<Cookbook – coding best practices>

|  |  |
| --- | --- |
|  |  |
| **Auteur(s)** | Alain DHEUR |
| **Responsable du document** | Alain DHEUR |
| **Etat** |  |

Mises à jour

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Version** | **Date** | **Description** |
|  | 1.0 | 24/07/2013 | Création du document |
|  | 1.1 | 02/08/2021 | Incorporation dans template NRB |
|  |  |  |  |

Validation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Version** | **Relu et validé par** | **Date** | **Etat** |
|  |  |  |  |  |
|  |  |  |  |  |

Liste de diffusion

|  |  |  |
| --- | --- | --- |
|  | **Nom** | **Société** |
|  |  |  |
|  |  |  |

Table des matières

[1 Objectif 4](#_Toc181267135)

[1.1 Champ d’application 4](#_Toc181267136)

[1.2 Préalables 4](#_Toc181267137)

[2 La programmation structurée 5](#_Toc181267138)

[2.1 Avantages 5](#_Toc181267139)

[2.2 Modélisation 5](#_Toc181267140)

[2.3 La séquence 6](#_Toc181267141)

[2.4 L’alternative 6](#_Toc181267142)

[2.5 Les Itérations 6](#_Toc181267143)

[2.6 Exemple d’utilisation 7](#_Toc181267144)

[3 L’écriture du code 8](#_Toc181267145)

[3.1 Eviter les trucs ou astuces 8](#_Toc181267146)

[3.2 Commentaires 9](#_Toc181267147)

[3.2.1 Commentaires explicatifs 9](#_Toc181267148)

[3.2.2 Les instructions en commentaires 10](#_Toc181267149)

[3.3 Privilégier les tests positifs 11](#_Toc181267150)

[3.4 Boucles 13](#_Toc181267151)

[3.4.1 Variable d’itération 13](#_Toc181267152)

[3.4.1.1 Déclaratation 13](#_Toc181267153)

[3.4.1.2 Utilisation de la variable en sortie de boucle 13](#_Toc181267154)

[3.4.1.3 Modification de la variable d’itération 14](#_Toc181267155)

[3.4.2 Définition de constante 15](#_Toc181267156)

[3.5 Eviter les mélanges de mode 16](#_Toc181267157)

[3.6 Indentation 17](#_Toc181267158)

[3.7 Tests imbriqués 18](#_Toc181267159)

[3.8 Boucle de traitements d’enregistrements 19](#_Toc181267160)

[3.9 Déclarations de variables 20](#_Toc181267161)

[3.10 Utilisations de tableaux 20](#_Toc181267162)

[3.10.1 Définition 20](#_Toc181267163)

[3.10.2 Manipulation de zones indicées 21](#_Toc181267164)

[3.11 Quelques spécificités PL/1 22](#_Toc181267165)

[3.11.1 Eviter les SUBSTR quand cela est possible 22](#_Toc181267166)

[3.11.2 Utilisation d’indices en BIN FIXED (31); 23](#_Toc181267167)

[3.11.3 Structure Dynamique 23](#_Toc181267168)

[3.12 Variables pré-processeur 27](#_Toc181267169)

[3.13 Relire le code 28](#_Toc181267170)

# Objectif

## Champ d’application

Le but de ce document est aussi de formaliser toute une série de consignes souvent émises uniquement de manière verbale.

La logique de programmation associée à la rédaction d’un code lisible va :

* Accentuer l'aspect qualitatif de la programmation
* Faciliter la lisibilité et donc la maintenance d'applicatifs
* Appréhender la logique de programmation d'une manière commune

Pour rappel, les qualités principales d’un programme sont :

* Facile à lire -> à comprendre -> maintenance plus aisée
* La structure logique doit être directement visible
* Modulaire
* Bon compromis avec l’aspect performance

## Préalables

**Avant de programmer**

* Prendre le temps de la réflexion
* Poser la structure
* Utiliser un approche Top-down
* Opter pour la programmation structurée
* Vérifier l’existence de modules permettant d’effectuer certaines fonctionnalités plutôt que récrire l’existant.

