

T.P. PARTE 1 – Instructivo de Uso

El siguiente instructivo estará dividido en 2 (dos) secciones principales conforme se responden todas las preguntas propuestas en la primera parte del trabajo práctico. A saber:

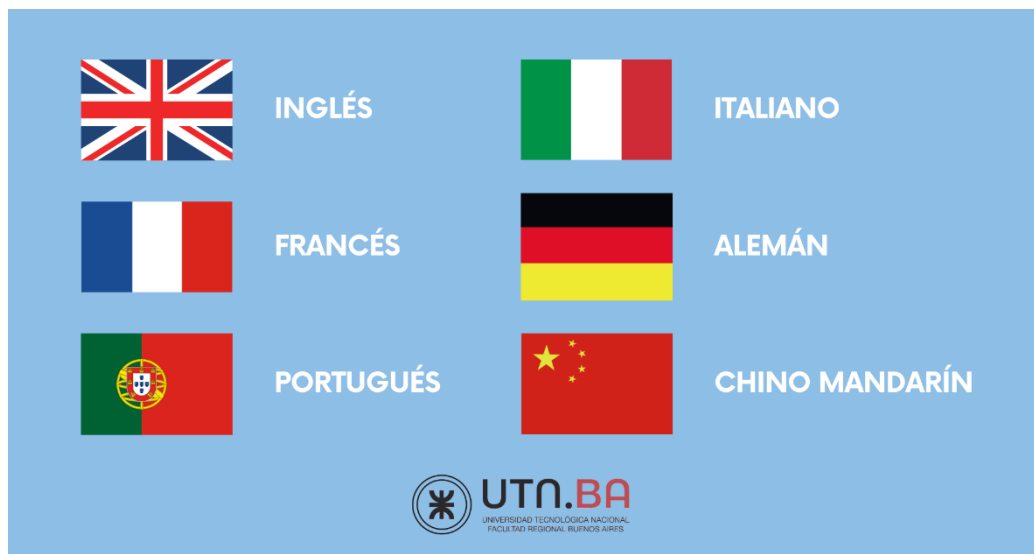
1. Introducción y funcionamiento del programa.

- Ingreso de Datos (input).
- Salida (output).
- Respuesta y manejo ante eventos inesperados.

2. Desarrollo y justificación de cómo se diseñó el programa.

Introducción y funcionamiento del programa:

El funcionamiento principal del programa *K1021-OsitosCariñositos-PARTE1.cpp* consiste en administrar la inscripción de alumnos a los distintos cursos de un instituto de idiomas. Siendo las opciones: Inglés, francés, portugués, italiano, alemán o chino mandarín.



Cada curso tendrá un profesor que imparta un idioma en un nivel específico del 1 al 8, siendo 1 el nivel más básico, y 8 el nivel más avanzado.

Ingreso de Datos:

El programa pedirá que el usuario ingrese los siguientes datos:

- **Ingresar el código del curso:** Se deberá ingresar un código únicamente numérico correspondiente al curso que se desea agregar.
- **Ingrese el idioma:** Según la oferta de idiomas, se debe elegir uno de los siguientes idiomas: *Inglés, francés, portugués, italiano, alemán o chino* (ingreso sin tilde).

- **Ingrese el nivel:** Deberá ingresar un número del 1 al 8, correspondiente al nivel que se desea enseñar. Sólo los siguientes niveles serán válidos: 1, 2, 3, 4, 5, 6, 7 u 8.
- **Ingrese el cupo:** Es la cantidad máxima (número) de alumnos que manejará el curso.
- **Ingrese el DNI del docente a cargo:** Deberá ingresar el DNI (SIN PUNTOS) del profesor que impartirá la materia.
- **Ingrese el nombre del docente:** Ingrese el nombre (sin el apellido) del docente a cargo.
- **Ingrese 1 para terminar o 0 para continuar:** Podrá ingresar únicamente dos opciones, 1 para finalizar, o 0 para continuar con el ingreso manual de los datos (se repetirán todos los pasos anteriores para agregar un nuevo curso).

