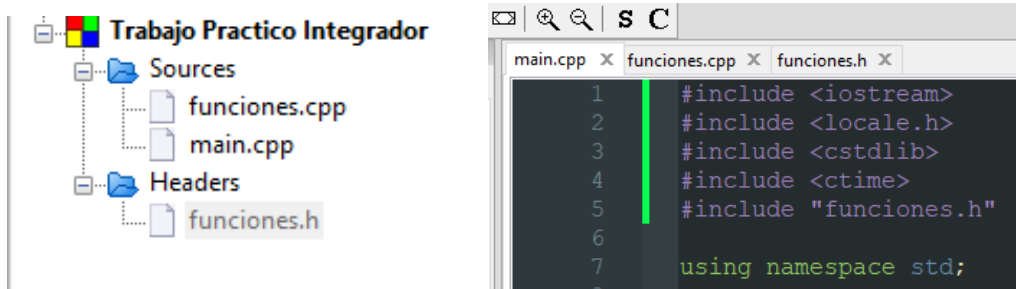


TRABAJO PRACTICO INTEGRADOR PROGRAMACIÓN 1

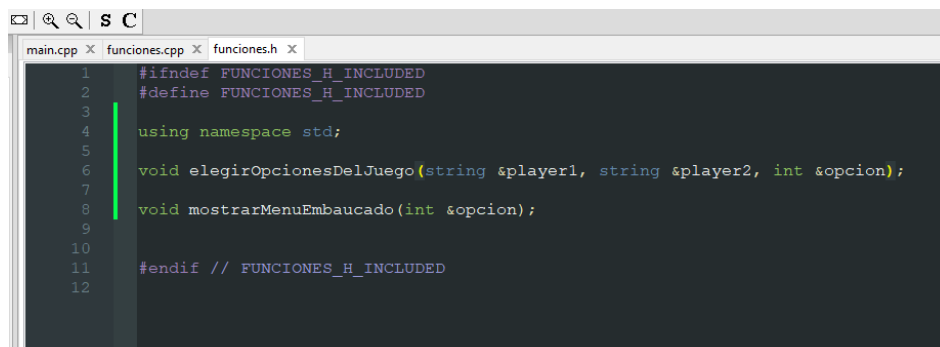
1- ARRANCAMOS CON LAS BIBLIOTECAS Y ESQUEMA

Para empezar, decimos usar el formato de librería .h para que sea más ordenado para el trabajo y lectura:

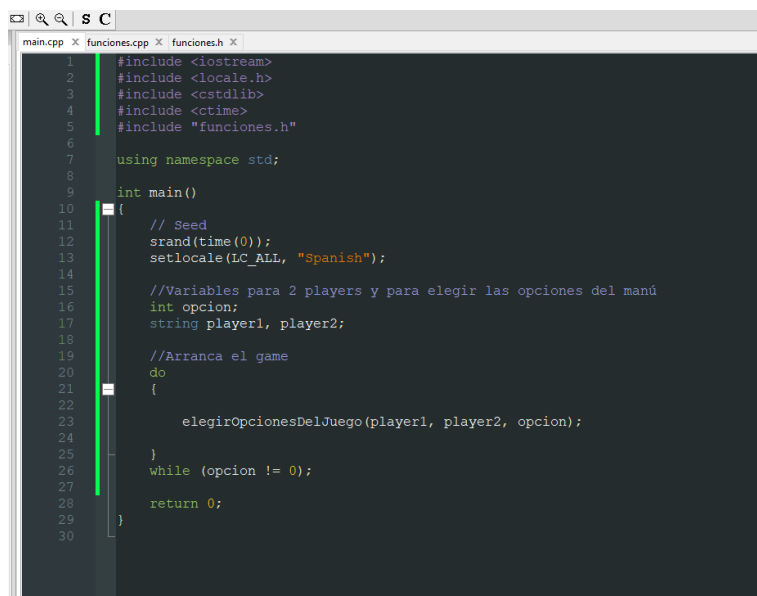


También decidimos agregar las librerías <locale.h> para que no tenga errores con caracteres especiales, <cstdlib> y <ctime> para crear la "seed" y poder usar "srand" para randomizar los naipes.

También ya agregamos la librería de "funciones.h".



A continuación, en "funciones.h" declaramos las funciones "elegirOpcionesDelJuego" y "mostrarMenuEmbaucado" las cuales van a recibir por referencia 3 variables para guardar más adelante los nombres de los jugadores y la opción que hace que se pueda navegar por el menú.



Seguidamente, decidimos utilizar la funcionalidad de un Do While para poder con el ingreso de o finalizar la ejecución total del programa o si no que mientras se ingrese un número predefinido, interactúe con un switch que comentamos en la siguiente función "elegirOpcionesDelJuego":

```
9  //Opciones de menú
10 void elegirOpcionesDelJuego(string &player1, string &player2, int &opcion)
11 {
12
13     mostrarMenuEmbaucado(opcion);
14
15     switch (opcion)
16     {
17     case 1:
18         ///Registrar Nombres
19         break;
20
21     case 2:
22         ///Estadisticas
23         break;
24
25     case 3:
26         ///Creditos
27         break;
28
29     case 0:
30         ///Fin del juego
31         break;
32
33     default:
34
35         break;
36     }
37 }
38
```

Dentro de la anterior función vamos a tener un Switch el cual tiene contemplado las 4 diferentes posibles opciones del menú principal (más adelante en el desarrollo contemplamos más opciones y excepciones), el cual también más avanzados en el desarrollo usamos para enviar a diferentes funciones las cuales realizaría su correspondiente acción.

```
void mostrarMenuEmbaucado(int &opcion)
{
    cout << "EMBAUCADO" << endl;
    cout << "-----" << endl;
    cout << "1 - JUGAR" << endl;
    cout << "2 - ESTADÍSTICAS" << endl;
    cout << "3 - CRÉDITOS" << endl;
    cout << "-----" << endl;
    cout << "0 - SALIR" << endl;
    cout << endl;

    cin >> opcion;
}
```

Dentro del Switch, la primera acción será mostrar el menú principal el cual está dentro de una función del tipo void, ya que no es necesario que retorne nada en particular y la única variable que maneja por ahora es por referencia la variable "opcion", que nos servirá para navegar por el menú de manera indefinida hasta que se ingrese o, el cual se usará para finalizar.

2- AGREGAMOS MÁS FUNCIONES Y DETALLES

Agregamos 2 funciones más `mostrarEstadisticas()` y `mostrarCreditos()` de tipo `void` por ahora ya que no necesitamos que retorne nada.

```
7
8 void mostrarMenuEmbaucado(int &opcion);
9
10 void mostrarEstadisticas();
11
12 void mostrarCreditos();
13
14
15 #endif // FUNCIONES_H_INCLUDED
16
```

Dentro de `mostrarMenuEmbaucado` en el `Switch` anteriormente creado podemos agregar en `case 2` y `case 3` que al seleccionar esa "opción" nos envíe a la información ahí albergada, la mismas van a ser completadas más adelante cuando contemos con los datos, en particular para las estadísticas.