# La programmation structurée

## Avantages

* Permet d’aborder la rédaction d’un programme de façon modulaire et ordonnée
* Empêche l’utilisation de l’instruction GO TO si elle est correctement respectée
* Donne une structure logique au programme
* Est indépendante d’un langage de programmation

## Modélisation

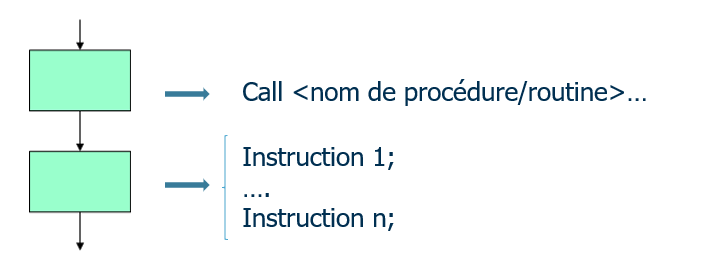
* Organigramme (alt: pseudocode)
* Les symboles



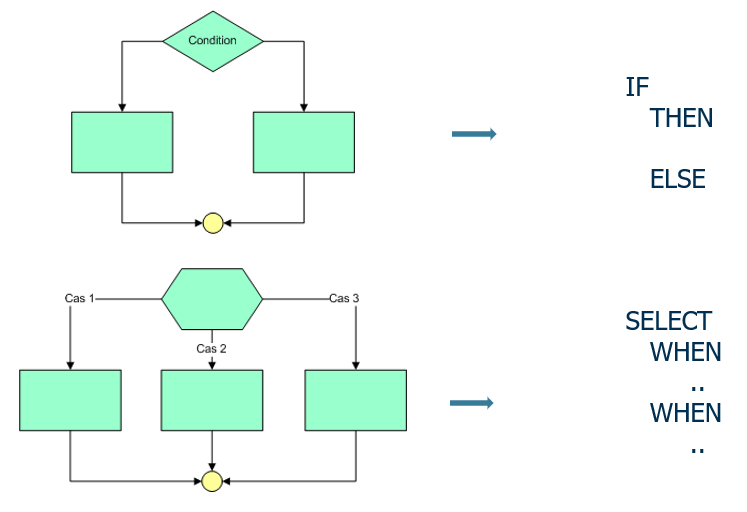
connecteur



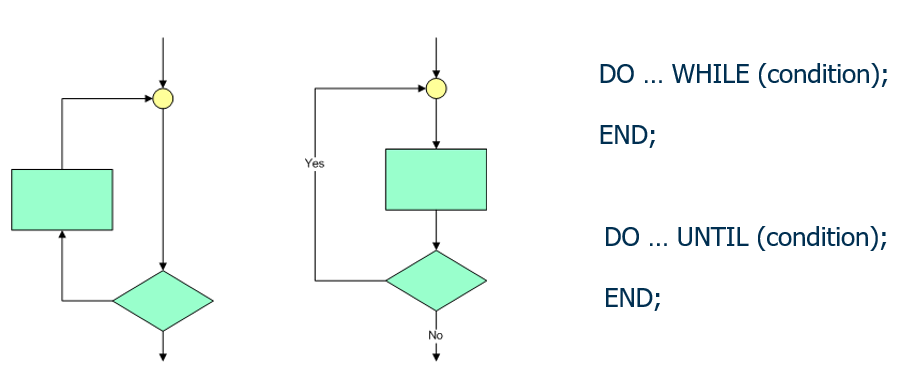
## La séquence



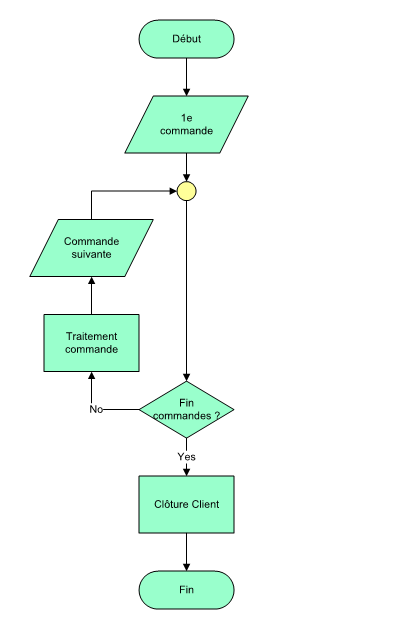
## L’alternative



## Les Itérations



## Exemple d’utilisation



🡪 coding lisible (chaque bloc sera développé de manière modulaire).

CALL READ\_ORDER ; /\* lecture 1e commande \*/

DO WHILE (EOF\_ORDER = OFF );

CALL PROC\_ORDER; /\* traitement de la commande \*/

CALL READ\_ORDER; /\* commande suivante \*/

END;

CALL CLOT\_CUSTOM; /\* Clôture relevé client \*/

# L’écriture du code

Les points suivants sont à considérer lors de l’écriture du code :

* Absence d’astuces
* Commentaires
* Utilisation prioritaire de tests positifs
* Lisibilité
* Boucles
* Définitions de tableaux
* Déclarations de variables
* Portée des variables
* Suivi des modifications

## Eviter les trucs ou astuces

Il est préférable d'utiliser un code simple et d'éviter de recourir à des astuces. Si le raccourci dans l'écriture d'un programme met en œuvre un raisonnement trop subtil, il est préférable d'allonger le code et de le mettre aisément à la portée du lecteur.

## Commentaires

### Commentaires explicatifs

**Consigne**

Ils permettent d’offrir une lisibilité du code, ils doivent donc être présents en suffisance.

Ils doivent cependant rester clairs et concis, un programme n’est pas non plus un roman, les commentaires doivent servir à clarifier le code, sans plus.

