Fundamentos de Programación

P2 Práctica 2. Versión 1

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

Raquel Hervás Ballesteros
Virginia Francisco Gilmartín
Facultad de Informática
Universidad Complutense

Objetivo

El objetivo es realizar un simulador de trucos de cartas. Trabajaremos con la baraja francesa, donde los palos son picas, tréboles, diamantes y corazones, y para cada palo tenemos los "números" A, 2, 3,…, 9, 10, J, Q, K.

Se podrán realizar diferentes operaciones sobre mazos de cartas (barajar, cargar de fichero, cortar, repartir,…), y luego realizar varios trucos de magia utilizando las operaciones anteriores.

Desarrollo incremental

- ✓ Versión 1: manipulación de mazos
  - ✓ Cargar
  - ✓ Barajar
  - ✓ Añadir mazo
  - ✓ Cortar mazo
  - ✓ Guardar
- ✓ Versión 2: Repartos de mazos
  - ✓ Separar en negras y rojas
  - ✓ Separar en bajas y altas
  - ✓ Repartir en tres montones
- ✓ Versión 3: Truco de los tres montones
- ✓ Versión 4: Truco de la posada
- ✓ Versión 5: Truco del jugador desconfiado (opcional)

Planificación

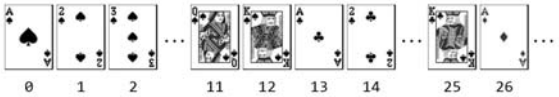
Entrega:
- ✓ 1 de febrero

Planificación recomendada:
- ✓ 8 de enero: versión 1
- ✓ 16 de enero: versión 2
- ✓ 23 de enero: versión 3
- ✓ 30 de enero: versión 4

¿Dudas de funcionamiento?
- ✓ Demo ejecutable en el campus virtual

Representación de los datos (I)

- ✓ Las 52 cartas de la baraja están representadas por números enteros entre 0 y 51



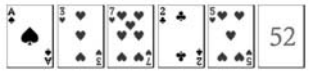
- ✓ Para representar las cartas:
  - ✓ Tipos enumerados para los palos (`tPalo`) y los números (`tNumero`)
    - — Para los números poned nombres representativos:
      `as, dos, tres, ..., jota, reina, rey`
  - ✓ Tipo para la carta como sinónimo de `int`

```
typedef int tCarta;
```

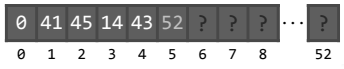
  - ✓ Constante para el número de cartas por palo

Representación de los datos (II)

- ✓ Los mazos no siempre estarán completos
  - ✓ Necesitamos entonces marcar dónde termina el mazo
  - ✓ Usamos un centinela, un valor especial dentro del mazo que marca el final
- ✓ Mazo de cartas:
  - ✓ Número máximo de cartas → 53
    - ➢ Tamaño del array (52 cartas + centinela)
  - ✓ Array de `tCarta` de `MAXCARTAS` posiciones
  - ✓ `CENTINELA = 52`. Para señalar el final del mazo



mazo



Recorridos en los mazos

- ✓ Como no sabemos cuántas cartas hay en el mazo, no podemos usar un `for`

```
int i = 0;
while (mazo[i] != CENTINELA) {
   ...
   i++;
}
```

No vale contar cuántas cartas hay en el mazo y luego usar un `for`. ¡Es muy ineficiente!

Búsquedas en los mazos

- ✓ Esquema típico de búsqueda, pero el final lo marca el centinela, no el tamaño

```
int i = 0;
while (mazo[i] != CENTINELA && !encontrado) {
   ...
   i++;
}
```

Subprogramas para dar el número y el palo

- ✓ `tPalo darPalo(tCarta carta)`
  - ✓ El valor del enumerado será el que esté en la posición
    `carta / CARTASPORPALO`
  - ✓ Con un molde se hace la transformación
    `tPalo(carta / CARTASPORPALO)`

- ✓ `tNumero darNumero(tCarta carta)`
  - ✓ El valor del enumerado será el que esté en la posición
    `carta % CARTASPORPALO`
  - ✓ Con un molde se hace la transformación
    `tNumero(carta % CARTASPORPALO)`

void escribirCarta(tCarta carta)

1. Se obtiene el palo de la carta con `darPalo(…)`
2. Se muestra por pantalla el palo
   - ➢ Recuerda que los enumerados no se pueden mostrar directamente por pantalla
3. Se obtiene el número de la carta con `darNumero(…)`
4. Se muestra por pantalla el número
   - ➢ Recuerda que los enumerados no se pueden mostrar por pantalla
     - — Para el as, jota, rey y reina se usa un texto
     - — Para los demás se puede calcular el número que hay que mostrar por pantalla automáticamente
       `int(numero) + 1`
       Ejemplo:
       `int(2) = 1 + 1 ⟶ 2`

bool cargarMazo(tMazo mazo)

- ✓ Cada línea del fichero contiene:
  - — Una letra indicando el palo de la carta
  - — Un número indicando el número de la carta

```
p 1
d 4
p 12
d 13
t 2
t 4
c 12
x
```

1. Leemos el palo y el número de cada carta
2. Obtenemos el `tCarta` correspondiente a dicho palo y número y lo guardamos en la posición del array que corresponda
   - ➢ Recuerda que en el array las cartas se representan como enteros del 0 al 51
     - – Si `palo=='p'`, entonces `tCarta(num-1)`
     - – Si `palo=='t'`, entonces `tCarta(CARTASPORPALO + num-1)`
     - – Si `palo=='d'`, entonces `tCarta(2*CARTASPORPALO + num-1)`
     - – Si `palo=='c'`, entonces `tCarta(3*CARTASPORPALO + num-1)`
3. Al final de la lectura, hay que poner el centinela del array en la posición siguiente a la última ocupada

void barajarMazo(tMazo mazo)