También unos pequeños detalles para cuando se ingrese cero y termine la ejecución.

```
mostrarMenuEmbaucado(opcion);

switch (opcion)
{
case 1:
    ///Registrar Nombres
    break;

case 2:
    ///Estadisticas
    mostrarEstadisticas();
    break;

case 3:
    ///Creditos
    mostrarCreditos();
    break;

case 0:
    system("cls");
    ///Fin del juego
    cout << "Fin!";
    break;

default:
    cout << "Error pruebe de nuevo!" << endl;
    mostrarMenuEmbaucado(opcion);
    break;
}
```

Dentro de mostrarMenuEmbaucado() colocamos unos pequeños arreglos con "system("cls") para que limpie la pantalla en cada interacción y no se vea todo de manera continua.

```
void mostrarMenuEmbaucado(int &opcion)
{
    system("cls");
    cout << "EMBAUCADO" << endl;
    cout << "-----" << endl;
    cout << "1 - JUGAR" << endl;
    cout << "2 - ESTADÍSTICAS" << endl;
    cout << "3 - CRÉDITOS" << endl;
    cout << "-----" << endl;
    cout << "0 - SALIR" << endl;
    cout << endl;

    cin >> opcion;
}

void mostrarEstadisticas()
{
    system("cls");
    ///Estadisticas
    cout << "Estadisticas" << endl;
    system("pause");
}

void mostrarCreditos()
{
    system("cls");
    ///Creditos
    cout << "Creditos:" << endl;
    system("pause");
}
```

Y por ultimo agregamos algunos detalles que iremos desarrollando más adelante para mostrar la funciones mostrarEstadisticas() y mostrarCreditos() para que limpie la pantalla anterior y haga una pequeña pausa para que se pueda apreciar los datos.

3- REGISTRAMOS LOS NOMBRES

Creamos la función registrarNombresJugadores() en funciones.h y le mandamos por referencia player1 y player2 ya que acá vamos a registrar los nombres de ambos jugadores y los vamos a necesitar a lo largo de la partida y para eventualmente las estadísticas.

```
void mostrarCreditos();  
void registrarNombresJugadores(string &player1, string &player2);  
  
#endif // FUNCIONES_H_INCLUDED
```

En case 1 colocamos la función para cuando se quiera comenzar el juego primero se deba registrar los nombres de los jugadores y que estas ambas acciones separadas en 2 funciones independientes

```
switch (opcion)  
{  
case 1:  
    ///Registrar Nombres  
    registrarNombresJugadores(player1, player2);  
    break;  
  
case 2:
```

Empezamos a trabajar dentro de la función con una estructura básica con algunas excepciones

```
void registrarNombresJugadores(string &player1, string &player2)  
{  
  
}
```

Creamos una variable de tipo char que guarde la opción de corte para el do while y confirme o niegue los datos ingresados.

```
void registrarNombresJugadores(string &player1, string &player2)  
{  
    char confirmar;  
    do  
    {  
  
    } while (confirmar != 'S' && confirmar != 's');  
}
```

Y armamos un pequeño cuadro para el ingreso de ambos datos que se van a guardar en las variables de tipo string que están alojadas en el main principal ya que son datos que podríamos llegar a necesitar más adelante

```
char confirmar;

do
{
    cout << "EMBAUCADO" << endl;
    cout << "-----" << endl;

    cout << "Antes de comenzar deben registrar sus nombres: " << endl;
    cout << endl;

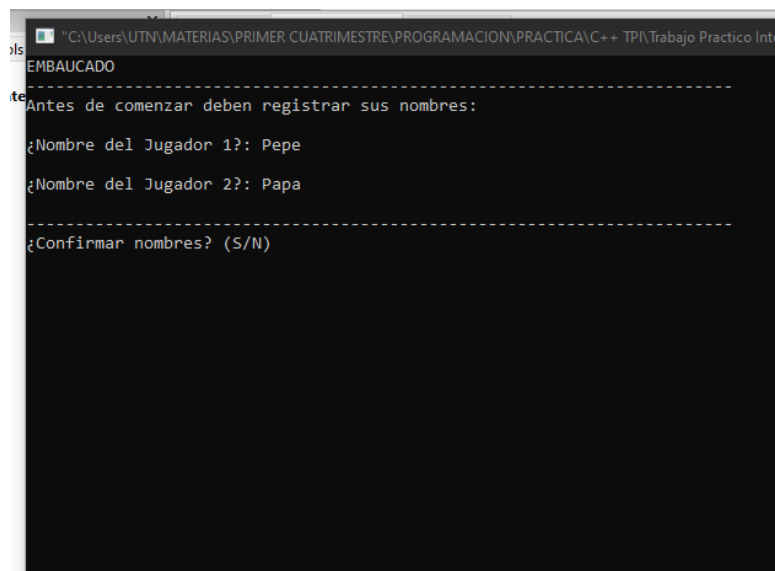
    cout << "¿Nombre del Jugador 1?: ";
    cin >> player1;
    cout << endl;

    cout << "¿Nombre del Jugador 2?: ";
    cin >> player2;
    cout << endl;
    cout << "-----" << endl;

    cout << "¿Confirmar nombres? (S/N) ";
    cin >> confirmar;
    cout << endl;

} while (confirmar != 'S' && confirmar != 's');
```

Una pequeña muestra de cómo va quedando



```
EMBAUCADO
-----
Antes de comenzar deben registrar sus nombres:

¿Nombre del Jugador 1?: Pepe
¿Nombre del Jugador 2?: Papa
-----
¿Confirmar nombres? (S/N)
```

¿Y si se ingresa alguna letra que no sea N o S? Bueno podemos hacer un pequeño While que siga iterando hasta que se ingresa una de las 4 opciones de corte o de reingreso de datos.

```
///Confirmamos que solo se ingresaen N o S

while(confirmar != 'S' && confirmar != 's' && confirmar != 'N' && confirmar != 'n')
{
    cout << "Error de tipeo, pruebe nuevamente." << endl;
    cout << endl;
    cout << "¿Confirmar nombres? (S/N): ";
    cin >> confirmar;
    cout << endl;
}
```

4- COMIENZA EL JUEGO

Bueno acá ya podemos empezar con las primeras incógnitas, siguiendo el esquema que nos propone la guía, creamos las funciones correspondientes en .h de tipo void por que por ahora no necesitamos que retorne nada

```
void registrarNombresJugadores(string &player1, string &player2);  
void comenzarJuego(string &player1, string &player2);  
  
#endif // FUNCIONES_H_INCLUDED
```

Y le enviamos cuando finaliza la función anterior en la que se ingresaban los datos, ambos nombres los cuales nos vana servir para los datos que se muestran en cada ronda del juego

```
1     }  
2     while (confirmar != 'S' && confirmar != 's');  
3  
4     comenzarJuego(player1, player2);  
5  
6 }  
7  
8 void comenzarJuego(string &player1, string &player2)  
9 {  
10  
11  
12  
13 }  
14
```

Creamos un pequeño ciclo For de solo 3 vueltas ya que solo son 3 rondas las que se juegan en total e ingresamos algunos datos para que se muestren por pantalla.

```
0  
1     for (int i = 0; i < 3; i++)  
2     {  
3         out << "EMBAUCADO" << endl;  
4         cout << "-----" << endl;  
5         cout << "RONDA #" << (i + 1) << endl;  
6         cout << player1 << " vs " << player2 << endl;  
7         cout << endl;  
8     }  
9  
10 }  
11  
12 void mostrarEstadisticas()  
13 {
```

