

INFORME DEL TALLER 1 – I.P.P.O.

Estudiante: Santiago Hernández Arias – Código 1631281

Fecha de entrega: 09 / 10 / 2016

Punto 1. Letrero de bienvenida

```
#include "letreroBienvenida.h"

int main() {

    generarLetrero();

}

void generarLetrero() {

    char letrero[37] = "          SACARÉ IPOO EN 5.0 :)          ";
    int posn = 0;

    cout << "*****\n";
    for(int a=0; a<29; a++){
        cout << "*";
        for(int b=0; b<8; b++){
            cout << letrero[posn];
            posn += 1;
        }
        cout << "*";
        cout << endl;
        posn = a+1;
    }
    cout << "*****";

}
```

Para generar el letrero de bienvenida, se declara una constante de tipo de char. Para producir el resultado pedido, es necesario realizar 29 impresiones, cada una de las cuales está constituida por otras 8. Esto se hace por medio de dos iterativos for. La constante **posn** guarda la posición del letrero que se imprime. En cada impresión horizontal, se recorre 8 veces el letrero, pero la posición inicial cambia para cada nueva impresión en sentido vertical, por ello se resetea **posn** y se le suma 1 por cada iteración horizontal concluida. Ejecución:

```

*****
*      S*
*      SA*
*      SAC*
*      SACA*
*      SACAR*
*      SACARÉ*
*      SACARÉ *
*SACARÉ I*
*ACARÉ IP*
*CARÉ IPO*
*ARÉ IPOO*
*RÉ IPOO *
*É IPOO E*
* IPOO EN*
*IPOO EN *
*POO EN 5*
*OO EN 5.*
*O EN 5.0*
* EN 5.0 *
*EN 5.0 :*
*N 5.0 :>*
* 5.0 :> *
*5.0 :> *
*.0 :> *
*0 :> *
* :> *
*:> *
*> *
* *
*****

```

Punto 2. Traductor

```

#include "traductor.h"

int main(){

    string diccionario[][2]={{"ANCIENT", "Antiguo"}, {"BRANCH", "Rama"},
    string clave;

    getClave(clave);
    translate(clave, diccionario);

}

```

El diccionario es un arreglo bidimensional de strings. El primer elemento de cada entrada es la palabra en inglés, o clave, y el segundo es su significado en español. La función **getClave** captura la entrada del usuario y le asigna este valor a **clave**. Sin embargo, el valor asignado es el ingresado por el usuario pero tras haber convertido en mayúsculas todas las letras, pues así se encuentran en el diccionario. De esta manera se pueden ignorar las discrepancias en caso de combinar mayúsculas y minúsculas. La función **translate** utiliza la clave que se generó para compararla con las entradas del diccionario. Ejecución:

```

Por favor ingrese la palabra que desea buscar en el diccionario:
Window
La palabra no existe en el diccionario

```

La palabra ventana no se encuentra en el diccionario, y el programa lo sabe cuando ha terminado de recorrer las entradas del mismo sin encontrar coincidencia.

```
Por favor ingrese la palabra que desea buscar en el diccionario:
sWArM
Traduce: Enjambre.
```

```
Por favor ingrese la palabra que desea buscar en el diccionario:
swarm
Traduce: Enjambre.
```

Arriba se observa como el programa ignora mayúsculas y minúsculas cuando intenta vincular una entrada del usuario con las claves del diccionario. Al encontrar coincidencia, la función **translate** devuelve el segundo elemento de la entrada del diccionario, que es el que corresponde a la palabra en español.

