**Logotipo, nombre de la empresa

Descripción generada automáticamente**

**UNIVERSIDAD TECNOLÓGICA NACIONAL**

**Facultad Regional Buenos Aires**

***Algoritmos y Estructuras de Datos***

**–2024–**

**Docente: Prof. Roxana Leituz**

**Trabajo Práctico**

***«ESCOBA DE 15»***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Localización: CAMPUS** | | | **Curso: X1071** | |
| **Integrantes del equipo:** | | | | |
| **Legajo** | **Apellido y Nombre** | | | |
| 215.449-3 | Mansilla Joseph Thomas | | | |
| 214.123-1 | Ignacio Scarfo | | | |
| 214.191-7 | Santiago Torres Franco | | | |
| 213.835-9 | Manuel Liniers | | | |
| **Entrega / Revisión** | **1** | **2** | | **3** |
| **Fecha de entrega** | **28/02/2024** | | | |
| **Firma del Docente** |  | | | |
| **Observaciones:** | | | | |

Contenido

[Pensamiento lógico del problema 3](#_Toc160006857)

[Manual de Usuario 5](#_Toc160006858)

[Pantallas de muestra 6](#_Toc160006859)

# Pensamiento lógico del problema

Primero nos preguntamos qué necesitábamos como base para resolver el problema. Por eso mismo incluimos las siguientes bibliotecas: iostream y cstream (ya usadas en la cursada) y también stdlib.h y ctime, estas dos últimas son necesarias para crear el número aleatorio y es lo único que investigamos de una biblioteca externa. Teniendo en cuenta esto, surge el *struct* Carta, para poder diferenciarlas por palo y número; un *struct* Nodo, para realizar la Mesa como una lista (ya que cuando se reparte son solo 4 cartas, pero ese número va variando mientras se juega, así que no se tiene un número fijo de cartas que pueden estar en la Mesa); y por último el *struct* Jugador, que van a ser quienes jueguen (como planteamos el código como si jugasen dos personas, por eso hay dos variables del tipo Jugador) y tienen un campo tipo Carta de dimensión 3, ya que son las que se reparten al inicio de cada mano, un puntaje para indicar cuándo gana, y una lista de las cartas que van levantando durante la partida.

Lo siguiente que hicimos fue inicializar todo para realizar el algoritmo dentro del main: Mesa=NULL; *int* ronda=0; uno.puntaje=0; dos.puntaje=0; uno.cartasLlevadas=NULL; dos.cartasLlevadas=NULL y el srand en NULL (para que no se repita siempre el mismo ganador). Seguido de eso, planteamos la condición de que el algoritmo funcione siempre y cuando ninguno de los jugadores tenga un puntaje mayor o igual a 15. Dentro de dicha condición, inicializamos la baraja (vector de tipo Carta de dimensión 40)

Una vez con las estructuras de datos que necesitábamos y el esqueleto del main planteado, teníamos que empezar a pensar en cómo se repartirían las cartas, cómo se manejaría el mazo o cómo se iban a guardar las cartas en la lista de cada jugador. Para eso, primero razonamos que tendríamos que desarrollar una función *void* repartir1, que recibe la baraja completa de cartas, ambos jugadores y la lista Mesa. Esta función lo que haría es primero sacar un número random de 0-39 y ese sería el índice a sacar de la baraja y lo que nos pareció más práctico es que cuando una carta ya fue utilizada ya sea en la baraja o en la mano del jugador su campo “num” sea igual a 0 esto serviría para no repetir las cartas y solo utilizarlas una vez. Y a la Mesa lo hace mediante un insertarPrimero. Este ciclo se puede realizar ya que usamos tres variables (i, j, k) para que no se repartan cartas de más.

Volviendo al main, después de repartir1, planteamos que mientras la ronda sea menor a 6 llame 6 veces a la función buscarPuntos (detallada más adelante). Esto los hicimos ya que la invoca una vez por carta de cada jugador, una carta del jugador 1, otra del jugador 2, la segunda del jugador 1, y así sucesivamente hasta terminar con las tres cartas de cada jugador. También planteamos que, si ronda acaba siendo distinta de 6, significa que todavía faltan cartas por jugar (ya que tenes las 10 cartas que se reparten en la ronda 0 y después 6 por ronda y 6x5=30 y 30+10=40), así que desarrollamos la función *void* repartir2, que la usaríamos para dicha situación en donde se repite el algoritmos de repartir 1 pero en ese caso solo repartiría a los jugadores.

También desarrollamos una función que simule la “búsqueda” de puntos, la función *int* buscarPuntos, donde buscaríamos todas las posibles combinaciones de cartas que se pueden realizar entre las del jugador y las de la Mesa donde se sume 15. Esta función solo se realiza mientras el jugador tenga cartas en su mano (es decir que no tenga ningún “num” igual a 0), y mientras haya cartas en la mesa (Mesa!=NULL). En esta función solo contemplamos hasta combinaciones de 5 cartas, es decir, una carta de la mano del jugador y cuatro del mazo pero se podrían contemplar hasta infinitas combinaciones, para eso con la función insertarPrimero añadimos las cartas combinadas a la lista de cartas llevadas de cada jugador, eliminamos las cartas de la Mesa para que no se vuelvan a utilizar, ponemos la carta de la mano del jugador en 0 para que no se vuelva a utilizar y ponemos una variable agarre en 1 que nos servirá luego, y si luego del agarre la Mesa queda en NULL (no hay más cartas) le sumamos un punto a ese jugador ya que interpretaría la escoba, y por último, si esa variable agarre está en 1 significa que se agarraron cartas entonces la función devuelve 0 y se cierra. En caso de no haber posible combinación con la mesa y ninguna de las cartas del jugador (recorridas con un for), directamente se hace que el jugador descarte una de sus cartas y la función devuelve -1.