ACLARACIÓN: No podrá haber dos cursos con un mismo idioma y nivel. Por cada idioma puede haber un máximo de 8 niveles. Por lo tanto la cantidad máxima de cursos será:

$$6 \text{ idiomas} \times 8 \text{ niveles} = 48 \text{ cursos posibles}$$

La validación está manejada correctamente en el programa.

Salida de Datos:

Una vez finalizado el ingreso de datos por medio del ingreso del número 1, o por agotamiento de los cursos posibles (48 cursos), el programa mostrará por pantalla la siguiente información:

- **Cantidad de cursos dictados de inglés**
- **Cantidad de cursos dictados de francés**
- **Cantidad de cursos dictados de portugués**
- **Cantidad de cursos dictados de italiano.**
- **Cantidad de cursos dictados de alemán.**
- **Cantidad de cursos dictados de chino mandarín.**
- **También se mostrará aquellos niveles en los que no se dictó ningún idioma.**

Vista previa de ejemplo:

```

D:\egust\Descargas\CinMadrakinp.exe
Ingrese el codigo de curso
1001
Ingrese uno de los siguientes idiomas: Ingles-Italiano-Frances-Aleman-Chino-Portugues
ingles
Ingrese un nivel entre del 1 al 8, siendo 1 el nivel mas basico y 8 el mas avanzado
7
Ingrese el cupo
30
Ingrese el DNI del docente a cargo
3399933
Ingrese el nombre del docente a cargo
Steve
Ingrese 1 para terminar con el ingreso de datos, 0 para continuar
1
Se dictaran 1 cursos de ingles
Se dictaran 0 cursos de frances
Se dictaran 0 cursos de portugues
Se dictaran 0 cursos de italiano
Se dictaran 0 cursos de aleman
Se dictaran 0 cursos de chino mandarin
No hay cursos de nivel 1
No hay cursos de nivel 2
No hay cursos de nivel 3
No hay cursos de nivel 4
No hay cursos de nivel 5
No hay cursos de nivel 6
No hay cursos de nivel 8
-----
Process exited after 75.73 seconds with return value 0

```

Respuesta ante eventos inesperados:

Código de curso repetido: El programa arrojará el mensaje “Código de curso ingresado ya está en uso, intente nuevamente” cuando el usuario haya ingresado un código existente, es decir, que ya había ingresado con anterioridad. Se solicitará su reingreso hasta que el código de curso no se halle repetido.

```

Ingrese 1 para terminar con el ingreso de datos, 0 para continuar
0
Ingrese el codigo de curso (Un numero entero mayor a 0)
100
Codigo de curso ingresado ya esta en uso, intente nuevamente
100
Codigo de curso ingresado ya esta en uso, intente nuevamente
101
Ingrese uno de los siguientes idiomas: Ingles-Italiano-Frances-Aleman-Chino-Portugues

```

Idioma no válido: Si el usuario ingresa un idioma que no se encuentra dentro de las opciones, inmediatamente el programa le informará que el idioma no es válido, y solicitará nuevamente su ingreso, hasta que sea válido.

```

D:\agust\Descargas\cimadarakintp.exe
Ingrese el codigo de curso
100
Ingrese uno de los siguientes idiomas: Ingles-Italiano-Frances-Aleman-Chino-Portugues
ruso
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
Hindi
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
Arabe
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
Japones
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
Coreano
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
Turco
No tenemos cursos de ese idioma, por favor seleccione uno de los mencionados
italiano
Ingrese un nivel entre del 1 al 8, siendo 1 el nivel mas basico y 8 el mas avanzado
3
Ingrese el cupo
20

```

Curso con nivel repetido: Si el programa produce la siguiente salida: “EL CURSO INGRESADO YA EXISTE, VUELVA A INTENTARLO”, significa que usted ya ingresó ese idioma con ese nivel en específico. Por lo tanto, el programa le solicitará que vuelva a ingresar el idioma y nivel nuevamente. Como se explicó anteriormente, no pueden existir 2 cursos con un mismo idioma y nivel. El nivel debe cambiar necesariamente.

