
# Fundamentos de la Programación

## Implementación de la Práctica 3

(Basado en la práctica de Clara María Segura Díaz y Alberto Núñez Covarrubias)

# Índice

# 1. Introducción

- ✓ La práctica consiste en desarrollar un programa en C++ para jugar a **robot turtles**: www.robotturtles.com/instructions.

- ✓ En este juego, cada jugador mueve su tortuga hasta que se alcanza una joya.

- ✓ Antes de comenzar el juego han de colocarse los siguientes elementos en el tablero:
  - Una tortuga por jugador (de 1 a 4).
  - Tantas joyas como jugadores haya. El objetivo del juego es ser el primero en conseguir una de estas joyas.
  - Entre 0 y 20 muros de piedra. Estos muros son fijos e impiden avanzar a las tortugas.
  - Entre 0 y 12 bloques de hielo. Estos bloques también impiden avanzar a las tortugas pero se pueden derretir si se usa una pistola láser.
  - Entre 0 y 8 cajas. Las cajas se pueden empujar en la dirección del movimiento de la tortuga siempre que haya una casilla libre justo detrás en esa dirección donde ponerla. Al empujar la caja, tanto la caja como la tortuga avanzan una casilla.

- ✓ Para alcanzar la joya, cada tortuga tiene disponibles cuatro tipos de **cartas**:
  - Carta “Avanza”: permite a la tortuga avanzar una casilla en la dirección en la que mira si dicho movimiento es posible. La tortuga no puede salirse del tablero, ni atravesar un muro, ni ocupar el lugar que ya está ocupado por otra tortuga, ni empujar una caja que detrás no tenga una casilla libre, por lo que en esos casos la carta no tendrá efecto.
  - Carta “Gira a la derecha”: permite a la tortuga girar en el sentido de las agujas del reloj permaneciendo en la misma casilla.
  - Carta “Gira a la izquierda”: permite a la tortuga girar en el sentido contrario a las agujas del reloj permaneciendo en la misma casilla.
  - Carta “Pistola láser”: la pistola se dispara en la dirección hacia la que mira la tortuga y golpea lo primero que alcanza en esa dirección. Si es un muro de hielo este se deshace. Y en cualquier otro caso no sucede nada.

- ✓ Los jugadores están numerados del 1 hasta el número de jugadores (máximo 4) y en cada ronda juegan en orden creciente de número.
- ✓ Cada jugador dispone de un juego de cartas llamado **mano** con las que poder jugar y otro llamado **mazo** de donde tomar mas cartas para incorporar a su mano (una a una).
- ✓ El mazo inicial consta de 38 cartas que se mezclan de manera aleatoria:
  - 18 cartas “Avanza”
  - 8 cartas “Gira a la derecha”
  - 8 cartas “Gira a la izquierda”
  - 4 cartas “Pistola láser”
- ✓ Inicialmente se asignan tres cartas de la parte superior de cada mazo para formar la mano inicial de cada jugador.

# 2. Consideraciones Iniciales

- ✓ Los módulos necesarios para la solución son: **secuencia**, **juego** y **puntuaciones**.
- ✓ Para la implementación de cada uno de estos módulos (ver las siguientes secciones) no olvides incluir las directivas de compilación condicional en los archivos de cabecera y añade los *#include* que necesites en función de las dependencias entre módulos (quien utiliza a quien).
- ✓ Si todavía no has visto programación modular, implementa todo el código en un mismo archivo *.cpp* siguiendo la estructura ya conocida (librerías, constantes, tipos de datos, prototipos, *main* y subprogramas). Una vez visto el tema de programación modular tendrás que repartir el código en los distintos módulos.

