Bello's Projects

HEZA DOCUMENTACION

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: tokentizacion, construcción de AST y ejecución; en el proceso de tokentizacion 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 de 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 declaras variables se usa la palabra reservada DATA, se pueden declara 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

```
LINE @salto de linea
ESP @espacio en blaco
TAB @tabulador
```

Valores:

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

program.hz

```
data x = 5
data y  = "Hello, World!"
data z = false
data w = LINE
data v = x
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

```
data x = 5
data y  = "Hello, World!"
data z = false

out->x              @imprime 5
out->LINE,y         @imprime "\n Hello, World!"
out->z or true      @imprime true
```

Entrada por consola:

Para recibir datos por la consolase 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

```
data x
data trace y,z

in->x
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

```
@aritmeticas
data x = 5+(4-5^2)
data y = 3%5/(9-2^x)*5

@logicas
data a = 5 > 3
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

```
@Ejemplo de uso de alias para un bloque de codigo
@Este ejemplo no es valido porque no cumple con la sintaxis correcta
@pero ilustra el concepto de alias para bloques de codigo

bloque as al

@bloque de codigo

end:al
```

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

Condicionales:

Como se menciono antes, los condicionales agrupan líneas de código en su cuerpo, y están se engloban 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
```

Actualmente no existe una sentencia que tenga la funcionalidad de Else, la solución por ahora es realizar otro condicional con la condición negada. 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
```

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

```
@Bucle FOR
loop(0:10:1) as i
    out->i,ESP
end:i

@Bucle WHILE
data x = 5
loop(x>0)? as j
    out->j,ESP
    x = x - 1
end:j
```

Al igual que en los condicionales, una ves 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

```
block saludo(name)
    ->"Hola, "+name+"!"
end:saludo

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.hz

```
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 linea
14  TAB @Tabulador
15  ESP @Espacio en blanco
```

```
Operadores aritmeticos:
+ @suma
- @resta
* @multiplicacion
/ @Division
^ @potencia
% @Mod
```

```
operador de asignacion =

operadores logicos:
not
or
and
```

```
Operadoreas de relacion:
>
<
==
!=
<=
>=
```

```
Operadores especiales:
? @Condicional o while
-> @flujo para in/out
: @Referenciacion

```

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:

```
Metodos integrados:
Reverse @para texto
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

```
Funciones integradas:

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

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 0.0.8, 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