Volviendo al main, en caso de quedar cartas en la Mesa cuando se finaliza la ronda, con la función recién mencionada se hace que estas vayan con el último jugador en levantar ya que si la última levantada del último jugador es distinta de -1 significa que levantó. Se liberan las cartas de la mesa con un pop y se llevan a la lista de cartas llevadas.

Y, por último, la función *void* contarPuntos, lo que hace es un pop a la lista de cartas de cada jugador y con diferentes variables cuenta para ver con cuántos puntos acabó cada uno la ronda teniendo en cuenta la cantidad de sietes, quién tiene el 7 de oro, quien tiene más oros y quien tiene más cartas, y por último se vuelve a 0 a la variable ronda asi, si ningún jugador supera los 15 puntos, se puede volver a repetir el algoritmos hasta que uno lo consiga.

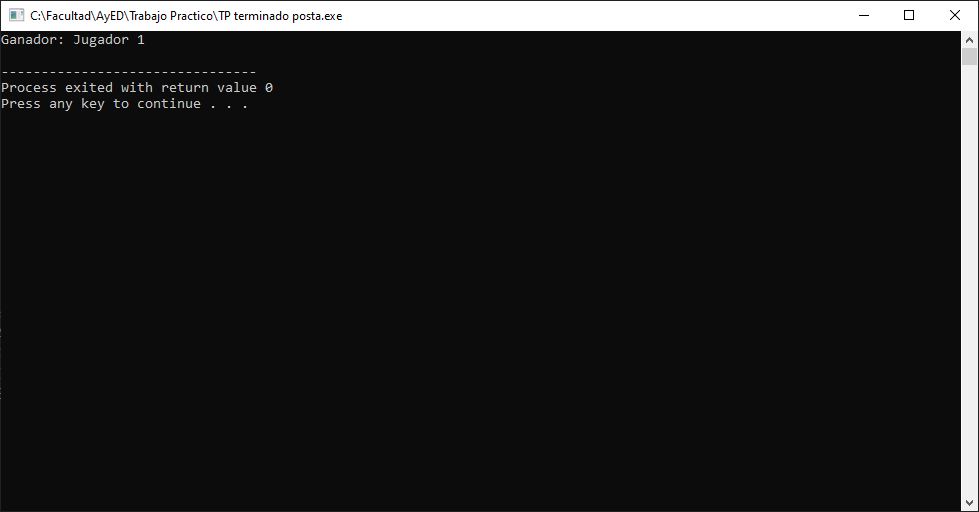
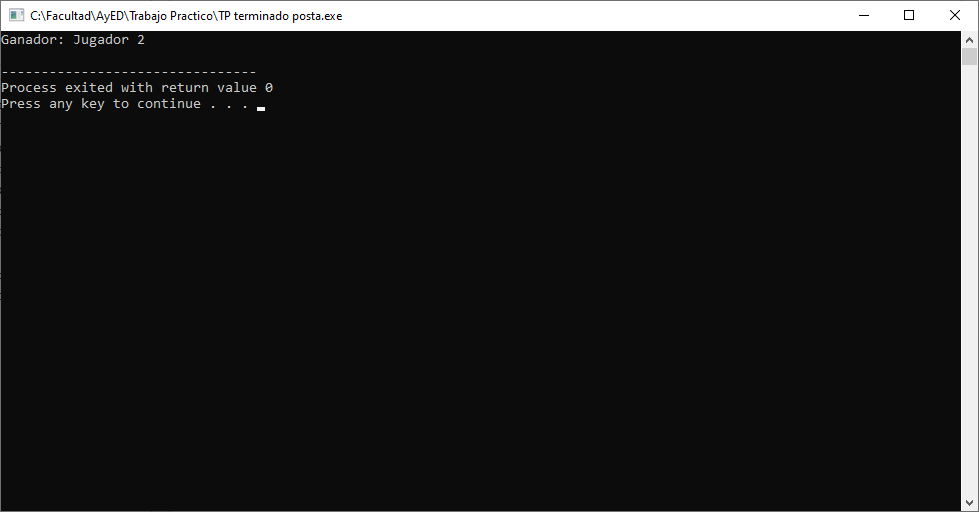
Finalmente, aparecerá quién es el ganador en pantalla según su puntaje.

# Manual de Usuario

La escoba de 15 es un juego de cartas donde se utiliza normalmente la baraja española. Los 8 y 9 dejan de tener validez para q la sota pase a valer 8, el caballo 9, y el rey 10. Se reparten 3 cartas a cada jugador y se ponen 4 cartas en la mesa, el objetivo es sumar 15 entre las cartas que tiene cada uno en su mano con las de la mesa. A medida que las rondas pasan, se va llevando un puntaje cumpliendo algunos requisitos, por ejemplo: el 7 de oro suma un punto, el que tenga la mayor cantidad de 7 en su mazo personal, quien tenga la mayor cantidad de cartas en dicho mazo, la mayor cantidad de oros, y las escobas, situaciones donde tu carta con las de la mesa suman 15 y al levantarlas no queda ninguna carta pendiente por jugar, entonces el siguiente jugador está obligado a tirar una carta para que continúe la ronda, en caso de tener más cartas.

Este algoritmo no necesita ninguna interacción del usuario, simula completamente una partida de escoba de 15. Desde el reparto hasta el puntaje final, todo está automatizado para que solo se necesite compilar el código.

# Pantallas de muestra

Uno de los problemas con el cual nos topamos antes de usar el “sran” es que el juego compilaría, sin embargo, siempre daría el mismo ganador. Al usar esa función pudimos resolver el problema.

Ahí podemos ver dos ejecuciones distintas del programa, ambas exitosas en donde en una el ganador es el jugador 1 y en la otra el 2