Si ya se ingresaron los 8 niveles de ese idioma, entonces deberá ingresar otro idioma distinto.

```

Ingrese uno de los siguientes idiomas: Ingles-Italiano-Frances-Aleman-Chino-Portugues
ingles
Ingrese un nivel entre del 1 al 8, siendo 1 el nivel mas basico y 8 el mas avanzado
1
EL CURSO INGRESADO YA EXISTE, VUELVA A INTENTARLO
Ingrese el idioma
ingles
Ingrese el nivel
2
Ingrese el cupo

```

Nivel inexistente: El programa producirá el siguiente mensaje en caso de que haya ingresado un nivel fuera del rango 1-8. Se le solicitará que ingrese nuevamente el nivel.

Hasta que no se ingrese un nivel correcto, el programa seguirá produciendo la misma salida.

```

Ingrese un nivel entre del 1 al 8, siendo 1 el nivel mas basico y 8 el mas avanzado
10
El nivel ingresado no se es un numero del 1 al 8, por favor vuelva a ingresarlo
-4
El nivel ingresado no se es un numero del 1 al 8, por favor vuelva a ingresarlo
5
Ingrese el cupo

```

Desarrollo y justificación de cómo se diseñó el programa:

Directivas del preprocesador: Las directivas del archivo de código fuente le dicen al preprocesador que tome medidas específicas. Las directivas utilizadas son `#include` para incluir las librerías `iostream`, `stdio.h`, y `string.h`.

Además el **using namespace std** es utilizado para que la computadora sepa el código para las funciones **cout** y **cin** utilizadas en el programa.

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <string.h>
4
5
6 using namespace std;
7
```

Estructuras (struct): Para almacenar la información ingresada por el usuario, se utilizaron dos estructuras (**struct**), una correspondiente a la información del docente, y la otra a la información de los cursos.

El **struct Docente** se compone de las variables **dni** y **nombre**. Para el DNI se utilizó el tipo de dato "int", y para el nombre se utiliza un "char" de 20 caracteres.

El **struct Curso** se compone de los miembros **codigo**, **idioma**, **nivel**, **cupos**, y **docente**. Asumimos que los códigos son enteros "int" y hardcodeamos el idioma como un "char" de 10 caracteres máximos (según el idioma más largo listado). Finalmente, se crea una variable docente, que usa la estructura de nombre "Docente" como el tipo de datos de la misma.

```
struct Docente
{
    int dni;
    char nombre[20];
};

struct Curso
{
    int codigo;
    char idioma[10]; //Portugues es el idioma mas largo y tiene 9 caracteres;
    int nivel;
    int cupos;
    Docente docente;
};
```

Prototipos de Función: Se declaran las funciones que decidimos utilizar para nuestro programa, con su retorno y parámetros.