Chaque paragraphe ou procédure doit avoir un commentaire , idéalement placé devant, décrivant succinctement la fonction de la procédure

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* VALIDATION SYNTAXIQUE CLOTURE DE CAISSE \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FA\_CTC: PROC REORDER ;

TRACE = $LTNMPRG !! 'FA\_CTC';

Un commentaire plus spécifique peut être associé à une instruction ou à un groupe d’instructions :

CALL FA\_CPR; /\* contrôle coupures \*/

/\* Adaptations pour passer en mode confirmation \*/

/\*----------------------------------------------\*/

IF CDRTPGM\_W = '' & SPA.NBPAGE1 > 0

& SUBSTR (SPA.CDEXTOP,4,3) ^= 'CNF'

THEN

DO;

SUBSTR (SPA.CDEXTOP,4,3) = 'CNF';

CALL DEF\_PGNAM;

END;

Des commentaires superflus, qui sont généralement qualifiés de « bruit », ont cependant l’effet inverse, en plus de ne rien apporter à la clarté, ils peuvent au contraire rendre le code moins lisible.

Le commentaire ci-dessous n’apporte pas grand chose

/\* Ajouter 1 à I \*/

I = I + 1 ;

Les commentaires ne doivent pas faire office de PV de reunion.

/\* ajout d’un test à la demande de AAA qui en essayant ..

a trouvé judicieux de .., bien que BBB pensait .. \*/

IF CDRTPGM\_W = '' & S PA.NBPAGE1 > 0

THEN

DO;

SUBSTR (SPA.CDEXTOP,4,3) = 'CNF';

CALL DEF\_PGNAM;

END;

### Les instructions en commentaires

**Consigne**

Le code périmé ne doit pas être laissé en commentaire, à terme ces instructions mises sans effet vont polluer la lecture du programme ; dans une très grande majorité des cas, personne n’osera les réactiver.

Le gestionnaire de source qui historise les versions va vous permettre de retrouver les instructions de ces versions antérieures.

