**ÍNDICE**

1 Pendiente o no soportado 6

1.1 RENAMES 6

2 Ideas 7

2.1 IVP 7

3 Multiples dialectos 8

4 Variables 9

4.1 Inicialización 9

4.2 Nombres duplicados 9

4.3 Estructura y gestión 9

4.4 Accesos 10

4.5 Colores 10

5 Issues 11

6 Arbol 12

7 Configuracion 13

8 Flags 14

9 15

**Índice de tablas, diagramas e ilustraciones**

**No se encuentran elementos de tabla de ilustraciones.**

Código 1: Ejemplo de codificación legible 11

**No se encuentran elementos de tabla de ilustraciones.**

**No se encuentran elementos de tabla de ilustraciones.**

# Pendiente o no soportado

## RENAMES

Pendiente de implementar

Hay que considerar el RENAMES THRU

# Ideas

Hacer un paquete especial para las pruebas. Este llama al paquete inicial y valida

Hacer un programa para cada sentencia cobol

MIPS Usage = CPU time \* Speed of the instruction processor

Speed of the instruction processor is 564.3 since we use z10.

<http://www.mainframesupport.dk/tips/tip1205.html>

<http://www-01.ibm.com/support/docview.wss?uid=swg1PM18842>

<https://listserv.uga.edu/cgi-bin/wa?A2=ind0611&L=CICS-L&P=209040>

# IVP

Cuando hemos cogido cada programa de IVP, nos guardamos si tiene mas de un juego de pruebas.

Cuando acabe el primer ciclo, empezamos con los siguientes, de esta forma no vamos cargando y descargando tablas.

Hay que considerar que hay diferentes programas. ¿Agrupar por ellos?

## Caso basico

La manera mas simple es incluir los resultados del IVP en el propio programa. Aquí no necesitamos generalización por que es para uso interno.

\*> SDP IVP caso\_uso objeto.método operador valor - desc

\*> Para indicar que es un comentario especial

SDP para indicar que es del producto

IVP para indicar que es un caso de IVP

Caso\_uso si existe, indica que hay varios casos de uso

Objeto clase que hay que investigar

Método método que hay que invocar

Valor valor esperado

Caso\_uso buscar un script que se llame igual y lo ejecutará antes de empezar

No enviamos datos al server

Si cuando test guardamos siempre el mismo usuario y las mismas fechas, se puede calcular el md5 y verificar los datos

Podemos coger el texto a sacar de SDPDesc

Necesito una lista de los programas y sus md5

En el proyecto de test o de prueba meter un parámetro para que cargue lo que ya hay chequeado y genere el nuevo en el mismo fichero: las firmas, el nombre, etc ….

# Multiples dialectos

Implementado a través de la configuración y la factoria.

Durante la fase de desarrollo, tomaremos como base OCCobol

# Variables

Para el tratamiento de las variables nos interesa conocer:

* Si tienen valor inicial o no
* Si se leen o se escriben o ambas

**Si** nos interesa conocer la picture ni el tamaño salvo en el caso de que necesitemos saber la memoria que consume el programa para los casos en que ocurre

Pic x(10) value “pepe”

## Inicialización

Si una variable esta inicializada hay que considerar si se inicializa en dos líneas con el carácter de continuación “-“ en la columna 7

Los filler se deben renombrar secuencialmente para poder identificar si son áreas inicializadas o no.

Este caso aplica a la construcción

01 DATA

02 F value …

02 F value ….

01 TB redefines DATA ….

Realmente TB contiene valores inicializados

## Nombres duplicados

Hay que considerar las construcciones del tipo VAR OF VAR-PADRE para poder identificar si un grupo se usa o no.

## Estructura y gestión

Por el caso de la duplicación de nombres, la tabla hash debe implementar una lista de padres y la variable debe tener un campo padre de nivel 01; es decir, deberíamos tener:

Id: Nombre de la variable

lstPadres:

padre de nivel 01

puntero al nivel superior

punteros a los niveles inferiores

punteros a los hermanos

Cuando una variable se lee, sus padres se deben marcar como utilizados pero de forma distinta a cuando se escribe.

