Universidad Nacional de Ingeniería
Facultad de Ciencias
Escuela de Matemática

Ciclo 2017-2.

[Curso CM − 094 : Estructúra de Datos]
[Prof.: Rósulo Perez Cupe]

02 de Octubre.

IMPLEMENTACION DEL JUEGO SOLITARIO EN C

1 Integrantes

1. Marco Leon R 20160775E
2. Porlles Chávez Miluska 20150167B
3. Jean Alvitres Palomino 20142591C
4. Lázaro Camasca Edson 20160468E
5. Rivera Granados Roger Guido 20130507B
6. Josué Zacarías Olivet Rojas 20032157G

2 Resumen

El solitario es considerado como un juego de cartas que necesita de mucha paciencia y destreza. Precisamente, el nombre de refiere al hecho de que sólo hay un jugador en competencia. No existe ninguna historia concreta atribuida a este juego, pero es probable que apareciera con el nacimiento de los naipes y como entretenimiento personal. Nuestra tarea en este proyecto será el implementar este juego en lenguaje C, usando la teoría de manejo de pilas, colas y colas dobles, aprendida en clase.

3 Descripción del juego

El juego consiste en usar un maso de cincuenta y dos cartas y distribuir veintiocho en siete columnas escalonadas de cartas volteadas, excepto la más superpuesta. El resto de cartas estarán en un pozo de donde se irán extrayendo de tres en tres y apilando en las cartas superpuestas siguiendo el orden decreciente del número y el no tener cartas del mismo color adyacentes. De la misma forma se dispondrán cuatro columnas donde se ordenarán de forma creciente, empezando por el `As` y sólo las del mismo palo.
El juego será ganado si se tienen todas las cartas del maso ordenadas en las once columnas.

4 Pilas

Una pila es una lista lineal en la que las inserciones y remociones de elementos ocurren sólo en el extremo de la misma, este extremo es llamado *tope*. Véase **Figura 1.** También se le conoce como lista LIFO (del inglés Last in First Out, "último en entrar, primero en salir"). Esta estructura se aplica en multitud de supuestos en el área de informática debido a su simplicidad y capacidad de dar respuesta a numerosos procesos. En el siguiente apartado explicaremos el funcionamiento del juego mediante el algorimo de funcionamiento en pseudocódigo

5 Implementación

Solitario
MAXPJ=13
MAXPS=13
MAXVP=63
Carta baraja: Array$[1, ..., 52]$de tipo Elemento
Base: Array$[1, ..., 14]$
Tope: Array$[1, ..., 14]$
Carta Mesa: Array$[1, ..., MAX_VP]$
inicializarPilas(Array tope, Array base,Entero 14)
ponerBaraja(Array Mesa,Array baraja,Array base, Array tope)
actualizar(Array Mesa, Array tope)
imprimirJuego(Array Mesa,Array base,Array tope) Base: Es un array de números enteros donde se almacenarán las Bases de cada pila.
Tope: Es un array de números enteros donde se almacenarán los topes de cada pila.
Carta Mesa: Es un array de números enteros donde coexisten las 14 pilas.
Carta Baraja; es un array de números enteros donde se almacenarán las cartas y es de tipo elemento tiene la siguiente estructura: Carta Baraja
valor: variable de tipo entero.
color: variable de tipo caracter
tipo: variable de tipo caracter
estado: variable de tipo caracter

- valor : es el valor de la carta $(1 - 12)$
- color : es el color de la cart (rojo-negro)
- tipo : es el tipo de carta (T:Trebol E:Espada C:Corazón O:Oros)
- estado : es el estado de la carta (A:Abierto C:Cerrado)

Base: Es un array de números enteros donde se almacenarán las Bases de cada pila.
Tope: Es un array de números enteros donde se almacenarán los topes de cada pila.
Carta Mesa: Es un array de números enteros donde coexisten las 14 pilas. inicializarPilas: Inicializa todas las pilas y las coloca en orden (dejando un espacio entre cada pila) Como al inicio ninguna pila tiene elemento, entonces el tope coincide con la base de cada pila; tiene la siguiente estructura: inicializaPilas(Array tope, Array base, Entero n)
i:variable de tipo entero
tope[0] = 0;
base[0] = 0;
base[1] = 24;

desde $i = 2$ hasta n
base[i] =base[$i-1$] + $i - 1$;
desde $i = 1$ hasta n
tope[i] =base[i];
tope[$n-1$] = $MAX_V P - 1$;
ponerBaraja: Colocar las cartas en cada pila, las inicializa y las barajea. ponerBaraja(Array Mesa, Array BarajaTotal, Array base, Array tope)
Entero $n = 13$
Entero numCartas
Caracter N, C, E, O, R, T
numCartas= $\times 4$
$n = n + 1$
Carta BarajaEspada[n];
inicializarBaraja(BarajaEspada,n, N, E, C);
barajearValor(BarajaEspada,n);
Carta BarajaOro[n];

inicializarBaraja(BarajaOro,n, R, O, C);
barajearValor(BarajaOro,n);
Carta BarajaTrebol[n];

inicializarBaraja(BarajaTrebol,n, N, T, C);
barajearValor(BarajaTrebol,n);
Carta BarajaCorazon[n];

inicializarBaraja(BarajaTrebol,n, N, T, C);
barajearValor(BarajaTrebol,n);

inicializarBaraja(BarajaCorazon,n, R, C, C);
barajearValor(BarajaCorazon,n);

numCartas=numCartas+1

Enteros i, j
desde $i = 0$ hasta 12
BarajaTotal[i] =BarajaEspada[j];
$j = j + 1$;
$j = 1$
desde $i = 13$ hasta 25
BarajaTotal[i] =BarajaOro[j];

