

El lenguaje de alto nivel es imperativo, estático y fuertemente tipado. Éste tiene los siguientes tipos de dato:

- **int**: Denota un entero de 32 bits con signo.
- **char**: Denota un caracter almacenado en un byte de 8 bits. El signo del byte no está especificado.
- **void**: Denota la ausencia de valor. No existen valores o variables de este tipo.
- **int[n]**: Denota un arreglo de **n** variables **int**. El valor de **n** debe ser no negativo y debe ser conocido en tiempo de compilación. Los índices del arreglo son enteros en el rango de 0 a **n-1**. Está definido por la implementación si se soportan arreglos de menos de dos elementos.
- **char[n]**: Denota un arreglo de **n** variables **char**. El valor de **n** debe ser no negativo y debe ser conocido en tiempo de compilación. Los índices del arreglo son enteros en el rango de 0 a **n-1**. Está definido por la implementación si se soportan arreglos de menos de dos elementos.
- **int'**: Denota un apuntador a una variable de tipo **int**. Se almacena usando 32 bits.
- **char'**: Denota un apuntador a una variable de tipo **char**. Se almacena usando 32 bits.

Una literal entera es una secuencia de uno o más dígitos (como 0, 123 y 1234545). Una literal de cadena es una secuencia de cero o más caracteres ASCII imprimibles o espacios, los cuales aparecen entre comillas dobles (como "@", "hola" y "saludos a todos"). Las secuencias de escape permitidas dentro de las literales de cadena son `\n` para denotar el caracter de salto de línea, `\\` para denotar la diagonal invertida y `\"` para denotar el caracter de comillas dobles. Se permite que una literal de cadena que abarque varias líneas del código fuente de un programa y éstos forman parte de la literal de cadena. Un identificador es una secuencia de una o más letras, dígitos o guiones bajos. Un identificador no debe comenzar con dígito. Se distingue entre mayúsculas y minúsculas.

Las literales de tipo **int** son literales enteras interpretadas en base decimal. Las literales de tipo **char** son literales de cadena con un único caracter. Las literales de tipo **int[n]** son secuencias de **n** valores de tipo **int** dentro de corchetes y separados por comas (como [1,23,456] para **int[3]**). Las literales de tipo **char[n]** son secuencias de **n** valores de tipo **char** dentro de corchetes y separados por comas (como ["h", "o", "l", "a"] para **char[4]**) o literales de cadena de **n** caracteres (como "hola" para **char[4]**). Está definido por la implementación el manejo de literales de arreglos de menos de dos elementos. No hay literales de otros tipos de dato.

Los operadores binarios del lenguaje deben tener operandos del mismo tipo y son los siguientes.

- **=**: Denota la asignación de valor. Disponible para todo tipo excepto **void**. El operando izquierdo de toda asignación (esto incluyendo asignaciones compuestas) debe ser una variable. No regresa un resultado.
- **+**, **+=**: Denotan la suma y la suma con asignación, respectivamente. Disponible para **int**. El operador sin asignación produce un temporal, mientras que el operador con asignación no regresa un resultado.
- **-**, **-=**: Denotan la resta y la resta con asignación, respectivamente. Disponible para **int**. El operador sin asignación produce un temporal, mientras que el operador con asignación no regresa un resultado.
- *****, ***=**: Denotan el producto y el producto con asignación, respectivamente. Disponible para **int**. El operador sin asignación produce un temporal, mientras que el operador con asignación no regresa un resultado.
- **/**, **/=**: Denotan la división y la división con asignación, respectivamente. Disponible para **int**. El operador sin asignación produce un temporal, mientras que el operador con asignación no regresa un resultado.
- **%**, **%=**: Denotan el residuo y el residuo con asignación, respectivamente. Disponible para **int**. El operador sin asignación produce un temporal, mientras que el operador con asignación no regresa un resultado.
- **&**: Denota la conjunción lógica *and*. Disponible para **int**. El resultado es un temporal que vale 1 si ambas variables son distintas de 0 y que vale 0 en otro caso. Se usa evaluación en corto circuito.
- **|**: Denota la disyunción lógica *or*. Disponible para **int**. El resultado es un temporal que vale 1 si alguna variable es distinta de 0 y que vale 0 en otro caso. Se usa evaluación en corto circuito.
- **<**, **<=**, **>**, **>=**: Denotan las comparaciones *menor que*, *menor o igual que*, *mayor que* y *mayor o igual que*, respectivamente. Disponible para **int**. El resultado es un temporal que vale 1 si la comparación es verdadera y que vale 0 en otro caso.
- **==**, **!=**: Denotan las comparaciones *igual a* y *diferente a*, respectivamente. Disponible para **int**, **char**, **int'** y **char'**. El resultado es 1 si la comparación es verdadera y 0 en otro caso.

