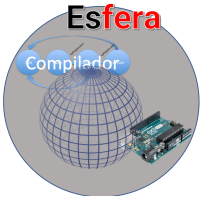
Proyecto Esfera

Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
CE3104 – Lenguaje, Compiladores e Intérpretes
I Semestre 2023



TEC | Tecnológico de Costa Rica

Objetivo general

- ➔ Compilador para maniobrar un artefacto por medio de la técnica y el arte.

Objetivos específicos

- ➔ Implementación de un compilador para un lenguaje de programación usando las herramientas de validación necesarias.
- ➔ Comunicación inalámbrica entre los insumos del intérprete y el arduino

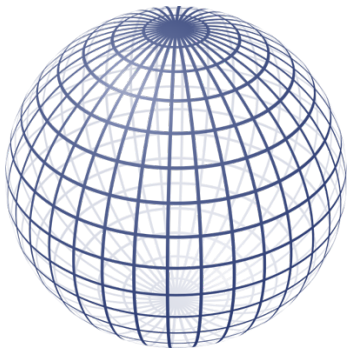
Datos Generales

- ➔ El valor del Proyecto: 25%
- ➔ Nombre código: Esfera
- ➔ El proyecto debe ser implementado por grupos de mínimo 2 personas, máximo de 4 personas.
- ➔ La fecha de entrega del proyecto es de **31/Marzo/2023**
- ➔ Cualquier indicio de copia de proyectos será calificado con una nota de 0 y será procesado de acuerdo al reglamento.

Teoría

En geometría, una superficie esférica es una superficie de revolución formada por el conjunto de todos los puntos del espacio que equidistan de un punto llamado centro.

Esfera proviene del término griego σφαῖρα, sphaîra, que significa pelota (para jugar). Coloquialmente hablando, se emplea la palabra bola, para describir al cuerpo delimitado por una esfera.



Ver
https://es.wikipedia.org/wiki/Esfera

https://youtu.be/pVtZ6yaM-s8

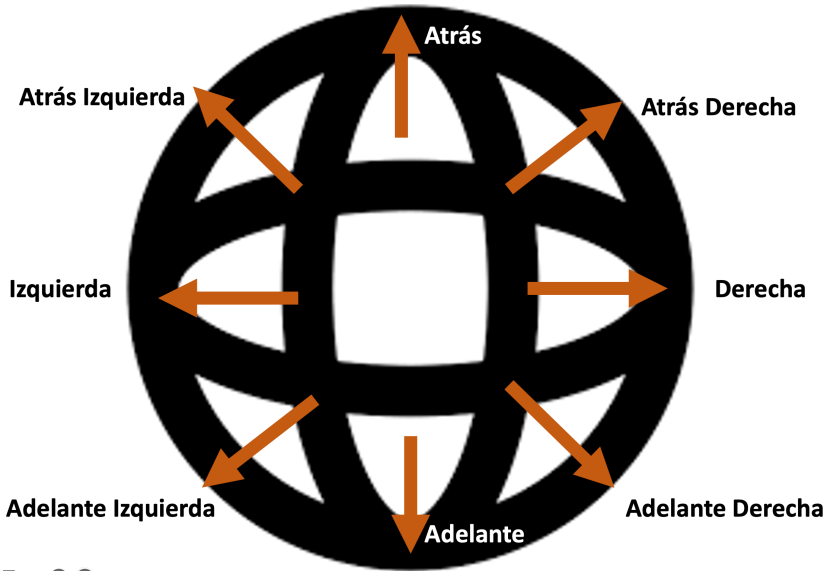
Descripción del problema

Para efectos del proyecto de "Compiladores e Intérpretes" del Tec, se desea desarrollar desde 0 (*desde el inicio*) el lenguaje de programación **Esfera**, el cual tendrá las siguientes características:

