Resumen de Teoría - Programación – 2020

***aclaro que intenté parafrasear lo menos posible. TODO es de la teoria y sin inventar palabras, invito a hacer modificaciones y a hacer Pull Request, tmbn en la practica.**

1. **Etapas para la Resolución de Problemas:**
    1. **Definición del Problema:**
        1. Formulación correcta y completa.
    2. **Análisis del Problema *(comprende...)*:**
        1. **Datos:** información para resolver el problema.
        2. **Resultados:** lo que se desea obtener.
        3. **Condiciones:** relaciones que vinculan los campos anteriores llegando al planteo de una solución.
    3. **Programación *(obtención de la solución)*:**
        1. **Elección y creación del método:** se busca un método eficiente[1].
            1. **Diseño de la estrategia**:
                1. Responde: ¿Qué hacer? La misma es el plan de un método que resuelve el problema.
                2. Diseñar una estrategia consiste en descomponer[2] un problema en subproblemas.
            2. **Definición del algoritmo**:
                1. Responde: ¿Como hacerlo? Es un conjunto finito[3], definido[4], general[5] y ordenado de instrucciones.
        2. **Codificación:** se expresa el método en un lenguaje de programación que será compilado o interpretado.
        3. **Prueba**:
            1. Ejecución del método elegido, suministrando datos y obteniendo resultados.
            2. Analizar los resultados y realizar las correcciones.
        4. **Depuración:** o “debugging” significa eliminar los errores de un programa. Tipos:
            1. Errores en Tiempo de Compilación: en reglas que definen la estructura de las declaraciones y sentencias.
            2. Errores en Tiempo de Ejecución: las sentencias y declaraciones son válidas pero se dan al ejecutarlo. Ej: división por cero.
            3. Errores de Lógica: el programa retorna una salida inesperada cuando NO hay errores en compilación y NO hay errores en ejecución.
    4. **Documentación:** debe realizarse durante todo el proceso de resolución de un problema; tanto en el programa como en archivos o impresiones adicionales.

---

1 No se debe confindir con Eficacia, donde esta es la capacidad de lograr el efecto que se desea o se espera. La Eficiencia es la capacidad de resolver un problema con una solución óptima.
2 Paradigma denominado top-down, diseño descendente, método de refinamiento sucesivo o diseño compuesto.
3 Tiene fin.
4 El enunciado debe estar en forma precisa y clara. Mismos datos conducen a una misma salida.
5 La solución no debe ser particular. Sino aplicable a un tipo de problemas.

2. **Formalización:**
    1. Ejecutante: o Procesador es toda entidad capaz de comprender un enunciado y ejecutar su algoritmo.
    2. Ambiente: los recursos existentes durante la ejecución del algoritmo conforman el ambiente del problema.
    3. Acción: evento que modifica al ambiente. Puede ser:
        1. Primitiva: puede ser ejecutada sin información adicional.
        2. No Primitiva: puede ser descompuesta en acciones primitivas.
    4. Secuencia: es un conjunto de acciones que se ejecutan en el mismo orden en que aparecen en el algoritmo, una a continuación de la otra.
    5. Condición: afirmación lógica sobre el estado de un recurso en el ambiente. Puede tomar un valor verdadero o falso.
    6. Programa: algoritmo codificado empleando un lenguaje de programación interpretable por una computadora.
    7. Abstracción: acción de abstraer, del latín abstractio, es separar las propiedades de un objeto.
    8. Sentencia compuesta o Bloque: es considerado como una única sentencia donde es un conjunto de sentencias encerradas entre llaves “{ }”.
    9. Ambito de validez de una variable: es el bloque del programa en donde fue declarada.
    10. Declaración: se reserva una posición en memoria. Se necesita tipo e indetificador.
    11. Definición: asignar un valor almacenándolo en la variable declarada.
    12. Redondeo: transformar un número decimal, al número entero más próximo.
    13. Truncamiento: transformar un número decimal en un número entero, considerando la parte entera del mismo, sin importar los decimales que posea.
    14. Estructura de datos: es un conjunto de datos homogéneos que se tratan como una sola unidad y bajo un mismo nombre colectivo.
    15. Variables Locales: aquellas declaradas dentro de una función, se destruye cuando la ejecución de la misma termina.
    16. Variables Globales: son accesibles por cualquier función que aparezca a partir de su declaración y hasta el final del programa.

