Convenios De Codificación

JOINT STRIKE FIGHTER

AIR VEHICLE

C++ CODING STANDARDS

Computación tolerante a fallas
21 de Octubre de 2020

Integrantes:

Aceves Saabedra Saúl - 214397353
Cortes Rodarte Luis Eligio - 217759329
López López Oscar Ricardo - 217759132
Márquez Rodríguez Isaac Andrés - 213742928
Vara Pérez Carlos Adonis - 216787671
Verdiales Valle Leonel - 216788112

Regla 1 - Tamaño de función / método
Cualquier función (o método) no **debe** tener más de 200 líneas de código fuente lógico.

Regla 2 - Código auto-modificable
No **debe** de haber ningún código auto-modificable para evitar errores de función, lectura, etc.

Regla 3 - Complejidad ciclomática
Todas las funciones **deben** de tener una complejidad ciclomática de 20 o menos.

Regla 4 - Romper regla "debería"
Para romper una regla "**debería**" debes tener aprobación del jefe de ingeniería de software.

Regla 5 - Romper regla "debe"
Para romper una regla "**debe**" debes tener aprobación del jefe de ingeniería de software y del product manager.

Regla 6 - Variaciones de "debe"
Cada variación de la palabra **"debe"** debe ser documentada en un documento.

Regla 7 - Excepciones
Si una regla tiene excepciones, no necesita aprobación para su variación.

Regla 8 - Standard de código
Todo el código **debe** cumplir con el standard de C++ ISO/IEC 14882:2002(E).

Regla 9 - Caracteres
Solo los caracteres especificados en el código base de C++ **deberán** ser usados.

Regla 10 - Valores de tipo de caracteres
Los valores de tipo de caracteres **deberán** ser restringidos a un subset de ISO 10646-1 definido y documentado.

Regla 11 - Trigrafos
Los trigrafos **no deben** de ser utilizados.

Regla 12 - Digrafos
Los siguientes digrafos **no deben** de ser utilizados.

Alternative	Primary	alternative	Primary
<%	{	:>	]
%>	}	%:	#
<:	[	%:%:	##

Regla 13 - Caracteres multi-bytes
Los caracteres multi-bytes y wide strings **no deben** de ser utilizados.

Regla 14 - Sufijos literales
Los sufijos literales **deben** de usar letras mayúsculas en lugar de minúsculas para evitar ser difíciles de leer.

Regla 15 - Provisiones
Las provisiones **deberían** ser hechas para chequeo de run-time.

Regla 16 - Librerías seguras
En código con seguridad crítica solo se **deben** utilizar librerías DO-178B nivel A o SEAL 1 C/C++.

Regla 17 - Indicador de error
El mensaje indicador de error *errno* **no debe** ser utilizado. Con excepción si se tiene documentado todos los posibles números de error presentables por el sistema.

Regla 18 - Macro offsetof
El macro *offsetof* de la librería <stddef.h> **no debe** de ser utilizado.

Regla 19 - Locale.h
La librería <locale.h> y la función *setlocale* **no deben** de ser utilizadas.

Regla 20 - Setjmp y Longjmp
El macro *setjmp* y la función *longjmp* **no deben** de ser utilizadas.

Regla 21 - Signal.h
Las utilidades de manejo de señales de <signal.h> **no deben** de ser utilizadas.

Regla 22 - Stdio.h
La librería de input/output <stdio.h> **no debe** de ser utilizada.

Regla 23 - Stdlib.h - atof, atoi, atol
Las funciones *atof, atoi* y *atol* de la librería <stdlib.h> **no deben** de ser utilizadas.

Regla 24 - Stdlib.h - abort, exit, getenv
Las funciones *abort, exit* y *getenv* de la librería <stdlib.h> **no deben** de ser utilizadas.

Regla 25 - Time.h
Las funciones de manejo de tiempo de la librería <time.h> **no deben** de ser utilizadas.

Regla 26 - Directivos pre-proceso
Solo los siguientes directivos pre-proceso pueden ser utilizados: *#ifndef, #define, #endif, #include.*

Regla 27 - Inclusiones múltiples de headers
#ifndef, #define y #endif serán utilizadas para prevenir múltiples inclusiones del mismo archivo header. Otras técnicas **no deben** de ser utilizadas.

Regla 28 - Ifndef, Endif
#ifndef y #endif solo **deberían** ser utilizadas como lo muestra en la **regla 27**.