- Debe ser un **lenguaje Compilado**
- Mínimo debe **comunicar una computadora** (PC) con un conjunto de dispositivos (hardware).
- La comunicación entre la computadora y el hardware será via **wifi** o **bluetooth** (lo importante es que es **inalámbrica**), esto para ejecutar el código guardado en el hardware.
- Se debe implementar un IDE en el cual se pueda escribir el código fuente. Este IDE deberá tener la capacidad de guardar y recuperar código fuente de la PC.
- El compilador recibirá un programa (con su debido y correcto nombre), realizará las validaciones respectivas y será capaz de enviar las instrucciones suficientes para que el dispositivo pueda realizar todas las instrucciones indicadas. El programa debe estar construido con la sintaxis que se indica más adelante, y podrá estar compuesto de instrucciones directamente colocadas en el cuerpo del programa o bien, llamados a procedimientos dentro del mismo programa.
- Deben construir una adecuada y correcta gramática que soporte todos los requerimientos indicados para el proyecto.
- Para efectos de la validación léxica y sintáctica, pueden utilizar **Lex y Yacc** (en cualquier de sus "sabores"). Esto quiere decir que el intérprete debe "reaccionar" a posibles errores y evitar su ejecución.
- El dispositivo (hardware) y sus componentes deben "obedecer" a las órdenes que se le den por medio del código escrito en un programa y "traducido" por el compilador.
- El objetivo del proyecto es permitir enviar una serie de instrucciones de tal manera que la esfera pueda realizar los movimientos solicitados.

Hardware

- Se debe tener un “dispositivo” similar a un esfera que reconozca las ordenes enviadas desde el compilador y puedan ser ejecutadas.
- Las ordenes enviadas por el compilador serán movimientos, los cuales permitirán a la esfera moverse.
- El conjunto de ordenes enviadas podrán ser ejecutadas cuantas veces se desee y de acuerdo con un dispositivo disparador del evento.
- El dispositivo deberá tener la posibilidad de realizar hasta 8 movimientos distribuidos de acuerdo con la siguiente imagen:



Compilador

- Debe implementar un Compilador con todas las características que este concepto trae consigo de acuerdo con lo solicitado.
- Deberá tener un IDE correctamente hecho en donde sea posible colocar el código fuente.
- De ninguna manera se permitirá la ejecución de comandos en línea entre el "compilador" y el microcontrolador sin pasar por las etapas de análisis de un intérprete visto en clase.
- Se validarán en la revisión los errores léxicos, sintácticos y semánticos, que pueden estar presentes en cualquier programa fuente de este lenguaje de programación.
- El hardware deberá recibir el mandato a ejecutar de forma inalámbrica. Pueden utilizar Wifi, bluetooth, etc.
- El lenguaje de programación para crear el programa a "quemar" en el microcontrolador igual que su ambiente queda a discreción de los estudiantes. Se permite la utilización de bibliotecas disponibles para la programación del ambiente inalámbrico.
- Se puede utilizar Lex y Yacc para la validación del compilador.
- El "editor" queda a criterio del estudiante, pero este debe ser implementado de una manera sencilla y amigable para que el usuario pueda "programar" todas las acciones disponibles en el proyecto. En este editor se escribirá el programa a interpretar. En la revisión NO se debe usar el debug del programa o algún tipo "informal" de revisión de código, todo debe ser por medio del IDE implementado.
- Tiene que existir una buena administración de errores (léxicos, sintaxis, semántico, etc.). Esto es, que el editor debe indicar explícitamente la aparición de un error y del tipo a que pertenece.
- El intérprete debe ser construido siguiendo las mejores prácticas y según lo aprendido en el curso.
- Se deben mostrar de una manera amigable los errores léxicos (ej. No cumple con las reglas en los identificadores, etc.)
- Se deben mostrar de una manera amigable los errores de sintaxis. (ej. Las sentencias no se encuentran escritas correctamente, etc.)
- Se deben mostrar de una manera amigables los errores semánticos.
- Se debe tener un dispositivo externo en el cual se dará la orden para ejecutar el código sobre la esfera.

