora:

- ✓ `bool excavadora(tArray fila, int tam, int posIni, int numDespla, int direccion)`: Primero se comprueba si el movimiento es posible, es decir, si la posición desde la que empezar a empujar es válida. Si es así, se lleva a cabo el movimiento de la excavadora: la excavadora se sitúa en la posición `posIni` y se mueve `numDespla` posiciones hacia la dirección `direccion` empujando los materiales que encuentre a su camino. La implementación se hará en pasadas sucesivas de 1 desplazamiento de toda la hilera de materiales empujados en cada una, ocupando un nuevo hueco en cada uno. Fíjate en que la hilera empujada podría hacerse cada vez más grande a lo largo de las pasadas.

  Si en algún desplazamiento se empujan materiales contra el muro inicial o final, el resto de desplazamientos hasta llegar a `numDespla` no tienen efecto (ej. si en el primer ejemplo de la página 1 empujásemos desde la posición **14** hacia la derecha 30 posiciones, el resultado sería igual que si lo hiciéramos sólo tres: **4**, **6** y **8** quedarían en las posiciones **17**, **18** y **19**).

Los huecos que inicialmente tuviera que atravesar la excavadora hasta tocar algún material y poder empezar a empujar también se descuentan en los `numDespla` desplazamientos (ej. si moviéramos la excavadora desde la posición **13** cuatro posiciones hacia la derecha, obtendríamos lo mismo, pero si lo hiciéramos tres, quedaría un hueco entre el **6** y **8**, ya que la excavadora se pararía antes de llegar a juntarlos).

Nota: Hacer pasadas de 1 desplazamiento de la hilera de materiales en cada una, como pedimos aquí, es ineficiente. En la versión opcional podrás hacer una segunda implementación mejor.

Versión 3 del programa. - Añade la siguiente funcionalidad al programa. El programa llevará la cuenta del número de movimientos de grúa o excavadora realizados. Cada vez que se cargue una fila desde fichero, el número de movimientos se pondrá a 0. Cada orden de mover la excavadora o la grúa sumará un movimiento sólo si la orden tuvo éxito (es decir, si el usuario indicó posiciones válidas para realizar el movimiento). Si tras realizar un movimiento de grúa o excavadora la fila queda ordenada (como en el tercer ejemplo que vimos al principio), el programa se lo indicará al usuario. En caso contrario, le indicará que la fila no está ordenada. En ambos casos, se le indicará después el número de movimientos realizados hasta ahora. El número de movimientos no se inicializa a 0 al ordenar la fila por primera vez, sólo se inicializa al volver a cargar una fila. Para facilitarte la tarea es conveniente que los procedimientos `ejecutarGrua` y `ejecutarExcavadora` devuelvan un `bool` que indique si el movimiento se realizó o no.

Para realizar esa nueva funcionalidad, añade al menos la siguiente función.

- ✓ `bool filaOrdenada(const tArray fila, int tam)`: Devuelve si los materiales en la fila están ordenados de menor a mayor, ignorando los huecos que los separen.

Ahora que ya tienes la funcionalidad obligatoria de la práctica, ¡compruébala mucho realizando muchas pruebas! Por ejemplo, empieza a comprobarla mientras intentas ordenar esta fila en cuatro movimientos o menos (se puede):

1	5			8	11		6		7			9	10	2					3					4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Versión 4, versión opcional. – Vamos a crear nuevas versiones de los movimientos de la grúa y de la excavadora para que sean más eficientes. En la grúa dejaremos de utilizar la fila auxiliar, y en la excavadora calcularemos el resultado del movimiento usando muchos menos cómputos.

```
5.- Usar grúa eficiente
6.- Usar excavadora eficiente
```

El movimiento de la grúa se puede realizar de una forma equivalente, pero **con menos uso de memoria**. Con nuestra versión previa del movimiento de la grúa, necesitas almacenar en memoria una segunda fila para guardar el segmento que se va a desplazar, lo que esencialmente duplica la cantidad de memoria usada por nuestro programa. En la nueva versión, desplazaremos los materiales sobre la misma fila, sin utilizar fila auxiliar.