Regla 29 - Define Macros
La directiva pre-proceso #define **no debe** de ser utilizada para crear macros. Se **debe** utilizar funciones inline.

Regla 30 - Define valores constantes
La directiva pre-proceso #define **no debe** de ser utilizada para definir valores constantes. Se **debe** de utilizar la palabra clave *const* al declarar un valor constante.

Regla 31 - Define archivos header
La directiva pre-proceso #define **sólo debe** utilizarse como lo muestra en la **regla 27**.

Regla 32 - Include
La directiva pre-proceso #include **solo debe** utilizarse para incluir archivos header (.h).

Regla 33 - Nomenclatura archivos header
La directiva #include **debe** de utilizar la nomenclatura <archivo.h> para incluir archivos header.

Regla 34 - Declaraciones en header
Los archivos de header **deberían contener** solo declaraciones lógicamente relacionadas.

Regla 35 - Inclusiones múltiples en header
Los archivos de header **deben** de utilizar un mecanismo que prevenga múltiples inclusiones de sí mismo.

Regla 36 - Dependencias de compilación
Las dependencias de compilación **deben** ser minimizadas siempre que sea posible.

Regla 37 - Archivos asociados
Los archivos header **deben** de tener asociados solamente a los archivos que requieran para la compilación, el resto se deben asociar al archivo .cpp.

Regla 38 - Clases por puntero o referencia
Declaraciones de clases que solo son accedidas por punteros (*) o referencias (&) **deberían** de ser provistas por archivos header que solo contengan declaraciones futuras.

Regla 39 - Definiciones en header
Los archivos header **no deben** de tener definiciones de variables no constantes o de funciones.

Regla 40 - Archivos de implementación
Cada archivo de implementación **deberá** incluir el archivo header que defina las funciones, tipos y plantillas utilizadas.

Regla 41 - Tamaño de líneas
Las líneas de código fuente **deben** mantenerse a un tamaño de 120 caracteres o menos.

Regla 42 - Expresión - Declaración
Cada expresión-declaración **deben** ser puestas en líneas separadas.

Regla 43 - Tabular
Las tabulaciones **deberían** ser evitadas para mantener continuidad entre editores.

Regla 44 - Indentación
Todas las indentaciones **deben** de ser de al menos 2 espacios y ser consistentes en cada archivo de código.

Regla 45 - Identificadores
Todas las palabras en identificadores **deben** de ser separadas por el caracter '_'.

Regla 46 - Tamaño de identificadores
Los identificadores especificados por el usuario (interno o externo) **no deben** de extenderse más de 64 caracteres.

Regla 47 - Inicio de identificador
Los identificadores no comenzarán con caracter de guión bajo (_).

Regla 48 - Diferencias entre identificadores
Las diferencias entre identificadores **deben** de ser mayores a solamente: cambios en minúscula/mayúscula, uso o falta de guión bajo, cambio de caracter/es por uno visiblemente parecido.

Regla 49 - Acrónimos en identificadores
Todos los acrónimos en identificadores **deben** de ser ingresados en mayúsculas.

Regla 50 - Inicio de nomenclatura
La primera palabra del nombre de una clase, estructura, namespace, enumeración o tipo creado con *typedef* **debe** de iniciar con mayúscula, todas las otras letras deben de ser minúsculas.

Regla 51 - Nombres de funciones y variables
Todas las letras incluidas en los nombres de funciones y variables **deben** de ser minúsculas.

Regla 52 - Identificadores para constantes y enumeraciones
Todos los identificadores para valores constantes y enumeraciones **deben** de ser minúsculas.

Regla 53 - Extensión de header
Todos los archivos de header **deben** de utilizar la extensión ".h".

Regla 53.1 - Caracteres prohibidos en header
Los siguientes caracteres **no deben** de aparecer en el nombre de archivo de headers: ‘, \, /*, //, o ".

Regla 54 - Extensión de archivos de implementación
Todos los archivos de implementación **deben** utilizar la extensión “.cpp”.

Regla 55 - Nombre de archivo header
El nombre del archivo header **debería** reflejar la entidad lógica para la cual provee declaraciones.

Regla 56 - Nombre de archivo de implementación
El nombre del archivo de implementación **debería** reflejar la entidad lógica para la cual provee definiciones.

Regla 57 - Secciones públicas, protegidas y privadas
Las secciones públicas, protegidas y privadas de las clases **deben** de ser declaradas en ese orden.