Software del usuario

- Se debe contar con un software que el usuario podrá utilizar.
- Se debe contar con algún tipo de "botón de inicio" que producirá el inicio de las acciones en el hardware.
- Cada vez que se presione "el botón de inicio" se ejecutarán los movimientos configurados en el hardware. En otras palabras, el compilador genera el código producto del código fuente, se carga dicho código en el hardware y usando el software del usuario se podrá ejecutar las veces que se desea.

Entorno de Programación (IDE)

Se debe crear una interfaz gráfica la cual será el **único medio de revisión del proyecto**. En esta interfaz se deben ver claramente los errores y warnings del compilador. Esta interfaz debe contener los siguientes elementos:

- Poder ser capaces de "cargar" programas predefinidos para modificarlos, compilarlos y ejecutarlos.
- Debe existir un **botón de "compilación"** el cual no ejecute el código sino simplemente realiza "pasadas" de validación según cada etapa de los compiladores.
- Debe haber una **ventana** en donde se pueda cargar o editar los programas y sobre los cuales se ejecutará la compilación y ejecución del código.
- Debe existir una ventana en donde se retornen los valores del comando "=>" (impresión).
- Debe haber una ventana en donde **aparecerán de forma "decente" los errores** que la "compilación" generará. No se permitirá el uso del IDE del lenguaje de programación usado para construir el Compilador ni ningún otro medio. Solamente se hará uso de la interfaz solicitada.
- Se debe tener **control sobre el número de línea del código fuente**, de tal manera que en caso de presentarse un error a la hora de la compilación, se muestre **exactamente** la línea sobre la cual se presenta el error, además el error presentado debe ser **amigable.**

Sintaxis

- Se debe respetar la sintaxis que más adelante se indica, pero el grupo puede "**enriquecer**" dicha gramática añadiendo las sentencias o comportamiento que consideren necesario, pero sin eliminar o cambiar nada de la sintaxis colocada en este documento.
- Se deben tomar en cuenta las siguientes instrucciones:
    - Las variables o Identificadores deben tener las siguientes características:
        - Siempre debe iniciar con una @ (arroa)
        - Un máximo de 10 posiciones (incluye la arroa)
        - Un mínimo de 3 posiciones (incluye la arroa)
        - Debe permitir letras, números y los símbolos "_" y #
        - La @ (arroa) solamente puede estar en la primer posición
        - Si no se cumple algunos de los puntos anteriores, se debe generar un error.
    - Los comentarios en el código fuente es por línea, y siempre debe iniciar con doble raya (--) y se comenta toda la línea, ejemplo:

      -- Esto es un comentario

      Todo código fuente debe tener al menos un comentario indicando el nombre y la funcionalidad del código, y esta debe encontrarse en la primer línea de todo programa. De lo contrario **debe generar un error.** Los demás comentarios pueden estar presentes en cualquier lugar dentro del código del programa, pero siempre debe existir al menos uno en la primera línea de este.

- Deben realizar todas las validaciones pertinentes a nivel léxico, sintáctico, semántico, etc.

- El lenguaje de programación permite y debe utilizar procedimientos.

- El cuerpo básico de un procedimiento debe cumplir con el siguiente formato:

```
Proc nombre_del_procedimiento
(
 ...  -- Instrucciones.
);
```

Donde **Proc** es una palabra reservada, los paréntesis agrupan las sentencias o cuerpo del procedimiento, y el nombre del procedimiento debe respetar las mismas reglas de las variables e identificadores.

- No puede existir más de un procedimiento con el mismo nombre, pues generaría un error.

- También como en otros entornos de programación un procedimiento puede llamar a otro procedimiento, lo que hará esto es realizar las instrucciones que tiene definidas el procedimiento al que se llama.