## Privilégier les tests positifs

**Consigne avec exception selon le contexte**

« L’œil humain » est beaucoup plus réceptif à la lecture de tests positifs, ceux-ci sont donc à privilégier au maximum

Exemple

|  |
| --- |
| IF ^BOVALID **🡨 privilégier un test positif**  THEN  BLOC D’INSTRUCTION 1  ELSE  BLOC D’INSTRUCTION 2; |

De meilleures solutions seraient :

|  |  |
| --- | --- |
| IF BOVALID = ‘0’B  THEN  BLOC D’INSTRUCTION 1  ELSE  BLOC D’INSTRUCTION 2; | IF BOVALID  THEN  BLOC D’INSTRUCTION 2  ELSE  BLOC D’INSTRUCTION 1 ; |

Ceci est encore plus vrai lors de l’imbrication avec des opérateurs logiques

|  |
| --- |
| IF (A ^= B) ! (C = D) & ^BOEXIST  THEN  BLOC 1  ELSE  BLOC 2; |

Ou cela devient pratiquement illisible.

L’utilisation de l’instruction « THEN; » peut permettre de pallier

|  |  |
| --- | --- |
| **Test négatif** | **Test positif avec utilisation du THEN** ; |
| IF (Test négatif très compliqué)  THEN  Bloc d’instructions ; | IF (Test positif plus lisible)  THEN;  ELSE  Bloc d’instructions; |

De manière générale, éviter le NOT global:

|  |
| --- |
| **Test avec NOT global** |
| IF ^(Test positif)  Then  Bloc d’instructions ; |

Exception :

Il est bien sûr évident qu’un test négatif simple reste acceptable et même parfois souhaité.

|  |
| --- |
| **Test négatif simple** |
| IF CDRTPGM ^= ‘’  Then  Bloc d’instructions ; |

## Boucles

### Variable d’itération

#### Déclaratation

**Consigne avec exception selon le contexte**

La variable d’itération de la boucle doit idéalement être définie en BINARY FIXED (31) ; c’est le format qui permet la meilleure optimisation du code.

#### Utilisation de la variable en sortie de boucle

**Consigne**

La valeur de la variable d’itération au-delà de la boucle est souvent sujette à discussion, quelle est sa valeur, comment le compilateur ordonnance-t-il ses tests ?

Dans l’exemple suivant, sommes-nous sûrs que I a été incrémenté ? Et avec la prochaine version du compilateur ? Et en cas de portée vers un autre langage ? Jouons donc la sécurité

Exemple : mettre un ‘\*’ dans le premier élément du tableau ≠’\*’

|  |  |
| --- | --- |
| BOFIN = ‘0’B;  DO I = 1 TO 1000  WHILE (BOFIN = ‘0’B);  IF TABLEAU(I) ^= ‘\*’  THEN  BOFIN = ‘1’B;  END;  IF BOFIN = ‘1’B  THEN  TABLEAU (I-1) = ‘\*’; | J = 0;  BOFIN = ‘0’B;  DO I = 1 TO 1000  WHILE (BOFIN = ‘0’B);  IF TABLEAU(I) ^= ‘\*’  THEN  DO;  BOFIN = ‘1’B;  J = I;  END;  END;  IF BOFIN = ‘1’B  THEN  TABLEAU (J) = ‘\*’; |

#### Modification de la variable d’itération

**Consigne**

Eviter de modifier la variable d’une boucle, soit pour sortir de la boucle, en forçant la variable d’itération à son maximum, soit pour toute autre raison

|  |
| --- |
| /\* comptage du nombre de caractères avant le 1er astérisque \*/  /\*---------------------------------------------------------\*/  NONASTER = 0;  DO I = 1 TO 1000;  IF TABLEAU(I) = ‘\*’  THEN  I = 1000; 🡨 modification de la variable d’itération  ELSE  NONASTER = NONASTER + 1;  END;  \_ \_ \_  NONASTER = 0;  BOFIN = ‘0’B;  DO I = 1 TO 1000 WHILE (BOFIN = ‘0’B);  IF TABLEAU(I) = ‘\*’  THEN  BOFIN = ‘1’B;  ELSE  NONASTER = NONASTER + 1;  END; |

Dans des cas plus complexes, cela peut donner des résultats inattendus voire des bouclages.