Regla 58 - Funciones con más de 2 parámetros
Al declarar y definir funciones con más de dos parámetros, el primer parámetro después de abrir paréntesis **debe** ser escrito en la misma línea, cada parámetro adicional **debe** ser escrito en diferente línea.

Regla 59 - Cuerpos de declaraciones
Los cuerpos de las declaraciones de *if, else if, else, while, do … while o for* **deben** siempre estar entre llaves, incluso si el cuerpo está vacío.

Regla 60 - Llaves
Las llaves (“{ }”) que encapsulan bloques de código **deben** de ser puestas en la misma columna en líneas separadas exactamente antes y después del bloque que limitan.

Regla 61 - Comentarios en llaves
Las llaves (“{ }”) que encapsulan bloques de código **no deben** de tener algo más que comentarios en la misma línea.

Regla 62 - Operador de desreferencia y direccionamiento
Los operadores de desreferencia (“ * “) y direccionamiento de operador (“ & “) **deben** estar directamente conectados con el especificador de tipo.

Regla 63 - Espacios
Los espacios **no deben** ser utilizados alrededor de “ . “ o “ -> “, ni entre operadores unitarios.

Regla 64 - Interface de clase
La interface de clase **debería** ser completa y mínima.

Regla 65 - Estructuras
Una estructura **debería** ser usada para modelar una entidad que no requiera una invariante.

Regla 66 - Clases
Una clase **debería** ser usada para modelar una entidad que mantenga una invariante.

Regla 67 - Datos públicos y protegidos
Los datos públicos y protegidos **deberían** ser utilizados solamente en estructuras, no clases.

Regla 68 - Funciones innecesarias
Las funciones innecesarias generadas implícitamente **deben** ser rechazadas explícitamente.

Regla 69 - Funciones constantes
Las funciones que no afectan el estado de un objeto **deben** ser declaradas como constantes.

Regla 70 - Amigos de clase
Una clase **debe** de tener amigos solo cuando una función u objeto requiere acceder a elementos privados de la clase, pero no es posible ser miembro de la clase por razones lógicas o de eficiencia.

Regla 70.1 - Uso de objeto
Un objeto **no debe** de ser utilizado impropiamente antes de que su tiempo de vida comience o después de que termine.

Regla 71 - Llamadas a operaciones de objeto
Las llamadas a operaciones externamente visibles de un objeto, que sea diferente a su constructor, **no deben** de ser permitidas hasta que el objeto se haya iniciado completamente.

Regla 71.1 - Funciones virtuales
Las funciones virtuales de una clase **no deben** ser invocadas desde su destructor o alguno de sus constructores.

Regla 72 - Invariante de clase
La invariante de clase **debería** ser: Una parte de la postcondición de cada constructor de clase, una parte de la precondición de cada destructor de clase y una parte de la precondición y postcondición de cada operación accesible públicamente.

Regla 73 - Constructores
No deben definir constructores predeterminados innecesarios.

Regla 74 - Inicialización de miembros de clase no estáticos
La inicialización de miembros de clase no estáticos **se realizará** mediante la inicialización de miembros de una lista en lugar de a través de la asignación en el cuerpo de un constructor.

Regla 75 - Listas de inicialización
Los miembros de la lista de inicialización **deben** enumerarse en el orden en que se declaran en la clase.

Regla 76 - Clases con punteros
La copia de un constructor y un operador de asignación **deberán** ser declaradas para las clases que contienen punteros a elementos de datos o destructores no triviales.

Regla 77 - Copia de constructor
La copia de un constructor **deberá** copiar todos los miembros de datos y bases que afectan el invariante de la clase.

Regla 77.1 - Función miembro
La definición de una función miembro **no debe** contener argumentos predeterminados que produzcan una firma idéntica a la copia del constructor declarado implícitamente para la correspondiente estructura de clase.

Regla 78 - Función virtual
Todas las clases base con una función virtual **deben** tener definido un destructor virtual.

Regla 79 - Destructor de clase
Todos los recursos adquiridos por una clase **serán** liberados por el destructor de la clase.

Regla 80 - Operadores por defecto
La copia y la asignación de los operadores por defecto **se pueden** usar para clases cuando los operadores ofrecen una semántica razonable.

Regla 81 - Autoasignación
El operador de asignación **deberá** manejar la autoasignación correctamente.