- Debe existir un procedimiento llamado "**@Principal**" que es el principal y el punto de entrada al programa. Este procedimiento no recibe parámetros. Se debe generar un error si este procedimiento no se encuentra, o bien, si hay más de uno en un programa en particular. Además, se debe generar un error si se le coloca parámetros.

- El programa "**@Principal**" puede estar colocado en cualquier parte del código.

- Las variables definidas dentro de "**@Principal**" se consideran variables globales y las variables definidas en cualquier procedimiento se consideran variables locales.

- Los procedimientos se ejecutan de la siguiente manera

CALL(nombre_del_procedimiento);

- **Sintaxis mínima solicitada:**

Sentencia en Lenguaje	Acción de la sentencia
Def(nombre_variable, tipodatos); **Def(**nombre_variable, tipodatos, valor);	Hay dos formas de definir una variable. En ambas se debe colocar el tipo de datos. En la definición de una variable se le puede asignar un valor, este valor puede ir cambiando a lo largo de la ejecución de un programa. El valor debe ser congruente con el tipo de dato que se quiera guardar en la variable. Los tipos de datos disponibles son los siguientes: - integer - boolean El valor integer puede tener valores positivos o valores negativos. Ej. **Def (**@variable1, integer**);** **Def (** @variable2, boolean, True**);** Los valores permitidos en el tipo de datos boolean son: • True • False Si el dato de inicialización de la variable no corresponde con el tipo dado, se debe generar un error. Una variable no puede ser usada (leída) si no tiene un valor previamente. En caso de suceder, se debe generar un error semántico. Si posteriormente se le asigna otro valor de distinto tipo de datos, se deberá generar un error "semántico". Si no se escribe la sentencia correctamente, o se obvia algún valor, se debe generar un error de sintaxis. Puede haber definiciones de variables en cualquier lado del código.

`nombre_variable(valor);`	Se altera el valor actual de la variable por el dato colocado en la instrucción. El valor usado puede ser un valor especifico, otra variable o el resultado de una operación matemática sobre valores o variables. Ej. @variable1(51); @variable1(-34); @variable2(False); @variable2(@Variable3); @variable1(51 + @variable4); @variable1(@variable4 * 2); @variable2(NOT(@Variable3)); En valores numéricos debe permitir: • Suma (+) • Resta (-) • Multiplicación (*) • División (/) Esto tanto para valores explícitos, como para variables, o combinación de estos últimos. En valor booleanos debe permitir: • Valor opuesto booleano al contenido (Not) • False o True
Alter(nombre_variable, valor)	Altera el valor de la variable ya sea adicional al valor actual de la variable un valor o disminuyéndole. -- Suma 5 unidades al valor de la variable **Alter (**@variable1, 5**);** -- Resta 2 unidades al valor de la variable **Alter (**@variable1, -2**);** -- Suma o resta las unidades de la @variable3 al valor de la variable @variable1 **Alter (**@variable1, @variable3**);** No es permitido usar una variable booleana. En caso de hacerlo deberá generar un error semántico.

Not (nombre_variable);	Modifica el valor de una variable booleana, colocando el valor contrario. -- @variable2 es True **Not** (@variable2); -- @variable2 ahora es False Si la variable es numérica, se debe generar un error.
Mover(A)	Realizará un único movimiento, y la dirección del movimiento será la del valor "A", donde este valor puede ser uno de los siguientes valores: • ATR (atrás) • ADL (adelante) • ADE (Atrás Derecha) • AIZ (Atrás Izquierda) • IZQ (Izquierda) • DER (Derecha) • DDE (Delante Derecho) • DIZ (Delante Izquierda) Cualquier otro valor agregado en esta función será catalogada como error semántico y así mismo se debe retornar en el intérprete.
Aleatorio()	Se deberá ejecutar por lo menos 10 movimientos **aleatorios** de la Esfera de forma automática
Condicionales Numéricas Operando1 operando Operando2	Las condicionales usadas en la sintaxis son las siguientes: • > (mayor qué) • < (menor qué) • == (igual) • <> (distinto qué) • <= (menor o igual qué) • >= (mayor o igual qué) Ej. @variabl1 > 10 @variabl1 <> 5

