



Fundamentos de la Programación

Ayuda Subprogramas de la Práctica 3

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

Índice

1. Introducción
2. Organización de Módulos
3. Módulo Puntuaciones
4. Módulo Secuencia
5. Módulo Juego
6. Programa Principal

1. Introducción

- ✓ Ya hemos presentado la práctica y el juego...
- ✓ En las próximas semanas os ayudaremos en la realización de los subprogramas.
- ✓ Recuerda que cada módulo es una unidad funcional y que llegado el momento (cuando veamos programación modular) podrás dividir el código en sus archivos correspondientes.
- ✓ De momento, añade el contenido de los distintos módulos a un único `.cpp` siguiendo la estructura vista en clase (constantes, tipos, prototipos, *main* y subprogramas).

2. Organización de Módulos

- ✓ Para realizar la práctica hemos ordenado los módulos según su complejidad y dependencias:
 1. **Módulo Puntuaciones.** Representa el registro de jugadores e implementa funcionalidad asociada.
 2. **Módulo Secuencia.** Representa los mazos y secuencias de cartas e implementa funcionalidades asociadas
 3. **Módulo Juego.** Representa el tablero y los jugadores e implementa funcionalidad asociada.
- ✓ Planificación:
 - **Módulo Puntuaciones:** semana del 21/2 (implementar y probar).
 - **Módulo Secuencia:** semana del 28/2 (implementar y probar).
 - **Módulo Juego:** semanas del 7/3, 14/3 y 21/3 (implementar y probar).
 - **Programa Principal:** semana del 28/3 (implementar y probar).

3. Módulo Puntuaciones

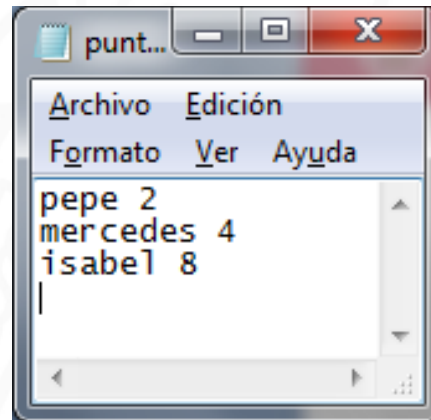
✓ Las constantes y tipos son:

```
const int MAX_JUGADORES_HISTORIAL = 4;
typedef struct{
    string nombre;
    int puntuacion;
} tPuntuacionJugador;
typedef tPuntuacionJugador tArray[MAX_JUGADORES_HISTORIAL];
typedef struct{
    tArray puntuaciones;
    int num_jugadores;
} tPuntuaciones;
```

✓ Los subprogramas pedidos son:

- `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 ya estaba, se incrementan sus puntos en nuevos puntos; si no está en el listado lo inserta con los nuevos puntos (si no hay espacio, se insertará solamente si hay algún jugador en el listado 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 de un fichero *puntuaciones.txt*.



- ✓ Cuando el juego finaliza, las puntuaciones obtenidas se guardan en este mismo fichero.

- ✓ Para probar el módulo implementa una función *main* desde donde se llamen a todos los subprogramas implementados.

```
int main(){
    tPuntuaciones puntos;
    if (cargarPuntuaciones(puntos)) {
        mostrarPuntuaciones(puntos);
        cout << "<----->\n";
        actualizarPuntuacion(puntos, "pepe", 4);
        mostrarPuntuaciones(puntos);
        cout << "<----->\n";
        actualizarPuntuacion(puntos, "carlos", 4);
        mostrarPuntuaciones(puntos);
        cout << "<----->\n";
        actualizarPuntuacion(puntos, "isabel", 4);
        mostrarPuntuaciones(puntos);
        // ... Añade tus pruebas
        system("pause");
        return 0;
    }
```


4. Módulo Secuencia

✓ Algunas constantes y tipos son:

```
const int MAX_CARTAS_MAZO = 38; // cuántas cartas en total
const int NUM_CARTAS_AVANZA = 18; // cuántas cartas avanza
const int NUM_CARTAS_IZQUIERDA = 8; // cuántas cartas girar izda
const int NUM_CARTAS_DERECHA = 8; // cuántas cartas girar dcha
const int NUM_CARTAS_LASER = 4; // cuántas cartas láser
const int NUM_TIPOS_CARTAS = 4; // cuántos tipos de cartas
typedef enum {AVANZAR, GIROIZQUIERDA, GIRODERECHA, LASER} tCarta;
typedef tCarta tCartas[MAX_CARTAS_MAZO];
typedef struct {
    tCartas cartas;
    int numCartas;
} tMazo;
```

✓ Define otro tipo estructurado *tMano* para representar las manos de los jugadores. Registra cuantas cartas de cada tipo tiene el jugador en la mano.