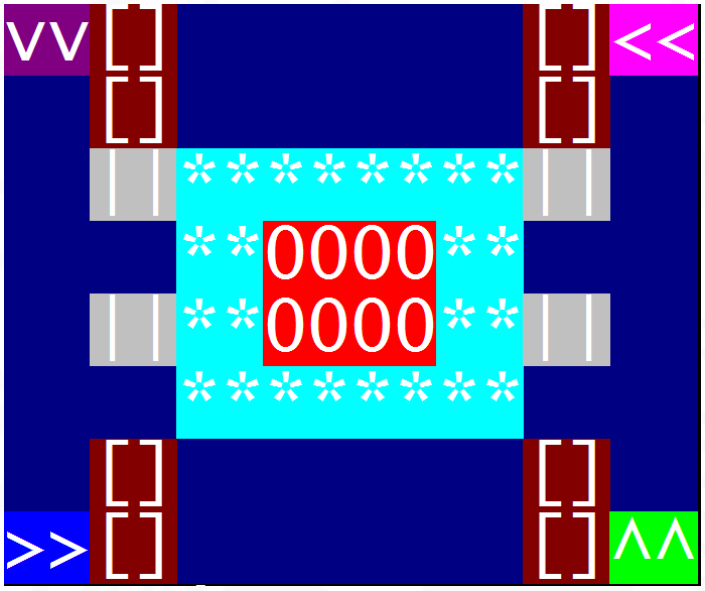
# 3. Funcionamiento General

- ✓ Al comenzar, el programa carga la información de la puntuación de los jugadores del fichero *puntuaciones.txt*.
- ✓ A continuación se muestra el siguiente menú:

```
1. Jugar
2. Mostrar puntuaciones
0. Salir
```

- ✓ El menú vuelve a aparecer después de cada opción (hasta 0).
- ✓ Si se elige la opción 1, el programa solicita el nombre del fichero de tableros, el número de jugadores y sus nombres y, carga el tablero correspondiente del fichero.
- ✓ Cuando el juego termina, el jugador que ha alcanzado la joya con su tortuga obtiene una puntuación igual al número de jugadores de la partida.
- ✓ Al salir se guarda la información de las puntuaciones.

- ✓ Si se elige jugar, el programa muestra el estado del tablero de juego y los jugadores con sus manos.




```
JUGADORES:

      1. Marina: 3 ^  3 <  2 >  2 ~
> 2. Alejandro: 4 ^  2 <  1 >  2 ~
```

- ✓ En cada turno (asignado por orden numérico), el programa pregunta que acción quiere realizar el jugador: **R** (para robar) o **E** (para crear una secuencia y ejecutarla).
- ✓ Si el jugador pulsa R, se roba una carta de la parte superior de su mazo y se añade a su mano.
- ✓ En caso de pulsar E, se solicita la secuencia que se desea crear, mediante las teclas de dirección ↑ para avanzar, → para girar a la derecha, ← para girar a la izquierda, y espacio para el láser. La tecla ENTER marca el fin de la secuencia. Y es necesario comprobar que la mano permite generar dicha secuencia.
- ✓ La ejecución de la secuencia modifica las cartas de la mano (eliminando las cartas seleccionadas), el estado del tablero (con el movimiento de las tortugas, etc.) y el mazo (las cartas utilizadas vuelven a la parte inferior del mazo).

# 4. Implementación del Módulo Secuencia

- ✓ En el archivo de cabecera de este módulo, define constantes para el máximo número de cartas de cada tipo.
- ✓ Define un tipo *tCarta* para representar las cartas.

```
typedef enum {AVANZAR, GIROIZQUIERDA, GIRODERECHA, LASER} tCarta;
```

- ✓ Define un tipo estructurado *tMazo* para representar los mazos con un array de cartas, la posición de la primera carta y el número de cartas. Este tipo sirve también para representar la secuencia de cartas a ejecutar.
- ✓ Define otro tipo estructurado *tMano* para representar las manos de los jugadores. Registra cuantas cartas de cada tipo tiene el jugador en la mano.