```
void punto1(Curso cursos[]);
void generarIdioma(Curso cursos[], char idioma[]);
void ordenarVector(Curso cursos[]);
void verCursos(Curso cursos[]);
void vaciarCursos(Curso cursos[]);
void ingresoCursos(Curso cursos[]);
bool verificarCodigo(int i, Curso cursos[]);
bool verificarNivel(int nivel);
void punto2(Curso cursos[]);
int contarIdioma(Curso cursos[], char idioma[]);
void punto3 (Curso cursos[]);
void punto4 (Curso curso[]);
bool verificarError(Curso cursos[], int i);
```

void punto1(Curso cursos[]): Llama/invoca a la función `ordenarVector()` y se pasa el array de cursos como argumento. Posteriormente recorre el vector, y según el idioma de cada elemento, lo agrega al archivo del idioma.dat correspondiente.

En caso de no existir el archivo, se creará.

void generarIdioma(Curso cursos[], char Idioma[]): Recibe un array de structs llamado "cursos". Agrega información de los cursos que dictan cada uno de los idiomas en un archivo "Idioma.dat", donde "Idioma" corresponde al idioma específico que dicta el curso. Si el archivo no existe, creará uno nuevo con el formato "Idioma.dat".

void ordenarVector(Curso cursos[]): Esta función se utiliza para ordenar, según el código, los cursos. El objetivo de esto es mostrar en orden ascendente los cursos en los archivos .dat. Para el ordenamiento se implementó el método de ordenamiento burbuja.

void verCursos(Curso cursos[]): Recorre los cursos existentes con un for, y los muestra por pantalla.

void vaciarCursos(Curso cursos[]): Recibe un array de structs llamado "cursos". Esta función se desarrolló para evitar los errores en tiempo de ejecución causados por la "basura" en memoria. Se aplica al principio del programa y no retorna ningún valor.

void ingresoCursos(Curso cursos[]): Recibe un array "cursos" que utiliza la estructura Curso. No retornará ningún valor. Se encarga de almacenar los datos ingresados por el usuario

bool verificarCodigo(int i, Curso cursos[]): El objetivo de esta función es verificar que un código de curso ingresado por el usuario no se repita. Recibe por argumento la posición donde se ingresó el código de curso. Devuelve 1(True) si se halló coincidencia, o 0 (False) si no encontró repeticiones.

bool verificarNivel(int nivel): Recibe un entero llamado "nivel". Retorna un 1 (True) si el nivel ingresado no está dentro del rango 1-8, o un 0 (False) si el nivel se encuentra dentro de las opciones válidas.

void punto2(Curso cursos[]): Resuelve la problemática propuesta en el punto 2 del trabajo. Recibe un array de structs llamado "cursos". Mostrará por pantalla cuántos cursos se dictaron de cada idioma. No retornará nada.

void contarIdioma(Curso cursos[], char idioma[]): Recibe la totalidad de los cursos y utiliza un contador para llevar un recuento de cada uno de los idiomas. Retorna el valor de dicho contador.

void punto3(Curso cursos[]): Resuelve la problemática propuesta en el punto 3 del trabajo. Recibe un array de structs llamado "cursos". Mostrará un mensaje por pantalla si se detectó que un idioma tiene cursos en los 8 niveles. No retornará ningún valor.

void punto4(Curso cursos[]): Resuelve la problemática propuesta en el punto 4 del trabajo. Recibe un array de structs llamado “cursos”. Dará una salida por pantalla si para un nivel en específico no se encontró ningún curso dictado de ningún idioma. No retornará nada.

bool verificarError(Curso cursos[], int i): Recibe un array de structs “cursos” y un entero “i”. Devolverá 1(True) o 0 (False) según se valide. La función consiste en verificar que para un curso e idioma recién ingresados, no exista otro curso con ese mismo idioma y nivel.

Función principal main(): El main es simple, consiste en realizar las llamadas a las funciones que se emplearán. Además, se declara el array “cursos” con un máximo de [48] espacios, según el máximo de cursos posibles. Cada uno de esos array utilizará la estructura de “Curso”.

Primero se llamará a la función vaciarCursos(cursos) para eliminar los posibles datos “basura”. Opcionalmente nosotros llamamos a verCursos(cursos) para comprobar que se haya eliminado cualquier dato basura posible. Por último ingresoCursos(cursos) permitirá ingresar los cursos y nuevamente verCursos(cursos) para poder visualizarlos. Una vez guardado los datos que el usuario ingresó, se llama a las 4 funciones correspondientes a las 4 problemáticas que el programa debe resolver.