	Devuelve un valor booleano de acuerdo con el resultado de la condición. El operando2 podría ser el resultado de una función. Si se utiliza como operando una variable booleana, se debe retornar error. Se puede utilizar `True` o `False` como valor de comparación en las condicionales.
`Condicionales booleanas` **`IsTrue`** `(nombre_variable);`	Devuelve True si el valor que contiene la variable es True de lo contrario devuelve False. Ej. `IsTrue (@variabl2)` Si se utiliza una variable numérica se debe retornar error.

`Repeat` `(instrucciones);`	Ejecuta todas las instrucciones que se encuentran en el cuerpo hasta que se encuentra una instrucción “break”.
	
	Si no hay en el cuerpo de instrucciones, una instrucción “break” se debe generar un error.
	
	En el cuerpo de instrucciones puede existir cualquier otra instrucción mencionado en este documento. El cuerpo de instrucciones debe estar entre paréntesis.
	
	Ej.
	`Repeat`
	`(`
	` New @variable1, (Num, 5);`
	` Mover(AAA);`
	` Break;`
	`);`

`Until`
`(instrucciones])`
`Condición;`
	Repite la lista de instrucciones tantas veces hasta que se cumpla la condición.
	
	Primero ejecuta el conjunto de instrucciones, de esta forma se asegura que las instrucciones se

<table>
<tr><td></td><td>ejecutan al menos una vez antes de comprobar la condición.

En caso de no cumplirse la condición, se vuelve a ejecutar el conjunto de instrucciones.

Ej.

-- inicializa la variable en 1
@variable1(1);

Until
(Mover(AAA);
-- Aumenta en 1 a la variable
Alter (@variable1, 1);
)
@variable1 > 10;

Observe que el cuerpo de la instrucción se encuentra entre paréntesis, y la condición viene inmediatamente después del cuerpo de la instrucción.</td></tr>
<tr><td>While condición
(instrucciones);</td><td>Repite la lista de instrucciones tanta veces hasta que se cumpla la condición. Si la condición expresada NO se cumple no se ejecutan las instrucciones ni una sola vez.

Observe que el cuerpo de la instrucción se encuentra entre paréntesis, y la condición viene inmediatamente después del cuerpo de la instrucción.

Ej.
-- Inicializa la variable en True
@variable2(True);

While IsTrue(@variable2)
(Mover(AAA);
-- Cambia variable a False
Not (@variable2);
);

-- Inicializa la variable en 100
@variable1(100);

While @variable1 > 10
(Mover(AAA);
-- Resta 10 de variable
Alter (@variable1, -10);
);</td></tr>
</table>

Case When (condición) **Then** (instrucciones) [**Else** (instrucciones)];	Realiza las instrucciones indicadas SI se cumple con la condición colocada. La instrucción ELSE es opcional. Ej. **Case When** (@variable1 > 2) **Then** (Mover(AAA);); **Case When IsTrue**(@variale2) **Then** (Mover(AAA);;) Else (Mover(ATR););
Case nombre_variable **When** Valor **Then** (instrucciones) **When** Valor **Then** (instrucciones) [**When** Valor **Then** (instrucciones)] **Else** (instrucciones);	Consulta el valor de la variable, y dependiendo de ese valor ejecuta el cuerpo de instrucciones asignado. La instrucción ELSE es opcional. Ej. **Case** @variable1 **When** 1 **Then** (Mover(AAA);) **When** 2 **Then** (Mover(ATR);) **When** 3 **Then** (Mover(IFA);); **Case** @variable2 **When** True **Then** (MoverLeft;) **Else** (MoverRight;);
=>(valor);	Imprime en pantalla el valor dado, donde el valor dado puede ser un string, una variable, o una o más combinación de ambos. **=>** ("Hola Mundo");

	Esta operación genera un cuadro de dialogo a nivel del software presentando en la pantalla el texto colocado.
	