$j = j + 1$
$j = 1$
desde $i = 26$ hasta 38
BarajaTotal[i] =BarajaTrebol[j];
$j = j + 1$
$j = 1$;
desde $i = 39$ hasta numCartas
BarajaTotal[i] =BarajaCorazon[j];
$j = j + 1$;

desde $i = 0$ hasta 9
BarajearTotal(BarajaTotal,numCartas);

$j = 2$;
Entero $k = 0$;
Desde $i = 0$ hasta 27
apilar(Mesa,tope[j],BarajaTotal[i]); $k = k + 1$
si $k = j - 1$
entonces $j = j + 1$
$k = 0$;
desde $i = 28$ hasta 51
Apilar(Mesa,base,tope,0,BarajaTotal[i]);

- inicializarBaraja: inicializa las barajas les da un valor, color, tipo y estado.

inicializarBaraja(Carta Carta[], int n, char color, char tipo, char estado)
desde $i = 0$ hasta $n - 1$
Carta[i].valor= i;
Carta[i].color=color;
Carta[i].tipo=tipo;
Carta[i].estado=estado;

- Baraja las cartas de una pila.

barajearValor(Carta Carta[],int n)
Entero i, j, aux, num
Desde $i = 0$ hasta $n - 1$
num=aleatorio$(1, 12)$;
aux=Carta[num].valor;

Carta[num].valor=Carta[i].valor;
Carta[i].valor=aux;

- barajearTotal: Baraja todas las cartas.

void barajearTotal(Carta carta[], int n)
Carta aux;
Entero i, num
Desde $i = 0$ hasta $n - 1$
num=aleatorio$(1, 52)$
aux=carta[num];
carta[num] =carta[i];
carta[i] =aux;

- apilar: coloca cartas sobre las pilas

apilar(Carta array Mesa, Entero tope, Carta dato)
tope= $tope + 1$
Mesa[$tope$] =dato;
Apilar(Carta Mesa[],Entero base[],Entero tope[],Entero i,Carta dato)
si pilaVacia(base[i],tope[i]))
si(dato.valor== 13)
entonces
apilar(Mesa,tope[i],dato)
si pilaLlena(base[i],tope[i]
entonces imprimir("error: no se puede apilar más")
sino
si tope[i] ==base[$i + 1$]
entonces
MoverPila(base,tope,i)
apilar(Mesa,&tope[i],dato);

- PilaVacia: Verifica si la pila está vacia y retorna un valor (verdadero o falso)

pilaVacia(Entero tope, Entero base)
return tope==base;

- PilaLlena: verifica si la pila esta llena y retorna un valor(verdadero o falso)

pilaLlena(Entero tope, Entero base)
si tope−base < 13
entoces
return falso;
sino
return verdadero;

- MoverPila: mueve las pilas cuando existe una colisión entre pilas.

MoverPila(Entero base[],Entero tope[],Entero i)
Entero j, k;
Entero izq= i,der= $i+1$;
encontro=falso;
mientras(encontro==false)
si tope$[izq-1]$ ==base$[izq]$&&tope$[der]$ ==base$[der+1]$
si $der < 14$
entonces
$der = der + 1$;
si $izq > 0$
entonces
$izq = izq - 1$;
sino
si tope$[izq-1]$! =base$[izq]$
$k = izq$;
sino
$k = der$;
encontro=true;
si $k = der$
desde $j = i + 1$ hasta der
tope$[j]$ =tope$[j]+1$;
base$[j]$ =base$[j]+1$;
sino
desde $j = i$ hasta 2
tope$[j]$ =tope$[j]-1$;
base$[j]$ =base$[j]-1$; actualizar: actualiza cuando se mueve una carta

actualizar(Carta Mesa[],int tope[])
Entero i;
desde $i = 2$ hasta 8
Mesa[tope$[i]$].estado='A'; imprimirJuego: Muestra el juego imprimirJuego(Carta Mesa[],int base[],int tope[])
imprimirPilaC(Mesa,base,tope,0);

```
imprimirPilaC(Mesa,base,tope,1);
imprimirPilaC(Mesa,base,tope,2);
imprimirPilaC(Mesa,base,tope,3);
imprimirPilaC(Mesa,base,tope,4);
imprimirPilaC(Mesa,base,tope,5);
imprimirPilaC(Mesa,base,tope,6);
imprimirPilaC(Mesa,base,tope,7);
imprimirPilaC(Mesa,base,tope,8);
imprimirPilaC(Mesa,base,tope,9);
imprimirPilaC(Mesa,base,tope,10);
imprimirPilaC(Mesa,base,tope,11);
imprimirPilaC(Mesa,base,tope,12);
```

imprimirPilaC: imprime las cartas de las pilas
imprimirPilaC(Carta Mesa[],int base[],int tope[],int k)

```
Entero i;
desde i =base[k] + 1 hasta tope[k]
imprimirCarta(Mesa[i]);
```

- imprimirCarta: imprime cartas

```
imprimirCarta(Carta k)
si k.estado==’A’
imprimir k.tipo
si k.estado==’C’
imprimir |##|
```

[1]

[1]hecho en LaTeX