

Información General:

Este es un lenguaje de programación interpretado no optimizado, hecho con fines completamente educativos, no recomendado para proyectos de alta demanda en eficiencia o rutinas complejas. Dado que es un lenguaje interpretado, su ejecución se basa en 3 partes: tokenización, construcción de AST y ejecución; en el proceso de tokenización se ignoran por completo espacios en blanco, tabulación, saltos de línea y comentarios, por lo cual todo el código fuente puede ser escrito en una única línea separando los tokens para evitar confusiones al Lexer, a pesar de la posibilidad, esta práctica no es recomendada. Este producto está hecho por Bello's Projects y todos los derechos están reservados.

Extensión de archivos:

Los archivos deben tener la extensión. HZ

```
program.hz
```

Comentarios:

Los comentarios se declaran con @, todo lo que este después del símbolo queda comentado

```
program.hz
1  @Este es un comentario
2
3  @Lineas
4  @comentadas
5  @con arroba
```

Declaración de variables:

Para declarar variables se usa la palabra reservada DATA, se pueden declarar con un valor inicial o no, se pueden declarar múltiples variables al mismo tiempo y si se quiere tener una vista de todos los valores que ha tenido una variable durante la ejecución del programa se puede usar la palabra reservada TRACE al declarar (útil para depuración)

```
program.hz
1  data x          @Declaracion simple
2  data y=5        @Declaracion con valor
3
4  data r,t        @Declaracion multiple
5
6  data trace z    @Declaracion con trace
```

Constantes:

Para valores como salto de línea, tabulación y espacio en blanco (representados en string) existen constantes que representan un valor, las cuales se pueden imprimir o asignar a otras variables, pasar como parámetro de función, usar como operando, etc, pero como toda constante no se puede editar

```
program.hz
1  LINE @salto de linea
2  ESP @espacio en blaco
3  TAB @tabulador
4
```

Valores:

En Heza existen 6 tipos de valores: enteros, string (entre comillas dobles), booleanos, decimales, constantes y variables (ya declaradas)

```
program.hz
1  data x = 5
2  data y = "Hello, World!"
3  data z = false
4  data w = LINE
5  data v = x
6  data u= 2.71
```

Note que no existen caracteres sueltos

Salida por consola:

Para imprimir unos o varios valores usamos la palabra reservada OUT, seguida del operador de flujo (->), mas la lista de valores a imprimir

```
program.hz
1  data x = 5
2  data y = "Hello, World!"
3  data z = false
4
5  out->x @imprime 5
6  out->LINE,y @imprime "\n Hello, World!"
7  out->z or true @imprime true
```

Entrada por consola:

Para recibir datos por la consola se usa la palabra reservada `IN`, seguida del operador de flujo, mas la lista de variables a las que se le asignara el valor recibido, pueden ser una o más variables obligatoriamente declaradas previamente, si estas variables fueron declaradas con `trace`, se les agregara a su historial este cambio

```
program.hz
1  data x
2  data trace y,z
3
4  in->x
5  in->y,z
```

Operaciones numéricas:

Como en la mayoría de los lenguajes de programación, en Heza existen dos tipos de operaciones con números: aritméticas y lógicas. Las aritméticas usan los operadores `+` `-` `*` `%` `^`, siguiendo el orden de jerarquía de operaciones PEMDAS, y las lógicas usan los operadores `not` `and` `or`, dando prioridad a los paréntesis y luego a la negación

```
program.hz
1  @aritmeticas
2  data x = 5+(4-5^2)
3  data y = 3%5/(9-2^x)*5
4
5  @logicas
6  data a = 5 > 3
7  data b = not 4==5 and (3 < 2 or 1 >= 0)
```

Alias:

Se usa la palabra reservada `AS`, para apodar un bloque de código como una función, bucle o condicional, por cada alias debe haber la palabra reservada `END`, que después del operador de referencia (`:`) lleva el alias, indicando que ese bloque de código termino

```
program.hz
1  @Ejemplo de uso de alias para un bloque de codigo
2  @Este ejemplo no es valido porque no cumple con la sintaxis correcta
3  @pero ilustra el concepto de alias para bloques de codigo
4
5  bloque as al
6
7  @bloque de codigo
8
9  end:al
```

Las funciones tienen un modo distinto de declarar los alias que se verá más adelante

Condicionales:

Como se menciona antes, los condicionales agrupan líneas de código en su cuerpo, y están englobados con un alias, primero se coloca la condición entre signos de interrogación, luego se declara un alias, seguidamente va el cuerpo del condicional y al final su respectivo end, si el alias del end no coincide con el declarado dará un error, puede haber condicionales anidados

```
program.hz
1  data x
2  in->x
3
4  ?x>0? as c1
5  |   out->"x is positive"
6  end:c1
```