Punto 3. Inventario

```
#include "inventario.h"

int main(){

    vector<string> nombres;
    vector<int> cantidades;

    int selector = 0;

    do{
        showmenu();
        cin >> selector;
        gestion(selector, nombres, cantidades);
    } while(selector != 4);

    return 0;
}
```

El inventario se compone de dos vectores, uno que guarda los nombres y otro las cantidades. El **selector** es la variable entera que la función **gestión** utiliza para seleccionar los casos de **switch**. El programa concluye si **selector** toma el valor de **4**, pues esta opción es la de salir. Ejecución:

```
*****
*BIENVENIDO A MIINVENTARIO*
*****
*Seleccione una opcion  *
*1. Ingresar un elemento *
*2. Eliminar un elemento *
*3. Mostrar elementos   *
*4. Salir                *
*****
Ingrese su opcion:

Ingrese su opcion: 1
*****
*Ingrese su elemento    *
*****
Ingrese el nombre:
Bebidas
Ingrese la cantidad:
350
```

Esta es la impresión del menú principal. Se muestra también la opción de ingresar un elemento en el inventario, en este caso Bebidas. También se ingresará “Frituras” con una cantidad de 512. La opción 3 muestra los elementos del inventario. Con la opción 2 se procede a eliminar elementos. Si el elemento no se encuentra en el inventario, se le avisa al usuario.

```

Ingrese su opcion: 3
*****
*Elementos almacenados *
*****
* Nombre      * Cantidad *
*****
* Bebidas     * 350      *
* Frituras    * 512      *
*****
Ingrese su opcion: 2
*****
*Eliminar un elemento *
*****
Ingrese el nombre:
Helados
El elemento no esta

```

Si el elemento sí está, se elimina y se informa de la eliminación exitosa. Ahora, el elemento Bebidas habrá sido eliminado del inventario. Como las opciones válidas son de los enteros 1, 2, 3 y 4, otras entradas serán notificadas como inválidas:

```

Ingrese su opcion: 2
*****
*Eliminar un elemento *
*****
Ingrese el nombre:
Bebidas
Eliminacion exitosa
*****
Ingrese su opcion: 5
Opcion no valida
*****

```

Punto 4. Generar pares

```

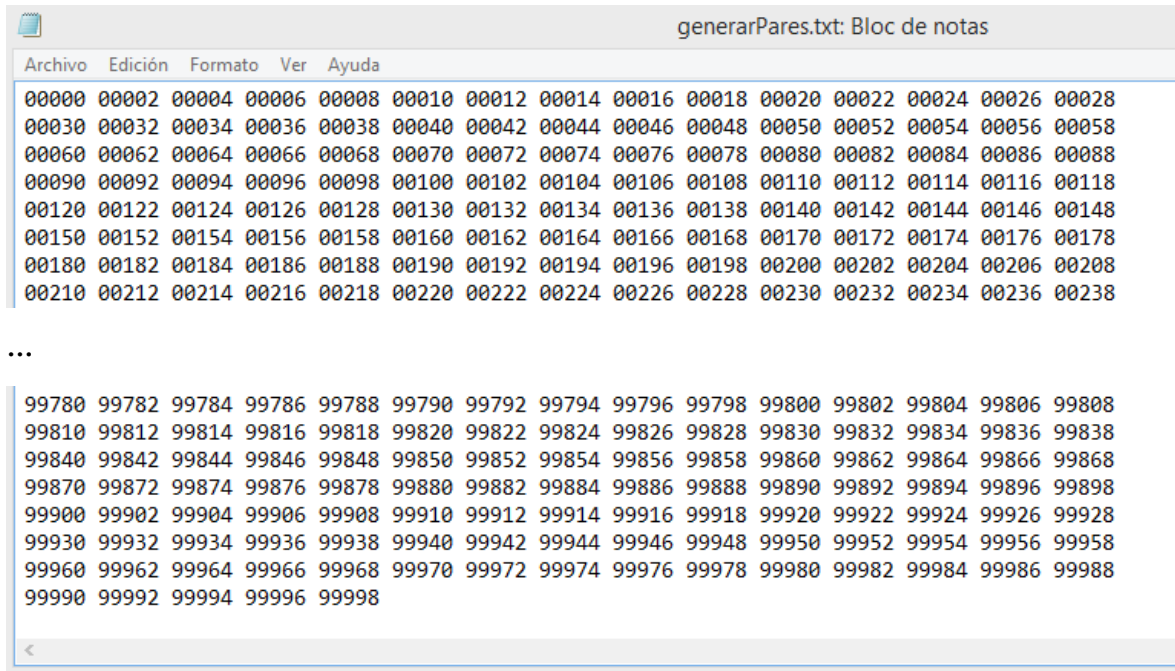
void generarArchivo(){
    ofstream output("generarPares.txt");

    for(int a=0; a<99999;){
        for(int b=0; b<15; b++){
            if(a<99999){
                if(a<=9){
                    output << "0000" << a << " ";
                }
                else{
                    if(a<=99){
                        output << "000" << a << " ";
                    }
                    else{
                        if(a<=999){
                            output << "00" << a << " ";
                        }
                        else{
                            if(a<=9999){
                                output << "0" << a << " ";
                            }
                            else{
                                output << a << " ";
                            }
                        }
                    }
                }
            }
            a += 2;
        }
    }
    if(a<99999){
        output << "\n";
    }

    output.close();
}

