DESCRIPCIÓN DEL PROYECTO #1
"CONVERTIDOR LENGUAJE NATURAL A LENGUAJE C++"

Objetivo General:

El objetivo de este proyecto es desarrollar una herramienta de conversión que permita transformar instrucciones escritas en lenguaje natural a código C++ estructurado. Los estudiantes deberán emplear conceptos fundamentales de programación como listas, arreglos, variables y estructuras de control para implementar el convertidor (obligatorio). El desafío consiste en traducir un conjunto de instrucciones simples escritas en un lenguaje más cercano al humano a un formato que la computadora pueda entender.

Descripción del Proyecto:

Los estudiantes desarrollarán un programa en C++ que reciba una serie de instrucciones escritas en lenguaje natural (por ejemplo, "sumar 5 y 3", "crear una lista de números", "imprimir el resultado") y las convierta a código C++ válido. El programa debe ser capaz de identificar las acciones requeridas y producir el fragmento de código correspondiente en C++.

Componentes Requeridos del Proyecto:

1. **Entrada de Lenguaje Natural**:
   El programa debe permitir al usuario introducir un conjunto de instrucciones en lenguaje natural (un archivo (.txt)).
2. **Procesamiento del Lenguaje Natural**:
   El programa debe interpretar las instrucciones de manera correcta. Esto puede implicar la identificación de palabras clave como "sumar", "restar", "crear lista", etc.
3. **Generación del Código C++**:
   A partir de la interpretación de las instrucciones, el programa debe generar código C++ que realice las operaciones indicadas. Por ejemplo:
   - Entrada: "sumar 5 y 3" → Salida: int resultado = 5 + 3;
   - Entrada: "crear lista de números con 10 elementos" → Salida: int lista[10];
4. **Manejo de Variables y Tipos de Datos**:
   El programa debe generar código con tipos de datos apropiados según las instrucciones (por ejemplo, int, float, string, etc.).

5. **Uso de Listas y Arreglos**:
   El proyecto debe incluir ejemplos donde se utilicen listas (o arreglos) que el programa debe generar dinámicamente en función de la entrada.
6. **Estructuras de Control**:
   El programa debe ser capaz de generar estructuras de control básicas, como condicionales (if, else) y bucles (for, while), basadas en las instrucciones del lenguaje natural.

Ejemplo de Conversión:

- **Instrucción**: "Sumar los números 10, 20, 30 y mostrar el resultado."
- **Código C++ generado**:

```
int suma = 10 + 20 + 30;
cout << "El resultado es: " << suma << endl;
```

- **Instrucción**: "Crear una lista de 5 números enteros."
- **Código C++ generado**:

```
int lista[5];
```

Aspectos Técnicos del Proyecto:

1. **Entradas**:
   - El programa debe ser capaz de recibir al menos 3 tipos de entrada: operaciones matemáticas simples, creación de arreglos/listas y estructuras de control (condicionales y bucles).
2. **Salidas**:
   - El programa generará código C++ estructurado según la entrada del usuario (archivo).
   - La salida debe ser un código C++ válido que el usuario pueda ejecutar en un entorno de programación.
3. **Validación**:
   - El sistema debe validar que las entradas sean comprensibles y correctas. Si la entrada es ambigua o no se puede procesar, el programa debe mostrar un mensaje de error adecuado.
4. **Interactividad**:
   - El programa debe permitir al usuario ingresar múltiples instrucciones de forma continua, por medio de archivo de carga original.
5. **Interfaz Gráfica**:

- El programa debe tener una interfaz gráfica intuitiva y sencilla, donde se permita cargar un archivo, se muestre el contenido del archivo y se pueda procesar la conversión del lenguaje natural y mostrarse en otra pantalla o en otro componente el resultado de dicha conversión, El resultado de esa conversión debe ser generado también en un archivo exportable a Visual Studio para ser ejecutado en dicho IDE y así probar la eficiencia del código generado.

Herramientas Recomendadas:

- **IDE de desarrollo**: Visual Studio Community 2022.

Aspectos para la Evaluación:

1. **Correctitud del Código (35%)**:
   - El programa debe generar un código C++ que cumpla con las instrucciones proporcionadas.
   - La salida debe ser un código válido y funcional que se pueda ejecutar en un compilador C++ sin errores.
2. **Implementación de Listas y Arreglos (20%)**:
   - Debe emplearse el uso correcto de arreglos y listas en el código generado, adaptándolos a las instrucciones que lo requieran.
3. **Manejo de Variables y Tipos de Datos (10%)**:
   - Las variables deben ser correctamente declaradas y deben ser del tipo adecuado según la operación solicitada (por ejemplo, int, float, char).
4. **Uso de Estructuras de Control (10%)**:
   - El programa debe ser capaz de generar estructuras de control como if, else, while, y for correctamente.
5. **Interfaz de Usuario y Usabilidad (10%)**:
   - El programa debe ser fácil de usar, con entradas claras y salidas bien estructuradas.
   - Debe manejar errores y entradas incorrectas de manera amigable.
6. **Creatividad y Complejidad (10%)**:
   - Se evaluará la capacidad de los estudiantes para crear un sistema que sea capaz de interpretar instrucciones más complejas y generar código que utilice conceptos avanzados de C++, como el uso de punteros, funciones o clases, si lo aplican.
7. **Defensa (5%)**:
   - Se realizará una defensa entre estudiante y académico para validar el funcionamiento del software desarrollado así como la originalidad, creatividad y particularidades inherentes al mismo desarrollo.

