Gramática del lenguaje de programación o (ómicron)

<programa>	→	**main** { <funciones> <declaraciones> <funciones> <sentencias> }
<programa>	→	**main** { <declaraciones> <funciones> <sentencias> }
<programa>	→	**main** { <funciones> <sentencias> }
<declaraciones>	→	<declaracion>
	|	<declaracion> <declaraciones>
<declaracion>	→	<modificador***es***_acceso> <clase> <identificadores> **;**
	|	<modificador***es***_acceso> <declaracion_clase>;
<modificadores_acceso>	→	hidden unique
	|	secret unique
	|	exposed unique
	|	hidden
	|	secret
	|	exposed
	|	unique
	|	**λ**
<clase>	→	<clase_escalar>
	|	<clase_puntero>
	|	<clase_vector>
	|	<clase_conjunto>
	|	<clase_objeto>
<declaracion_clase>	→	<modificador***es***_clase> **class** <identificador> **inherits** <identificadores> **{** <declaraciones> <funciones> **}**
<declaracion_clase>	→	<modificador***es***_clase> **class** <identificador> **{** <declaraciones> <funciones> **}**
<modificadores_clase>	→	**λ**
<clase_escalar>	→	<tipo>
<tipo>	→	**int**
	|	**boolean**
	|	**float**
<clase_objeto>	→	**{** <identificador> **}** /* variables que apuntarán a un objeto de una clase */
<clase_puntero>	→	<tipo> *

	|	<clase_puntero> *****
<clase_vector>	→	**array** <tipo> **[** <constante_entera> **]**
	|	**array** <tipo> **[** <constante_entera> **,** <constante_entera> **]**
<clase_conjunto>	→	**set of** <constante_entera>
<identificadores>	→	<identificador>
	|	<identificador> **,** <identificadores>
<funciones>	→	<funcion> <funciones>
	|	**λ**
<funcion>	→	**function** <modificador***es***_acceso> <tipo_retorno> <identificador> **(** <parametros_funcion> **) {** <declaraciones_funcion> <sentencias> **}**
<tipo_retorno>	→	**none**
	|	<tipo>
	|	***<clase_objeto>***
<parametros_funcion>	→	<parametro_funcion> <resto_parametros_funcion>
	|	**λ**
<resto_parametros_funcion>	→	**;** <parametro_funcion> <resto_parametros_funcion>
	|	**λ**
<parametro_funcion>	→	<tipo> <identificador>
	|	***<clase_objeto> <identificador>***
<declaraciones_funcion>	→	<declaraciones>
	|	**λ**
<sentencias>	→	<sentencia>
	|	<sentencia> <sentencias>
<sentencia>	→	<sentencia_simple> **;**
	|	<bloque>
<sentencia_simple>	→	<asignacion>
	|	<lectura>
	|	<escritura>
	|	<liberacion>
	|	<retorno_funcion>
	|	<operacion_conjunto>
	|	<identificador_clase> **.** <identificador> **(** <lista_expresiones> **)**
	|	<identificador> **(** <lista_expresiones> **)**

	|	<destruir_objeto>
<destruir_objeto>	→	**discard** <identificador>
<bloque>	→	<condicional>
	|	<bucle>
	|	<seleccion>
<asignacion>	→	<identificador> = <exp>
	|	<elemento_vector> = <exp>
	|	<elemento_vector> = **instance_of** <identificador> **(** <lista_expresiones> **)**
	|	<acceso> = <exp>
	|	<identificador> = **malloc**
	|	<identificador> = **&** <identificador>
	|	<identificador> = **instance_of** <identificador> **(** <lista_expresiones> **)**
	|	<identificador_clase> **.** <identificador> = <exp>
<elemento_vector>	→	<identificador> **[** <exp> **]**
	|	<identificador> **[** <exp> **,** <exp> **]**
<condicional>	→	**if (** <exp> **) {** <sentencias> **}**
	|	**if (** <exp> **) {** <sentencias> **} else {** <sentencias> **}**
<bucle>	→	**while (** <exp> **) {** <sentencias> **}**
	|	**for (**<identificador> = <exp> **;** <exp> **) {** <sentencias> **}**
<lectura>	→	**scanf** <identificador>
	|	**scanf** <elemento_vector>
<escritura>	→	**printf** <exp>
	|	**cprintf** <identificador>
<liberacion>	→	**free** <identificador>
<acceso>	→	***** <identificador>
	|	***** <acceso>
<retorno_funcion>	→	**return** <exp>
	|	**return none**
<seleccion>	→	**switch (** <exp> **) {** <casos_seleccion> **}**
<casos_seleccion>	→	<casos_estandar> <caso_defecto>
<casos_estandar>	→	<caso_estandar>
	|	<casos_estandar> <caso_estandar>

