1. Orientaciones Generales

Hacer un programa en lenguaje ensamblador que evalúe expresiones matemáticas.

- En la siguiente sección se expondrán una serie de funcionalidades que puede tener o no el proyecto.
- La primera funcionalidad (funcionalidad común) estará presente en todos los equipos.
- Cada una de estas funcionalidades aporta una cantidad de puntos a la nota del proyecto y a la nota final.
- Los estudiantes pueden alcanzar una nota superior a 5 en el proyecto lo cual los ayudará con la nota final de la asignatura.
- Las mejores tareas se considerarán para **premio** de la asignatura.
- Puede ser que ninguna tarea alcance la categoría de premio.
- La tarea se hará en equipos de 2 estudiantes.
- Los equipos pueden ser de estudiantes del mismo grupo o de grupos distintos.
- Los equipos se definen en la 1ra clase práctica despúes de la orientación.
- Se aceptan equipos de un solo estudiante (en casos contados) pero nunca de 3.
- La revisión es por equipo. Si en un equipo hay un estudiante que sabe mucho y otro estudiante que sabe poco la nota del que más sabe no tiene por que ser alta.
- Lo principal que se va a evaluar en el proyecto es la habilidad de programar en ensamblador y la capacidad de resolver problemas de manera independiente.
- Tendrán un final más feliz las tareas que no están completas pero los integrantes del equipo saben todo lo que ocurre (y son capaces de hacer modificaciones y arreglos) que las tareas en las que las funcionalidades están hechas con nivel alto de terminación pero los miembros del equipo no saben defender el proyecto.
- La nota del proyecto es una combinación de:
  - Programa corriendo.
  - Cantidad de errores detectados durante la ejecución.
  - Funcionalidades que haya escogido el equipo.
  - Preguntas que surjan durante la revisión.
  - Defensa del proyecto, explicación, etc.
- La suma total de puntos de todas las funcionalidades es 13 lo cual corresponde con el máximo de puntos que puede alcanzar un equipo.

2. Funcionalidades

2.1. Parte Común (2 puntos)

1. El programa debe permitir 2 tipos de instrucciones:

   - `<variable> = <expresión>` (asignaciones)
   - `print <expresion>` (imprimir)

   `<variable>` es el nombre válido para un identificador en cualquier lenguaje de programación. La longitud máxima que tendrán los nombres de las variables es 3 caracteres.

   `<expresion>` es una expresion matemática compuesta por lo siguiente:

   - numeros enteros de 16 bits con signo
   - operadores aritméticos (`+ - * / %`)
   - signos de agrupacion (paréntesis)

2. La entrada del programa es una secuencia de instrucciones.

3. La salida del programa es un resultado por cada instrucción `print`

4. la comunicación entre el programa y los usuarios será usando el teclado y la pantalla.

5. El usuario en cada momento introduce una línea de texto que será procesada.

6. Si la línea es vacía el programa termina.

7. Los espacios se ignoran siempre. El programa debe permitir los dos siguientes casos (el único espacio obligado es el que va después del `print`)

   - `a=5+1`
   - `a = 5   + 1`

8. Si una expresión no puede ser evaluada se imprimirá un cartel `ERROR` y el programa pasa a esperar la próxima instrucción.

9. Si se utiliza en parte derecha una variable que no haya sido declarada se debe visualizar un cartel de error.

10. Ejemplo:

```
> a = 10
> b = a + 4
> print a * a + b
= 114
>
FIN
```

2.2. Errores Descriptivos (0.5 puntos)

En vez de poner `ERROR` para todos los errores ponerle errores descriptivos a todas las situaciones posibles, por ejemplo:

- `ERROR: expresion mal balanceada.`
- `ERROR: variable no declarada.`
- `ERROR: resultado de la operación fuera de rango`
- `ERROR: división por cero`

2.3. Matrices y Vectores (2 puntos)

1. Aparece un nuevo tipo de datos que son las matrices y como caso particular están las matrices de una sola columna (vectores columna) y las matrices de una sola fila (vectores fila)
2. El formato de representación es el siguiente:
   - vectores columna (ejemplo: `[1; 3; 4]` o `[20; 30]`)
   - vectores fila (ejemplo: `[1 & 2 & 3]` o `[21 & 31]`)
   - matrices (ejemplo `[1 & 2; 2 & 3; 3 & 4]`)
3. Las operaciones aritméticas que a apartir de ahora existen son:
   - suma: (+)
     - entero + entero = entero
     - columna + columna = columna
     - fila + fila = fila
     - matriz + matriz = matriz
   - resta: (-) similar a la suma
   - multiplicacion: (*)
     - entero * entero = entero
     - fila * entero = fila
     - columna * entero - columna
     - matriz * entero = matriz
     - fila * fila = fila (producto escalar)
     - columna * columna = columna (producto escalar)
     - fila * columna = matriz
     - matriz * matriz = matriz
   - division: (/)
     - entero / entero = entero
     - fila / entero = fila
     - columna / entero = columna