Observaciones Finales:

Este proyecto permite a los estudiantes experimentar con las bases de la programación en C++ a través de la creación de un sistema que involucra conceptos fundamentales como la manipulación de listas, variables y estructuras de control.

Este proyecto es ideal para afianzar los fundamentos de C++ y promover el pensamiento algorítmico en estudiantes que están comenzando en el mundo de la programación.

Instrucciones Generales:

1. El trabajo deberá entregarse en forma digital (fuentes y ejecutable) el día y hora indicados en el aula virtual según se indica en el cronograma del curso.
2. El proyecto será desarrollado de **forma individual**.
3. El proyecto será desarrollado en la herramienta ***VS Community 2022 - C++.***
4. El proyecto deberá llevar su control con una herramienta de control de versiones elegida por el profesor a cargo del curso. Para ello se hará una revisión de avance según las fechas establecidas.
5. El proyecto será probado en dos ocasiones, la primera será con el profesor y el estudiante y la segunda será por parte del profesor a cargo del grupo.

Rúbrica de la Evaluación del Proyecto

Criterio	Puntaje
Correctitud del Código	**35%**
Implementación de Listas y Arreglos	**20%**
Manejo de Variables y Tipos de Datos	**10%**
Uso de Estructuras de Control	**10%**
Interfaz de Usuario y Usabilidad	**10%**
Creatividad y Complejidad	**10%**
Defensa	**5%**
TOTAL	**100%**

Fecha de Entrega

Entrega del enunciado: Jueves 31 de julio del 2025.
Revisión y defensa: Jueves 25 de setiembre del 2025.

Anexos

Anexo #1

Buenas Prácticas solicitadas en el proyecto

	Código
	Utiliza estándares propios del lenguaje C++.
	Utiliza control de versiones Git
	Utiliza múltiples commits para la realización del proyecto, se denota una planeación en la realización del proyecto (Existe una cantidad de commits y orden razonable)
	Cada commit corresponde a una tarea única, está bien comentado en inglés. (buena documentación)
	Utiliza código correctamente formateado, escrito en inglés.
	El código de cada archivo esta ordenado, variables declaradas primero, luego métodos públicos (empieza con el constructor y destructor) y de ultimo métodos privados
	Archivos
	Maneja archivos separados para cada estructura o clase, así como de encabezado.
	El nombre de cada archivo cumple con las buenas prácticas: extensiones correctas, nombre representativo, utiliza los caracteres correctos.
	Nomenclatura
	El nombre de cada clase cumple con las buenas prácticas: nombre representativo, PascalCase, utiliza los caracteres correctos, es un sustantivo, está en inglés.
	El nombre de cada atributo, variable o parámetro cumple con las buenas prácticas: nombre representativo, no incluye tipo, pero representa su identidad (por ejemplo, booleanos positivos, o nombres plurales), camelCase, utiliza los caracteres correctos, es un sustantivo, está en ingles
	El nombre de cada método o función cumple con las buenas prácticas: nombre representativo, camelCase, utiliza los caracteres correctos, inicia con un verbo, indica correctamente lo que devuelve, realiza función única, muestra de manera explícita lo que debe recibir, está en ingles
	Estructuras de control
	Utiliza curly brackets siempre
	Utiliza los diferentes ciclos para lo que es
	Evita anidaciones de más de 3 en cada función
	La lógica booleana esta optimizada, prefiere expresiones de condición legibles, utiliza el else solo cuando es necesario, utiliza mejores alternativas al switch
	Mensajes al usuario y comentarios
	El código no contiene comentarios
	Los mensajes al usuario son positivos, están en español, tienen buena gramática y ortografía
	Programación orientada a objetos

	Utiliza herencia cuando es necesario
	Utiliza polimorfismo cuando es necesario
	Utiliza un nivel de abstracción adecuado
	Utiliza encapsulamiento (abierto a extensión, cerrado para modificación)
	Utiliza bajo acoplamiento y alta cohesión
	Utiliza el principio de responsabilidad única
	Utiliza el principio de DRY, reutiliza código, evita repetirlo