Una vez cerrado el condicional se puede reutilizar el alias para otro bloque, pero si los condicionales están anidados se debe buscar otro alias

```
program.hz
1  data x
2  in->x
3
4  ?x>0? as c1
5  |   out->"x is positive",LINE
6  end:c1
7
8  ?x>5? as c1
9  |   out->"x is greater than 5"
10 end:c1
```

Para la funcionalidad del Else, Heza maneja una variable llamada 'condition', que representa el valor booleano de la última condición evaluada, se puede hacer un condicional normal con la condición 'not condition', pero como el Else es una operación común, ¿la sintaxis se simplifica a 'default?'. Igualmente debe llevar un alias y end ya que engloba bloques de código.

```
example.hz
1  ?not condition? as name
2
3  end:name
4
5  @equivalente
6  default? as name
7
8  end:name
```

Note que 'condition' es una variable con trace activo, puede ser modificada por el usuario libremente y no necesita estar necesariamente después de una condición, si se llama a una función y en esa función hay un condicional, el valor de condition cambiara a pesar de que el cambio fue en otro contexto. La variable condition se puede usar para operar, asignar su valor a otra variable, recibir por consola, etc. Tenga en consideración que el usuario es responsable de la manipulación de la variable condition, si se le asigna un valor no booleano como un string puede haber un error al hacer default?, lo mas recomendable es no cambiar su valor manualmente para que pueda cumplir con su función.

Observaciones:

Las constantes LINE y TAB no pueden ser sustituidas por el literal "\n" o "\t", ya que el programa imprimirá tal cual la secuencia de caracteres, para eso existen las constantes que funcionan como secuencia de escapes. Heza viene con las constantes matemáticas PI y E, las cuales se pueden usar como un valor decimal, no se pueden declarar variables con estos nombres. Para recuperar el valor de trace una variable se usa la palabra reservada TRACE, seguida del operador de referencia seguida del nombre de la variable, la cual obligatoriamente debe ser declarada con trace, de lo contrario dará error. Este trace es un valor de tipo string, por lo cual se puede imprimir, asignar a otras variables y operar

```
program.hz
1  data trace x = 5
2
3  out-> trace:x @imprime el historial de x
4  data y = trace:x @guarda en y el historial de x
```

Ciclos:

En Heza tenemos dos tipos de ciclos, numérico (FOR) y condicional (WHILE), ambos se declaran con la palabra reservada LOOP, y ambos llevan un alias, la diferencia radica en que el while lleva una condición como parámetro y termina con un signo de interrogación, y el for lleva 3 valores numéricos (inicio, final, incremento), separados por el operador de referencia. En los ciclos de tipo numérico, el alias contiene el valor actual del ciclo, por lo cual se puede usar como variable para imprimir, reasignar, operar etc.

```
program.hz
1  @BuCLe FOR
2  loop(0:10:1) as i
3  |    out->i,ESP
4  end:i
5  |
6  @BuCLe WHILE
7  data x = 5
8  loop(x>0)? as j
9  |    out->j,ESP
10 |    x = x - 1
11 end:j
```

Al igual que en los condicionales, una vez cerrado el bloque se puede reutilizar el alias, también se puede anidar, pero con alias distintos. Los únicos alias que son tratados como variables son los del loop FOR, por lo cual el alias de estos bucles no puede tener el nombre de otras variables ya existentes

Funciones:

Para las funciones se usa la palabra reservada BLOCK, estas también llevan un alias y cierran con end, pero no usan la palabra reservada Aspara declarar el alias porque el nombre de la función actúa como el propio alias, para retornar un valor dentro de una función se usa el operador de flujo seguido de un único valor a retornar

```
program.hz
1  block saludo(name)
2  |    ->"Hola, "+name+"!"
3  end:saludo
4
5  out->saludo("Mundo")
```

Las funciones pueden tener múltiples parámetros, pero retornan un solo valor, también pueden no retornar nada, si se le asigna el valor de una función que no retorna nada a una variable, se hará el procedimiento correspondiente en la función, pero a la variable se le asignará el valor de None. Actualmente no se soportan parámetros por defecto en caso de no enviarse en la llamada a la función

Reasignación múltiple:

En Heza se puede hacer una reasignación múltiple de varias variables con varios valores, Heza hace un respaldo de los valores actuales de las variables antes de la reasignación para evitar perder los valores en caso de reutilizarse, por lo cual es útil y seguro para swaps. Esta forma de asignar es como con variables ya declaradas mas no en la línea de declaración

```
program.nz
1  data x,y = 5,6 @Error de sintaxis
2
3  @Buen uso
4  data x,y
5  in->x,y
6
7  x,y = y,x @Swap variables
8
9  out->x,y
```