### Définition de constante

**Consigne**

Une telle définition va consommer inutilement des ressources, la valeur exprimée étant toujours la même.

Exemple 1 : le calcul du volume de la sphère est réalisé par la formule V = (4תּR³)/3

|  |  |
| --- | --- |
| PI = 3.1415927 ;  DO R = 1 to 1000;  V = (4\*PI\*(R\*\*3))/3 ;  PUT SKIP LIST (R,V);  END; | PI = 3.1415927 ;  CONSTANT = 4 \* PI / 3;  DO R = 1 to 1000;  V = CONSTANT \* (R\*\*3));  PUT SKIP LIST (R,V);  END; |
| La constante 4 \* PI / 3 est recalculée 1000 fois, | Permet de ne calculer la constante qu’une seule fois |

Exemple 2 : calcul d’une ristourne

|  |  |
| --- | --- |
| MTRISTO = 5;  DO I = 1 to 1000;  PRIX\_RIST = PRIX\_NET \*  (1-MTRISTO/100);  END; | MTRISTO = 5 ;  CONSTANT = 1-(MTRISTO/100);  DO I = 1 to 1000;  PRIX\_RIST = PRIX\_NET \* CONSTANT;    END; |
| La constante est recalculée 1000 fois, | Permet de ne calculer la constante qu’une seule fois |

## Eviter les mélanges de mode

Un champ défini en caractère ne doit pas se voir assigner une zone numérique définie en decimal ou en binary ; une telle assignation doit passer par une routine prédéfinie ou par des variables intermédiaires de type pic(ture).

De même un champ défini en decimal ou en binary  ne doit pas être assigné directement dans un champ de type caractère.

Les mêmes contraintes s’appliquent en ce qui concernent les tests, une zone caractère ne doit pas être testée par rapport à une zone numérique et inversement.

## Indentation

**Consignes**

Mettre en concordance les instructions de type – bloc - de même niveau :

DO ….. et END

SELECT et END

WHEN… et OTHER d’un SELECT

|  |  |
| --- | --- |
| **Exemples idéaux** | |
| IF BOFIN = ‘1’B  THEN  DO;  TABLEAU (J) = ‘\*’;  CALL PROC1;  END;  ELSE  DO;  TABLEAU (J) = ‘0’;  CALL PROC2;  END; | IF BOFIN = ‘1’B  THEN DO;  TABLEAU (J) = ‘\*’;  CALL PROC1;  END;  ELSE DO;  TABLEAU (J) = ‘0’;  CALL PROC2;  END; |

|  |  |
| --- | --- |
| Exemple acceptable | Exemple incorrect |
| IF BOFIN = ‘1’B THEN  DO;  TABLEAU (J) = ‘\*’;  CALL PROC1;  END;  ELSE  DO;  TABLEAU (J) = ‘0’;  CALL PROC2;  END; | IF BOFIN = ‘1’B  **THEN** DO;  TABLEAU (J) = ‘\*’;  CALL PROC1;  **END**; 🡨 non aligné avec son ‘DO’  **ELSE** DO;  TABLEAU (J) = ‘0’;  CALL PROC2;  **END**; |

Chaque bloc doit être indenté par rapport à un éventuel bloc parent :

|  |  |  |
| --- | --- | --- |
| **Incorrect** | **Raison de l’ « erreur »** | **Correct** |
| SELECT (A)  WHEN (‘1’)  DO;  WK1 = ‘A’;  WK2 = ‘1’;  END;  WHEN (‘2’)  DO;  ...  END;  OTHER;  END; | 🡨 doit être indenté par rapport au bloc WHEN  🡨 doit être aligné avec WHEN | SELECT (A)  WHEN (‘1’)  DO;  WK1 = ‘A’;  WK2 = ‘1’;  END;  WHEN (‘2’)  DO;  ...  END;  OTHER;  END; |

Important : quelle que soit la méthode choisie, il est important de garder au maximum un code harmonisé.