Acá ya vamos a necesitar algunas variables extras para empezar a guardar los puntos de cada ronda, que más adelante deberías calcular en otro algoritmo, por ahora solo vamos maquetando de manera general

```
void comenzarJuego(string &player1, string &player2)
{
    int puntosPlayer1 = 0, puntosPlayer2 = 0;

    for (int i = 0; i < 3; i++)
    {
        out << "EMBAUCADO" << endl;
        cout << "-----" << endl;
        cout << "RONDA #" << (i + 1) << endl;
        cout << player1 << " vs " << player2 << endl;
        cout << endl;

        ///Mostrar Puntos player 1
        cout << player1 << " (" << puntosPlayer1 << " puntos)" << endl;

        ///Mostrar cartas de player 1

        ///Mostrar Puntos player 2
        cout << player2 << " (" << puntosPlayer2 << " puntos)" << endl;

        ///Mostrar cartas de player 2
    }
}
```

Seguimos avanzando y al finalizar la tirada de cartas, que también vamos a necesitar que se realice en otro algoritmo, mostramos la carta embaucadora que es crucial para el calculo de los puntos, tiene que ser de tipo string ya que solo tiene figura.

```
void comenzarJuego(string &player1, string &player2)
{
    int puntosPlayer1 = 0, puntosPlayer2 = 0;

    string embaucadora;

    for (int i = 0; i < 3; i++)
    {
        out << "EMBAUCADO" << endl;
        cout << "-----" << endl;
        cout << "RONDA #" << (i + 1) << endl;
        cout << player1 << " vs " << player2 << endl;
        cout << endl;

        ///Mostrar Puntos player 1
        cout << player1 << " (" << puntosPlayer1 << " puntos)" << endl;

        ///Mostrar cartas de player 1

        ///Mostrar Puntos player 2
        cout << player2 << " (" << puntosPlayer2 << " puntos)" << endl;

        ///Mostrar cartas de player 2

        ///Mostramos embaucadora
        cout << "Embaucadora: " << embaucadora << endl;
        cout << "-----" << endl;

        ///Y aca hay que calcular y mostrar los puntajes generados de cada player menos las embaucadas
    }
}
```


5- EMPEZAMOS A RANDOMIZAR

Creamos 2 funciones para randomizar los naipes para ambos jugadores en el archivo .h

```
void randomizarCartasParaPayers(string figura_ActualP1[], int TAM_P1, string naipe_ActualP1[], int TAM_N1, string figura_ActualP2[], int TAM_P2, string naipe_ActualP2[], int TAM_N2);
string randomizarCartaEmbaucadora();

#endif // FUNCIONES_H_INCLUDED
```

Y vamos a enviarle a la función randomizarCartasParaPayers() los vectores y variables que vamos a mostrar posteriormente en comenzarJuego()

```
///Mostrar Puntos player 1
cout << player1 << " (" << puntosPlayer1 << " puntos)" << endl;

///Randomizamos las cartas
randomizarCartasParaPayers(figura_ActualP1, TAM_N, naipe_ActualP1, TAM_N, figura_ActualP2, TAM_N, naipe_ActualP2, TAM_N);

///Mostrar cartas de player 1
```

Para eso vamos a tener que declarar dichos vectores y variables para por enviarlos, en este caso usaremos constantes para que sea mas ordenado y más fácil por si necesitamos eventualmente hacer cambios. Los inicializamos vacíos ya que vamos a modificarlos dentro de randomizarCartasParaPayers().

```
void comenzarJuego(string &player1, string &player2)
{
    int const TAM_N = 5, TAM_R = 3;
    int puntosPlayer1 = 0, puntosPlayer2 = 0;
    int puntosPorRondaP1[TAM_R] = {}, puntosPorRondaP2[TAM_R] = {};
    char opcion;

    string figura_ActualP1[TAM_N] = {}, naipe_ActualP1[TAM_N] = {};
    string figura_ActualP2[TAM_N] = {}, naipe_ActualP2[TAM_N] = {};
    string embaucadora;

    for (int i = 0; i < 3; i++)
    {
        out << "EMBAUCADO" << endl;
        cout << "-----"
```

Hacemos lo mismo con la función randomizarCartaEmbaucadora() pero en este caso haremos la función de tipo string, para que esa variable que retorne sea la figura de la carta embaucadora y podamos mostrarlo y eventualmente usarla para el cálculo de puntos.

```
///Devuelve la embaucadora randomizada:
embaucadora = randomizarCartaEmbaucadora();

///Mostramos embaucadora
cout << "Embaucadora: " << embaucadora << endl;
cout << "-----" << endl;

///Y aca hay que calcular y mostrar los puntajes generados de cada player menos las embaucadas
}
```

Empezamos randomizarCartaEmbaucadora() que más sencillo para probar el funcionamiento de "rand()".

```
void randomizarCartasParaPayers(string figura_ActualP1[], int TAM_F1, string naipes_ActualP1[], int TAM_N1, string figura_ActualP2[], int TAM_F2, string naipes_ActualP2[], int TAM_N2)
{
}

string randomizarCartaEmbaucadora()
{
    ///Randomizador de las figuras de para la embaucadora, devuelve uno al "azar"
}
```

Creamos las variables necesarias y un vector de 4, para que en cada uno de sus índices se le asigne una de las cuatro variables posibles con el nombre de cada figura.

```
string randomizarCartaEmbaucadora()
{
    ///Randomizador de las figuras de para la embaucadora, devuelve uno al "azar"
    int const TAM_F = 4;
    int RAN_F;

    string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};

}

void mostrarEstadisticas()
{
}
```

Usamos la funcionalidad de "rand()" y le colocamos el % de la constante del tamaño del vector el cual alberga las figuras, en este caso devolverá un numero entre 0 y 3 que este mismo será el que usemos para asignar al índice del vector figura y que devuelva en el retorno una de esas cuatro posibles figuras.

```
string randomizarCartaEmbaucadora()
{
    ///Randomizador de las figuras de para la embaucadora, devuelve uno al "azar"
    int const TAM_F = 4;
    int RAN_F;

    string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};

    RAN_F = (rand() % TAM_F);

    return figura[RAN_F];
}
```

6 – RANDOMIZAMOS LAS CARTAS PARA CADA PLAYER

Bueno acá empezamos con la parte más compleja, declaramos los vectores y sus tamaños con constantes y sus posibles figuras y numero de naipe que podría tocar.

```
}  
  
void randomizarCartasParaPayers(string figura_ActualP1[], int TAM_P1, string naipe_Act  
={  
    int const TAM_F = 4, TAM_N = 5;  
  
    string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};  
    string naipe[TAM_N] = {"10", "J", "Q", "K", "A"};  
-}  
  
string randomizarCartaEmbaucadora()  
={  
    ///Randomizador de las figuras de para la embaucadora, devuelve uno al "azar"  
    int const TAM_F = 4;  
    int RAN_F;
```

Creamos los vectores para, en este caso de que sean 2 jugadores, solo se barajen 10 cartas, pro eso creamos 2 vectores para las 10 combinaciones.

```
}  
  
void randomizarCartasParaPayers(string figura_ActualP1[], int TAM_P1, string naipe_Act  
={  
    int const TAM_F = 4, TAM_N = 5, TAM_T = 10;  
  
    string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};  
    string naipe[TAM_N] = {"10", "J", "Q", "K", "A"};  
  
    string figura_Actual[TAM_T];  
    string naipe_Actual[TAM_T];  
-}  
  
string randomizarCartaEmbaucadora()  
={
```