```
int main()
{
    Curso cursos[48];
    vaciarCursos(cursos);
    verCursos(cursos);
    ingresoCursos(cursos);
    verCursos(cursos);
    punto1(cursos);
    punto2(cursos);
    punto3(cursos);
    punto4(cursos);

    verCursos(cursos);
    return 0;
}
```

Definiciones de Funciones:

La función **ingresoCursos()** declara una variable iterador “i” y un bucle while, para ir almacenando toda la información de cada curso. Cada posición [i] del vector, se corresponderá a un curso específico, que use la estructura *Curso*.

```

void ingresoCursos(Curso cursos[])
{
    int i = 0;
    Docente auxiliar;
    int salir = 0;

    while (salir != 1)
    {

        do{

            cout << "Ingrese el codigo de curso (Un numero entero mayor a 0) \n";
            cin >> cursos[i].codigo;

        }while(verificarCodigo(i,cursos) || cursos[i].codigo == 0);
    }
}

```

Ingreso de Código del Curso: Se invocará la función “**verificarCodigo(i, cursos)**” que verificará que el código no se repita comparándolo con el código recién ingresado por el usuario. Por su parte, también se evitará que el usuario ingrese un código igual a 0 (cero).

Ingreso de Idioma: Se pedirá por pantalla que se ingrese uno de los siguientes idiomas (inglés, italiano, francés, alemán, chino, portugués).

Ingreso de Nivel del Curso: Se pide que el usuario ingrese un nivel dentro del rango 1-8

```

    cout << "Ingrese uno de los siguientes idiomas(EN MINUSCULA): ingles-italiano-frances-aleman-chino-portugues \n";
    cin >> cursos[i].idioma;

    cout << "Ingrese un nivel entre del 1 al 8, siendo 1 el nivel mas basico y 8 el mas avanzado \n";
    cin >> cursos[i].nivel;

    while (verificarNivel(cursos[i].nivel))
    {
        cout << "El nivel ingresado no se es un numero del 1 al 8, por favor vuelva a ingresarlo \n";
        cin >> cursos[i].nivel;
    }

    while (verificarError(cursos, i))
    {
        cout << "EL CURSO INGRESADO YA EXISTE, VUELVA A INTENTARLO \n";
        cout << "Ingrese el idioma \n";
        cin >> cursos[i].idioma;
        cout << "Ingrese el nivel \n";
        cin >> cursos[i].nivel;

        while (verificarNivel(cursos[i].nivel))
        {
            cout << "El nivel ingresado no se es un numero del 1 al 8, por favor vuelva a ingresarlo \n";
            cin >> cursos[i].nivel;
        }
    }
}

```

Se hará una llamada de la función **verificarNivel()** pasándole el nivel como argumento. Se implementa un while para que mientras no escriba un nivel válido, se siga pidiendo su ingreso.

Posteriormente se invoca a la función **verificarError()** la cual valida que no exista un curso con ese mismo idioma y nivel. También se usó un while para pedir el ingreso nuevamente del idioma y nivel hasta que no encuentre coincidencias.


```

cout << "Ingrese el cupo \n";
cin >> cursos[i].cupo;

cout << "Ingrese el DNI del docente a cargo \n";
cin >> auxiliar.dni;

cout << "Ingrese el nombre del docente a cargo \n";
cin >> auxiliar.nombre;
cursos[i].docente = auxiliar;

cout << "Ingrese 1 para terminar con el ingreso de datos, 0 para continuar \n";
cin >> salir;
i++;

```

Ingreso de Cupo, DNI y Nombre de Docente: Estos datos se leerán directamente. Para el caso del dni y nombre, se emplea una *variable auxiliar que utiliza el struct Docente*, y luego su valor se le asigna al vector `cursos[i].docente`

Finalmente, la variable “salir” leerá 1 o 0 para terminar o seguir con el ingreso de datos según lo establecido en el primer while, y se incrementará el iterador para continuar con el correcto almacenamiento de los cursos.

La función **verificarCodigo()** recibe como argumento la posición del curso ingresado, y un vector con todos los cursos almacenados. A su vez, establece una variable bool en false.

Luego recorre el vector *cursos* con un for, y setea la variable “salida” en “true” en caso de hallar coincidencia en los códigos de cursos almacenados, en alguna posición (por eso el `i != j`).