Remarque : L’alignement du THEN et du ELSE correspondant représente un plus ; le IF pouvant être considéré comme une ouverture de bloc, même si en PL/1, le END-IF n’existe pas.

## Tests imbriqués

**Consigne**

Le nombre de tests imbriqués ne devrait pas dépasser 3, au-delà, la découpe en procédures devrait être envisagée, quoi de plus désagréable que de lire un code dont les instructions sont comprises entre les colonnes 50 et 72.

## Boucle de traitements d’enregistrements

**Consigne**

Le processus logique de traitement d’enregistrements venant d’un fichier (DB ou autre) doit respecter le schéma suivant :

|  |
| --- |
| Lecture enregistrement 1  Boucle jusqu’à fin des enregistrements  Traitement d’un enregistrement  Lecture enregistrement suivant  Fin de boucle |

Dans le cas du traitement du résultat d’une exploration DB, la procédure de lecture doit fournir l’élément à traiter, cette même procédure de lecture doit vérifier si il est nécessaire de procéder à un nouvel accès DB pour obtention du groupe d’enregistrements suivant.

Le « code » doit être développé de manière modulaire, la fonction de lecture quelle que soit sa complexité ne pilote pas le traitement, elle doit pouvoir se résumer à un simple appel d’une procédure.

Des méthodes de lecture de l’élément suivant d’un tableau d’exploration dans une boucle sont décrites dans la partie annexe de ce document.

## Déclarations de variables

**Consigne**

Les déclarations de variables locales sont à regrouper en début de paragraphe avant toute instruction exécutable, les variables globales sont bien sûr à placer à l’endroit adéquat (cfr charte contrôle qualité).

**Consigne avec exceptions selon le contexte**

Les variables globales sont à privilégier, les variables locales devraient se limiter aux indices, aux paramètres (arguments), et à quelques variables de travail spécifiques à une fonction.

## Utilisations de tableaux

### Définition

En plus de la description du tableau, définir un (des) indice (s) spécifique (s) au tableau qui seront exclusivement utilisés à cet effet (le COBOL intègre en partie cette fonctionnalité par la clause INDEXED BY).

Une variable reprenant le nombre d’éléments actif dans le tableau peut également se révéler très utile.

DCL 1 TBCOMB1($NBCOMBI),

2 NOBIENS FIXED BIN (31),

2 NOGARAN FIXED BIN(31),

2 DADBFPR PIC '(8)9',

2 DAFNFPR PIC '(8)9',

2 NOIDGAR CHAR(03),

2 MTBASVR FIXED(13,2), …

DCL I\_COMB1 FIXED BIN (31);

DCL NBCOMB1 FIXED BIN (31);

I\_COMB1 ne devra être utilisé que pour référencer un élément de TBCOMB1, cela évite de se perdre avec des indices génériques (de type I, J) dont la valeur n’est jamais très sûre.

### Manipulation de zones indicées

Tenter d’éviter un trop grand nombre de manipulations de variables indicées, chaque calcul d’adressage est couteux en temps CPU, l’utilisation d’une zone intermédiaire peut permettre une optimisation.

DCL 1 TBCOMB1($NBCOMBI),

2 NOBIENS FIXED BIN (31),

2 NOGARAN FIXED BIN(31),

...

2 MTBASVR FIXED(13,2), …

DCL I\_COMB1 FIXED BIN (31);

DCL NBCOMB1 FIXED BIN (31);

DCL TBCOMB1\_CUR LIKE TBCOMB1 ;

TBCOMB1\_CUR = TBCOMB1 (I\_COMB1);