<caso_estandar>	→	**case** <constante_entera> **:** <sentencias>
<caso_defecto>	→	**default** <sentencias>
<operación_conjunto>	→	**union (** <identificador> **,**<identificador> **,** <identificador> **)**
	|	**intersection (** <identificador> **,** <identificador> **,** <identificador> **)**
	|	**add (** <exp> **,** <identificador> **)**
	|	**clear (** <identificador> **)**
<exp>	→	<exp> + <exp>
	|	<exp> **-** <exp>
	|	<exp> / <exp>
	|	<exp> ***** <exp>
	|	**-** <exp>
	|	<exp> **&&** <exp>
	|	<exp> || <exp>
	|	**!** <exp>
	|	<identificador>
	|	<constante>
	|	**(** <exp> **)**
	|	**(** <comparacion> **)**
	|	<acceso>
	|	<elemento_vector>
	|	**size (** <identificador> **)**
	|	**contains (** <exp> **,** <identificador> **)**
	|	<identificador> **(** <lista_expresiones> **)**
	|	<identificador_clase> **.** <identificador> **(**<lista_expresiones>**)**
	|	<identificador_clase> **.** <identificador>
<identificador_clase>	→	<identificador>
	|	**itself**
<lista_expresiones>	→	<exp> <resto_lista_expresiones>
	|	**λ**
<resto_lista_expresiones>	→	**,** <exp> <resto_lista_expresiones>
	|	
<comparacion>	→	<exp> == <exp>
	|	<exp> **!=** <exp>

	|	<exp> **<=** <exp>
	|	<exp> **>=** <exp>
	|	<exp> **<** <exp>
	|	<exp> **>** <exp>
<constante>	→	<constante_logica>
	|	<constante_entera>
	|	<constante_real>
<constante_logica>	→	**true**
	|	**false**
<constante_entera>	→	<numero>
<numero>	→	<digito>
	|	<numero> <digito>
<constante_real>	→	<constante_entera> **.** <constante_entera>
<identificador>	→	<letra>
	|	<letra> <cola_identificador>
<cola_identificador>	→	<alfanumerico>
	|	<alfanumerico> <cola_identificador>
<alfanumerico>	→	<letra>
	|	<digito>
<letra>	→	**a** | **b** | ... | **z** | **A** | **B** |...| **Z**
<digito>	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Consideraciones sintácticos Adicionales

El lenguaje ÓMICRON (**o**) permite incluir comentarios encerrados entre los caracteres **//** y el final de la línea (son comentarios de una sola línea)

Los identificadores se limitan a una longitud de 50 caracteres.

Observa que por tu claridad

- Se han marcado en gris reglas que este año no se aplican
- Se han marcado en naranja reglas que tienen que ver con la parte orientada a objetos.

Consideraciones sintácticos adicionales (procedural)

En este apartado se describe la semántica del lenguaje **o** sin componentes de orientación a objetos agrupando temáticamente las restricciones semánticas.

- Declaración y uso de identificadores. Las restricciones semánticas que afectan a la declaración y el uso de los identificadores son las siguientes:
    - Las variables globales, las locales y las funciones deben ser definidas antes de ser utilizadas.
    - Los parámetros de las funciones se definen en la propia cabecera de la función.
    - Las variables globales, las locales, las funciones y sus parámetros deben ser únicos dentro de su ámbito de aplicación.