✓ Los subprogramas pedidos para el módulo son:

- `void crearVacia(tMazo &mazo) // crea un mazo o una secuencia vacía de cartas.`
- `bool sacar(tMazo &mazo, tCarta &carta) // saca una carta de la parte superior del mazo (la primera). 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 (al final) 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.`

✓ Para probar el módulo *Secuencia* con otra función *main*, elimina la que habías creado con el módulo *Puntuaciones* e implementa los siguientes subprogramas:

- `string cartaToString(tCarta carta) // devuelve el string que corresponde a la carta dada como parámetro`
- `void mostrarMazo(const tMazo &mazo) // muestra por consola el contenido de un mazo llamando a la función cartaToString()`

✓ Para la generación de un mazo aleatorio realiza las siguientes operaciones:

```
Inicializar contador a 0
Desde contador hasta (contador + NUM_CARTAS_AVANZA)
    Poner carta AVANZAR en el mazo
contador += NUM_CARTAS_AVANZA
Desde contador hasta (contador + NUM_CARTAS_IZQUIERDA)
    Poner carta GIROIZQUIERDA en el mazo
contador += NUM_CARTAS_IZQUIERDA
Desde contador hasta (contador + NUM_CARTAS_DERECHA)
    Poner carta GIRODERECHA en el mazo
contador += NUM_CARTAS_DERECHA
Desde contador hasta (contador + NUM_CARTAS_LASER)
    Poner carta LASER en el mazo
contador += NUM_CARTAS_LASER
mazo.numCartas = MAX_CARTAS_MAZO
srand(time(NULL))
random_shuffle(&(mazo.cartas[0]), &(mazo.cartas[MAX_CARTAS_MAZO-1]))
```

✓ La nueva función *main()* puede ser la siguiente:

```
int main() {  
    tMazo mazo;  
    tCarta carta;  
    bool exito = true;  
    crearVacia(mazo);  
    crearMazoAleatorio(mazo);  
    mostrarMazo(mazo);  
    cout << "<----->\n";  
    exito = sacar(mazo, carta);  
    if (exito)  
        cout << cartaToString(carta) << endl;  
    cout << "<----->\n";  
    insertar(mazo, carta);  
    mostrarMazo(mazo);  
    cout << "<----->\n";  
    // ... Añade tus pruebas  
    system("pause");  
    return 0;  
}
```

5. Implementación del Módulo Juego

- ✓ Algunas de las constantes necesarias para la representación del tablero son:

```
const int NUM_FILAS = 8; // alto tablero
const int NUM_COLUMNAS = 8; // ancho tablero
const int MAX_HIELOS = 12;
const int MAX_JUGADORES = 4;
const int MAX_CARTAS_JUGADA = 10;
```

- ✓ Define también una paleta de colores constante para las casillas, los jugadores y las cartas

```
const int NUM_TIPOS_CASILLAS = 6; // nº de tipos de estado de casilla
const int NUM_COLORES = NUM_TIPOS_CASILLAS + MAX_JUGADORES;
const int PALETA[NUM_COLORES] = {1, 11, 7, 4, 12, 5, 13, 9, 10, 3};
```

- ✓ Los colores de la paleta se asignan en este orden: casilla vacía, hielo, muro, caja, joya, tortuga1, tortuga2, tortuga3, tortuga4, cartamano. Por ejemplo, cartamano = 3.