1. Se calcula cuántas cartas hay en el mazo
   - ✓ Usando la función `cuantasEnMazo(const tMazo mazo)`
2. Se calcula el número de intercambios (3 * cartas en el mazo)
3. En cada intercambio:
   - ✓ Se eligen dos posiciones aleatorias válidas
   - ✓ Se intercambian las cartas de esas posiciones aleatorias

bool unirMazos(tMazo mazoOriginal, const tMazo nuevoMazo)

1. Se calcula cuántas cartas hay en los dos mazos
   - ✓ Usando la función `cuantasEnMazo(const tMazo mazo)`
2. Si la suma es mayor que 52, no caben las cartas en el mazo
   - ✓ Se devolverá `false` al final de la función
3. Si sí caben, a partir de la última posición ocupada, hay que introducir las cartas del mazo nuevo
   - ✓ Se añaden cartas hasta que se llegue al centinela del mazo nuevo
   - ✓ Se coloca el centinela al final del mazo original con las cartas nuevas

bool partirMazo(tMazo mazoOrigen, int cuantasCoger, tMazo mazoDestino)

1. Se comprueba si hay suficientes cartas para coger
   - ✓ Usando la función `cuantasEnMazo(const tMazo mazo)`
2. Si no hay suficientes cartas
   - ✓ Se devolverá `false` al final de la función
3. Si sí hay suficientes
   - ✓ Pasamos al `mazoDestino` las cartas que nos indiquen en `cuantasCorger`
   - ✓ Ponemos el centinela en el `mazoDestino`
   - ✓ Eliminamos del mazo origen las cartas que hemos pasado al `mazoDestino`
     - — Para ello desplazamos hacia la izquierda todas las cartas que quedan, tantas posiciones como cartas hayamos llevado al `mazoDestino`

void cortarMazo(tMazo mazo, int cuantas)

1. Se parte el mazo usando la función `partirMazo(...)`
2. Si todo ha salido bien, se unen los mazos con `unirMazos(...)`

Otros subprogramas

- ✓ `void guardarMazo(const tMazo mazo)`
  - ✓ La traducción para escribir en fichero se hace similar a la de `escribirCarta(…)`
- ✓ `void crearMazoVacio(tMazo mazo)`
  - ✓ Se pone el centinela en la posición 0 del array
- ✓ `int cuantasEnMazo(const tMazo mazo)`
  - ✓ Recorrido para contar las posiciones ocupadas del array
- ✓ `void escribirMazo(const tMazo mazo)`
  - ✓ Recorrido. Para cada carta del array se llama a `escribirCarta(…)`

Programa principal (I)

- ✓ El programa principal guardará un mazo sobre el que se realizarán las operaciones
  - ✓ Hay que inicializarlo al principio usando `crearMazoVacio(…)`
- ✓ Tendremos:
  - ✓ Un menú (función)
  - ✓ Un `while` (mientras el usuario no pulsa la opción 0, se sigue en el programa)
  - ✓ Un `switch` con la ejecución de las opciones

Programa principal (II)

- ✓ Dentro de cada opción hay que realizar varias operaciones
  - ✓ Opción cargar mazo: Carga el mazo y muestra el mazo cargado o un error
  - ✓ Opción barajar: Muestra el mazo, lo baraja, y lo vuelve a mostrar
    - ➢ Si el mazo del programa está vacío no hace nada e informa del error
  - ✓ Opción añadir mazo: Carga el mazo a añadir, muestra el mazo del programa, el mazo que se va a añadir, los une y muestra el mazo resultante
  - ✓ Opción cortar mazo: Muestra el mazo del programa, pregunta al usuario por dónde cortar, corta y muestra el mazo cortado
    - ➢ Si el mazo del programa está vacío no hace nada e informa del error
  - ✓ Opción guardar: Sólo hace el guardado del mazo del programa

Programa principal (III)

- ✓ Para que el main sea más claro, hacemos un subprograma para cada opción

```
switch (opcion){
   case 1:  opcionCargarMazo(mazo);
            break;
   case 2:  opcionBarajarMazo(mazo);
            break;
   case 3:  opcionUnirMontones(mazo);
            break;
   case 4:  opcionCortarMazo(mazo);
            break;
   case 5:  guardarMazo(mazo);
            break;
}
```


Tratamiento de errores

- ✓ Los errores deben tratarse en los subprogramas de las opciones, no dentro de las funciones básicas
- ✓ Ninguno de los siguientes subprogramas debería usar `cin` o `cout`
  - `void barajarMazo(tMazo mazo)`
  - `bool unirMazos(tMazo mazoOriginal, const tMazo nuevoMazo)`
  - `bool partirMazo(tMazo mazoOrigen, int cuantasCoger, tMazo mazoDestino)`
  - `void cortarMazo(tMazo mazo, int cuantas)`
    - ➢ Cuando hace falta, devuelven como resultado si ha habido algún error

Tratamiento de errores. Ejemplo

```
void opcionCargarMazo(tMazo mazo) {
   //El nombre del fichero se pide dentro de cargarMazo
   if (cargarMazo(mazo))
      escribirMazo(mazo);
   else
      cout << "Error: fichero no encontrado." << endl;
}
```


Fundamentos de Programación

P2 Práctica 2. Versión 1

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

Raquel Hervás Ballesteros
Virginia Francisco Gilmartín
Facultad de Informática
Universidad Complutense