- Expresiones lógicas. Las restricciones semánticas relativas a las expresiones lógicas son las siguientes:
    - En las expresiones lógicas sólo pueden aparecer datos de tipo lógico. Por lo tanto, todas las subexpresiones, variables y constantes empleadas en una expresión lógica tienen que ser de ese tipo.
    - Como puede observarse en la gramática de **o**, el lenguaje dispone de los siguientes operadores:
        - disyunción
        - conjunción
        - negación

- Expresiones aritméticas. En relación a las expresiones aritméticas, las restricciones semánticas son las siguientes:
    - En las expresiones aritméticas sólo pueden aparecer datos de tipo numérico. Por lo tanto, todas las subexpresiones, variables y constantes empleadas en una expresión aritmética tienen que ser de ese tipo.
    - Como puede observarse en la gramática de **o**, los operadores binarios disponibles para operaciones aritméticas son los siguientes:
        - suma
        - resta
        - multiplicación
        - división
        - El operador monádico disponible para operaciones aritméticas es el cambio de signo.

- Expresiones de comparación. Las restricciones semánticas que afectan a las expresiones de comparación son:
    - Las comparaciones sólo pueden operar con datos de tipo numérico y el resultado de la comparación es de tipo lógico.
    - Los operadores disponibles en **o** son los siguientes:

        - igualdad de operandos
        - desigualdad de operandos
        - el primer operando menor o igual que el segundo
        - el primer operando mayor o igual que el segundo
        - el primer operando menor que el segundo
        - el primer operando mayor que el segundo

- Asignaciones. Las asignaciones válidas son aquellas en las que las partes izquierda y derecha son del mismo tipo.

- Vectores. Las variables de tipo vector tienen la semántica habitual de los lenguajes de programación con las siguientes peculiaridades:
    - Sólo pueden contener datos de tipo básico (no existen vectores de vectores).
    - Sólo son de una dimensión.
    - El tamaño de los vectores no podrá exceder nunca el valor de 64.
    - Para acceder a los elementos de los vectores se utilizará cualquier expresión de tipo entero. Esta expresión debe tener un valor entre 0 y el tamaño definido para el vector menos 1 (ambos incluidos).
    - Tras la declaración de una variable de tipo vector, ésta aparecerá siempre indexada, y se utilizará de la misma manera que cualquier otro objeto que pueda ocupar su misma posición.

- Estructuras de control de flujo de programa iterativas y condicionales. La semántica de las estructuras de control de flujo de programa iterativas (while) y condicionales (if e if-else) es similar a la de otros lenguajes de programación de alto nivel.

- Operaciones de entrada/salida
    - La operación de entrada lee datos escalares y los almacena en variables. Está contemplada la lectura de datos enteros y lógicos.
    - La operación de escritura de datos de tipo escalar trabaja con expresiones de tipo lógico o numérico.

- Funciones. Las funciones se rigen por las siguientes restricciones semánticas:
    - En las llamadas a funciones, los parámetros actuales no pueden ser llamadas a otras funciones.
    - Una sentencia de retorno de función solamente debe aparecer en el cuerpo de una función.
    - En el cuerpo de una función obligatoriamente tiene que aparecer al menos una sentencia de retorno.
    - En una sentencia de retorno el tipo de la expresión debe de coincidir con el tipo de retorno de la función.

Consideraciones semánticos adicionales (orientación a objetos)

Debes tener en cuenta además las siguientes cuestiones:

- No se permite anidar funciones ni clases (no hay funciones internas a funciones ni a métodos ni clases internas a clases).

- Las funciones tienen que tener al menos un return. Las none deben tener return none.

- No se pueden sobrecargar métodos sólo cambiando el tipo de retorno o los cualificadores de acceso o de clase.

- Puede haber argumentos y variables locales de tipo "clase" ({<ID>} ID).

- No se permite extender una clase que no se haya declarado previamente. Si se necesitara (casos de relaciones mutuas entre clases que quisieran ser implementadas con objetos de unas clases como atributos de la otra y no se pudiera encontrar un orden de declaración adecuado) se debería recurrir al polimorfismo y utilizar clases más generales que se puedan utlilizar como atributos de forma que luego puedan albergar referencias más específicas.