CLS:

La palabra reservada CLS limpia la consola, no hace falta enviarla como parámetro ni nada simplemente escribirla y funciona sola

```
program.hz  
1    CLS
```

Resumen:

A continuación, se muestra la lista de todas las palabras reservadas y operadores que existen en Heza

```
documentacion.hz  
1    Palabras clave:  
2    data @Declara una variable  
3    out @Salida por consola  
4    in @ entrada por consola  
5    trace @Activa trace sobre una variable  
6    as @alias  
7    loop @ciclo while-for  
8    end @termina un bloque  
9    block @function  
10   CLS @Limpia la consola
```

```
12   Constantes que representan un valor:  
13   LINE @Salto de línea  
14   TAB @Tabulador  
15   ESP @Espacio en blanco
```

```
17   Operadores aritmeticos:  
18   + @suma  
19   - @resta  
20   * @multiplicacion  
21   / @Division  
22   ^ @potencia  
23   % @Mod
```

```
25   operador de asignacion =  
26  
27   operadores logicos:  
28   not  
29   or  
30   and
```

```
32   Operadoreas de relacion:  
33   >  
34   <  
35   ==  
36   !=  
37   <=  
38   >=
```

```
40 Operadores especiales:
41 ? @Condicional o while
42 -> @flujo para in/out
43 : @Referenciacion
44
```

Extras:

Adicionalmente, en Heza existen unas funciones ya establecidas que se consideran necesarias para un programa común, con el tiempo se irán agregando más en versiones posteriores, también hay dos métodos que solo pueden ser aplicados con variables, no con valores inmediatos, las funciones declaradas por el usuario no pueden contener el nombre de ninguna de las funciones/métodos predefinidos:

```
133 Metodos integrados:
134 Reverse @para texto
135 Equals(c) @devuelve la cantidad de conindicencias del caracter.
```

No todas las funciones/métodos tiene resaltado de sintaxis en la extensión.

Todos los métodos/funciones que viene incorporados retornan un dato y deben tener () para identificarlos como una llamada

```
116 Funciones integradas:
117
118 Integer(texto) @Convierto string a entero si es posible
119 String(variable) @Convierte a string si es posible
120 R(min,max) @Numero random entre min y max
121 Wait(time) @Espera time segundos
122 Size(objeto) @Largo de string o cantidad de digitos de un numero
123 Sin,Cos,Sqrt,Ln @Funciones matematicas de un solo parametron
124 Log(base,n) @Logaritmo base base de n
125 ToUpperCase(txt),ToLowerCase(txt) @trasformacion de texto
126 Pressed(key) @Devuelve true si la tecla key esta presionada en ese momento
127 Vocalas(txt) @Devuelve solo las vocales del texto
128 Abs(n) @valor absoluto
129 Date() @fecha actual
130 Time() @hora actual
131 Repeat(text, n) @Repite text n veces en una cadena
```

Novedades:

Recientemente se agregaron las funciones integradas Talk() y Ear() que interactúan fácilmente con el entorno. La función Talk(text, out) convierte text a voz (con el idioma predefinido del computador) y lo reproduce, también lo imprime en pantalla si out es verdadero. La función Ear(notify), enciende el microfono y devuelve un texto con todo lo que escuche desde que inicia hasta que detecta silencio, si notify es verdadero imprime en pantalla 'Escuchando...' cuando inicie. Esta función requiere conexión a internet para funcionar

Bello's Projects
HEZA DOCUMENTACION

example.hz

```
1  data voz = Ear(false)
2
3  out-> voz
4
5  Talk(["Hello, world!",true])
```

Como usar:

Para programar en Heza ve a Visual Studio Code e instala la extensión Heza Extensión de Bello's Projects, la versión mas reciente es la 0.1.0, esta versión viene con resaltado de sintaxis, snippets y un botón en el menú llamado Run Heza. Para que funcione se debe instalar el intérprete, cuyo enlace se encuentra en los detalles de la extensión, no se requieren permisos de administrador para instalar ni editar variables de entorno. Al instalar el intérprete se asociarán automáticamente los archivos .hz con Heza.exe, y al presionar el botón Run Heza en VS Code puedes ver el resultado en la consola de VS Code sin abrir pestañas ni programas externos.

Todo el código fuente del interprete se encuentra en mi GitHub Bello's Projects en el repositorio Heza, junto al interprete:

Enlaces:

Extension: <https://marketplace.visualstudio.com/items?itemName=BellosProjects.hezaextension>

Intérprete: <https://github.com/bellosprojects/Heza/raw/main/HezaSetup.exe>

Código Fuente: <https://github.com/bellosprojects/Heza>