Creamos las variables para randomizar las figuras y los naipes, decidimos separar los naipes y las figuras para que sea más fácil hacer la comparación con la embaucadora y posteriormente hacer más sencillo el calculo de los puntajes por cada player. También un contador de cartas para saber cuantas cartas se han randomizado por que esta va a ser nuestra condición de corte.

```
}  
  
void randomizarCartasParaPayers(string figura_ActualP1[], int TAM_P1, string naipe_Act  
={  
    int const TAM_F = 4, TAM_N = 5, TAM_T = 10;  
    int RAN_F, RAN_N;  
    int contCartas = 0;  
  
    string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};  
    string naipe[TAM_N] = {"10", "J", "Q", "K", "A"};  
  
    string figura_Actual[TAM_T];  
    string naipe_Actual[TAM_T];  
-}  
  
string randomizarCartaEmbaucadora()
```

Inciso 1: Esta fue una de las opciones anteriores a las cuales habíamos llegado, en este caso usábamos varios ciclos For, para randomizar las 10 posibles variables y luego comparaba que no haya coincidencias, funcionaba bastante bien pero se daba una situación muy particular, que era que si justo la ultima carta consultada era igual a alguna de las anteriores la randomizaba pero como era la última había una posibilidad que randomizara en una que ya existía y como era la ultima del ciclo, podía salir igualada....

```
int palo;
int cont = 0;

palo = ((rand() % 4));

for (int i = 0; i < 10; i++)
{
    RAN_F = (rand() % 4);
    RAN_N = (rand() % 5);

    figura_Actual[i] = figura[RAN_F];
    naipe_Actual[i] = naipe[RAN_N];
}

for (int y = 0; y < 10; y++)
{
    cout << naipe_Actual[y] << " " << figura_Actual[y] << endl;
}

cout << endl;

for (int w = 0; w < 10; w++)
{
    for (int a = 0; a < 10; a++)
    {
        if (naipe_Actual[w] == naipe_Actual[a] && figura_Actual[w] == figura_Actual[a])
        {
            cont++;
            if (cont > 1)
            {
                RAN_F = (rand() % 4);
                RAN_N = (rand() % 5);

                naipe_Actual[a] = naipe[RAN_N];
                figura_Actual[a] = figura[RAN_F];
                cont = 1;
            }
        }
    }
}

cont = 0;
}
```

En esos caso se nos ocurrió entonces que no deberían ser ciclos For si no un Ciclo que solo corte cuando las 10 cartas hayan sido randomizadas y sean diferentes entre si:

```
int const TAM_F = 4, TAM_N = 5, TAM_T = 10;
int RAN_F, RAN_N;
int contCartas = 0;

string figura[TAM_F] = {"Picas", "Corazon", "Trebol", "Diamantes"};
string naipe[TAM_N] = {"10", "J", "Q", "K", "A"};

string figura_Actual[TAM_T];
string naipe_Actual[TAM_T];

/// Genero 10 cartas randoms todas distintas (al fin...)
while (contCartas < TAM_T)
{
    /// Numeros randoms
    RAN_F = rand() % TAM_F;
    RAN_N = rand() % TAM_N;

}

}

string randomizarCartaEmbaucadora()
{
}
```

En este le damos valor a ambas variables para que tenga un numero random que sirve para figura y naipes. Y creamos una bandera que nos diga si la carta sea o no repetida.

```
string figura[TAM_F] = { "10", "J", "Q", "K", "A" };
string naipes[TAM_N] = { "10", "J", "Q", "K", "A" };

string figura_Actual[TAM_T];
string naipes_Actual[TAM_T];

// Genero 10 cartas randoms todas distintas (al fin...)
while (contCartas < TAM_T)
{
    // Numeros randoms
    RAN_F = rand() % TAM_F;
    RAN_N = rand() % TAM_N;

    // Bandera repetidos
    bool noRepetida = true;

}

string randomizarCartaEmbaucadora()
{
```

Creamos un For con la cantidad de ciclos igual a la cantidad de cartas generadas, esto sirve para en cada iteración del ciclo while y donde se agregue una carta random se haga una verificación de coincidencias. De encontrarla la bandera se coloca en False y no guardara la carta generada y no contará esa carta, volverá a iterar el while y randomizara nuevamente la carta. De ser diferente a las generadas la bandera se mantendrá el True y la guardara en los vectores correspondiente en el índice del contador de cartas que va de 0 a 9.

```
string figura[TAM_F] = { "10", "J", "Q", "K", "A" };
string naipes[TAM_N] = { "10", "J", "Q", "K", "A" };

string figura_Actual[TAM_T];
string naipes_Actual[TAM_T];

// Genero 10 cartas randoms todas distintas (al fin...)
while (contCartas < TAM_T)
{
    // Numeros randoms
    RAN_F = rand() % TAM_F;
    RAN_N = rand() % TAM_N;

    // Bandera repetidos
    bool noRepetida = true;

    // Comparo en cada carta con las ya randomizadas anteriormente
    for (int i = 0; i < contCartas; i++)
    {
        if (figura_Actual[i] == figura[RAN_F] && naipes_Actual[i] == naipes[RAN_N])
        {
            noRepetida = false;
        }
    }

    // Guardo la carta con figura y naipes random
    if (noRepetida)
    {
        figura_Actual[contCartas] = figura[RAN_F];
        naipes_Actual[contCartas] = naipes[RAN_N];

        // Cuento cuantas cartas se van randomizando
        contCartas++;
    }
}
```

Y por ultimo dividimos en los vectores que recibimos desde la función comenzarJuego(), acá hago un inciso 2: había generado un For de 10, por que eran los 10 naipes que se randomizaba anteriormente y los guardaba en los vectores del índice [x], lo que no tuve en cuenta es que los vectores iban de 0 a 4 ambos, no de 5 a 9 en el caso del player2, esto me saco el sueño una noche hasta que me di cuenta....

Y por eso en player2 se debe usar el índice [x - 5] para que se guarden las 5 cartas restantes en los índices correspondiente.

```
    /// Divido en los 2 players
    for (int x = 0; x < 10; x++)
    {
        if (x < 5)
        {
            figura_ActualP1[x] = figura_Actual[x];
            naipe_ActualP1[x] = naipe_Actual[x];
        }
        else
        {
            figura_ActualP2[x - 5] = figura_Actual[x];
            naipe_ActualP2[x - 5] = naipe_Actual[x];
        }
    }
}
```