3. **Programación Modular:**
    1. Método de diseño que divide el problema en partes (módulos) diferenciadas para ser analizadas, resueltas.
        1. Un módulo corresponde a una función lógica bien definida.
        2. Un módulo debe ser pequeño y de poca complejidad.
        3. La salida debe ser exclusivamente de la entrada.
        4. Cada módulo tiene una única entrada y una única salida.
    2. **Objetivos**:
        1. Disminuir la complejidad.
        2. Aumentar la claridad.
        3. Aumentar la fiabilidad.
        4. Facilitar modificaciones y conexiones.
    3. **Alternativas a la programación modular.**
        1. Procedimentales.
        2. Imperativos y Declarativos.
        3. Orientados a Objetos.
        4. Orientados a Eventos.
        5. Programación Estructurada, hacen uso de estructuras de control[6].

4. **Algoritmos:**
    1. Deben ser separados por punto y coma;
    2. Encontramos:
        1. Constante: su valor *no puede alterarse* durante la ejecución del algoritmo, poseen un significado *propio y único*.
        2. Variable: elemento o lugar que *puede variar* durante la ejecución del algortimo. Un nombre representa una posición en memoria. Puede asignarse un único valor por vez. Reglas para su declaraión *(ver punto 8, identificadores)*:
            1. Utilizar letras, dígitos o guión bajo.
            2. Comenzar con una letra.
            3. No emplear palabras reservadas.
            4. Hacer distinción entre mayúsculas y minúsculas.
        3. Expresión: conjunto de operandos ligados por operadores, desbribiendo un cáclulo y presentando un único resultado.

5. **Código ejecutable:**
    1. Compilación: proceso en el que se traduce un código fuente llegando a una imagen ejecutable.
    2. Interpretación: este procesa instrucción por instrucción sin generar un código ejecutable completo. Cada vez que se ejecute un programa este será invocado.
    3. Ventajas/Desventajas

Ventajas

Compilación	**Interpretación**
Errores de sintaxis antes de la ejecución Velocidad de ejecución Protección del código.	Ejecución en una sola etapa Proceso interactivo de depuración

Desventajas

Compilación	**Interpretación**
Proceso en varias etapas y a menudo engorroso. (*) Depuración laboriosa	Errores de sintaxis detectados durante la ejecución. Baja velocidad de ejecución. Código abierto.

6 Secuenciales, Condicionales e Iterativas.

6. **Lenguajes de Programación:**
    1. Aquellos procesados por una computadora y posteriormente convertidos en programas.
    2. Encontramos:
        1. Lenguajes de Máquina: generan programas usando instrucciones que son procesadas directamente por el procesador de la computadora como dispositivo electrónico, procesando dos señales eléctricas *(encendido, apagado)* representadas en sistema binario *(0, 1)*. Poco legibles, dificultad al depurar y codificar, ejecutados con velocidad óptima y dependientes del procesador.
        2. Lenguajes de Bajo Nivel: lenguaje formado por sentencias nemotécnicas y agradable a la interpretación humana. Continua siendo una tarea compleja, uso limitado al control de dispositivos electrónicos, requierer traducción a código máquina.
        3. Lenguajes de Alto Nivel: permitió abordar nuevos paradigmas y modelos. Son portables[7], poseen palabras reservadas y símbolos. Poseen sofisticados entornos de desarrollo para trabajar. Requieren muchos recursos del ordenador y tiempo de ejecución.