Cuando una variable se escribe, sus hijos se escriben indirectamente

Para mantener la tabla Hash seria clave-objeto la clave el nombre de la variable y el objeto.

## Accesos

En el caso de lectura solo nos interesa saber la variable concreta, por que los hijos deberían ser filler si no se referencian

01 LINEA

03 CAMPO1

03 CAMPO2 pic x(30) value “un valor”

03 CAMPO3

Si hacemos MOVE LINEA TO ALGO y no usamos CAMPO2 entonces no haría falta declararlo

Por el contrario, si hacemos READ INTO LINEA, entonces hay que tener en cuenta que se ha perdido la inicialización de CAMPO2

Por el caso de la duplicación de nombres, la tabla hash debe implementar una lista de padres y la variable debe tener un campo padre de nivel 01; es decir, deberíamos tener:

~~Para optimizar el proceso, el seguimiento de padres, hijos y hermanos lo hacemos al final del proceso; es decir, durante el proceso solo marcamos leído o escrito la variable afectada y al final del proceso, propagamos esos datos por todo el árbol.~~

Para optimizar el proceso, lo hacemos cuando se referencia preguntando si ya ha sido procesada; es decir si se lee o se escribe se pregunta si ya esta marcado, y en ese caso ya se ha hecho la propagación.

Hay que considerar si las copys se utilizan o no. Es decir, si un nivel 01 no se utiliza y ese esta en una copy habría que indicar que esa copy no se utiliza, pero también hay que considerar que una copy podría tener varios niveles 01

## Colores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Leido | Escrito | I | No usado |
|  |  | Directo | Indirecto |  |
|  | Normal | Negrita |  | Cursiva gris |
|  |  |  |  |  |

# Issues

En la primera versión, hacemos la gestión de los issues en línea, en la siguiente hay que hacerlo en otros hilos, en modo productor consumidor

## Prioridades

Las propiedades son:

0 – Se procesan todas y se lanzan todas

> Mayor que 0, se procesan por orden de prioridad y por secuencia, se para si se activa la primera

## Resto

Se procesan todos los issues en el analizador.

Ademas de los issues, controlar el uso “No usado”. Este es otro issue

Hay que definir las reglas y la tabla para guardar tanto las reglas como los issues

Hay issues implícitos, es decir:

* Uso de variables
* Codigo muerto
* …

Tipo de issue (controlar 0 para las variables)

Codigo de issue

Subcodigo de issue ¿

Target: A que aplica

Objeto

Tipo de objeto

Linea inicio

Linea fin

# Arbol

A cada variable la enriquecemos con sus movimientos identificando si es una variable o un literal.

Cada CALL dinamico lo guardamos en la tabla TBCall y en el postproceso revisamos las variables.

Este proceso consume mas memoria y tiempo en process, pero nos ahorra el preproceso y las itereaciones posteriores.

Al contar las actualizaciones, si no coinciden con la lista de variables o valores entonces es que se ha producido movimientos no controlados, y entonces no podemos asegurar.

# Configuracion

La configuración es estatica; esto quiere decir que no se puede añadir o quitar nada via web, únicamente cambiar los valores.

Por eso, en la parte web no se muestra los limites ni las mascaras, simplemente la clave y el valor.

Cuando se modifique alguna, se envía al servidor para que lo valide y devuelva ok o no.

Si es necesario incluir mas claves, estas se hacen a mano y el sistema se encarga de gestionarlas.

# Flags

Es una tabla de 16 bytes

| Posicion | Tipo |
| --- | --- |
|  |  |
| 1 | Verificar profiler o hacer defecto |
| 2 | Perfilar o no |
| 3 | Hacer Suspend o no (Suspend actualiza SMF) |
| 4 |  |
| 5 | Modulo |
| 6 | Parrafo |
| 7 | Perform |
| 8 | Call |
| 9 | SQL |
| 10 | CICS |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |

# 