7 – PROBAMOS Y VERIFICAMOS LOS DATOS

Ahora vamos a realizar la lógica para que se muestre por pantalla las cartas que se randomizaron en randomizar en randomizarCartasParaPlayers() y la embaucadora que nos retorna la función randomizarCartaEmbaucadora() con dos ciclos For

```
///Mostrar Puntos player 1
cout << player1 << " (" << puntosPlayer1 << " puntos)" << endl;

///Randomizamos las cartas
randomizarCartasParaPayers(figura_ActualP1, TAM_N, naipe_ActualP1, TAM_N, figura_ActualP2, TAM_N, naipe_ActualP2, TAM_N);

///Mostrar cartas de player 1
for (int x = 0; x < 5; x++)
{
    cout << naipe_ActualP1[x] << " de " << figura_ActualP1[x] << endl;
}

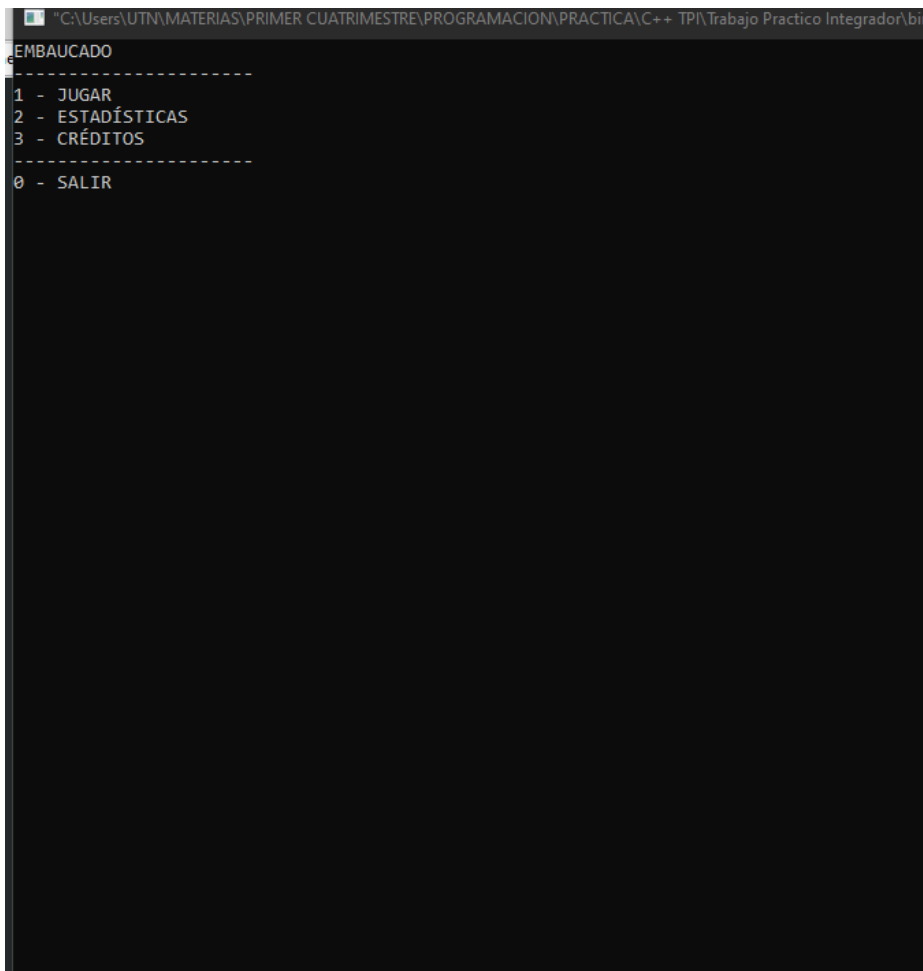
///Mostrar Puntos player 2
cout << player2 << " (" << puntosPlayer2 << " puntos)" << endl;

///Mostrar cartas de player 2
for (int y = 0; y < 5; y++)
{
    cout << naipe_ActualP2[y] << " de " << figura_ActualP2[y] << endl;
}

///Devuelve la embaucadora randomizada:
embaucadora = randomizarCartaEmbaucadora();

///Mostramos embaucadora
cout << "Embaucadora: " << embaucadora << endl;
cout << "-----" << endl;
```

Hacemos la prueba



```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrador\bi
EMBAUCADO
-----
1 - JUGAR
2 - ESTADÍSTICAS
3 - CRÉDITOS
-----
0 - SALIR
```

De paso comprobamos lo que se fue programando hasta ahora:

```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrador\bin
EMBAUCADO
-----
Antes de comenzar deben registrar sus nombres:

¿Nombre del Jugador 1?: Pepe
¿Nombre del Jugador 2?: Papa
-----
¿Confirmar nombres? (S/N) s
```

Y comprobamos que se muestran bien por pantalla los nombres de ambos jugadores, se dan las 3 rondas y que las 10 cartas sean diferentes. Además, que se randomiza bien la carta embaucadora:

```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrador\bin
EMBAUCADO
-----
RONDA #1
Pepe vs Papa

Pepe (0 puntos)
Q de Corazon
J de Trebol
Q de Picas
10 de Corazon
Q de Diamantes

Papa (0 puntos)
A de Corazon
J de Picas
K de Diamantes
10 de Picas
K de Corazon

Embaucadora: Trebol
-----
Presione una tecla para continuar . . .
```


8- VERIFICACIÓN PARA RE-RANDOMIZAR LA EMBAUCADORA

Acá acorde a las reglas de juego decidimos hacer 2 validaciones con condicionales para que en la segunda ronda pregunte al player1 si desea cambiar la embaucadora solo si tiene más de 20 puntos (los puntajes los calcularemos en el próximo capítulo), y de ser positiva la respuesta del player1 se le resta los 20 puntos y se re-randomiza la embaucadora y muestra la nueva embaucadora. De ser negativo la opción de re-randomizar pasa al player2 al cual se le aplica las mismas condiciones de puntos, etc.

```
if(i == 1)
{
    if (puntosPlayer1 >= 20)
    {
        cout << endl;
        cout << player1 << ", ¿Deseas modificar la carta embaucadora actual "<< embaucadora <<"? (S/N): ";
        cin >> opcion;
        cout << endl;
        if (opcion == 's' || opcion == 'S')
        {
            ///Resto el sacrificio de randomizar la embaucadora
            puntosPlayer1 -= 20;

            ///Nueva embaucadora
            embaucadora = randomizarCartaEmbaucadora();
            cout << "Nueva Embaucadora: " << embaucadora << endl;
            cout << "-----" << endl;
        }
        else if (opcion == 'n' || opcion == 'N')
        {
            if (puntosPlayer2 >= 20)
            {
                cout << player2 << ", ¿Deseas modificar la carta embaucadora actual "<< embaucadora <<"? (S/N): ";
                cin >> opcion;
                cout << endl;
                if (opcion == 's' || opcion == 'S')
                {
                    ///Resto el sacrificio de randomizar la embaucadora
                    puntosPlayer2 -= 20;

                    ///Nueva embaucadora
                    embaucadora = randomizarCartaEmbaucadora();
                    cout << "Nueva Embaucadora: " << embaucadora << endl;
                    cout << "-----" << endl;
                }
            }
        }
    }
}
```