Los operadores prefijos del lenguaje son los siguientes.

- `@`: Denota la obtención de un apuntador que contiene la dirección de memoria de una variable. Disponible para `int` y `char`. El resultado es un temporal de tipo `int'` o `char'`, respectivamente.
- `+`: Denota el mismo valor. Disponible para `int`. Produce un temporal.
- `-`: Denota el inverso aditivo del valor. Disponible para `int`. Produce un temporal.
- `!`: Denota la negación lógica *not* del valor. Disponible para `int`. El resultado es un temporal que vale 1 si el valor es 0 y vale 0 en otro caso.
- `#`: Denota el número de elementos de un arreglo. Disponible para `int[n]` y `char[n]`. Produce un temporal de tipo `int`.

Los operadores posfijos del lenguaje son los siguientes.

- `[i]`: Obtiene el *i*-ésimo elemento de un arreglo. Disponible para `int[n]`, `char[n]`, `int'` y `char'`. El resultado es una variable de tipo `int` o `char`, respectivamente.
- `'`: Desreferencia un apuntador. Disponible para `int'` y `char'`. El resultado es una variable de tipo `int` o `char`, respectivamente.

No existen conversiones implícitas en el lenguaje. Los moldeados disponibles en el lenguaje son los siguientes.

- `int(v)`: Produce un temporal de tipo `int` reinterpretando el valor de *v*. El tipo de *v* puede ser `int` o `char`. El valor del temporal se obtiene mediante la extensión de signo de *v* usando el tipo `int`.
- `char(v)`: Produce un temporal de tipo `char` reinterpretando el valor de *v*. El tipo de *v* puede ser `int` o `char`. El valor del temporal se obtiene mediante la copia del primer byte de *v* usando el tipo `char`.

Un comentario de línea comienza con `//` y termina con el fin de línea o con el fin de archivo. Un comentario de bloque comienza con `/*` y termina con `*/`. Los comentarios de bloque no se anidan. El código fuente sólo debe contener caracteres ASCII imprimibles y espacios. Un compilador es libre de tratar cualquier otro carácter como el fin de archivo.

El lenguaje usa los separadores `,` y `;` como en C pero el primero no funciona como operador ni puede listar expresiones fuera de llamadas a función o literales de arreglo. El espaciado no es significativo semánticamente. No se permiten las sentencias vacías y todas las declaraciones son definiciones. La declaración de una variable usa la notación de C:

```
tipo_variable identificador = inicializador;
```

Es posible dejar una variable sin inicializar, en cuyo caso su valor está indefinido. No se pueden declarar varias variables en la misma declaración. En los parámetros de función se debe especificar el tipo de cada variable individual al igual que en C. La declaración de una función sigue la notación de C:

```
tipo_retorno identificador(parámetros) { /* código */ }
```

Es posible que una función llame a otra que está declarada abajo en el código fuente. No existe sobrecarga de funciones. No se permite que los parámetros o el valor de retorno de una función sean arreglos. La ejecución del programa comienza en la función `main`, la cual tiene la siguiente definición:

```
void main( ) { /* código */ }
```

El programa termina al finalizar `main` o si se ejecuta la sentencia de control global `exit` desde cualquier parte del programa. Las sentencias de control local disponibles son `if`, `else`, `while`, `do`, `for`, `break`, `continue` y `return`. El comportamiento de las sentencias locales es el mismo que el de C, excepto que las llaves de sus bloques son obligatorias pero la sección de control no lleva paréntesis, además de que la inicialización de un `for` no puede ser una expresión. Como caso especial, se permiten encadenar `else` con `if`.

```
if condición { /* código */ } else if condición { /* código */ } else { /* código */ }
```

Las declaraciones de bloques internos ocultan declaraciones de bloques externos, tal como ocurre en C. No se permiten variables globales ni funciones locales. No se permiten bloques anónimos.

La palabra reservada `scan` se usa como función y sirve para leer una variable de tipo `int` o `char`. La palabra reservada `print` se usa como función y sirve para imprimir un valor de tipo `int`, `char` o `char[n]`. Las llamadas a estas dos funciones pueden aparecer dentro de una expresión y no regresan un resultado.