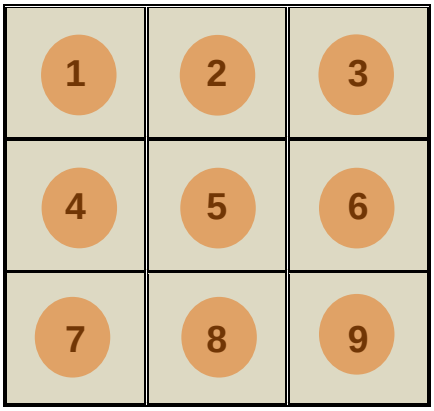
Práctica 1
CodeBreaker

Fecha de entrega: 8 de diciembre

Te has quedado encerrado en una cámara acorazada y la puerta tiene una cerradura codificada. Tienes que introducir la clave para poder salir. Iluminas el teclado numérico de la cerradura con tu luz ultravioleta y descubres que la clave está compuesta por una secuencia de dígitos. Cada vez que pulsas una tecla escuchas un "bip" si el dígito es correcto y un "bop" si no lo es. Sospechas que tras un número de intentos saltará una alarma y si no consigues abrir, te quedarás encerrado todo el fin de semana.



1. Descripción de la práctica

En esta práctica tienes que ir desarrollando de manera incremental un programa que permita jugar a *CodeBreaker*. El programa generará una clave aleatoria y el jugador tiene que adivinar la clave en un número limitado de intentos. En cada intento se lee un código y se muestra el número de dígitos iniciales que se han acertado. Si se aciertan todos se habrá ganado.

Versión 1

El ordenador elige una clave al azar, compuesta, en principio, por una secuencia de 4 dígitos entre 1 y 3. El jugador tiene que adivinar la clave escribiendo otra secuencia. Una vez escrita el ordenador responde mostrando "bip" por cada dígito que sea correcto y "bop" para el primer dígito que no lo es. Una vez que muestra un "bop" ya no muestra nada más para los dígitos restantes. El jugador tiene 3 intentos para adivinar la clave. La longitud de la clave, el número máximo de intentos y el intervalo de dígitos válidos tienen que poderse modificar fácilmente.

Ejemplo de ejecución:

```
Introduce un código (0 para abandonar): 2223
  bip bop -- Acceso denegado!
Introduce un código (0 para abandonar): 2123
  bip bip bop -- Acceso denegado!
Introduce un código (0 para abandonar): 2113
  bip bip bip bip -- OK!
Has utilizado 3 intentos
```

Define al menos las constantes `INTENTOS`, `LONGITUD` e `INTERVALO` con los valores 3, 4 y 3 respectivamente. Implementa al menos las siguientes funciones:

- ✓ `int codeBreaker()`: Conduce el desarrollo del juego y devuelve el número de intentos que ha utilizado el jugador para adivinar la clave. Si no consigue adivinarla después de `INTENTOS` veces, devuelve `INTENTOS+1`; y si abandona devuelve `0`. Utiliza, directa o indirectamente, las funciones que siguen.
- ✓ `int claveAleatoria()`: Genera una secuencia de `LONGITUD` dígitos entre `1` e `INTERVALO` y lo devuelve como resultado.
- ✓ `int invertir(int n)`: Devuelve el número `n` invertido (se invierte el orden de los dígitos). Por ejemplo `invertir(2113)` devolverá `3112`.
- ✓ `int longitudCodigo(int codigo)`: Devuelve el número de dígitos del `codigo` proporcionado.
- ✓ `bool codigoValido(int codigo)`: Devuelve `true` si el código proporcionado tiene exactamente `LONGITUD` dígitos y todos ellos están entre `1` e `INTERVALO`; en caso contrario devuelve `false`.
- ✓ `int numeroBips(int clave, int codigo)`: Devuelve el número de dígitos iniciales coincidentes entre `clave` y `codigo`. Por ejemplo, `numeroBips(2123, 2113)` devolverá `2` y `numeroBips(3113, 2113)` devolverá `0`.

Para generar números aleatorios debes utilizar las funciones `rand()` y `srand()` de la biblioteca `cstdlib`. Una secuencia de números aleatorios comienza en un primer número entero que se denomina semilla (*seed*). Una vez iniciada la secuencia, la función `rand()` genera, de forma pseudoaleatoria, otro entero positivo a partir del anterior. Si quieres que la secuencia esté compuesta por números en un determinado intervalo, deberás utilizar el operador % .

Para obtener un entero entre `1` e `INTERVALO`: `(rand() % INTERVALO) + 1`

Lo que hace que la secuencia se comporte de forma aleatoria es la semilla. Una semilla habitual es el valor de la hora del sistema `time(NULL)`, de la biblioteca `ctime`, ya que es siempre distinta para cada ejecución.