✓ Define los siguientes enumerados para el juego:

```
typedef enum {NORTE,SUR,ESTE,OESTE} tDir;  
typedef enum {VACIA,HIELO,MURO,CAJA,JOYA,TORTUGA} tEstadoCasilla;  
typedef enum {EJECUTAR_JUGADA,ROBAR_CARTA,NINGUNA} tAccion;  
typedef enum {AVANZA,DERECHA,IZQUIERDA,PLASER,SALIR,NADA} tTecla;
```

✓ Los tipos necesarios son:

```
typedef struct {  
    int numero; // asocia tortuga con jugador  
    tDir direccion;  
} tTortuga;  
typedef struct {  
    tEstadoCasilla estado;  
    tTortuga tortuga; // 0 si no hay tortuga en la casilla  
} tCasilla;  
typedef tCasilla tTablero[NUM_FILAS][NUM_COLUMNAS];
```



```
typedef struct {  
    int x;  
    int y;  
} tCoordenada;  
typedef struct {  
    string nombre;  
    tMazo mazo;  
    tMano jugada;  
    tDir direccionActual;  
    tCoordenada posicionActual;  
} tJugador;  
typedef struct {  
    int numJugadores;  
    int turnoActual;  
    tJugador jugadores[MAX_JUGADORES];  
    tTablero tablero;  
} tJuego;
```

✓ Incluye al menos los siguientes subprogramas:

- `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 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.

- ✓ Según la planificación, tenéis tres semanas para implementar y probar este código:
 - Semana 7/3: cargar y mostrar el juego.
 - Semana 14/3: acción robar y acción secuencia.
 - Semana 21/3: ejecutar y cambiar el turno.
- ✓ El mayor esfuerzo estará concentrado en la semana 14/3 por la complejidad de la acción secuencia.

✓ En la función *cargarJuego()* realiza las siguientes operaciones:

Inicializar ok = true

Pedir y leer el nombre del fichero de tableros

Pedir y leer un número válido de jugadores

ok = cargar el tablero del fichero (ver la siguiente transparencia)

Si ok

 Poner el turno actual a 0

 Para cada jugador

 Pedir y leer el nombre del jugador

 Crear un mazo aleatorio

 Inicializar la mano del jugador (todos los contadores a 0)

 Sacar del mazo e incluir en la mano las tres primeras cartas

Devolver ok

✓ La carga del tablero consiste en:

Inicializar ok = false

Inicializar numJugador = 0

Abrir el fichero que contiene el tablero

Si el fichero se ha abierto

Leemos el número de jugadores del fichero

Leemos una línea del fichero

Mientras no sea el número de jugadores del juego

Repetir getline() hasta NUM_FILAS

Leemos el número de jugadores

Leemos una línea del fichero

ok = true

Para cada fila

Para cada columna

Leemos un carácter

Guardamos coordenada

Cargamos la casilla (juego, coordenada, numJugador, carácter)

leemos línea del fichero

Devolvemos ok

✓ Cuando cargamos la casilla:

Vemos el tipo de casilla según el caracter

Guardamos el tipo en el estado de la casilla

Si tipo no es TORTUGA

Ponemos el número de la tortuga de la casilla a 0

Si tipo es TORTUGA

Vemos la dirección según el caracter

Guardamos coordenada en posición actual de numJugador en el juego

Guardamos dirección en direccion actual de numJugador en el juego

Incrementamos numJugador

Guardamos número y dirección de la tortuga en la casilla

Guardamos la casilla en la posición [coordenada.x][coordenada.y] del tablero

- ✓ En la función *mostrarJuego()* realiza las siguientes operaciones:

```
system("cls");
```

Para cada fila

Para cada columna

```
dibujaCasilla (tablero[fila][columna])
```

Para cada jugador

Poner color de fondo según Paleta

Si turnoActual == numJugador mostrar ">"

Mostrar el nombre del jugador

Poner color de fondo 0

Mostrar número de cartas Avanzar

Poner color de fondo 3

Mostrar carácter " ^ "

```
// ... Añade todas las cartas de la mano del jugador
```

- ✓ Actualización de la planificación:
 - Semana 14/3: cargar y mostrar el juego (continuación).
 - Semana 21/3: acción robar y acción secuencia.
 - Semana 28/3: ejecutar turno y cambiar turno.
 - Semana 4/4: programa principal y pruebas finales.
- ✓ La entrega es el día 5/4.