```

137 bool verificarCodigo(int i, Curso cursos[]){
138     bool salida=false;
139
140     for(int j=0; j<48;j++){
141         if((i!=j) && (cursos[i].codigo==cursos[j].codigo)){
142             salida=true;
143         }
144     }
145     return salida;
146 }

```

La función **verificarNivel(int nivel)** es igual de simple. Cuando es invocada, recibe el nivel como argumento, y dependiendo si el nivel está dentro del rango 1-8 o no, devolverá true (si es un nivel fuera del rango), o false (si es un nivel válido).

```

205 bool verificarNivel(int nivel)
206 {
207     bool salida = true;
208     for (int i = 0; i < 8; i++)
209     {
210         if (nivel == (i + 1))
211         {
212             salida = false;
213         }
214     }
215     return salida;
216 }
217

```

La función **verCursos()** utiliza un *for* y una estructura condicional *if* para recorrer solo aquellos cursos existentes, y posteriormente imprime sus datos por pantalla (código, nivel, idioma, cupos).

La función **vaciarCursos()** es llamada siempre al principio del programa, para establecer en 0 o ‘’, cada curso almacenado en el vector `cursos[]`. Se implementa para eliminar los datos “basura”.

```

void vaciarCursos(Curso cursos[])
{
    for (int i = 0; i < 48; i++)
    {
        cursos[i].codigo= 0;
        cursos[i].idioma = "";
        cursos[i].nivel = 0;
        cursos[i].cupu = 0;
    }
}

```

La función **punto1()** recibe como argumento cada curso ingresado por el usuario.

Realiza una llamada a la función **ordenarVector(cursos)** para que se ordenen según su código de forma ascendente. Posteriormente, recorre todos los cursos con un bucle *for*, y con ayuda de las cláusulas *else if* (que equiparan el funcionamiento de un *switch case*), se hará una llamada a la función `generarIdioma()`, pasándole como argumento “palabra” que es un char de 20 caracteres utilizado para almacenar el idioma que se lee.

Se aplica la función **strcmpi()** para evitar que distinga entre idiomas (que no sea case-sensitive)

```

void punto1(Curso cursos[]){
    ordenarVector(cursos);
    char palabra[20];
    for (int i = 0; i < 48; i++)
    {
        if (strcmpi( cursos[i].idioma,"ingles") == 0 )
        {
            strcpy(palabra,"Ingles");
            generarIdioma(cursos,palabra);
        }
        else if (strcmpi( cursos[i].idioma,"italiano") == 0 )
        {
            strcpy(palabra,"Italiano");
            generarIdioma(cursos,palabra);
        }
    }
}

```

```

        else if (strcmpi( cursos[i].idioma,"Frances") == 0 )
        {
            strcpy(palabra,"Frances");
            generarIdioma(cursos,palabra);
        }
        else if (strcmpi( cursos[i].idioma,"Aleman") == 0 )
        {
            strcpy(palabra,"Aleman");
            generarIdioma(cursos,palabra);
        }
        else if (strcmpi( cursos[i].idioma,"Chino") == 0 )
        {
            strcpy(palabra,"Chino");
            generarIdioma(cursos,palabra);
        }
        else if (strcmpi( cursos[i].idioma,"Portugues") == 0 )
        {
            strcpy(palabra,"Portugues");
            generarIdioma(cursos,palabra);
        }
    }
}

```

La función **strcpy()** copia la cadena correspondiente al idioma hacia la variable char "palabra".

La función **generarIdioma()** es la encargada de añadir la información de los cursos en el archivo *Idioma.dat* o en su defecto, crearlo, según el modo "ab" correspondiente a la apertura de archivos binarios.

Se define archParcial como puntero a FILE, y abre el archivo *Idioma.dat*. En caso de no poder crearse, se imprime por pantalla "ERROR". Sino, se procede a escribir el archivo con los datos del vector cursos, y finalmente se cierra archivo asociado.