Y en la ronda 3 se invierte el orden pasando a ser el player2 el primero al que se le pregunta y aplica las mismas reglas que en la ronda 2 pero invertido

```
else if(i == 2)
{
    if (puntosPlayer2 >= 20)
    {
        cout << endl;
        cout << player2 << ", ¿Deseas modificar la carta embaucadora actual "<< embaucadora <<"? (S/N): ";
        cin >> opcion;
        cout << endl;
        if (opcion == 's' || opcion == 'S')
        {
            ///Resto el sacrificio de randomizar la embaucadora
            puntosPlayer2 -= 20;

            ///Nueva embaucadora
            embaucadora = randomizarCartaEmbaucadora();
            cout << "Nueva Embaucadora: " << embaucadora << endl;
            cout << "-----" << endl;
        }
        else if (opcion == 'n' || opcion == 'N')
        {
            if (puntosPlayer1 >= 20)
            {
                cout << player1 << ", ¿Deseas modificar la carta embaucadora actual "<< embaucadora <<"? (S/N): ";
                cin >> opcion;
                cout << endl;
                if (opcion == 's' || opcion == 'S')
                {
                    ///Resto el sacrificio de randomizar la embaucadora
                    puntosPlayer1 -= 20;

                    ///Nueva embaucadora
                    embaucadora = randomizarCartaEmbaucadora();
                    cout << "Nueva Embaucadora: " << embaucadora << endl;
                    cout << "-----" << endl;
                }
            }
        }
    }
}
```

9 – CALCULAR PUNTAJES

Declaramos la función `calcularPuntajePlayers()` de tipo `INT` por que necesitamos que nos devuelva una variable de tipo entero para mostrar el total de los puntos:

```
string randomizarCartaEmbaucadora();  
int calcularPuntajesPlayers(string player, int &puntosPlayer, string naipes_ActualP[], int TAM_N, string figura_ActualP[], int TAM_A, string embaucadora);  
  
#endif // FUNCIONES_H_INCLUDED
```

Le pasamos las variables, vectores y la embaucadora para calcular los puntos de cada player para que además nos muestren por separado cada uno de los puntos de cada ronda.

```
}  
//t aca hay que calcular y mostrar los puntajes generados de cada player menos las embaucadas  
cout << endl;  
cout << "Puntajes:" << endl;  
cout << "-----" << endl;  
  
//Mostramos el retorno para cada jugador  
puntosPorRondaP1[i] = calcularPuntajesPlayers(player1, puntosPlayer1, naipes_ActualP1, TAM_N, figura_ActualP1, TAM_N, embaucadora);  
cout << endl;  
puntosPorRondaP2[i] = calcularPuntajesPlayers(player2, puntosPlayer2, naipes_ActualP2, TAM_N, figura_ActualP2, TAM_N, embaucadora);  
cout << "-----" << endl;  
  
cout << endl;  
system("pause");  
}  
}
```

Dentro de la función creamos una variable de tipo `INT` `contadorDePunto = 0`, el cual va a ir sumando los números en cada ronda para que lo retorne al final.

```
int calcularPuntajesPlayers(string player, int &puntosPlayer, string naipes_ActualP[], int TAM_N, string figura_ActualP[], int TAM_A, string embaucadora)  
{  
    // Contador de puntos para retornar al final, guardar en un vector mostrar en el resumen  
    int contadorDePuntos = 0;  
  
    cout << player << ": ";
```

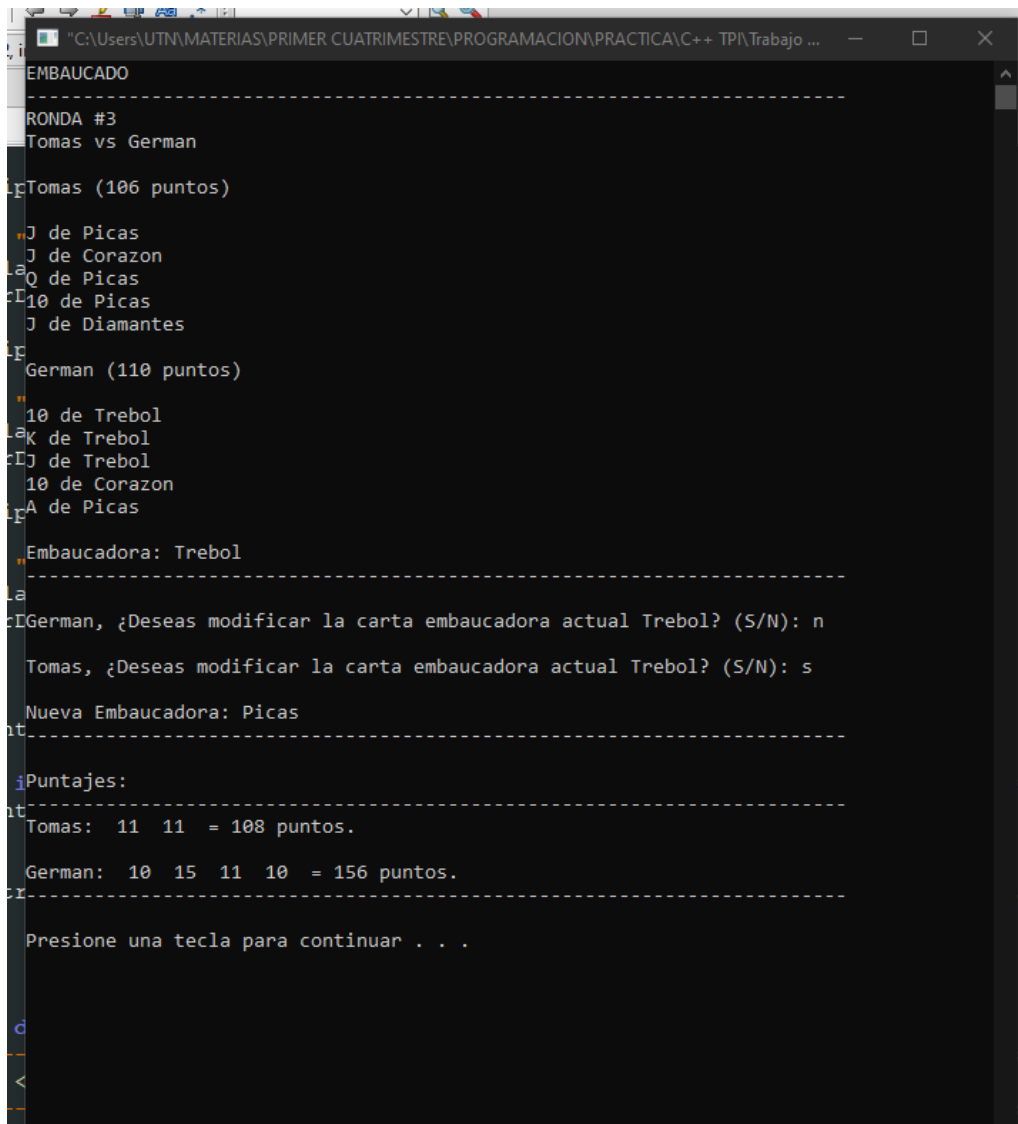
Luego en un ciclo `For` de 5 vueltas (por la cantidad de naipes) usaremos un condicional para buscar coincidencias con la embaucadora y dentro junto con el índice que nos brinda el ciclo, buscaremos coincidencias para que muestre y sume los correspondientes para el total.

```
    // Ciclo de 5 vueltas para comparar con la embaucadora y solo sumar las que no coinciden  
    for (int i = 0; i < 5; i++)  
    {  
        if (figura_ActualP[i] != embaucadora)  
        {  
            if (naipes_ActualP[i] == "10")  
            {  
                cout << " 10 ";  
                puntosPlayer += 10;  
                contadorDePuntos += 10;  
            }  
            else if (naipes_ActualP[i] == "J")  
            {  
                cout << " 11 ";  
                puntosPlayer += 11;  
                contadorDePuntos += 11;  
            }  
            else if (naipes_ActualP[i] == "Q")  
            {  
                cout << " 12 ";  
                puntosPlayer += 12;  
                contadorDePuntos += 12;  
            }  
            else if (naipes_ActualP[i] == "K")  
            {  
                cout << " 15 ";  
                puntosPlayer += 15;  
                contadorDePuntos += 15;  
            }  
            else if (naipes_ActualP[i] == "A")  
            {  
                cout << " 20 ";  
                puntosPlayer += 20;  
                contadorDePuntos += 20;  
            }  
        }  
    }  
    cout << " = " << puntosPlayer << " puntos." << endl;
```