✓ Para *accionRobar()* implementa el siguiente pseudo-código:

Inicializar resultado = true

Si podemos sacar carta del mazo del jugador juego.turnoActual

Aumentamos uno de los campos de la mano del jugador según la carta

En caso contrario (no se ha podido sacar carta)

resultado = false

Mostrar un mensaje de error

Devolver resultado

✓ Para *accionSecuencia()* implementa:

Inicializar hayGanador = false

Mientras queden cartas que ejecutar y !hayGanador

 Sacar carta de la secuencia

 switch (carta)

 Si la carta es un giro a la derecha

 girarDerecha(juego)

 Sino si la carta es un giro a la izquierda

 girarIzquierda(juego)

 Sino si la carta es avanzar

 avanzarTortuga(juego) y actualizar hayGanador

 Sino

 dispararLaser(juego)

Devolver la carta al mazo del jugador que tiene el turno

Eliminar la carta de la mano del jugador que tiene el turno actual

Devolver hayGanador

✓ *avanzarTortuga(tJuego &juego)*

Inicializar haGanado = false

Obtener la coordenada destino según la dirección

Obtener la coordenada destino de una caja según la dirección

Si la coordenada destino está en el tablero

 Guardar coordenada origen (posición actual)

 Si destino es casilla VACIA

 Actualizamos el tablero

 Actualizamos la posición actual del jugador

 Sino, si el destino es una CAJA

 Si el destino de la caja está en el tablero

 Si el destino de la caja es casilla VACIA

 Mover la caja

 Mover la tortuga

 Actualizamos la posición actual del jugador

 Sino, si el destino es una JOYA

 Actualizamos la posición actual del jugador

 Poner VACIA la casilla origen

 Poner VACIA la casilla destino donde está la joya

 haGanado = true

✓ *lanzarLaser(tJuego &juego)*

Inicializar buscarObstaculo = true

Guardar en destino la posición actual del jugador que tiene el turno

Mientras buscarObstaculo == true

 Obtener el destino según la dirección

 Si el destino se sale del tablero

 buscarObstaculo = false

 Sino, si la casilla destino es MURO, CAJA TORTUGA o JOYA

 buscarObstaculo = false

 Sino, si la casilla destino es HIELO

 buscarObstaculo = false

 Derretir el hielo poniendo casilla destino a VACIA

✓ *bool ejecutarTurno(tJuego &juego):*

```
jugadaValida = false; haGanado = false
```

```
Repetir mientras !jugadaValida
```

```
Leer y validar la acción del jugador con "introduce acción [R, E]"
```

```
switch (accion)
```

```
case 'R':
```

```
    jugadaValida = accionRobar(juego)
```

```
case 'E':
```

```
    Creamos una secuencia vacía de cartas y la pedimos
```

```
    Repetir
```

```
        tecla = leerTecla()
```

```
        Si tecla no es SALIR ni NADA
```

```
            Vemos la carta y la insertamos en la secuencia
```

```
            mientras tecla distinta de SALIR y NADA y secuencia valida
```

```
            Si tecla == NADA o secuencia no valida
```

```
                jugadaValida = false
```

```
            Sino,
```

```
                jugadaValida = true;
```

```
                haGanado = accionSecuencia(juego, secuencia)
```

```
Devolver haGanado
```

✓ *tTecla leerTecla().*

```
Declarar tTecla tecla
cin.sync();
dir = _getch(); // dir: tipo int
if (dir == 0xe0) {
    dir = _getch();
    if (dir == 72) tecla = AVANZA;
    else if (dir == 77) tecla = DERECHA;
    else if (dir == 75) tecla = IZQUIERDA;
    else tecla = NADA;
} else if (dir == 32) {
    tecla = PLASER;
} else if (dir == 13) {
    tecla = SALIR;
} else
    tecla = NADA;
Devolver tecla
```

6. Programa Principal

- ✓ El programa principal debe contener las funciones *main()* y *menu()*.
- ✓ En la función *main()* declara variables adecuadas para el juego y las puntuaciones y realizar al menos las siguientes operaciones:

```
finJuego = false
```

```
Si la carga de las puntuaciones es correcta:
```

```
  Mientras !finJuego:
```

```
    Mostrar el menú y leer la opción
```

```
    Si la opción es 1:
```

```
      Si la carga del juego es correcta
```

```
        partidaActiva = true
```

```
        Mientras partidaActiva
```

```
          Mostrar el juego
```

```
          Ejecutar el turno
```

Si se interrumpe la ejecución del turno porque la tortuga ha alcanzado una joya:

Actualizar puntuaciones.

partidaActiva = false

Si no se ha interrumpido la ejecución del turno:

Cambiar el turno

Si la opción es 2:

Mostrar las puntuaciones

Si la opción es 0:

finJuego = true

Guardar las puntuaciones