- ✓ Declara los siguientes prototipos en el archivo de cabecera e implementa los subprogramas en el archivo de implementación:
  - `void crearVacia(tMazo &mazo) // crea una secuencia vacía de cartas.`
  - `bool sacar(tMazo &mazo, tCarta &carta) // saca una carta de la parte superior. Devuelve true si se ha podido sacar la carta o false en caso contrario.`
  - `void insertar(tMazo &mazo, tCarta carta) // inserta la carta en la parte inferior del mazo.`
  - `void crearMazoAleatorio(tMazo &mazo) // crear un mazo aleatorio. Para ello rellenamos un mazo inicial con las cartas disponibles y aplicamos el método random_shuffle de la librería <algorithm> para generar una mezcla aleatoria del mismo.`

# 5. Implementación del Módulo Juego

- ✓ En el archivo de cabecera, define constantes para los distintos tipos de elementos del tablero. Por ejemplo, declara:

```
const int MAX_HIELOS = 12;
```

- ✓ Define una paleta de colores fija para las casillas, los jugadores y las cartas.

```
const int PALETA[NUM_TIPOS_CASILLAS + MAX_JUGADORES] =
          {1,11,7,4,12,5,13,9,10,3};
```

- ✓ Define un enumerado *tDir* para los cuatro tipos de dirección.

```
typedef enum {NORTE, SUR, ESTE, OESTE} tDir;
```

- ✓ Define un tipo estructurado *tTortuga* para representar el número del jugador que maneja una tortuga y la dirección en la que mira:

```
typedef struct {int numero; tDir direccion;} tTortuga ;
```

- ✓ Declara un enumerado *tEstadoCasilla* con los siguientes valores:

```
typedef enum {VACIA, HIELO, MURO, CAJA, JOYA, TORTUGA} tEstadoCasilla;
```

- ✓ Define un tipo *tCasilla* para representar las casillas del tablero:

```
struct tCasilla {tEstadoCasilla estado; tTortuga tortuga;};
```

- ✓ El tipo *tTablero* es un tipo array bidimensional de *tCasilla*:

```
typedef tCasilla tTablero[NUM_FILAS][NUM_COLUMNAS];
```

- ✓ Define un enumerado *tAccion* para representar las posibles acciones de los jugadores:

```
typedef enum {EJECUTAR_JUGADA, ROBAR_CARTA, NINGUNA} tAccion;
```

- ✓ Define el tipo *tCoordenada* para representar la fila y la columna de una casilla:

```
typedef struct {int x; int y;} tCoordenada;
```

- Define el tipo *tJugador* con al menos los siguientes campos:

```
typedef struct {
    string nombre;
    tMazo mazo;
    tMano jugada;
    tDir direccionActual;
    tCoordenada posicionActual;
} tJugador;
```

- Define el tipo *tJuego* con al menos los siguientes campos:

```
typedef struct {
    int numJugadores;
    int turnoActual;
    tJugador jugadores[MAX_JUGADORES];
    tTablero tablero;
} tJuego;
```

- ✓ Por último, define un tipo enumerado *tTecla* con los siguientes valores:

```
typedef enum {AVANZA, DERECHA, IZQUIERDA, LASER, SALIR} tTecla;
```

- ✓ El archivo de implementación debe contener al menos:
  - `bool cargarJuego(tJuego &juego) // solicita al usuario el nombre del fichero y el número de jugadores, y carga el tablero correspondiente desde ese fichero. También inicializa los mazos y manos de los jugadores.`
  - `void mostrarJuego(const tJuego &juego) // visualiza el estado actual del juego.`
  - `bool ejecutarTurno(tJuego &juego) // lee la acción del jugador actual y la ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya.`
  - `bool accionRobar(tJuego &juego) // el jugador con el turno roba una carta de su mazo si hay cartas disponibles. El booleano indica si ha sido posible robar la carta.`
  - `bool accionSecuencia(tJuego &juego, tMazo &cartas) // el jugador con el turno ejecuta una secuencia de cartas. El segundo parámetro contiene un subconjunto de las cartas del jugador actual y se va consumiendo a medida que se ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya en la ejecución de esta secuencia.`