```

La función **generarArchivo** hace el trabajo. Primero, crea el archivo **generarPares.txt**, que es donde se realizará la impresión. Ésta se realiza por medio de dos iteradores for, siendo el segundo el que imprime las filas de 15 elementos cada una. Cada impresión aumenta en 2 el valor de **a**, pues solo se imprimen números pares. Para lograr que se impriman siempre 5 dígitos, el iterador revisa el valor de **a** e imprime 0 (ceros) para completar según sea necesario. Por ejemplo, si **a** está entre 100 y 999, harán falta dos ceros para llegar a los 5 dígitos (00###). La comprobación se realiza por medio de condicionales anidados. La operación concluye cuando se han impreso los números hasta 99998. Archivo generado:



```

generarPares.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
00000 00002 00004 00006 00008 00010 00012 00014 00016 00018 00020 00022 00024 00026 00028
00030 00032 00034 00036 00038 00040 00042 00044 00046 00048 00050 00052 00054 00056 00058
00060 00062 00064 00066 00068 00070 00072 00074 00076 00078 00080 00082 00084 00086 00088
00090 00092 00094 00096 00098 00100 00102 00104 00106 00108 00110 00112 00114 00116 00118
00120 00122 00124 00126 00128 00130 00132 00134 00136 00138 00140 00142 00144 00146 00148
00150 00152 00154 00156 00158 00160 00162 00164 00166 00168 00170 00172 00174 00176 00178
00180 00182 00184 00186 00188 00190 00192 00194 00196 00198 00200 00202 00204 00206 00208
00210 00212 00214 00216 00218 00220 00222 00224 00226 00228 00230 00232 00234 00236 00238
...
99780 99782 99784 99786 99788 99790 99792 99794 99796 99798 99800 99802 99804 99806 99808
99810 99812 99814 99816 99818 99820 99822 99824 99826 99828 99830 99832 99834 99836 99838
99840 99842 99844 99846 99848 99850 99852 99854 99856 99858 99860 99862 99864 99866 99868
99870 99872 99874 99876 99878 99880 99882 99884 99886 99888 99890 99892 99894 99896 99898
99900 99902 99904 99906 99908 99910 99912 99914 99916 99918 99920 99922 99924 99926 99928
99930 99932 99934 99936 99938 99940 99942 99944 99946 99948 99950 99952 99954 99956 99958
99960 99962 99964 99966 99968 99970 99972 99974 99976 99978 99980 99982 99984 99986 99988
99990 99992 99994 99996 99998

```

Punto 5. Triqui

```

#include "triqui.h"

int main() {

    char ** matrix;
    int turno = 1;
    int row, column;

    initializeMatrix(matrix);

    do{

        if(turno == 10){
            printMatrix(matrix);
            cout << "Empate\n";
            return 0;
        }

        printMatrix(matrix);
    }
}