Regla 82 - Operador de asignación
Un operador de asignación **debe** devolver una referencia a *this.

Regla 83 - Miembros de datos y bases
Un operador de asignación **deberá** asignar todos los miembros de datos y bases que afectan el invariante de la clase.

Regla 84 - Sobrecarga de operadores
La sobrecarga de operadores **se utilizará** con moderación y de manera convencional.

Regla 85 - Operadores opuestos
Cuando dos operadores son opuestos, ambos **se pueden** definir y uno **será** definido en términos del otro.

Regla 86 - Representación de conceptos
Se deben utilizar tipos concretos para representar conceptos simples e independientes.

Regla 87 - Jerarquía
Las jerarquías **deben** basarse en clases abstractas.

Regla 88 - Herencia múltiple
La herencia múltiple **será** solo permitida con la siguiente forma restringida: “n” interfaces de plus “m” implementaciones privadas y una implementación protegida.

Regla 88.1 - Base virtual
Se declarará explícitamente una base virtual con estado en cada clase derivada que acceda a ella.

Regla 89 - Clase base
Una clase base **no debe** ser virtual y no virtual en la misma jerarquía.

Regla 90 - Interfaces más utilizadas
Las interfaces más utilizadas **deben** ser mínimas, generales y abstractas.

Regla 91 - Herencia pública
La herencia pública **se utilizará** para implementar relaciones "is-a".

Regla 92 - Subclase
los métodos de subclase **deben** esperar menos y entregar más que la clase base de los métodos que anulan.

Regla 93 - Relaciones
Las relaciones "tiene-un" o "se-implementa-en-términos-de" **serán** modeladas a través de la membresía o herencia no pública.

Regla 94 - Función no virtual
Una función no virtual heredada **no se** redefinirá en una clase derivada.

Regla 95 - Parámetros
Un parámetro predeterminado heredado **nunca deberá** ser redefinido.

Regla 96 - Matrices
Las matrices **no deben** tratarse polimórficamente.

Regla 97 - Matrices en interfaces
Las matrices **no deben** utilizarse en interfaces. En su lugar, debería utilizarse la clase Array.

Regla 97.1 - Operador de igualdad
Ningún operando de un operador de igualdad (== o !=) **Será** un puntero a un miembro virtual de la función.

Regla 98 - Nombres no locales
Todos los nombres no locales, excepto main (), **deben** colocarse en algún espacio de nombres.

Regla 99 - Espacio de nombres
Los espacios de nombres **no deben** anidarse a más de dos niveles de profundidad.

Regla 100 - Elementos de un espacio de nombres
Los elementos de un espacio de nombres **deben** seleccionarse de la siguiente manera: 1.- Usar una declaración o una calificación explícita para algunos (aproximadamente cinco) nombres. 2.- Directiva de uso para muchos nombres.

Regla 101 - Revisión de plantillas
Las plantillas **deben** ser revisadas de la siguiente manera: 1.- Con respecto a la plantilla de forma aislada considerando los supuestos o requisitos que se imponen a sus argumentos. 2.- Con respecto a todas las funciones instanciadas por argumentos reales.

Regla 102 - Pruebas de plantillas
Las pruebas de plantilla **deben** crearse para cubrir todas las instancias de plantilla reales.

Regla 103 - Argumentos de las plantillas
Deben aplicarse comprobaciones de restricciones a los argumentos de la plantilla.

Regla 104 - Especialización de plantilla
Una especialización de plantilla **debe ser** declarada antes de su uso.

Regla 105 - Definición de plantilla
Se **debe** minimizar la dependencia de una definición de plantilla de sus contextos de instanciación.

Regla 106 - Tipos de punteros
Se **deben** realizar especializaciones para tipos de puntero cuando sea apropiado.

Regla 107 - Declaración de funciones
Las funciones **deberán** siempre ser declaradas en el ámbito de archivo.

Regla 108 - Funciones de argumento variable
No deben utilizarse funciones de argumento variable.

Regla 109 - Definición de función
Una definición de función **no debe** colocarse en una especificación de clase a menos que la función esté destinada a estar insertada.

Regla 110 - Argumentos en funciones
No se usarán funciones que contengan más de 7 argumentos ya que las hace difíciles de leer.

Regla 111 - Retornar objetos locales
Una función no deberá retornar un puntero o una referencia a un objeto local no estático.

Regla 112 - Propiedad de los recursos
Los valores que retorna una función no debe invalidar la propiedad de los recursos.