- Este es el significado de los cualificadores de acceso cuando el ámbito desde el que se está tratando de acceder a ellos está vinculado con una clase:

    - hidden sólo visible en la clase donde se declara.

    - secret sólo visible en la jerarquía de clases que extiendan la clase que lo declara.
    - exposed (o nada) visible por todos.

- En el caso de querer acceder a un símbolo desde main, el significado de los cualificadores es el siguiente:
    - hidden si el símbolo que se busca para ser accedido desde main es hidden, no se permite su acceso.

    - secret y exposed se tratan de la misma manera, al buscar el símbolo desde main (que está fuera de la jerarquía) sólo se tiene en cuenta que no sea hidden y que sea visible desde main por lo tanto secret y exposed (o ausencia de cualificador) no impiden el acceso al símbolo.

- Éste es el significado de los tipos de miembro:

    - unique sólo hay "una versión" para todas las instancias. Los atributos son compartidos por todos ellos y no se pueen sobreescribir los métodos unique heredados.

    - (nada) son de instancia, cada instanciene su copia de los atributos no unique y los métodos no unique pueden ser sobreescritos.

- Al margen de que la gramática lo permita o no, por homogeneidad de forma general los cualificadores sólo tienen sentido (se ignorarán o prohibirán) cuando no afecten a miembros.
- Cuando hay herencia múltiple, en el caso de que exista ambigüedad respecto al padre del que debe heredar algún miembro, se priorizará al último padre declarado en el programa fuente independientemente de cualquier otra consideración (por ejemplo el orden de las cláusulas inherits).

- Restricciones de declaración de atributos de instancia

    - Cuando no exista en la jerarquía el atributo, se puede declarar.

    - Cuando exista en la clase no se puede declarar.

    - Cuando exista en la jerarquía y sea accesible (accesos distintos de hidden) no se puede declarar.
    - Cuando exista en la jerarquía y no sea accesible (acceso hidden) se puede declarar.

- Restricciones de declarciones de métodos sobreescribibles

    - Cuando no exista en la jerarquía ni en el ámbito actual se puede declarar, es un nuevo método sobreescribible.

    - Cuando no exista en la clase actual pero exista en la jerarquía y no sea accesible (acceso hidden) se puede declarar y es un nuevo método sobreescribible con un nuevo offset acumulado para la tabla de métodos.

    - Cuando no exista en la clase actual pero sí en la jerarquía y sea accesible (acceso distinto de hidden):

        - Si el método encontrado es sobrescribible se puede declarar como sobreescritura utilizando el mismo offset acumulado para la tabla de métodos que el encontrado.

        - Si el método encontrado no es sobreescribible no se pude declarar.

    - De forma general hay que mantener los cualificadores cuando se sobresccribe.

- La cuestión de la sobrecarga

    - Para sobrecargar un método o función, debe tener el mismo nombre que otro y diferentes argumentos (en número, tipo u orden).

    - No se puede sobrecargar un método sólo por cambiar el tipo de retorno o intentar cambiar los cualificadores.

    - Cuando la sobrecarga es posible, de hecho se está definiendo otro método con todas sus características.

- Sobre miembros de clase

    - Para poder declarar un miembro de clase (ya sea instancia o método) sólo se tiene en cuenta que no esté definido previamente en el ámbito actual, sólo en él. En ese caso se declara como un nuevo miembro de clase.

- Sobre acceso a símbolos desde diferentes ámbitos

    - En general, las restricciones de acceso a símbolos desde diferentes ámbitos son las que se deducen de las restricciones anteriores.

    - Hay que añadir el hecho de que desde un método de clase (unique) sólo se podrá acceder a miembros de clase (propios o de los antecesores en la jerarquía) y a variables y funciones globales declaradas previamente.

- El hecho de que las clases se definan antes que las funciones globales hace que desde las clases no se puedan usar funciones globales.

- Observa que la declaración de variables y clases se puede entremezclar.

- A los miembros de instancia se puede acceder sólo a través de (cualificando) instancias.
- A los miembros de clase se puede acceder indistintamente cualificándolos mediante el nombre de la clase o mediante instancias.