7. **Historia de C++**

En 1967 Martin Richards creó un lenguaje de programación BCPL para escribir sistemas operativos y compiladores y Ken Thompson creó el lenguaje B basándose en el BCPL. Estos 2 lenguajes eran muy rústicos y dejaban muchas tareas al programador. En 1972 Denis Ritchie -también de Laboratorios Bell—escribe un lenguaje basado en BCPL y B con varias mejoras que contribuyeron a su posterior popularidad; lo llamó C. La eficiencia del C en términos de ejecución y administración de recursos lo hizo el preferido de las empresas de software que diseñaban sistemas operativos y compiladores. Una de sus principales características era su independencia del hardware, lo cual permitía inicialmente correr programas C en cualquier plataforma con mínimas modificaciones. Pero las empresas de software comenzaron a diseñar versiones de C particulares que le quitaban portabilidad a los programas. Por eso, en 1983 el American National Standards Institute (ANSI) creó un comité técnico para su estandarización. La versión aprobada junto a la Organización Internacional de Estandarizacón (ISO) vio la luz en 1990 y se la conoce como ANSI C. En 1980 Bjarne Stroustrup de Laboratorios Bell, comenzó a experimentar con versiones mejoradas de C (C con clases) con la única finalidad de escribir programas de simulación orientados a eventos. Stroustrup llamó a su nuevo lenguaje C++. Este compilador fue creciendo con ren23ovadas características que lo hicieron muy original, manteniendo la compatibilidad con su antecesor C. C++ incorpora clases y funciones virtuales basándose en SIMULA67, tipos genéricos y excepciones de ADA, la posibilidad de declarar variables en cualquier lugar de ALGOL68, así como otras características originales que no existían antes: herencia múltiple, espacios con nombre, funciones virtuales puras, etc. Alex Stepanov y Andrew Koenig idearon la biblioteca de plantillas standard (STL), la cual le da a C++ una potencia única entre los lenguajes de alto nivel. Debido a la enorme difusión del C++, y –nuevamente- a las diferentes versiones que fueron apareciendo, las organizaciones ANSI e ISO se reunieron en 1990 para definir el Standard de este lenguaje, el cual fue aprobado en 1998. Hoy día, C++ posee una notable inserción en el mundo de las computadoras y es uno de los lenguajes clásicos de programación: tanto de sistemas operativos y compiladores (software de base) como de aplicaciones.

7 Independientes del hardware.

8. **Estructura de un programa C++**
    1. Include y Macros *(directivas del procesador):*
        1. Son encabezados, instruyen al procesador acerca de cómo deben interconectarse con los componentes de la biblioteca y los componentes escritos por el usuario.
    2. La función main()
        1. Las instrucciones de plantean dentro de sí, delimitando el bloque con {}.
    3. **Elementos o Tokens**:
        1. Idetificadores:
            1. Son nombres que representan variables, constantes y tipos de datos.
        2. Palabras Reservadas:
            1. Son identificadores predefinidos y estándares del lenguaje.

Asm	*Continue*	*float*	*new*	*signed*	*try*
Auto	*Default*	*for*	*operator*	*sizeof*	*typedef*
Break	*Delete*	*friend*	*private*	*stact*	*union*
Case	*Do*	*goto*	*protected*	*struct*	*unsigned*
Match	*double*	*if*	*public*	*switch*	*virtual*
Char	*Else*	*inline*	*register*	*template*	*void*
Class	*Enum*	*int*	*return*	*this*	*volatile*
Const	*Extern*	*long*	*short*	*trhow*	*while*

