# Lenguaje

## Fin de fichero

Un programa fuente COBOL correcto puede acabar de una de las siguientes maneras:

1. Una sentencia END PROGRAM, indicando de manera explícita el fin del fichero fuente
2. Una sentencia EXIT, como consecuencia de un párrafo de control de salida
3. Una sentencia STOP RUN, indicando que el programa no se ejecutará más allá de esa sentencia
4. Una sentencia GOBACK, indicando que el programa no se ejecutará más allá de esa sentencia
5. Otra sentencia

Los cuatro primeros casos son controlados por el analizador, que inyectará el código adecuado para cada situación. El problema se presenta cuando el fin del programa está definido implícitamente al enco

# Bloques

Each DIVISION may consist of a variety of SECTIONs and each SECTION consists of one or more PARAGRAPHs. A PARARAPH consists of SENTENCEs, each of which consists of one or more STATEMENTs.

Desde el punto de vista del código, este se puede ver formado por:

Secciones : Una sección esta formada por uo o mas párrafos

Parrafo: Un párrafo esta formado uno o mas sentencias

Sentencia: UN conjunto de bloques

Bloque: Un conjunto de instrucciones que se ejecutarán como una unidad; en el sentido de que, si el flujo de programa alcanza ese punto, se ejecutarán todas ellas

Instrucción: Cada uno de los verbos del lenguaje

## Insercion de variables

Las variables necesarias para el fucnionamiento se alamacenan en Working

Puede no haber esa sección, entonces se mira si hay data división, si la hay hay otras cosas para ponerla en su sitio

La secuencia es local-storage, linkage, report, screen si no hay ninguna de ellas

Va encima de procedure

Si no hay data división

Va antes de procedure incluyendo data division

# Bloques

Each DIVISION may consist of a variety of SECTIONs and each SECTION consists of one or more PARAGRAPHs. A PARARAPH consists of SENTENCEs, each of which consists of one or more STATEMENTs.

Un bloque es un conjunto de instrucciones

# Collector

El collector es el responsable de capturar los diferentes eventos que se van produciendo durante la ejecución del programa impactando lo menos posible en el sistema objeto de análisis.

Por ello este componente se implementa mediante una arquitectura Cliente/Servidor cuyo escenario más sencillo se muestra en la siguiente figura:



Ilustración : Arquitectura básica Collector

La idea subyacente a esta arquitectura se basa en los siguientes criterios:

## Tipos de mensajes

Los mensajes enviados al servidor son mensajes en modo datagrama separados por el carácter delimitador punto y coma (;), de manera que en función del tipo de mensaje puede contener diferente información.

Los mensajes que se generan son:

* Módulo: Contiene la información relativa al consumo asociado a un determinado módulo.
* Parráfo: contiene la información relativa a los tiempos asociados a un determinado párrafo.
* Llamada: Contiene la informaci

### Módulo

Contiene la información relativa a los consumos realizados dentro de un determinado módulo.

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Tipo de mensaje.  Valor: **1** |
| 2 | Id Sesion | Contiene el identificador único de la sesión |
| 3 | Id Modulo | Contiene el identificador único del módulo.  Cuando este valor coincide con el identificador de la sesión los tiempos obtenidos indican los tiempos totales de ejecución del programa |
| 4 | Nombre | Nombre del módulo ejecutado |
| 5 | Elapsed | Tiempo total transcurrido en microsegundos |
| 6 | Kernel | Tiempo total de CPU consumido en modo Kernel |
| 7 | User | Tiempo total de CPU consumido en modo Usuario |
| 8 | Elapsed In | Tiempo en microsegundos consumidos dentro del módulo excluyendo todas las llamadas a módulos externos |
| 9 | Kernel In | Tiempo de CPU consumido en modo Kernel excluyendo el consumido en llamadas externas |
| 10 | User In | Tiempo de CPU consumido en modo Usuario excluyendo el consumido en llamadas externas |

### Párrafo

Contiene la información relativa a los consumos realizados dentro de un determinado módulo.