- `void cambiarTurno(tJuego &juego) // cambia el turno al jugador siguiente.`
- `bool esFinDePartida(tJuego &juego) // comprueba si la partida ha finalizado.`
- `void incluirCarta(tMano &mano, tCarta carta) // incluye una nueva carta en la mano del jugador.`

✓ Los tableros del juego se cargan de un archivo de texto que contiene un tablero para cada número de jugadores y después el tablero. En el siguiente ejemplo se muestra la parte del fichero que se corresponde con el tablero visto:

```
4
DC     CL
 C     C
 #@@@@#
  @$$@
 #@$$@#
  @@@@
 C     C
RC     CU
```

- En la matriz de caracteres, ’#’ representa muro de piedra, ’@’ es muro de hielo, ’ ’ (blanco) es casilla vacía, ’$’ es joya y ’C’ es caja . Para representar una tortuga aparece una letra que indica la dirección en la que mira U, D, R, L (up, down, right y left). Los jugadores se van añadiendo en el orden en que los encontramos al leer los datos de arriba abajo y de izquierda a derecha.
- Cada vez que visualices el estado del tablero, borra primero el contenido de la ventana de la consola, de forma que siempre se muestre el tablero en la misma posición. Para borrar la consola utiliza:

```
system(“cls”);
```

- Por defecto, el color del primer plano con el que se muestran los trazos de los caracteres es el blanco mientras el color de fondo es negro. Podemos cambiar esos colores.

- Disponemos de 16 colores diferentes con valores de 0 a 15. El 0 es el negro y el 15 el blanco. Los demás son azul, verde, cian, rojo, magenta, amarillo y gris, en dos versiones, oscuro y claro.
- Visual Studio incluye una biblioteca, *Windows.h*, que tiene, entre otras, rutinas para la consola. Una de ellas es *SetConsoleTextAttribute()*, que permite ajustar los colores de fondo y primer plano. Incluye en el módulo juego esa biblioteca y esta rutina:

```
void colorFondo(int color) {
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(handle, 15 | (color << 4));
}
```

- Basta proporcionar un color para el fondo (1 a 14) y esa rutina lo establecerá, con el color de primer plano en blanco (15).

- ✓ Debes cambiar el color de fondo cada vez que tengas que dibujar una casilla y volverlo a poner a negro (0) a continuación.
- ✓ Ten en cuenta que cada tortuga tiene un color de fondo distinto, y diferente de los demás elementos del juego. Para facilitar la visualización usa siempre el mismo color para el resto de los elementos de cada tipo, por ejemplo, el gris para los muros de piedra, rojo para las joyas, azul clarito para el hielo y granate para las cajas.
- ✓ Puedes utilizar la paleta de colores vista en las constantes.
- ✓ A través de esta paleta pueden conocerse los colores de los elementos allí donde haga falta.

- ✓ Utiliza los siguientes símbolos para visualizar los elementos del tablero: || para los muros de piedra, ** para los muros de hielo, [] para las cajas, 00 para las joyas; y para las tortugas que miran en las cuatro direcciones posibles usa >>, <<, ^^ y vv.
- ✓ Las manos de los jugadores se mostrarán al lado de su nombre (en el color correspondiente) por orden, apareciendo un símbolo > delante del que tiene el turno.
- ✓ Las cartas se visualizarán con un símbolo del tipo de carta y al lado el número de cartas de ese tipo: ^ para las cartas avanzar, < para girar a la izquierda, > para girar a la derecha, ~ para el láser.
- ✓ Para leer las teclas pulsadas por el usuario implementa también un subprograma:

```
tTecla leerTecla() // devuelve un valor de tipo tTecla, que
puede ser una de las cuatro direcciones si se pulsan las
flechas de dirección correspondientes;  el valor LASER si se
pulsa la tecla espacio; el valor SALIR, si se pulsa la tecla
ENTER; o NADA si se pulsa cualquier otra tecla.
```