        3. Operadores:
            1. Elementos del léxico de C++ que permiten conectar operandos.
            2. Reglas:
                1. Operadores del mismo grupo tienen igual prioridad y asociatividad.
                2. Si dos operadores se aplican al de un operando, se resuelve primero el de mayor prioridad.
                3. La precedencia puede alterarse con paréntesis.
            3. Econtramos:
                1. Aritméticos.
                2. Asignación.
                3. Relativos de asignación.
                4. Relacionales.
                5. Lógicos.
                6. Otros: *(ver precedencia de operadores al final)*
        4. Separadores.
            1. Espacios en blanco.
            2. Avances de línea.
            3. Retornos de carro.
            4. Tabulaciones.
        5. Literales o Constantes.
            1. *Constantes Literales:* de notación y sintaxis determinada
                1. **Entera Decimal**. 10
                2. **Entera Octal**. 012 comienza con cero.
                3. **Entera Hexadecimal**. 0XFA4 comienza con 0X.
                4. **Real o Punto Flotante**. Emplea punto decimal o notacion exponencial.
                5. **Char.** ‘\n’ . Lleva comillas simples.
                6. **Cadenas**. “Hello World!”. Lleva comillas dobles.
            2. *Constantes Definidas:* definidas en preprocesador. **#define** <nombre> <valor>;
            3. *Constantes Declaradas:* definidas con **const** <tipoDato> <nombre> = <valor>
            4. *Constantes Enumeradas:* definidas con **enum** <nombre_lista> = {constantes}

9. **Tipos de datos** *(ver tipos al final)*
    1. Enteros **(int)**.
    2. Reales **(float, double)**.
    3. Caracter **(char)**: de tipo alfanumérico comprendido en el código ASCII.
    4. Lógico **(bool)**.
    5. Nulo **(void)**.
    6. Cadena de caracteres **(string).**

10. Flujos de Entrada y Salida

1. Es una secuencia de caracteres que se envían (fluyen) desde o hacia un dispositivo.
2. Los flujos **cin**, **cout**, **cerr** y **clog**, son clases predefinidas de C++, las cuales se hallan en el archivo **iostream.h**
3. El flujo de salida **cout** requiere del **operador de inserción o salida** << para enviar la información a la pantalla.
4. El comando de entrada de flujo **cin** pero empleando los operadores de extracción o entrada >>.
5. También se dispone de otros dos flujos **cerr** y **clog** para manejo de errores.
6. Caracteres especiales o Manipuladores de I/O:

Secuencia de escape	Caracter	Efecto
\a	BEL	Campana o pitido de alerta
\b	BS	Retroceso (Backspace)
\f	FF	Avance de página
\n	LF	Avance de línea
\r	CR	Retorno de carro
\t	HT	Tab horizontal
\v	VT	Tab Vertical
\\	\	Barra invertida (backslash)
\'	'	Apóstrofo
\"	"	Doble comilla
\?	?	Marca de interrogación

Manipulador	Efecto	Ejemplo
endl	Avance de línea (“\n”)	cout << “a=”<< a << endl << “b=”<< b
Hex	Exhibirá el siguiente valor en formato hexadecimal	cout << hex << 1000
Dec	Exhibirá el siguiente valor en formato decimal	cout << dec << x
Oct	Exhibirá el siguiente valor en formato octal	cout << oct << 105
setbase( )	Establece la base para mostrar el siguiente valor	cout << setbase(8) << dato
setw( )	Determina ancho de campo para mostrar la información	cout << “Resultado:”<< setw(20)<< r
setfill( )	Establece un caracter de relleno	cout << setfill(‘.’)
setprecision( )	Determina el número de dígitos de la fracción decimal en la presentación de números reales	cout << seprecision(4) << 10.0/3.0