Y por ultimo retornamos el valor de la suma de los naipes que no coincidan con la embaucadora y lo guarde dentro de la variable puntosPorRondaP1[i] (i por el número de ronda en el que se encuentre que además nos va a servir para el resumen más adelante) en la función comenzarJuego().

```
    }  
    }  
    cout << " = " << puntosPlayer << " puntos." << endl;  
  
    /// Retona el punta individual de cada player por cada una de las 3 rondas para que se muestre en el resumen  
    return contadorDePuntos;  
}
```

Acá podemos apreciar como se verá representados los puntos de manera individual para cada player, el uso del cambio de embaucadora y te muestra el total generado (acumulándolo en cada ronda) que además esto nos va a servir para el resumen y para el valor máximo para las estadísticas.



```
EMBAUCADOR  
-----  
RONDA #3  
Tomas vs German  
  
Tomas (106 puntos)  
J de Picas  
J de Corazon  
Q de Picas  
10 de Picas  
J de Diamantes  
  
German (110 puntos)  
10 de Trebol  
K de Trebol  
J de Trebol  
10 de Corazon  
A de Picas  
  
Embaucadora: Trebol  
-----  
German, ¿Deseas modificar la carta embaucadora actual Trebol? (S/N): n  
Tomas, ¿Deseas modificar la carta embaucadora actual Trebol? (S/N): s  
Nueva Embaucadora: Picas  
-----  
  
¡Puntajes:  
-----  
Tomas:  11  11  = 108 puntos.  
  
German:  10  15  11  10  = 156 puntos.  
-----  
  
Presione una tecla para continuar . . .
```

10 – RESUMEN DE PARTIDA

Hacemos la declaración de la función en h.

```
string randomizarCarterEmbaucadora();
int calcularPuntajesPlayers(string player, int &puntosPlayer, string naipes_ActualP[], int TAM_N, string figura_ActualP[], int TAM_A, string embaucadora);
void finalizarJuegoMostrarEstadisticas(string player1, string player2, int puntosPorRondaP1[], int TAM_R1, int puntosPorRondaP2[], int TAM_R, int puntosPlayer1, int puntosPlayer2);
#endif // FUNCIONES_H_INCLUDED
```

Le pasamos las variables, vectores necesarios para el resumen:

```
    cout << endl;
    system("pause");
}

finalizarJuegoMostrarEstadisticas(player1, player2, puntosPorRondaP1, TAM_R, puntosPorRondaP2, TAM_R, puntosPlayer1, puntosPlayer2);
```

Creamos la función:

```
void finalizarJuegoMostrarEstadisticas(string player1, string player2, int puntosPorRondaP1[], int TAM_R1, int puntosPorRondaP2[], int TAM_R, int puntosPlayer1, int puntosPlayer2)
{
}

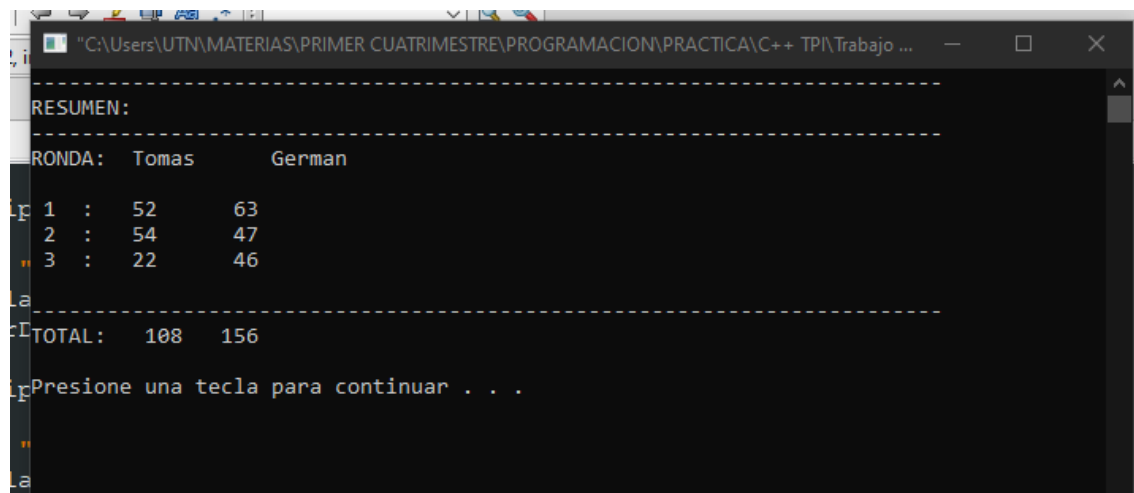
void mostrarEstadisticas()
{
}
```

Y creamos un breve cuadro donde se muestren los nombres de los player, los puntos individuales y sus totales al final.

```
void finalizarJuegoMostrarEstadisticas(string player1, string player2, int puntosPorRondaP1[], int TAM_R1, int puntosPorRondaP2[], int TAM_R, int puntosPlayer1, int puntosPlayer2)
{
    system("cls");
    ///Resumen en final de las 3 partidas, muestra los resultados de cada ronda y un total
    cout << "-----" << endl;
    cout << "RESUMEN: " << endl;
    cout << "-----" << endl;
    cout << "RONDA: " << player1 << " " << player2 << endl;
    cout << endl;
    cout << " 1 : " << puntosPorRondaP1[0] << " " << puntosPorRondaP2[0] << endl;
    cout << " 2 : " << puntosPorRondaP1[1] << " " << puntosPorRondaP2[1] << endl;
    cout << " 3 : " << puntosPorRondaP1[2] << " " << puntosPorRondaP2[2] << endl;
    cout << endl;
    cout << "-----" << endl;
    cout << "TOTAL: " << puntosPlayer1 << " " << puntosPlayer2 << endl;
    cout << endl;

    system("pause");
}
```

Y por último podemos ver cómo va quedando:



```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo ...
RESUMEN:
-----
RONDA:  Tomas      German
1 :    52      63
2 :    54      47
3 :    22      46
-----
TOTAL:   108    156
Presione una tecla para continuar . . .
```

11-ESTADISTICAS

Para las estadísticas vamos a necesitar declarar dos variables dentro del main, ya que necesitamos que esos datos se mantengan en memoria hasta que finalice el juego y que muestre cual fue el jugar con el puntaje máximo.

```
using namespace std;

int main()
{
    // Seed
    srand(time(0));
    setlocale(LC_ALL, "Spanish");

    //Variables para 2 players y para elegir las opciones del menú
    int opcion, puntajeMaximo = 0;
    string player1, player2, nombrePlayerPuntajeMaximo;

    //Arranca el game
    do
    {
        elegirOpcionesDelJuego(player1, player2, opcion, puntajeMaximo,
```