Regla 113 - Puntos de salida de la función
Las funciones sólo tendrán un único punto de salida debido a que si hay varios puntos de salida las funciones son más difíciles de entender y analizar.

Regla 114 - Declaraciones del retorno de las funciones
Todos los puntos de salida de funciones que retornan un valor deben ser retornos declarados.

Regla 115 - Retorno de errores
Si una función retorna un error, esa información retornada debe ser probada ya que si se ignora el retorno la aplicación puede continuar procesando esa falsa suposición.

Regla 116 - Pasar un parámetro por valor
Los parámetros de tipo concreto serán enviados por valor si los cambios a esos parámetros no son reflejados en la función llamada.
Los objetos no concretos deberán ser pasados por puntero o referencia.

Regla 117 - Pasar parámetros por referencia
Los parámetros serán enviados por referencia si el valor no puede ser NULL.
Un objeto será enviado como const T& si la función no cambia el valor del objeto.
Un objeto será enviado como T& si la función puede cambiar el valor del objeto.

Regla 118 - Pasar parámetros por puntero
Los parámetros deberán ser enviados por puntero si los valores NULL son posibles.
Un objeto deberá ser enviado como const T* si este valor no va a ser modificado, y como T* si el valor puede ser modificado.

Regla 119 - Recursividad
Las funciones no deben llamarse a ellas mismas, ya sea directa o indirectamente, debido a que puede provocar un desbordamiento.
La recursión solo se puede usar si se puede probar que existen los recursos necesarios para soportar el máximo número de recursiones posibles.

Regla 120 - Sobrecarga de operaciones
Un método o función sobrecargada debe formar parte de una familia que use la misma semántica, compartan el mismo nombre, tengan el mismo propósito y sea diferenciado por un parámetro formal.

Regla 121 - Funciones inline
Solo las funciones con una o dos declaraciones pueden ser consideradas para ser funciones inline.
O sea funciones insertadas en el espacio de código que deban funcionar.

Regla 122 - Funciones triviales inline
Las funciones con acceso trivial o funciones mutantes deberán ser funciones inline.

Regla 123 - Número de funciones de acceso
El número de funciones de acceso debe de ser minimizado, de lo contrario indica que una clase simplemente sirve para agregar una colección de datos.

Regla 124 - Funciones triviales
Las funciones de reenvío cortas y simples pueden ser funciones inline ya que pueden ahorrar tiempo y espacio.

Regla 125 - Objetos temporales
Los objetos temporales innecesarios deben ser evitados debido a que la creación y destrucción de estos objetos pueden provocar un peor desempeño.

Regla 126 - Comentarios
Solo será usado el estilo válido o standard de comentarios en C++ (//).

Regla 127 - Código comentado
El código que no es usado y esté comentado debe ser eliminado para una mejor lectura.

Regla 128 - Comentarios externos
Comentarios que documenten acciones que sucedan fuera del archivo documentado no serán permitidas.

Regla 129 - Comentarios en la cabecera
Los comentarios en la cabecera de los archivos deberá describir el comportamiento de las funciones o clases visibles que están siendo documentadas.

Regla 130 - Comentar código
El propósito de cada cada línea de código ejecutable puede ser explicado por un comentario, un comentario pueda describir más de una línea de código.
Esta regla no indica que todas las líneas de código deben ser comentadas.

Regla 131 - Comentarios redundantes
Se debe evitar explicar algo en los comentarios si ya está mejor explicado en el código, ya que puede ser redundante y hasta complejo de leer.

Regla 132 - Comentar declaraciones
Cada declaración de variable, enumeración de valores y de estructuras debe de ser comentada, a excepción de casos donde es redundante.

Regla 133 - Comentarios de introducción
Cada archivo fuente tiene que tener documentada una introducción que brinde información sobre el nombre, los requerimientos y el contenido del archivo.

Regla 134 - Limitaciones de funciones
Las limitaciones de una función deben ser comentadas, ya que el mantenimiento se puede volver muy difícil si se desconocen las limitaciones de la función.

Regla 135 - Nombre de identificadores
Los identificadores de un espacio interno no deben ser los mismos que los identificadores de un espacio externo debido a que puede ser confuso.

Regla 136 - Alcance de las variables
La declaración de las variables debe tener el menor alcance posible, esto para intentar tener el menor número de variables "vivas" simultáneamente.