11. Programación Estructurada en C++

1. Paradigma basado en tres estructuras de control. Tienen un **único punto de entrada y un único punto de salida**, el programa es más **fácil de entender** y **permite detectar los errores de lógica rápidamente**:
    1. Secuencia. es un conjunto de acciones que se ejecutan en el mismo orden en que aparecen en el algoritmo, una a continuación de la otra. Delimitados por ;
    2. Selección. Conformada por:
        1. **Condicional, Estructura de Decisión o if:** se evalúa la expresión lógica planteada a continuación del if y si es distinta de cero (verdadero) se realizan las acciones indicadas; si la expresión lógica es cero (falso), se realizan las acciones a continuación del else. Es posible anidarlas, incluir condicionales dentro de sí.
        2. **Selección Múltiple o switch:** sentencia que permite efectuar una selección entre múltiples opciones en base al valor de una variable de control[8] que permite administrar la estructura Iteración. La acción propuesta a continuación de default se ejecutará si el valor de la variable de control no coincide con ninguno de los valores propuestos.
    3. Iteración. Permiten anidamiento.Conformada por:
        1. **while:** las acciones abarcadas por esta estructura se ejecutan repetidamente mientras la expresión lógica arroje un valor distinto de cero (verdadero). El bucle se ejecuta de 0 a n veces.
        2. **do while:** las acciones abarcadas por esta estructura se ejecutan repetidamente hasta que la expresión lógica arroje el resultado cero (falso). Se ejecuta de 1 a n veces.
        3. **for:** las acciones abarcadas por esta estructura se ejecutan repetidamente hasta que la exp2 arroje cero (falso); exp1 es una expresión de inicialización y se ejecutan una única vez; exp3 se realiza al final del grupo de acciones y generalmente se emplea para incrementar la variable que controla la estructura.

12. Funciones y sentencias útiles:

1. **break.** Abandona la estructura iterativa inmediatamente.
2. **continue.** Salta al final de la estructura repetitiva para continuar con su ejecución.
3. **exit().** Permite interrumpir un programa.

13. Cadenas de Caracteres:

1. Es posible usarlas mediante:
    1. Una cadena como tipo predefinido del lenguaje **(string)**.
    2. Usar una array cuya estructura sea de tipo char.
2. Si no hay asignación, dicha variable tendrá por defecto la cadena vacía ("").
3. Para usar el tipo de dato string se debe implementar **#include <string>**
4. Para realizar lecturas tomando espacios en blanco emplear **getline(cin, cadena);**
5. Operaciones con string:
    1. Asignación (**=**).
    2. Concatenación[9] (**+**).
    3. Generación de subcadenas (**cadena.substr(posicion, largo)**).
    4. Insertar cadena (**cadena.insert(posicion, "texto"**).
    5. Reemplazar en una cadena (**cadena.replace(inicio, fin, "texto")**).
    6. Longitud de una cadena (**cadena.size()**).
    7. Acceso a un caracter (**cadena[posicion]**).
    8. Comparación (**==, !=, <, <=, >, >=**).
    9. Búsqueda (**cadena.find("texto_a_buscar")**).
    10. Convertir número en cadena (**to_string(variable)**).
    11. Convertir cadena a número:
        1. entero, **stoi()**
        2. entero largo, **stol()**
        3. real, **stof()**

8 Solo puede ser int o char.
9 Almenos la primera o segunda cadena debe estar declarada string.

14. Array o Arreglo

1. Se define arreglo como la estructura de datos formada por una secuencia de elementos homogéneos[10].Cada elemento tiene una posición relativa[11] dentro de la secuencia.
2. Todo arreglo tiene asociado:
   1. **Identificador[12].**
   2. **Tipo.**
   3. **Dimensión[13].**
   4. **Indice[14].**
3. Los arreglos son estructuras estáticas que requieren establecer la cantidad de elementos que pueden almacenar.
4. La estructura completa ocupa un segmento de memoria único y sus elementos se almacenan en forma contigua.
5. C++ admite operar fuera del rango preestablecido por la dimensión, pero con resultados impredecibles (L>D)[15], por eso si se desconoce la cantidad de elementos se debe **sobredimensionar**.
6. Operaciones con arreglos:
   1. Asignación.
   2. Lectura.
   3. Escritura.
   4. En expresiones.
7. Tipos de arreglos:
   1. **Unidimensional.** Cuando la referencia a uno de sus elementos se realizatravés de un único valor llamado índice.
   2. **Bidimensional**[16]. cuando sus elementos están dispuestos en filas y columnas y la referencia a cada uno de sus elementos se realiza a través de 2 (dos) índices.
   3. **Multidimensionales**. se requerirán más de 2 índices para posicionar un elemento en la colección.