```

218 void generarIngles(Curso cursos[])
219 {
220     FILE *archParcial = fopen("Ingles.dat", "ab");
221     if (archParcial == NULL)
222         cout << "ERROR" << endl;
223     else
224     {
225         fwrite(&cursos, sizeof(Curso), 1, archParcial);
226         fclose(archParcial);
227     }
228 }
229
230
231

```

La función **ordenarVector()** recibe el vector con todos los cursos ingresados por el usuario, y los ordena. Para ello, se usan dos iteradores, "i" y "j".

El iterador "j" va comparando el código del curso en la posición actual, con el código del curso en la siguiente posición. Si este último es menor, entonces se los cambia de lugar.

```

void ordenarVector(Curso cursos[])
{
    unsigned i = 1, j;
    Curso aux;
    bool cambio;
    do
    {
        cambio = false;
        for (j = 0; j < 47; j++)
        {
            if (cursos[j].codigo > cursos[j + 1].codigo)
            {
                aux = cursos[j];
                cursos[j] = cursos[j + 1];
                cursos[j + 1] = aux;
                cambio = true;
            }
        }
        i++;
    } while (i < 48 && cambio);
}

```

punto2() es la función que muestra por pantalla la cantidad de cursos que se crearon por cada idioma, gracias a las llamadas a la función *contarIdioma()*. Se usa también **strcpy**

```

void punto2(Curso cursos[])
{
    char palabra[20];
    strcpy(palabra, "Ingles");
    cout << "Se dictaran " << contarIdioma(cursos, palabra) << " cursos de ingles \a\t\n";
    char palabra1[20];
    strcpy(palabra1, "Frances");
    cout << "Se dictaran " << contarIdioma(cursos, palabra1) << " cursos de frances \n";
    char palabra2[20];
    strcpy(palabra2, "Portuges");
    cout << "Se dictaran " << contarIdioma(cursos, palabra2) << " cursos de portugues \n";
    char palabra3[20];
    strcpy(palabra3, "Italiano");
    cout << "Se dictaran " << contarIdioma(cursos, palabra3) << " cursos de italiano \n";
    char palabra4[20];
    strcpy(palabra4, "Aleman");
    cout << "Se dictaran " << contarIdioma(cursos, palabra4) << " cursos de aleman \n";
    char palabra5[20];
    strcpy(palabra5, "Chino");
    cout << "Se dictaran " << contarIdioma(cursos, palabra5) << " cursos de chino mandarin \n";
}

```

La función **contarIdioma()** establece una variable contador, y recorre todos los cursos. Por cada idioma encontrado, se incrementará su respectivo contador en 1.

```

int contarIdioma(Curso cursos[], char idioma[]){
    // cuenta la cantidad de un idioma que hay

    int cont = 0;

    for(int i=0; i < 48 ; i++){
        if (strcmpi( cursos[i].idioma, idioma) == 0 )
        {
            cont++;
        }
    }

    return cont;
}

```

La función **punto3()** se invoca en el main luego de que el usuario haya ingresado todos los cursos. Recibe todos los cursos como argumento, y hace un llamado a las funciones **contarIdioma()**. Si el retorno de dichas funciones es igual a 8 (ocho), entonces imprime por pantalla que ese idioma tiene curso en los 8 niveles. En caso contrario no hará nada.