Para establecer la semilla deberás usar: `srand(time(NULL))`

Versión 2

Añade un menú de opciones que permita al jugador configurar el número máximo de intentos, la longitud de la clave y el intervalo de dígitos válidos.

Ejemplo de ejecución:

```
1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
0 - Salir
Opción: 4

Introduce el dígito máximo: 9
Intervalo de dígitos modificado

1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
0 - Salir
Opción: _
```

Añade al programa tres variables (`maxIntentos`, `longitud` e `intervalo`) y tres funciones (`int pedirIntentos()`, `int pedirLongitud()` e `int pedirIntervalo()`) para las opciones del menú 2, 3 y 4 respectivamente. Además, tendrás que adaptar las funciones de la versión 1 añadiendo parámetros. Para cada función que utilice alguna de las constantes (`INTENTOS`, `LONGITUD` o `INTERVALO`), habrá que sustituir el uso de cada una de ellas por un parámetro. Por ejemplo, la función `claveAleatoria();` utiliza las constantes `LONGITUD` e `INTERVALO`, luego habrá que añadir dos parámetros: `int claveAleatoria(int longitud, int intervalo);`

Añade otra función para el menú:

- ✓ `int menu()`: Muestra el menú, pide la opción y la devuelve como resultado. Sólo devolverá una opción válida.

Versión 3

Para mantener los valores de las variables `maxIntentos`, `longitud` e `intervalo` entre ejecuciones del programa, se guardarán los valores en un archivo de texto denominado `configCB.txt`. El archivo contendrá solamente tres enteros separados por un espacio en este orden: intentos, longitud e intervalo.

Haz que al final del programa se guarden en el archivo los valores de las variables en el orden indicado. Y que al iniciarse el programa se lean de dicho archivo los valores

iniciales de las variables. Si no se encuentra el archivo se tomarán como valores iniciales las constantes.

Además añade otra opción al menú para mostrar información acerca del programa:

```
5 - Acerca de CodeBreaker
```

La información que se mostrará (créditos, instrucciones) será la que se encuentre en el archivo `versionCB.txt`.

Añade también una función `bool mostrar(string nombArch);` para la nueva opción del menú. Si el archivo no se encuentra devolverá `false`; en otro caso `true`.

Ejemplo de ejecución:

```
1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
5 - Acerca de CodeBreaker
0 - Salir
Opción: 5

------------------------------------------------------
                 Acerca de CodeBreaker

Práctica 1 Versión 3.1 (30/11/2013)

Fundamentos de la Programación
Facultad de Informática
Universidad Complutense de Madrid

Autores:
  nombre y apellidos (grupo)
  nombre y apellidos (grupo)
------------------------------------------------------

1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
5 - Acerca de CodeBreaker
0 - Salir
Opción: _
```

Incluye en el archivo `versionCB.txt` los nombres de los autores y, si quieres, unas sencillas instrucciones del juego.

Versión 4 (Opcional)

Añade una opción al menú para que el ordenador adivine su propio código, mostrando la misma salida que si se tratase de un jugador humano, pero en un archivo de texto.

```
6 - Rompedor automático
```

Ejemplo de ejecución (con límite de 8 intentos, longitud de 3 e intervalo de 1 a 2):

```
1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
5 - Acerca de CodeBreaker
6 - Rompedor automático
0 - Salir
Opción: 6

Introduce el nombre del archivo: automatico.txt
Proceso realizado!

1 - Jugar
2 - Cambiar el máximo de intentos
3 - Cambiar la longitud de los códigos
4 - Cambiar el intervalo de dígitos
5 - Acerca de CodeBreaker
6 - Rompedor automático
0 - Salir
Opción: _
```

El contenido del archivo `automatico.txt` sería algo así:

```
Introduce un código (0 para abandonar): 111
  bip bop -- Acceso denegado!
Introduce un código (0 para abandonar): 121
  bip bip bop -- Acceso denegado!
Introduce un código (0 para abandonar): 122
  bip bip bip -- OK!
Has utilizado 3 intentos
```

2. Requisitos de implementación

No olvides incluir los prototipos de tus funciones. No utilices variables globales: cada función, además de los parámetros con los que se le pasa información en la llamada, debe declarar localmente las variables que requiera.

3. Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea **Entrega de la Práctica 1**, que permitirá subir el archivo con el código fuente de la última versión (`practica1.cpp`) y el archivo `versionCB.txt`. Asegúrate de poner el nombre de los miembros del grupo en un comentario al principio del archivo de código fuente, ya que basta con que uno de los dos miembros del grupo entregue la práctica en el Campus.

Fecha límite de entrega: **8 de diciembre**