```

```

// Solicitar fila y columna de la jugada
askForInput(turno, row, column);
// Validar rango de argumentos
while(!isInputValid(row, column)){
    cout << "Fila o columna inválida (Rango [1, 3]). Intente de nuevo.\n";
    askForInput(turno, row, column);
}
// Validar disponibilidad de entrada
while(!isInputAvailable(row, column, matrix)){
    cout << "La entrada ya fue marcada. Intente de nuevo.\n";
    askForInput(turno, row, column);
}
// Cambiar la matriz con argumentos válidos
changeMatrix(turno, row, column, matrix);

turno++;

} while(!triqui(matrix));

printMatrix(matrix);
if(turno%2==0){
    cout << "Gana Jugador 1\n";
}
else{
    cout << "Gana Jugador 2\n";
}
}

```

La matriz del juego inicia como un puntero de punteros de tipo char. El entero **turno** lleva cuenta de cuántos turnos han transcurrido, **row** y **column** guardan las entradas de fila y columna del usuario. Al principio, se inicializa la matriz que contiene las entradas del juego. A continuación se accede al bucle principal del juego, que no se detiene hasta que la condición de que exista “**triqui**” sea cumplida. En caso de que **turno** llegue al valor de 10, se declara empate pues se habrá llenado la matriz del juego sin un ganador.

```

*****
*Estado del juego          *
*****
| X | X | O |
| O | O | X |
| X | O | X |
*****
Empate

```

La función **askForInput** guarda las entradas del usuario, reconociendo a quién pertenece el turno por el valor de esta variable (impares jugador 1, pares jugador 2).

```

Jugador 1 (X)
Ingrese fila
1
Ingrese columna
1
*****
*Estado del juego          *
*****
| X | - | - |
| - | - | - |
| - | - | - |
*****

```

La función **isInputValid** repetirá el proceso de adquisición de valores por parte del usuario en caso de que se intente ingresar una columna o fila fuera del rango de la matriz de juego, es decir, valores diferentes a 1, 2 y 3.

```
Jugador 2 (O)
Ingrese fila
7
Ingrese columna
2
Fila o columna inválida (Rango [1, 3]). Intente de nuevo.
Jugador 2 (O)
Ingrese fila
```

La función **isInputAvailable** revalida las entradas checando que la posición deseada se encuentre disponible. Para esto, se fija en el valor de esta posición en la matriz, reconociendo que la entrada está disponible si su valor corresponde al carácter '-'.

```
*****
*Estado del juego          *
*****
| X | - | - |
| - | 0 | - |
| - | - | - |
*****
Jugador 1 (X)
Ingrese fila
2
Ingrese columna
2
La entrada ya fue marcada. Intente de nuevo.
Jugador 1 (X)
Ingrese fila
```

Si la entrada es válida, ambos ciclos permiten continuar y la función **changeMatrix** cambiara la entrada de la matriz de '-' a 'X' (jugador 1) o 'O'.

```
Ingrese fila
1
Ingrese columna
3
*****
*Estado del juego          *
*****
| X | - | X |
| - | 0 | - |
| - | - | - |
*****
```

El ciclo principal se rompe cuando se llega al **triqui**, es decir, cuando esta función reconoce que se ha alcanzado alguna de las condiciones de victoria (que toda una fila, columna o diagonal posea el mismo valor, sea 'X' o 'O'). Se conoce al ganador por el valor de **turno**, pues con este se sabe quién efectuó la jugada que culminó en victoria.