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Tipo de mensaje.  Valor: **2** |
| 2 | Id Sesion | Contiene el identificador único de la sesión |
| 3 | Id Modulo | Contiene el identificador único del módulo. |
| 4 | Nombre | Nombre del párrafo |
| 5 | Elapsed | Tiempo total transcurrido en microsegundos |
| 6 | Kernel | Tiempo total de CPU consumido en modo Kernel |
| 7 | User | Tiempo total de CPU consumido en modo Usuario |
| 8 | Elapsed In | Tiempo en microsegundos consumidos dentro del módulo excluyendo todas las llamadas a otros párrafos o módulos externos |
| 9 | Kernel In | Tiempo de CPU consumido en modo Kernel excluyendo el consumido en llamadas a otros párrafos o módulos externos |
| 10 | User In | Tiempo de CPU consumido en modo Usuario excluyendo el consumido en llamadas a otros párrafos o módulos externos |

### CALL

Contiene la información relativa a los consumos realizados durante la llamada a un módulo externo

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Tipo de mensaje.  Valor: **3** |
| 2 | Id Sesion | Contiene el identificador único de la sesión |
| 3 | Id Modulo | Contiene el identificador único del módulo. |
| 4 | Nombre | Nombre del párrafo |
| 5 | Elapsed | Tiempo total transcurrido en microsegundos |
| 6 | Kernel | Tiempo total de CPU consumido en modo Kernel |
| 7 | User | Tiempo total de CPU consumido en modo Usuario |
| 8 | Elapsed In | Tiempo en microsegundos necesario para realizar la carga del módulo |
| 9 | Kernel In | Tiempo de CPU en modo Kernel necesario para realizar la carga y asignación de recursos para el módulo llamado |
| 10 | User In | Tiempo de CPU en modo Usuario necesario para realizar la carga y asignación de recursos para el módulo llamado |

### Cobertura

Contiene la información relativa a los bloques de código que se han ejecutado durante la sesión

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Tipo de mensaje.  Valor: **0** |
| 2 | Id Sesion | Contiene el identificador único de la sesión |
| 3 | Id Modulo | Contiene el identificador único del módulo. |
| 4 | Cobertura | Tabla de indicadores con las marcas de ejecución de bloques.  Es una lista de caracteres donde el valor ‘1’ indica que el bloque asociado a su posición relativa dentro de la lista de caracteres se ha ejecutado |

|  |  |
| --- | --- |
|  |  |
| 1 | Tipo de mensaje |
| 2 | Identificador de la sesión |
| 3 | Identificador del modulo |
| 4 | Nombre del componente |
| 5 | Tiempo total consumido en el compon |
|  |  |
|  |  |

|  |
| --- |
| #!/bin/bash |
|  |
| mvn deploy:deploy-file -Durl=sftp://web.sourceforge.net:/home/project-web/emustudio/htdocs/repository -DrepositoryId=emustudio-repository -Dfile=./java-cup-runtime-0.11a.jar -DpomFile=./pom.xml |
|  |

GiTHUB:

https://github.com/

JavierGlez

Fjgg2185

[Fcojavier.glez@gmail.com](mailto:Fcojavier.glez@gmail.com)

<http://www.carlospinan.com/2014/03/11/instalando-opencobol-en-windows-7/>

Oracle

SYS y SYSTEM password jgg

APEX en el Puerto 9090

Acceso SYS/jgg

Sonar

Puerto 9000

Tomcat

Admin/jgg

# Datos tecnicos

Entrada de datos

El formato de un programa COBOL es posicional y distingue tres áreas:

Maquinas:

Serendipity cliente

Usuario: jgonzalez password: jgg la misma que para root

Collector:

SDPxTRPn ->

X Indica el sistema operativo

N indica si es de 32 o 64 bits

### Building DLLs

This page gives only a few simple examples of gcc's DLL-building capabilities. To begin an exploration of the many additional options, see the gcc documentation and website, currently at <http://gcc.gnu.org/>

Let's go through a simple example of how to build a dll. For this example, we'll use a single file myprog.c for the program (myprog.exe) and a single file mydll.c for the contents of the dll (mydll.dll).

Fortunately, with the latest gcc and binutils the process for building a dll is now pretty simple. Say you want to build this minimal function in mydll.c:

#include <stdio.h>

int

hello()

{

printf ("Hello World!\n");

}

First compile mydll.c to object code:

gcc -c mydll.c

Then, tell gcc that it is building a shared library:

gcc -shared -o mydll.dll mydll.o

That's it! To finish up the example, you can now link to the dll with a simple program:

int

main ()

{

hello ();

}

Then link to your dll with a command like:

gcc -o myprog myprog.c -L./ -lmydll

However, if you are building a dll as an export library, you will probably want to use the complete syntax:

gcc -shared -o cyg${module}.dll \

-Wl,--out-implib=lib${module}.dll.a \

-Wl,--export-all-symbols \

-Wl,--enable-auto-import \

# Maquina virtual

openSuSe 13.2 32 bits

disco swap /dev/sda1

discoduro /dev/sda2

Usuarios:

Jgonzalez373:jgg