15. Funciones[17]

1. es un conjunto de acciones, diseñado generalmente en forma separada y cuyo objetivo es resolver una parte del problema.
2. Pueden ser invocados desde diferentes puntos de un mismo programa, desde otros subprogramas, e incluso desde otros programas por medio de Librerías.
3. Ventajas:
   1. Reducir la complejidad del programa y lograr mayor modularidad.
   2. Permitir y facilitar el trabajo en equipo[18].
   3. Facilitar la prueba de un programa[19].
   4. Optimizar el uso y administración de memoria.
   5. Crear librerías de subprogramas para su posterior reutilización en otros programas.
4. Cuando emplearlos?
   1. Existe un conjunto de operaciones que se utilizan más de una vez.
   2. Existe un conjunto de operaciones útiles que pueden ser utilizadas por otros programas.
   3. Se desea agrupar procesos para lograr una mayor claridad en el código del programa.
   4. Se pretende crear bibliotecas que permitan lograr mayor productividad en el desarrollo de futuros programas.

---

10 de igual tipo.
11 que puede ser establecida por uno o más índices.
12 El nombre del arreglo identifica a una variable que contiene la dirección de memoria donde comienza el bloque donde se aloja el arreglo.
13 valor o valores numéricos que indican la cantidad máxima de elementos que componen el arreglo.
14 variable, constante o expresión numérica que hace referencia a una posición del arreglo. Cualquier expresión que arroje un número entero positivo dentro del rango establecido (dimensión).
15 Longitud del arreglo mayor a la dimensión.
16 También conocido como tabla o matriz.
17 También conocidos como subprogramas, procedimientos, subrutinas.
18 Cada diseñador puede atacar diferentes módulos o subprogramas.
19 cada subprograma puede ser probado previamente y en forma independiente.

5. Cómo funcionan?
    1. Se llamará programa principal (main), que incluirá entre sus acciones una sentencia especial que permitirá llamar al subprograma.
    2. Al encontrar la llamada al subprograma, se transfiere el control de ejecución a éste y comienzan a ejecutarse las acciones previstas en él.
    3. Al finalizar la ejecución del subprograma y obtenidos los resultados planeados, el control retorna al programa que produjo la llamada, y continúa la ejecución del programa principal.
    4. Puede existir intercambio de información o no[20] entre el programa o módulo que llama a la función. La función devuelve un solo valor (o ninguno) a través de la sentencia **return**.
6. Intercambio de información:
    1. Una función envía información por medio de **parámetros** o **argumentos** donde:
        1. **Parámetros de formales o de diseño**: son empleados en el prototipo y definición de la función.
        2. **Parámetros actuales o de llamada**: son empleados cuando se invoca a la función.
        3. Los pasajes **de** infomación pueden ser mediante:
            1. **Pasaje por valor**: los parámetros formales se asignan en posición y tipo con los valores de los parámetros de llamada.
            2. **Pasaje por referencia**: se utiliza un parámetro formal como una referencia a la posición de memoria de un parámetro de llamada. Usando el operador referencia **&** se define un alias para una variable y se emplean los parámetros de llamada para obtener resultados.
7. Sobrecarga de funciones
    1. C++ admite que dos funciones diferentes puedan tener el mismo nombre pero para ser distinguidas es necesario que difieran sus parámetros donde se pueden hacer dos funciones con el mismo nombre pero con comportamientos totalmente diferentes.
8. Recursividad
    1. Es una técnica que permite definir una función en términos de sí misma.
    2. main() no puede llamarse a sí misma.
    3. Los compiladores utilizan una pila (stack) de memoria temporal, la cual puede causar una interrupción del programa si se sobrepasa su capacidad (stack overflow).
    4. Condiciones necesarias:
        1. Realizar llamadas a sí misma para efectuar versiones reducidas de la misma tarea.
        2. Incluir uno o más casos donde la función realice su tarea sin emplear una llamada recursiva.