Regla 137 - Declaraciones estáticas
Todas las declaraciones en el alcance del archivo deben ser estáticas siempre que sea posible ya que minimiza las dependencias entre transacciones de unidades.

Regla 138 - Vínculos con los identificadores
Los identificadores no deben tener vínculos internos ni externos simultáneamente en la misma unidad de transacción, evitar ocultar nombres de variables ya que puede ser confuso.

Regla 139 - Objetos externos
Los objetos externos no deben ser declarados en más de un archivo.

Regla 140 - Almacenamiento de registro
El almacenamiento de registro de clases no será utilizado.

Regla 141 - Declaración y definición
Ninguna clase, estructura o enumerado se va a declarar en la misma definición.
La definición se hará independientemente de la declaración del objeto.

Regla 142 - Inicializar variables
Todas las variables deben ser inicializadas antes de ser utilizadas, exceptuando objetos que no se pueden inicializar antes de usarse, como input streams.

Regla 143 - Valor de inicialización
Las variables no se inicializarán hasta que puedan ser inicializadas con un valor significativo para esa variable.

Regla 144 - Uso de corchetes
Los corchetes deben ser usados para indicar la estructura en la inicialización de un arreglo o estructura.

Regla 145 - Enumerador de lisa
En un enumerador de una lista él “=” no debe ser usado para inicializar miembros, a no ser que los miembros ya estén inicializados explícitamente.

Regla 146 - Estándar de punto flotante
Las implementaciones del punto flotante debe de cumplir con un estándar definido, el cual es el ANSI/IEEE Std 754.

Regla 147 - Representación de bits en flotantes
El programador **no utilizará** de ninguna manera las representaciones de bits subyacentes de los números de coma flotante, ya que esta manipulación es propensa a errores.

Regla 148 - Uso de enumeraciones
Se **deben utilizar** datos de tipo de enum en lugar de tipos enteros (y constantes) para seleccionar entre una serie limitada de opciones, puesto que mejora la depuración, la legibilidad y el mantenimiento del código.

Regla 149 - Constantes octales
Ya que cualquier constante entera que comience con cero ("0") está definida por el estándar de C++ como una constante octal, **no se deben utilizar** constantes octales, sin embargo, los números hexadecimales y cero (que también es una constante octal) son permitidos .

Regla 150 - Constantes hexadecimales
Las constantes hexadecimales **se representarán** con todas las letras mayúsculas.

Regla 151 - Valores numéricos
No se utilizarán valores numéricos en el código; en su lugar, se utilizarán valores simbólico a excepción de la inicialización de un miembro de un arreglo.

Regla 151.1 - Cadenas literales
Las cadenas literales **no deben modificarse**.

Regla 152 - Declaración múltiple de variables
La declaración múltiple de variables en una misma línea **no está permitida**.

Regla 153 - Uniones
Las uniones **no deben utilizarse** ya que no son de tipo estático seguro e históricamente se sabe que son una fuente de errores.

Regla 154 - Campos de bits
Los campos de bits **deben tener** únicamente datos explícitos de tipo enumeración o enteros sin signo.

Regla 155 - Empaquetamiento de datos en campos de bits
Los campos de bits **no se utilizarán** para empaquetar datos en una palabra con el único propósito de ahorrar espacio.

Regla 156 - Acceso a miembros de estructura/clase
Todos los miembros de una estructura (o clase) serán nombrados y solo se podrá acceder a ellos a través de sus nombres.

Regla 157 - Efectos secundarios en los operadores
El operador a la derecha de los operadores *&&* o || no debe contener efectos secundarios (es decir, operaciones que realicen cambios de estado del entorno de ejecución).

Regla 158 - Paréntesis en los operadores
Los operadores *&&* o || **deberán estar contenidos** entre paréntesis si los operandos contienen operadores binarios

Regla 159 - Sobrecarga de operadores
Los operadores *&&*, || y el operador unario *&* **no deberán sobrecargarse**.

Regla 160 - Expresión de asignación
Una expresión de asignación **se utilizará sólo** como expresión en una declaración de expresión.

Regla 161 - Valores con signo/sin signo en operaciones
Los valores con signo y sin signo **no deben mezclarse** en operaciones aritméticas o de comparación.

Regla 162 - Aritmética sin signo
No se utilizará aritmética sin signo

Regla 163 - Operando derecho del operador de desplazamiento
El operando de la derecha de un operador de desplazamiento **deberá estar** entre cero y uno menos que el ancho de los bits del operador de la izquierda.