- matriz / entero = matriz

4. Los casos que no se contemplan en el punto anterior se consideran `ERROR`.
5. **Notas**:
   - Los vectores columna estan separados por punto y coma.
   - Los vectores fila estan separados por ampersand (`&`).
   - Los vectores columna se puede interpretar como matrices de un solo elemento por fila.
   - Los vectores fila se pueden interpretar como matrices de una sola fila.
   - Cuando una operación no se pueda efectuar se imprimirá `ERROR`.
   - Cuando se sumen dos filas o dos columnas o dos matrices estas deben tener la misma dimensión en caso contrario se imprime `ERROR`.
   - En la multiplicacion se deben verificar las dimensiones de las matrices y los vectores en caso de que no se pueda multiplicar se debe imprimir `ERROR`.

2.4. Operaciones Extras con Matrices (1 punto)

1. Depende de **Matrices y Vectores (2.3)**
2. Cada estudiante definirá como será la sintaxis para las operaciones.
3. Las operaciones nuevas que aparecerán son las siguientes:
   - Invertir matrices.
   - Hallar la traspuesta.
   - Hallar determinante.

2.5. Operaciones Aritméticas Adicionales (1 punto)

1. las operaciones nuevas son:
   - potencia (`^`)
   - raiz cuadrada
   - factorial
   - raiz enésima
   - sumatoria
   - shift left (`<<`)
   - shift right (`>>`)
   - not (`~`)
   - or (`!`)
   - and (`&`)
2. El estudiante decidirá la sintaxis para las operaciones no tienen definido el operador.

2.6. Funciones (2 puntos)

1. A partir de ahora se podrán efinir funciones de cualquier cantidad de variables.
2. Tanto el nombre de la función como el nombre de las variables puede ser una secuencia de hasta 3 caracteres validos para un identificador.
3. La parte izquierda de la función (lo que está a la izquierda del signo =) es del tipo `f(x) o fun(pri, seg)`.
4. La parte derecha es una expresión que puede depender de:
   - Los parámetros definidos en la parte izquierda.
   - Otras variables definidas anteriormente (tienen que estar definidas si no se imprimer `ERROR`)
   - Otras funciones ya definidas evaluadas en los parámetros: `f(x) = g(x) + 5`

2.7. Boolean (1 punto)

1. Aparecen los operadores de comparación (`< <= > >= == !=`) que devuelve bool por ejemplo: `a = 3 > 2` (a = true)
2. Aparecen los operadores lógicos entre boolean (`||, &&, !`)
3. Aparece el operador: `(condicion) ? resultado_true : resultado_false`

2.8. Números Grandes (1 punto)

1. Todos los operandos y resultados pueden tener hasta 10 cifras.
2. Se recomienda implementas la multiplicacion y la division en galera (método que se aprende en la primaria)

2.9. Float (1 punto)

1. Depende de **Números Grandes (2.8)**
2. Todas las operaciones se pueden efectuar con números con punto flotante.
3. Los números podrán tener 10 cifras antes del punto decimal y 10 después.

2.10. Funciones recursivas (1 punto)

1. Depende de **Funciones (2.6)**
2. Depende de **Float (2.9)**
3. Se pueden definir de la siguiente forma
   `fact(n) = n < 2 ? 1 : n * fact(n-1)`

2.11. Nombres de variables largos (0.5 puntos)

1. Depende de **Funciones (2.6)**
2. Los nombres de las variables y las funciones pueden tener cualquier longitud.
3. Puede haber funciones y variables con el mismo nombre.
4. Puede haber funciones con el mismo nombre siempre y cuando reciban distinta cantidad de parámetros (sobrecarga).

2.12. Otras Funcionalidades

Cualquier otra funcionalidad que se implemente se valorará en el momento de la revisión la cantidad de puntos que eso puede aportar o no a la nota del mismo.