20 Una función puede no devolver valores si se la define de tipo nulo (void). La función no incluye la sentencia return.

16. Archivos

1. Es un conjunto de registros con una estructura común y organizados para un propósito particular.
2. Un conjunto de archivos relacionados a su vez, constituye una BASE DE DATOS.
3. Organización de los datos de un archivo:
    1. **Secuencial**: cada registro, salvo el primero, tiene otro que le antecede y todos, excepto el último, uno que le sigue. El orden en que aparecen, es el orden en que han sido escritos. El orden físico coincide con el orden lógico. Para acceder al registro n es necesario leer los n-1 registros anteriores.
    2. **Directa**: el orden físico de los registros no corresponde con el orden lógico. La información se accede aleatoriamente mediante su posición, donde puede accederse directamente y no de forma secuencial.
    3. **Secuencial con índice**: hay una tabla con índices, que permite indicar la localización de registros almacenados secuencialmente.
4. Acciones para manipular archivos:
    1. **Crear una variable lógica que maneje el archivo físico en el disco**.
        1. El archivo[21], tendrá un nombre, una extensión, y estará almacenado en una ruta.
        2. En el programa se da un nombre[22] que referencia al archivo.
    2. **Abrir el archivo**.
        1. Se especifica si se abrirá para entrada o salida de datos.
            1. Entrada de datos: recuperar desde memoria externa la información, mediante la lectura.
            2. Salida de información: escribir en memoria externa los datos que deseamos almacenar.
    3. **Leer y Escribir en el archivo**.
    4. **Cierre del archivo**.
5. Clasificación:
    1. Archivos de Texto.
    2. Archivos Binarios.
6. Modos de apertura:
    1. El flujo de entrada por defecto es el teclado y el flujo de salida por defecto es la pantalla
    2. Entrada. **ostream**
    3. Salida. **Ifstream**

- **`ios::app`** Operaciones de añadidura.
- **`ios::ate`** Coloca el apuntador del archivo al final del mismo.
- **`ios::in`** Operaciones de lectura. Esta es la opción por defecto para objetos de la clase ifstream.
- **`ios::out`** Operaciones de escritura. Esta es la opción por defecto para objetos de la clase ofstream.
- **`ios::nocreate`** Si el archivo no existe se suspende la operación.
- **`ios::noreplace`** Crea un archivo, si existe uno con el mismo nombre la operación se suspende.
- **`ios::trunc`** Crea un archivo, si existe uno con el mismo nombre lo borra.
- **`ios::binary`** Operaciones binarias.

21 Este archivo junto a sus atributos será la variable FÍSICA.
22 Este nombre corresponde a la variable LÓGICA.

7. Buenas prácticas en el manejo de archivos:
    1. Incluir la biblioteca **<fstream>**
    2. Usar el espacio de nombres estándar **using namespace std;**
    3. Declarar las variables lógicas.
    4. Abrir el flujo de datos.
    5. Comprobar errores en la apertura.
    6. Realizar la transferencia de la información.
    7. Comprobar si al transferencia se realizó correctamente.
    8. Cerrar el flujo.

17. Struct

1. Un struct en C++ es una colección de componentes, los cuales pueden ser de diferente tipo. Cada componente o miembro debe declararse individualmente.
2. La palabra clave **struct** introduce la definición de la estructura con el identificador seleccionado.
3. Los datos declarados dentro de las llaves, son los **miembros**.
4. La definición de un struct no reserva espacio en memoria; crea un nuevo tipo de datos utilizado para declarar variables de estructura.
5. Para asignar un nombre correspondiente a una definición de tipo en C++ deberá emplearse la palabra reservada **typedef**.

