C++ según yo

Introducción:

Los lenguajes de programación permiten “decirle” a las computadoras lo que queremos que hagan.

Hay 2 tipos principales de lenguajes de programación:

- Compilados: se escriben en un archivo de código fuente que un programa llamado “compilador” traduce a código “Ensamblador” (Assembly), y este es luego traducido a código máquina (machine code). Ej.: C, C++, Rust, Swift.
- Interpretados: su archivo de código fuente es leído por un programa llamado “intérprete”, el cual lo ejecuta en tiempo real. Así funcionan los navegadores (Chrome, Firefox, Edge, etc.). Ej.: Javascript, php, Python.

Para programar, se necesitan al menos 2 programas:

- Un programa que permita escribir el código fuente. Puede ser sencillo, como el bloc de notas o Notepad++, o más complejo. A estos últimos se les llama Entorno de Desarrollo Integrado (IDE). Algunos ejemplos son Visual Studio, Netbeans y Zinjai para C++.
- Un compilador o intérprete, según el lenguaje a utilizar. Los IDE suelen incluir el compilador, automatizando el proceso.

C++:

C++ es un lenguaje compilado muy popular, inspirado en su predecesor C, con una sintaxis que muchos lenguajes más modernos han adoptado. Sus archivos de código fuente tienen extensión “.cpp”.

C++ tiene un preprocesador que interpreta directivas (líneas en el código fuente que empiezan con “#”), lo cual añade algunas características interesantes al lenguaje. Estas directivas son leídas antes del compilado, automáticamente.

Estructura típica de un programa simple de C++:

```
1. #include <iostream>
2. #include <string>
3. #include “archivoDeEncabezado”
4.
5. using namespace std;
6.
7. const int CONSTANTE = -5;
8.
9. struct TipoDeDato {
10.       //Propiedades del registro
11. } ejemplo;
12.
13. int a;
14. float b = 2.0;
15. char c = ‘T’;
16. string d = “Cadena de texto”;
17. unsigned long int e[10];
18.
19. int funcion1(int parametro1, char parametro2);
20. void funcion2(int parametro1[10], string & parametro2);
```

```
21.
22. int main() {
23.       //Programa
24.       return 0;
25. }
26.
27. int funcion1(int parametro1, char parametro2){
28.       //Definición de la funcion 1
29.       return varLocal;
30. }
31. void funcion2(int parametro1[10], string & parametro2){
32.       //Definición de la función 2
33. }
```

- Directivas `#include` (líneas 1-3): le indican al preprocesador qué archivos de encabezado utilizar. Estos añaden funcionalidades extra al programa y permiten, entre otras cosas, separar el código fuente en distintos archivos. El uso de símbolos "<>" indica que el archivo de encabezado debe ser buscado en la carpeta en donde se ubican por defecto, y "" indican que se busquen en la carpeta raíz del proyecto (donde se encuentra el archivo actual). Muchos archivos de encabezado vienen ya incluídos, como `iostream` (para entrada/salida con la consola), `string` (para simplicidad al usar cadenas de texto), `ifstream`, `ofstream` y `fstream` (para manipulación de archivos).
- `Using namespace` (línea 5): esta combinación de palabras clave es opcional, pero simplifica el código. Un `namespace` es una "herramienta" que, básicamente, ayuda a juntar múltiples archivos de encabezado con diferentes funciones, estructuras (`struct`) y clases (`class`), entre otras, que se llamen igual sin generar conflictos por esto. Si no colocamos esta línea, deberíamos escribir `std::cout` en lugar de `cout`, `std::cin` en lugar de `cin`, `std::string` en lugar de `string`, etc.
- Declaración y definición de constantes (línea 7): como a cualquier variable, pero agregando la palabra clave `const` antes del tipo de dato.
- Declaración de tipos (líneas 9-11): la palabra reservada `struct` permite la implementación de registros. Inmediatamente después de esta, se coloca el nombre del tipo de registro (opcional, utilizado si queremos crearlos más adelante en el programa), la definición de las variables (propiedades) que contiene (entre "`{}`") y la declaración de algunas variables de este tipo (opcional).
- Declaración (y opcionalmente definición) de variables globales (líneas 13-17): colocando:
    1) Modificadores (`unsigned` (sin signo, no admiten números negativos), `long` (aumenta el tamaño de la variable), `short` (disminuye el tamaño de la variable), etc.).
    2) El tipo de dato (`int`: entero de 32 bits, `float`: número en formato de punto flotante de simple precisión de acuerdo al estándar IEEE 754, `char`: caracter (entre `''`), `string`: cadena de texto según la implementación del archivo de encabezado `string.h`, etc.).
    3) Nombre de la variable (que contenga letras minúsculas o mayúsculas del alfabeto inglés, números o "_". No puede comenzar con un número).
    4) Contenido (opcional).