Regla 163.1 - Operando izquierdo del operador de desplazamiento a la izq.
El operando de la izquierda de un operador de desplazamiento a la derecha no **deberá tener** un valor negativo.

Regla 164 - Operador menos unario
El operador menos unario **no deberá ser aplicado** a una expresión sin signo.

Regla 165 - Efectos secundarios en el operador *sizeof*
El operador *sizeof* **no será usado** en expresiones que contengan efectos secundarios (es decir, operaciones que realicen cambios de estado del entorno de ejecución).

Regla 166 - Compilador para división de enteros
Se determinará, documentará y tendrá en cuenta la implementación de la división de enteros en el compilador elegido.

Regla 167 - Operador coma
El operador coma **no será utilizado**.

Regla 168 - Punteros a punteros
Punteros a punteros **deberán ser evitados** cuando sea posible.

Regla 169 - Indirección de punteros
No deberán de utilizarse más de dos niveles de indirección de punteros.

Regla 170 - Punteros y operadores relacionales
Los operadores relacionales **no deberán ser aplicados** a punteros de tipo excepto cuando ambos operandos sean del mismo tipo y apunten a:

- el mismo objeto,
- la misma función,
- miembros del mismo objeto, o
- elementos del mismo arreglo (incluyendo uno más allá del arreglo)

Regla 172 - Dirección de objetos y persistencia de datos
La dirección de un objeto con almacenamiento automático no se asignará a un objeto que persista después de que el objeto haya dejado de existir.

Regla 173 - Puntero nulo
El puntero nulo **no deberá ser** des-referenciado.

Regla 174 - Punteros y el valor NULL
Un puntero **no deberá ser comparado** con NULL o asignado a NULL, en su lugar se deberá usar un 0.

Regla 175 - typedef
Se utilizará typedef para simplificar la sintaxis del programa al declarar punteros de función.

Regla 176 - Funciones de conversión definidas por el usuario
Funciones de conversión definidas por el usuario **deberán ser evitadas**.

Regla 177 - Casteo hacia abajo
Un casteo hacia abajo (castear una clase derivada de otra hacia su padre) **sólo se permitirá** a través de uno de los siguientes mecanismos:

- Funciones virtuales que actúen como casteos dinámicos.
- Utilizando el patrón de diseño visitor.

Regla 178 - Punteros a clase base virtual
Un puntero a una clase base virtual **no se convertirá** en un puntero a una clase derivada.

Regla 179 - Conversiones implícitas
Las conversiones implícitas que puedan resultar en pérdida de información **no deberán de ser utilizadas**.

Regla 180 - Casteos explícitos redundantes
No se utilizarán casteos explícitos redundantes.

Regla 181 - Casteo de tipos desde o hacia punteros
Los casteos de tipos desde cualquier tipo hacia o desde punteros **no deberán utilizarse**.

Regla 182 - Casteo de tipos
Cualquier medida posible para evitar un casteo de tipos **deberá ser tomada**.

Regla 183 - Casteo de números de coma flotante a números enteros
Los números de coma flotante no se convertirán en números enteros a menos que dicha conversión sea un requisito algorítmico específico o sea necesaria para una interfaz de hardware.
Regla 184 - Casteo de C++ contra casteo de C
Se deberá usar el estilo de casteo de C++ en lugar del estilo de casteo de C

Regla 185 - Código no accesible
Se deberá evitar el código no accesible, es decir, que por cuestiones del control de flujo del mismo no se pueda ejecutar.

Regla 186 - Declaraciones no nulas
Todas las declaraciones deben de ser potencialmente usadas en algún momento de la ejecución del programa.

Regla 187 - Etiquetas
Las etiquetas no serán usadas, excepto en las declaraciones del switch.

Regla 188 - Uso de *goto*
El goto no deberá de ser usado. Solamente podrá ser usada para romper un ciclo anidado.

Regla 189 - Uso de *continue*
El continue no debe de ser usado.

Regla 190 - Uso del *break*
El break no debe ser usado, excepto para terminar los casos de una declaración switch.

Regla 191 - Terminación de condicionales con else
Toda declaración de un if/else if deberá de ser finalizada con un *else,* a pesar de no ser necesario. En caso de que ese else no sea necesario, se deberá de indicar dentro de esa condición el por qué no es necesario usar el *else.*

Regla 192 - Case en sentencia switch
Todos los cases de una declaración de switch, deben de finalizar con un break.