```

void punto3(Curso cursos[])
{
    char palabra[20];
    strcpy(palabra, "Ingles");
    if (contarIdioma(cursos, palabra) == 8)
    {
        cout << "\nIngles tiene curso en los 8 niveles.";
    }

    char palabra1[20];
    strcpy(palabra1, "Frances");
    if (contarIdioma(cursos, palabra1) == 8)
    {
        cout << "\nFrances tiene curso en los 8 niveles.";
    }

    char palabra2[20];
    strcpy(palabra2, "Portugues");
    if (contarIdioma(cursos, palabra2) == 8)
    {
        cout << "\nPortugues tiene curso en los 8 niveles.";
    }

    char palabra3[20];
    strcpy(palabra3, "Italiano");
    if (contarIdioma(cursos, palabra3) == 8)
    {
        cout << "\nItaliano tiene curso en los 8 niveles.";
    }

    char palabra4[20];
    strcpy(palabra4, "Aleman");
    if (contarIdioma(cursos, palabra4) == 8)
    {
        cout << "\nAleman tiene curso en los 8 niveles.";
    }

    char palabra5[20];
    strcpy(palabra5, "Chino");
    if (contarIdioma(cursos, palabra5) == 8)
    {
        cout << "\nChino Mandarin tiene curso en los 8 niveles.";
    }
}

```

La función **punto4()** también recibe todos los cursos que el usuario ingresa.

Declara un vector `niveles[8]`, con 8 posiciones correspondientes a los niveles.

Recorre con un for todos los cursos, y anida otro for que “va del 1 al 8”(k+1) según los niveles posibles. El objetivo es detectar el nivel de cada curso. Por lo tanto, cuando encuentre una coincidencia entre (cursos[i].nivel) y (k+1), se establece el vector niveles[i] en 1.

En otras palabras, cuando se establece en 1, significa que existe un curso con dicho nivel. Como el vector niveles tiene 8 posiciones, será fácil saber aquellos niveles que tienen algún curso, y aquellos que no.

Para mostrar los niveles que no tienen cursos, se utilizó el for de la parte inferior.

```
void punto4(Curso cursos[])
{
    int niveles[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // este vector va a tener 1 donde haya al menos 1 curso de ese nivel siendo niveles[0]-->el nivel 1

    for (int i = 0; i < 48; i++) // recorremos el vector de cursos
    {
        for (int k = 0; k < 8; k++)
        { // nos fijamos en cada curso si existe algun nivel entre el 1 y el 8
            if (cursos[i].nivel == (k + 1))
            { // si existe un nivel
                niveles[k] = 1; // colocamos un 1 en la posición equivalente a ese nivel. Ej nivel 3--> niveles[2]=1 (donde 2 va a ser k)
            }
        }
    }

    for (int j = 0; j < 8; j++)
    {
        if (niveles[j] == 0)
        { // mostramos del vector nivel las posiciones donde no haya un 1, es decir donde no haya nivel
            cout << "No hay cursos de nivel " << j + 1 << "\n";
        }
    }
}
```



Finalmente, la función **verificarError()** se utiliza para verificar que no exista un curso con el mismo idioma y nivel.

Cuando el usuario ingresa el nivel, se realiza una llamada a esta función, y se le pasa como argumento los cursos y la posición del curso que está ingresando.

Se declara una variable booleana en false, y con un for recorre todos los cursos. Si detecta una coincidencia en el idioma ingresado con un idioma existente y además una coincidencia en el nivel ingresado con un nivel existente, la variable se establece en true. Finalmente, la función retorna el valor de la variable.

```
bool verificarError(Curso cursos[], int i) // verifica que no exista ya el curso de ese idioma a es
{
    bool var = false;
    for (int k = 0; k < 48; k++)
    {
        if (cursos[i].idioma == cursos[k].idioma && cursos[i].nivel == cursos[k].nivel && i != k)
        { // el i!=k es para que no tome el que ya ingresamos
            var = true;
        }
    }
    return var;
}
```

Para finalizar, un ejemplo de los archivos .dat generados:

 Ingles	07/08/2022 19:24	Archivo DAT	1 KB
 Italiano	07/08/2022 19:24	Archivo DAT	1 KB