El movimiento de la excavadora se puede realizar de una forma equivalente pero **más rápida**. Imagina que queremos desplazar cinco materiales cinco posiciones hacia la derecha. Con nuestra versión previa del movimiento de la excavadora, esto supone 25 desplazamientos individuales de materiales: movemos cinco veces todos los materiales (los cinco) una posición hacia la derecha. ¿No sería más inteligente mover cada uno de ellos directamente a su destino final, sin que tenga que moverse cada uno de ellos "por pasadas"? El número de pasos totales de nuestra versión anterior es del orden del cuadrado del número de posiciones de la fila. En la nueva versión, el número de pasos será lineal. Esto no parece muy importante para mover 5 materiales, pero imagina que fueran un millón. ¡No es lo mismo hacer un millón de movimientos que hacer un billón!

Tu versión opcional podría incluir una de las dos funcionalidades nuevas o ambas. Para lograr la máxima puntuación en la parte opcional, realiza ambas.

Detalles de implementación de la Versión 4.- Incluye, al menos, las siguientes nuevas funciones:

Métodos de grúa:

- ✓ `void grua_VO(tArray fila, int tam, int posIni, int posFin, int posSoltar)`: Realiza lo mismo que la función con el mismo nombre pero sin "VO" (Versión Optimizada), pero lo hace sin utilizar ninguna fila ni array auxiliar: todos los cambios se hacen directamente sobre el array de `fila`.

  Ojo: Algo que no deberías olvidar comprobar en tu función es qué pasa cuando dejas caer el segmento una posición a la derecha de donde está, y qué pasa si lo dejas caer una posición a la izquierda de donde está. ¿En qué orden estás sobrescribiendo los elementos del array?

Métodos de excavadora:

- ✓ `int posNHuecoLibreDir(const tArray fila, int tam, int posIni, int numHuecos, int direccion)`: Empezando en `posIni` y yendo en la dirección `direccion`, devuelve el `numHuecos`-ésimo hueco que se encuentre, o la primera posición fuera del array en dicha dirección.

- ✓ `void excavadora_V0 (tArray fila, int tam, int posIni, int numDespla, int direccion)`: Realiza el movimiento de la pala desde `posIni` hacia `direccion` un número `numDespla` de posiciones. Su resultado final deberá ser siempre el mismo que en nuestra versión anterior de la función de la excavadora. La única diferencia es cómo lo conseguiremos: el número de pasos del algoritmo será lineal en lugar de cuadrático.

  Una posible forma de lograrlo: Al principio, avanzaremos la pala a lo largo de todos los huecos que encuentre en la dirección indicada hasta la primera posición ocupada (estos pasos se descuentan de los desplazamientos, igual que hicimos en la otra función). Entonces buscaremos, desde la posición alcanzada en adelante (adelante según `direccion`) tantos huecos como desplazamientos nos queden por hacer, y nos quedaremos con la posición del último de dichos huecos. Si no hubiera tantos huecos, nos iríamos al extremo de la fila en la dirección en que estemos avanzando y, regresando, nos quedaríamos con el primer hueco que encontrásemos. Entonces, en cualquiera de los dos casos, iremos recorriendo la fila hacia atrás desde dicho hueco, recordando dos posiciones en cada momento: la del último hueco, que empezará siendo el hueco encontrado de una forma u otra como hemos dicho, y la de la posición a analizar en cada momento, que empezará siendo la posición anterior a dicho hueco. Si la segunda no está vacía, se intercambia con el hueco, y retrasamos el hueco hasta el hueco anterior yendo hacia atrás. En cualquier caso, la posición a analizar retrocederá una posición. Continuamos así mientras dichas dos posiciones no alcancen la posición desde la que comenzamos a empujar hacia delante.

  Antes de programar esta idea, hazte un ejemplo en papel para ver cómo funciona.

Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea **Entrega de la Práctica 2**, que permitirá subir el archivo `main.cpp` con el código fuente de la versión 3 (o de la 4, si haces la parte opcional). Uno de los dos miembros del grupo será el encargado de subirlo, no lo suben los dos.

Recordad poner el nombre de los miembros del grupo en un comentario al principio del archivo de código fuente.

Fecha límite de entrega: **22 de enero de 2017.**