Regla 193 - Default en sentencia switch
Cuando todos los posibles cases de una sentencia switch no son testeados, deberá de colocarse la sentencia default para esos casos.

Regla 194 - Booleanos en sentencia switch
Un switch nunca debe de representar un valor booleano.

Regla 195 - Cantidad de casos en una sentencia switch
Una sentencia switch debe de tener, al menos, dos casos y un potencial default.

Regla 196 - Contadores flotantes
Los números flotantes no deben de ser tomados en cuenta para ser iteradores en un ciclo.

Regla 197 - Inicialización del ciclo for
La expresión de inicialización de un for no ejecutará ninguna instrucción además de inicializar el ciclo.

Regla 198 - Función incremental del ciclo for
La expresión de incrementación de este ciclo no ejecutará otra acción más que la de cambiar el parámetro del ciclo.

Regla 199 - Funciones incrementales e iniciaciones nulas en ciclos
No se usarán funciones incrementales e iniciaciones nulas en el ciclo for. Si se requiere eso, deberás usar el ciclo while.

Regla 200 - Variables numéricas en ciclo for.
Las variables numéricas que son usadas sin un ciclo for para el conteo de las iteraciones, no deben ser modificadas dentro del cuerpo del ciclo.

Regla 201 - Testeo en variables flotantes
Las variables de punto flotante no deben ser probadas para demostrar una igualdad o desigualdad de forma exacta, por el tema de que al ser de este tipo, se pueden encontrar truncadas.

Regla 202 - Evaluación de expresiones
La evaluación de expresiones no debe dar lugar a desbordes o sub desbordamientos.

Regla 203 - Operaciones únicas
Una única operación solo debe ser usada en los siguientes casos:

1. Por ella misma
2. Al lado derecho de una asignación
3. En una condición
4. Cuando es el único argumento de una función como efecto secundario
5. Como condición de un ciclo
6. Como condición de un switch
7. Como parte única de una operación encadenada

Regla 204 - Valor de una expresión
El valor de una expresión siempre deberá de ser el mismo ante cualquier condición.

Regla 205 - Teclado volátil
El teclado volátil no deberá de ser usado, a menos que se necesite una interfaz directa con el hardware del sistema.

Regla 206 - Asignación y desasignación de pilas
La asignación de desasignación de las pilas no debe de ocurrir después de la inicialización.

Regla 207 - Datos globales
Los datos globales no encapsulados deberán de ser evitados.
Regla 208 - Excepciones de C++
Las excepciones de C++ (throw, catch, try) no deben de ser usadas.

Regla 209 - Uso de tipos de datos
Los tipos de datos básicos (int, float, short, long, etc) no deben de ser usados. En su lugar, se deberán de usar los typedef de acuerdo a cada compilador.

Regla 210 - No asumir la representación de los datos
Los algoritmos no deberán de asumir cómo se representan los datos en la memoria del sistema.

Regla 211 - No asumir la asignación de los datos
Los algoritmos no deberán de asumir el orden de la asignación de los datos no estáticos.

Regla 212 - Dependencia del desbordamiento
No se dependerá de ninguna forma del funcionamiento del desbordamiento.

Regla 213 - Dependencia de operadores
No se dependerá de la regla de precedencia de operadores en las expresiones.

Regla 214 - Orden de los objetos no estáticos
Los objetos no estáticos deben de ser inicializados en orden.

Regla 215 - Apuntadores aritméticos
Los apuntadores aritméticos no deben de ser usados.

Regla 216 - Optimización del código
Los programadores no deben de optimizar el código de forma prematura.

Regla 217 - Errores principales
Los errores de compilación y de enlace deben de ser resueltos sobre los errores de ejecución.

Regla 218 - Advertencias del compilador
Los niveles de advertencia del compilador deben ser establecidos en conjunto con las políticas del proyecto.

Regla 219 - Testing en interfaces
Todo test que se haga a una interfaz principal, deberá de ser realizado en el resto de las interfaces que deriven de la primera.

Regla 220 - Cobertura estructural
Los algoritmos de cobertura estructural deben de ser aplicados contra las clases aplanadas.

Regla 221 - Cobertura estructural con herencia
La cobertura estructural dentro de clases con herencia y que además, contenga funciones virtuales, deberá incluir pruebas para cada posible resolución.