18. RESPUESTAS A PREGUNTAS DE PARCIALES:
    1. Un switch se puede remplazar por un conjunto de ifs y un conjunto de ifs se pueden remplazar por un switch. FALSO
        1. Un switch podes reemplazarlo por cadenas de ifs pero la cadena de ifs no se puede reemplazar por switch ya que este solo trabaja con numeros enteros y positivos, y char; la condicion de if puede ser cualquier tipo de dato.
    2. La búsqueda binaria funciona tanto con arreglos ordenados como con arreglos desordenados. FALSO
        1. Solo con arreglos ordenados.
    3. Para que una función sea recursiva y esté bien planteada tiene que llamarse a si misma y existir al menos 1 caso en que no se autoinvoque. VERDADERO
    4. Una salida por pantalla (cout) con una expresión que contiene una variable preincrementada es igual a una salida por pantalla con una expresión que contiene una variable posincrementeda. FALSO
        1. Los valores mostrados son diferentes. En uno de los casos se incrementa antes de mostrar su valor.
    5. Una constante es una variable que no cambia. FALSO
        1. Es aquella en la que su valor *no puede alterarse* durante la ejecución del algoritmo, poseen un significado *propio y único.*
    6. Un arreglo puede dimensionarse con constantes y variables enteras. FALSO
        1. Sólo con constantes.
    7. ¿Dónde está el pasaje por referencia o la dirección de memoria del inicio del arreglo?
        1. En el nombre del arreglo.

Prioridad y categoría	Operador	Función o significado	Asociati-vidad
1. (Prioridad más alta)	( ) [ ] → :: .	Llamada a función Subíndice de arreglos Selector indirecto de miembro Selector de ámbito de resolución Selector directo de miembro	I-D
2. Unarios	! ~ + - ++ -- & * sizeof new delete	Negación lógica Complemento a uno (bits) Más (unario) Menos (unario) Incremento Decremento Dirección Indirección Tamaño del operando en bytes Alocación dinámica en memoria Eliminación dinámica	D-I
3.Acceso a miembros	* →	Lee o modifica el valor apuntado Accede a un miembro de un objeto apuntado	I-D
4.Multiplicativos	* / %	Multiplicación División entera o flotante Resto de la división entera (módulo)	I-D
5.Aditivos	+ -	Más binario Menos binario	I-D
6.Desplazamiento	>> <<	Desplazamiento a la derecha Desplazamiento a la Izquierda	I-D
7.Relacional	< <=	Menor que Menor o igual que	I-D
	> >=	Mayor que Mayor o igual que	
8. Igualdad	== !=	Igual que Distinto que	I-D
9.	&	And (manipulación de bits)	I-D
10.	^	Xor (manipulación de bits)	I-D
11.	|	Or (manipulación de bits)	I-D
12.	&&	Conjunción lógica and	I-D
13.	||	Disyunción lógica or	I-D
14.Condicional	?:	a ? x : y (significa: if a then b, else c)	D-I
15.Asignación	= *= /= %= += -= &= ^= | = <<= >>=		D-I
16. Coma (prioridad más baja)	,	Evaluación múltiple	

Enteros

Tipo	Rango	Tamaño (bytes)
Char	--128 .. 127	1
Unsigned char	0 .. 255	1
Short	-32768 .. 32767	2
Unsigned short	0.. 65535	2
Int	-2.147.483.648 .. 2.147.483.647	4
Unsigned int	0.. 4.294.967.295	4
Long	-2.147.483.648 .. 2.147.483.647	4
Unsigned long	0.. 4.294.967.295	4
Float	3.4 x 10-38 .. 3.4 x 1038	4
Double	1.7 x 10 -308.. 1.7 x 10 308	8
long double	3.4 x 10 -4932 .. 3.4 x 10 4932	10

Lógico

Tipo	Rango	Tamaño (bytes)
Bool	false,true	1

Caracter

Tipo	Rango	Tamaño (bytes)
Char	-128 .. 127	1
unsigned char	0 .. 255	1