	Puede recibir como parámetro un texto, una variable o una constante.
	
	Puede recibir más de un parámetro a la vez.
	
	`=> ("Este es el Proyecto número ", @variable1, " del Compi 2022");`
	
	`Imprime`
	
	"Este es el proyecto número 1 de Compi 2023"
	
	Este comando => es sumamente importante pues será la manera de revisar a nivel de software la lógica del código.
	
	Si no se implementa no será posible revisar el código del compilador, y se obtendrá una nota de 0 en dicho rubro.

Para la solución del problema del proyecto presentado se espera del estudiante:

- Fuerte investigación sobre las posibles soluciones en las distintas etapas del problema
- Búsqueda de distintas fuentes que servirán de insumo técnico-práctico para las soluciones así como la ayuda de otras áreas para cumplir con el objetivo del proyecto.
- Selección de criterios acordes al nivel de nuestro ambiente Tec en donde se seleccionen las mejores soluciones correspondientes.
- Una solución solvente de acuerdo con lo esperado.
- Cualquier otro elemento de sintaxis que se considere necesario para que el proyecto llegue a un cumplimiento satisfactorio, se debe agregar como responsabilidad del grupo de trabajo, manteniendo la base de la sintaxis previamente expuesta.
- A parte de las instrucciones anteriormente indicadas, se **deben proveer tres (3) instrucciones adicionales** inventadas por el grupo, las cuales serán evaluadas en la revisión. Tienen que ser distintas entre sí y distintas con las planteadas en este enunciado, y deben generar algún tipo de valor agregado en lo que funcionamiento del dispositivo de hardware se refiere.
- Se espera que el proyecto sea **completamente funcional**, o que muestre una funcionalidad aceptable.
- La esfera debe ser manipulada sobre algún tipo de superficie plana (el piso puede ser posible).
- Al momento de la revisión se debe presentar la funcionalidad de todo el sistema,

- Todo el proyecto debe ser implementado por el grupo de estudiantes, en caso de querer la utilización de bibliotecas, módulos u otras facilidades (no mencionadas anteriormente) y que disminuiría la carga de trabajo, se debe consultar previamente con el profesor y obtener su aprobación en dicho uso, en caso contrario se tomará como trabajo inconcluso.
- Como parte de la documentación, se debe presentar la **gramática** usada y aplicada según los requerimientos indicados anteriormente.
- Se debe generar una interfaz en la computadora o dispositivo remoto agradable al usuario, en donde se pueda realizar la codificación y llamado de programas. Además, donde se muestre los errores encontrados tanto a nivel léxico como sintáctico, u otro tipo.

Puntos extras

- Al final de las revisiones del proyecto, se realizará una "competencia" con los grupos interesados. El profesor colocará una ruta "complicada". Se dará puntos extras al grupo que realice la trayectoria en el menor tiempo posible y sin salirse de la misma. Podrán tener hasta 3 oportunidades. Los dispositivos se agruparán según tamaño del mismo, y otras particularidades que podrían dar algún tipo de ventaja.
- Previo a la revisión, los grupos deben indicar si están de acuerdo en participar en la competencia.
- Se realizará la competencia si hay más de un grupo interesado.

Documentación y aspectos operativos

➔ Al finalizar el proyecto en la fecha indicada, se deberá entregar un documento que contenga lo siguiente:

- ✓ Diagrama de arquitectura de la solución que refleje un nivel de detalle suficiente para entender a términos generales el funcionamiento del proyecto.
- ✓ Alternativas de solución consideradas y justificación de la seleccionada.
- ✓ Problemas conocidos: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.
- ✓ Problemas encontrados: descripción detallada, intentos de solución sin éxito, solución encontrada con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.
- ✓ Conclusiones y Recomendaciones del proyecto.
- ✓ Bibliografía consultada en todo el proyecto

Atributos

Se debe **entregar un documento aparte** que contenga una portada y el siguiente detalle.