Vamos a necesitar mandar por referencias esas variables de tipo int y string en varias funciones:

```
using namespace std;

///Opciones de menú
void elegirOpcionesDelJuego(string &player1, string &player2, int &opcion, int &puntajeMaximo, string &nombrePlayerPuntajeMaximo)
{
    mostrarMenuEmbaucado(opcion);

    switch (opcion)
    {
        case 1:
            ///Registrar Nombres
            registrarNombresJugadores(player1, player2, puntajeMaximo, nombrePlayerPuntajeMaximo);
            break;

        case 2:
            ///Estadísticas
            mostrarEstadísticas(puntajeMaximo, nombrePlayerPuntajeMaximo);
            break;
    }
}
```

```
using namespace std;

void elegirOpcionesDelJuego(string &player1, string &player2, int &opcion, int &puntajeMaximo, string &nombrePlayerPuntajeMaximo);

void mostrarMenuEmbaucado(int &opcion);

void mostrarEstadísticas(int puntajeMaximo, string nombrePlayerPuntajeMaximo);

void mostrarCreditos();

void registrarNombresJugadores(string &player1, string &player2, int &puntajeMaximo, string &nombrePlayerPuntajeMaximo);

void comenzarJuego(string &player1, string &player2, int &puntajeMaximo, string &nombrePlayerPuntajeMaximo);
```

Hasta llegar a mostrarEstadisticas() donde vamos a tener ambos resultados finales de cada partida, uno por cada jugador, acá con un par de condicionales vamos a poder saber quien es el ganador y comparar si el puntaje generado es mayor al que esta alojado en dicha variable puntajeMaximo. Guardamos el puntaje y el nombre del player.

```
cout << endl;
cout << "-----" << endl;
cout << "TOTAL:  " << puntosPlayer1 << "    " << puntosPlayer2 << endl;
cout << endl;

if( puntosPlayer1 > puntosPlayer2)
{
    if(puntosPlayer1 > puntajeMaximo)
    {
        puntajeMaximo = puntosPlayer1;
        nombrePlayerPuntajeMaximo = player1;
    }
}
else
{
    if(puntosPlayer2 > puntajeMaximo)
    {
        puntajeMaximo = puntosPlayer2;
        nombrePlayerPuntajeMaximo = player2;
    }
}

system("pause");
```

Y finalmente podemos mostrarlo en la opción del menú principal de "Estadísticas", en la función mostrarEstadisticas():

```
system("pause");
}

void mostrarEstadisticas(int puntajeMaximo, string nombrePlayerPuntajeMaximo)
{
    system("cls");
    ///Estadisticas
    cout << "EMBAUCADO - ESTADISTICAS" << endl;
    cout << "-----" << endl;
    cout << "JUGADOR : " << nombrePlayerPuntajeMaximo << endl;
    cout << endl;
    cout << "PUNTOS : " << puntajeMaximo << endl;
    cout << "-----" << endl;
    system("pause");
}

void mostrarCreditos()
{
    system("cls");
    ///Creditos
    cout << "Creditos:" << endl;
    system("pause");
}
```

Un poco de su funcionamiento:

```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrado
EMBAUCADO - ESTADISTICAS
}
JUGADOR : Papa
PUNTOS : 165
Presione una tecla para continuar . . .
```

Y si durante la sesión de juego, alguien supera el puntaje máximo se actualiza:

```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrado
RESUMEN:
RONDA: Lele      Lala
1 : 50      45
2 : 78      45
3 : 51      66
TOTAL: 179 136
Presione una tecla para continuar . . .
```

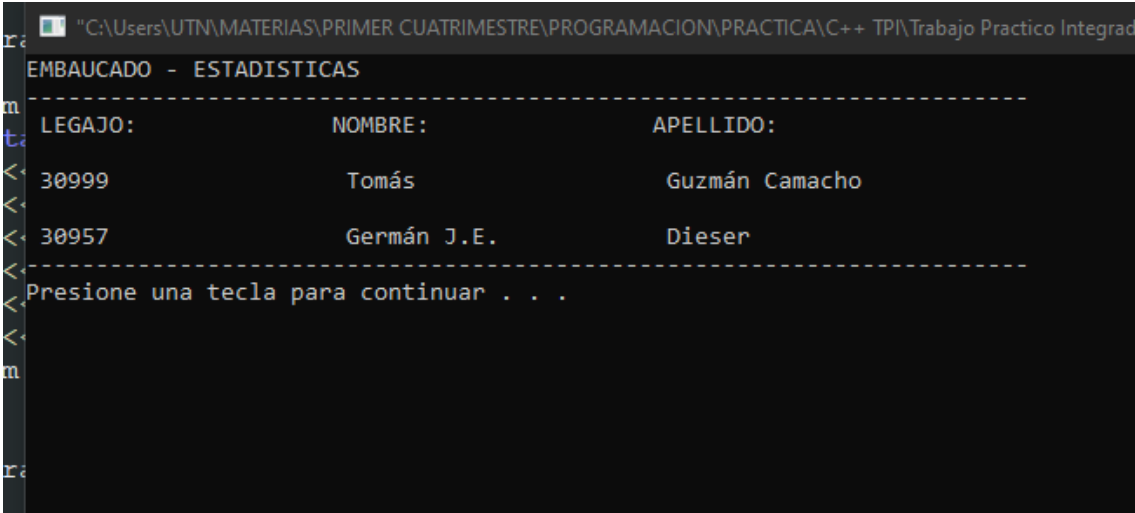
```
"C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrado
EMBAUCADO - ESTADISTICAS
}
JUGADOR : Lele
PUNTOS : 179
Presione una tecla para continuar . . .
```

12-CREDITOS

Y por último, nos quedaría agregar el apartado de créditos donde se mostrarán los datos de los participantes del grupo:

```
void mostrarCreditos()
{
    system("cls");
    ///Creditos
    cout << "EMBAUCADO - ESTADISTICAS" << endl;
    cout << "-----" << endl;
    cout << "LEGAJO:          NOMBRE:          APELLIDO: " << endl;
    cout << endl;
    cout << " 30999          Tomás          Guzmán Camacho " << endl;
    cout << endl;
    cout << " 30957          Germán J.E.      Dieser " << endl;
    cout << "-----" << endl;
    system("pause");
}
```

Que quedaría de la siguiente manera:



The screenshot shows the output of the C++ program in a terminal window. The title bar indicates the file path: "C:\Users\UTN\MATERIAS\PRIMER CUATRIMESTRE\PROGRAMACION\PRACTICA\C++ TPI\Trabajo Practico Integrado". The output displays the program's title, a separator line, a table of credits, another separator line, and a prompt to press a key to continue.

LEGAJO:	NOMBRE:	APELLIDO:
30999	Tomás	Guzmán Camacho
30957	Germán J.E.	Dieser

Presione una tecla para continuar . . .

Estos sería una aproximación al programa que vinimos desarrollando, la lógica principal que usamos pero con algunos agregados estético, todo sin agregar librerías que no se hayan presentado en clase hasta ahora, la utilización de la mayoría de las herramientas y estrategias presentada a lo largo de la cursada.