- Declaración de funciones (líneas 19-20): se realiza antes del programa principal para que el compilador puede utilizarlas en este. Los modificadores y el tipo de dato se colocan antes del nombre, los parámetros a pasar por referencia se indican colocando "`&`" antes del nombre del parámetro; los punteros (como los arreglos) siempre son pasados por referencia, y no se les debe colocar "`&`". La palabra clave `void` indica que la función no debe retornar nada.
- Programa principal (líneas 22-25): la función `main` es una función que es llamada automáticamente al ejecutarse el programa. Puede tomar parámetros, pero son opcionales. La línea `return 0` es el valor que devuelve la función (es utilizada en otras funciones, no solo en `main`).
- Definición de funciones (líneas 27-33): se realiza al final del archivo para claridad a la lectura. El nombre, tipo de dato y los parámetros formales deben coincidir, por lo general (hay un caso especial llamado sobrecarga de función (*function overload*) que permite tener múltiples funciones con el mismo nombre y tipo de dato, útil en determinadas situaciones)

Programación en C++:

C++ tiene algunas reglas de sintaxis a seguir siempre, como que "`{}`" son utilizados para agrupar bloques de código, y el uso de "`;`" para separar instrucciones.

C++ tiene además "buenas prácticas", como el escribir una instrucción por línea, la indentación/tabulación, la definición de funciones luego de `main`, el nombrado de las constantes en mayúsculas, el uso de comentarios (con "`//`" o "`/* Comentario */`") , etc.

C++ tiene múltiples operadores, tales como los aritméticos (`+`, `-`, `*`, `/`, `%`), los de asignación (`=`, `+=`, `-=`, etc.), los de comparación (`==`, `<`, `<=`, `>`, `>=`, `!=`), los lógicos (`&&`, `||`, `!`), los de incremento/decremento (`++`, `--`), los binarios (bitwise) (`&`, `|`, `^`, `~`, `<<`, `>>`) y otros, como los que permiten manipular punteros (`*`, `&`), el operador ternario (`?`), los que permiten acceder a variables u objetos dentro de registros y objetos (`.`, `->`), los que permiten entrada/salida de datos desde archivos (`<<`, `>>`);

C++ tiene múltiples estructuras de control:

```
1. if(condición){
2.         //Si condición es verdadera
3. } else if (condición2) {
4.         //Si condición es falsa y condición2 es verdadera
5. } else {
6.         //Si todas las condiciones anteriores son falsas
7. }
8.
9. switch(variable) {
10. case a: //Si variable == a
11.       break;
12. case b: //Si variable == b
13.       break;
14. case c: //Si variable == c
15.       break;
16. default://Si variable != a, variable != b y variable != c
17. }
18. /*Si no se coloca "break;", si se cumple un case,
19. se ejecutará todo el código de los siguientes
```

```
20. case hasta hallar un break;*/
21.
22. while(condición) {
23.       /*Código a ejecutar hasta que condición deje de ser
24.       verdadera. Puede nunca ejecutarse*/
25. }
26.
27. do {
28.       /*Código a ejecutar hasta que condición deje de ser
29.       verdadera. Siempre se ejecuta al menos 1 vez*/
30. } while (condición)
31.
32. for (definición de variable; condición; modificación de
   variable) {
33.       //Código a ejecutar siempre que se cumpla la condición
34. }
35. //Ejemplo de for
36. for (unsigned int a = 0; a < 10; a++) {
37.       cout<<a<<endl;
38. }
```

Actividades de práctica propuestas:

A programar se aprende por prueba y error hasta que se le pierde el miedo, así que a continuación sugiero algunas actividades y programas sencillos que podrías intentar: hacer.

1. Investigar sobre el archivo de encabezado `string.h` y realizar un programa que invierta el orden de los dígitos de una cadena. ¿Qué hace `c_str()`?
2. Investigar la función `getline()`, y realizar un programa que separe la entrada del usuario en palabras y las almacene en un arreglo.
3. ¿Cómo implementarías las estructuras de control vistas en pseudocódigo en C++? ¿Todas tienen alguna equivalente? ¿En qué se diferencian?
4. Investigar el tipo de dato `char`, y realizar un programa que imprima todos los caracteres en pantalla.
5. Investigar el tipo de dato `double`. ¿En qué se diferencia de `float`? ¿Qué modificadores pueden ser usados con él?
6. Investigar cómo convertir cadenas de caracteres a otros tipos de datos.
7. ¿Cuándo usamos los corchetes (`[]`)? ¿Cómo definimos un arreglo multidimensional?
8. Investigar cómo obtener números aleatorios.
9. Investigar qué son las listas enlazadas y los arreglos dinámicos, sus ventajas y desventajas, y qué archivos de encabezado son utilizados para implementarlos. ¿Qué hace el archivo de encabezado `matrix.h`?
10. Investigar sobre los archivos de encabezado `ifstream.h` y `ofstream.h`, y realizar un programa que lea lo que contiene un archivo y luego lo reemplace por la entrada del usuario.
11. ¿Los structs pueden contener funciones?
12. Investigar qué son las clases. ¿En qué se diferencian de los structs?
13. Investigar qué son los punteros (pointers).
14. Investigar qué es wxWidgets.
15. Realizar un programa con menú que permita manipular pilas y columnas, agregando y quitando elementos.
16. Aprovechando las funciones de la actividad 14., realizar un algoritmo que juegue a “Las Torres de Hanói”.