***** *Estado del juego ***** X X X - 0 - 0 - - ***** Gana Jugador 1	***** *Estado del juego ***** X - 0 X 0 - 0 - X ***** Gana Jugador 2	***** *Estado del juego ***** X 0 0 X X 0 - X 0 ***** Gana Jugador 2
---	---	---

Punto 6. Matrices (cuadrados mágicos)

```
int main() {

    int ** matrix;
    int matrixsize;
    vector<int> numbersInMatrix;
    int magicconstant=0;

    getmatrix(matrixsize, matrix);
    getvector(matrixsize, matrix, numbersInMatrix);
    findMagicConstant(matrixsize, matrix, magicconstant);

    cout << "La matriz ingresada:\n";
    if(isMatrixMagic(matrixsize, numbersInMatrix, matrix, magicconstant)){

        printMatrix(matrixsize, matrix);
        cout << "\nSí es un cuadrado mágico";

    }
    else{

        printMatrix(matrixsize, matrix);
        cout << "\nNo es un cuadrado mágico";

    }

}
```

La matriz sobre la cual se realizarán verificaciones inicia como un puntero de punteros de tipo int. **numbersInMatrix** es un vector cuyos elementos serán los de la matriz, ordenados ascendentemente, para efectos que se explicarán adelante.

```
void getvector(int matrixsize, int ** &matrix, vector<int> &numbersInMatrix){

    // Añade los elementos de la matriz al vector
    for(int a=0; a<matrixsize; a++){
        for(int b=0; b<matrixsize; b++){
            numbersInMatrix.push_back(matrix[a][b]);
        }
    }

    // Organiza los elementos del vector de manera ascendente
    sort(numbersInMatrix.begin(), numbersInMatrix.end());

}
```


magicconstant es el valor de la razón mágica, es decir, el que debe resultar de sumar cualquier fila, columna o diagonal de la matriz. Se obtiene sumando los elementos de la primera columna de la matriz.

```
void findMagicConstant(int matrixsize, int ** matrix, int &magicconstant){  
  
    for(int a=0; a<matrixsize; a++){  
        magicconstant += matrix[a][0];  
    }  
  
}
```

matrixsize es el valor de dimensión de la matriz, para este caso siempre cuadrada y que se extrae del primer entero del archivo fuente. La función **getmatrix** es la encargada de solicitar al usuario el archivo fuente donde se encuentra la matriz a analizar. En caso de que no se encuentre el archivo especificado por el usuario, la petición del mismo continuará:

```
Ingrese el nombre del archivo que tiene la matriz:  
matrizInexistente  
Archivo no encontrado, por favor intente de nuevo
```

Después de que **getmatrix** reciba un nombre válido, del archivo se extraerán la matriz, el vector, la constante mágica y el tamaño de la matriz. A continuación, se imprimirá la matriz con la determinación de si es mágica o no. La comprobación es realizada por la función **isMatrixMagic**, que devuelve **true** si y solo si se cumplen las tres condiciones especificadas, **false** de otro modo.

```
bool isMatrixMagic(int matrixsize, vector<int> &numbersInMatrix, int ** matrix, int &magicconstant){  
  
    if(!areNumbersInRange1_n2(matrixsize, numbersInMatrix)){  
        return false;  
    }  
    else{  
        if(areThereDuplicates(matrixsize, numbersInMatrix)){  
            return false;  
        }  
        else{  
            if(!magicSum(matrixsize, matrix, magicconstant)){  
                return false;  
            }  
            else{  
                return true;  
            }  
        }  
    }  
}
```

areNumbersInRange verifica la primera condición, que todos los números de la matriz (contenidos en el vector) se encuentren entre 1 y n al cuadrado. La función **areThereDuplicates** también hace uso del vector generado, esta es la que comprueba que no hayan entradas repetidas en la matriz, para lo que necesita que el vector se encuentre organizado. **magicSum** trabaja con la matriz. Esta función comprueba que el resultado de sumar cada fila, columna y diagonal de la matriz sea igual a la constante mágica.

```
matriz1.txt  
La matriz ingresada:  
1      1      1  
1      1      1  
1      1      1  
  
No es un cuadrado mágico
```

En este caso, la suma mágica y el rango de entradas se cumplen, pero no el que los elementos sean únicos.

```
La matriz ingresada:
1      15      14      4
12     6       7       9
8      10      11      5
13     3       2      16

Sí es un cuadrado mágico
```

En el caso anterior se cumplen todas las condiciones para una matriz 4x4.