- La función *leerTecla*() detectará la pulsación de teclas especiales, concretamente las teclas de flecha (direcciones) y la tecla ENTER (salir). La tecla ENTER sí genera un código ASCII (13), al igual que el espacio (32), pero las de flecha no tienen códigos ASCII asociados. Cuando se pulsan en realidad se generan dos códigos, uno que indica que se trata de una tecla especial y otro que indica cuál es.
- Las teclas especiales no se pueden leer con *get()*, pues esta función sólo devuelve un código. Podemos leerlas con la función *_getch()*, que devuelve un entero, y se puede llamar una segunda vez para obtener el código, si el entero devuelto en la primera llamada corresponde a una tecla especial.

- Esta función requiere que se incluya la biblioteca *conio.h*.

```
cin.sync();
dir =_getch(); // dir: tipo int
if (dir == 0xe0) {
   dir = _getch();
   // ...
}
// Si aquí dir es 13, es la tecla ENTER
```

- Si el primer código es *0xe0*, se trata de una tecla especial. Sabremos cuál con el segundo resultado de *_getch()*. A continuación puedes ver los códigos de cada tecla especial:

↑ 72 → 77 ↓ 80 ← 75

# 6. Implementación del Módulo Puntuaciones

- ✓ Para el módulo de puntuaciones, define una constante:

```
const int MAX_JUGADORES_HISTORIAL = 50;
```

- ✓ y los siguientes tipos:

```
typedef struct{
    string nombre;
    int puntuacion;
}tPuntuacionJugador;
typedef tPuntuacionJugador tArray[MAX_JUGADORES_HISTORIAL];
typedef struct{
        tArray puntuaciones;
        int num_jugadores;
}tPuntuaciones;
```

- En este último módulo debes implementar al menos los siguientes subprogramas:
  - `bool cargarPuntuaciones(tPuntuaciones &puntos) // carga las puntuaciones de los jugadores a partir del fichero.`
  - `bool guardarPuntuaciones(const tPuntuaciones &puntos) // guarda las puntuaciones de los jugadores en el fichero.`
  - `void mostrarPuntuaciones(const tPuntuaciones &puntos) // muestra las puntuaciones de los jugadores.`
  - `bool actualizarPuntuacion(tPuntuaciones &puntos, const string &nombre, int nuevos) // si el jugador no está en el listado lo inserta con los nuevos puntos; si ya estaba, se incrementan sus puntos en nuevos puntos. Si no hay espacio para uno nuevo, se insertará solamente si hay algún jugador con menos puntuación que él, y en tal caso ocupará el lugar del que menos puntos tiene. Devuelve false si no se ha podido insertar.`
- La información de las puntuaciones se carga del fichero *puntuaciones.txt*. En este fichero aparece en cada línea el nombre de un jugador (sin espacios en blanco), separado por un blanco, y un numero natural que representa sus puntos.

- ✓ Se puede suponer que los nombres de los jugadores no están repetidos en el fichero.

# 7. Implementación del Programa Principal.

- El programa principal debe contener *main() y* la función *menu() .*
- En *main()*, declara variables adecuadas para el juego y las puntuaciones y realizar al menos las siguientes operaciones:
- Si la carga de las puntuaciones es correcta:
  - Mientras no sea el final del juego:
    - Mostrar el menú y leer la opción
    - Si la opción es 1:
      - Si la carga del juego es correcta
        - Mientras la partida esté activa
          - Mostrar el juego
          - Ejecutar el turno

- ✓ Si se interrumpe la ejecución porque la tortuga ha alcanzado una joya:
  - ✓ Actualizar puntuaciones.
  - ✓ Partida activa = false
- ✓ Si no se ha interrumpido la ejecución del juego:
  - ✓ Cambiar el turno

✓ Si la opción es 2:

- ✓ Mostrar las puntuaciones

✓ Si la opción es 0:

- ✓ Fin del juego = true
- ✓ Guardar las puntuaciones