Debe comentar de qué manera se aplicaron los atributos que posee el curso y su impacto, esto por medio de una tabla en donde se detalle para cada atributo lo siguiente:

- Aplicación del atributo en la solución del proyecto
- Impacto del proyecto en la sociedad
- Retroalimentación obtenida gracias al proyecto

Favor colocar la información antes solicitada para cada uno de los atributos siguientes:

- **Conocimiento de ingeniería (CI):** Capacidad para aplicar los conocimientos a nivel universitario de matemáticas, ciencias naturales, fundamentos de ingeniería y conocimientos especializados de ingeniería para la solución de problemas complejos de ingeniería.

- **Aprendizaje Continuo (AC):** Capacidad para reconocer las necesidades propias de aprendizaje y la habilidad de vincularse en un proceso de aprendizaje independiente durante toda la vida, en un contexto de amplio cambio tecnológico.

Evaluación

1. La implementación corresponde a un 90% de la nota final del proyecto.
2. La documentación tendrá un valor de un 10% de la nota final del proyecto, cumplir con los puntos especificados en la documentación no significa que se tienen todos los puntos. Se debe presentar la documentación de los atributos.
3. Al momento de la revisión, el profesor llevará un listado de cada uno de los puntos solicitados en el proyecto, y en base a este se dará la calificación final del proyecto. Este listado contendrá cada uno de los requerimientos indicados en el presente documento. Este programa se generará en base a las instrucciones mencionadas en el presente documento.
4. Cada grupo recibirá una nota en cada uno de los siguientes apartados: Funcionalidad y Documentación.
5. En lo que a funcionalidad se refiere, se evaluará la generación del compilador (con toda la validación del lenguaje de programación) y además la funcionalidad y correcta ejecución del hardware.
6. De las notas mencionadas en el punto anterior se calculará la Nota Final del Proyecto.
7. No debe imprimirse la documentación, y esta debe encontrarse en formato PDF.
8. La entrega debe hacerse usando el **TECDigital**
9. Las citas de revisión oficiales serán determinadas por el profesor durante las lecciones o mediante algún medio electrónico.
10. Aún cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones

    a. Sí la documentación no se entregan en la fecha indicada se obtiene una nota de 0.
    b. Sí la implementación no es funcional se obtendrá una nota de 0
    c. Se deben respetar los requerimientos del proyecto
11. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto. El único requerimiento que se consultará durante la defensa del proyecto es el diagrama de arquitectura, documentación interna y cualquier otra documentación que el profesor considere necesaria.
12. Cada grupo tendrá como máximo 20 minutos para exponer su trabajo al profesor y realizar la defensa de éste, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo cual se recomienda tener todo listo antes de ingresar a la defensa, y será responsabilidad del grupo administrar el tiempo dispuesto para su revisión de tal manera que el profesor pueda observar todo el producto terminado. Contratiempos de último momento no son excusa para presentarse en la hora acordada y realizar la revisión correspondiente.
13. El profesor llevará a la revisión un programa con errores léxicos, sintacticos y semánticos, los cuales deberían ser reconocidos por el compilador implementado por el grupo.
14. El profesor llevará un programa con movimientos que la esfera deberá realizar, el cual debería de comprobar la funcionalidad del programa realizado por el propio grupo.
15. El profesor llevará un listado con todos los requerimientos colocados en el presente enunciado, el cual será usado para la calificación final del proyecto.
16. Cada grupo es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.
17. Al momento de la revisión, se tomará como producto final, lo entregado por el grupo en ese momento y dadas las circunstancias del mismo, y sobre esto se realizará la revisión. En ningún caso se dará más tiempo para mejorar o presentar el proyecto, ya sea por problemas de último momento o por que las circunstancias hayan generado algún inconveniente (ej. Falta de internet).