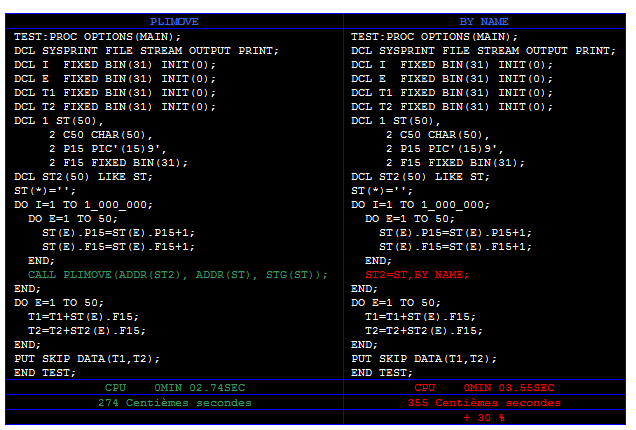
**TBCOMB1\_CUR.NOBIENS** = NOBIENS;

...

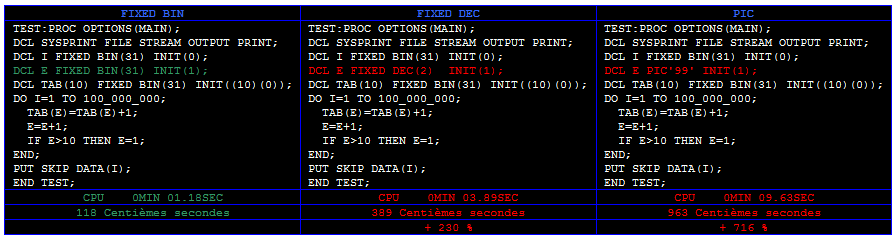
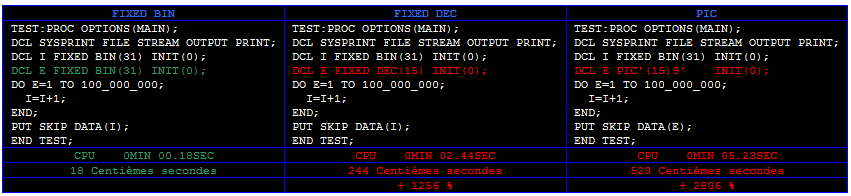
## Quelques spécificités PL/1

### Eviter les SUBSTR quand cela est possible

* Tenter d’éviter au maximum l’utilisation de SUBSTR, leur préférer des redéfinitions,
* Gain de temps sur les BY NAME,
  + Uniquement si structures IDENTIQUES



### Utilisation d’indices en BIN FIXED (31);



### Structure Dynamique

Lorsqu’on crée un interface dans laquelle il y a une partie Dynamique et une partie Fixe, il faut obligatoirement placer la partie Dynamique à la fin de la définition.

*Voici un exemple concret :*

(exemple ci-joint la sous-structure POL\_CHAN\_RR).

C’est une structure d’appel à Flexi-tarif. L’existant c’est la sous-structure en vert. Le test que j’ai fait, suite à divers strobes et au PM que tu viens d’ouvrir récemment, j’ai déplacé cette sous-structure à la fin de l’interface (en rouge).

DCL 1 CPG01J\_I01 BASED(CPG01J\_I\_PTR) UNALIGNED,

2 PARAMETERS,

3 PRDU\_CD VAR CHAR(32), /\* Product Code \*/

3 SPRDU\_CD VAR CHAR(32), /\* Sub Product Code \*/

3 PRCG\_TP\_CD CHAR(15), /\* Pricing Type Code \*/

3 PRCG\_RF\_TP\_CD CHAR(15), /\* Type of reference \*/

3 PRCG\_RF\_DS VAR CHAR(255), /\* Desc. pricing ref \*/

3 TARIF\_CD VAR CHAR(32), /\* Tarif Code \*/

3 CALCU\_DT CHAR(10), /\* Calculation Date \*/

3 SAV\_PRCG\_FL BIT(1), /\* Flag save pricing \*/

3 CRT\_USR\_CD CHAR(8), /\* User logged in \*/

3 APP\_NM CHAR(20), /\* Application name \*/

3 ENTPR\_DATA\_OWN\_CD CHAR(1), /\* Enterprise owner cd \*/

2 BODY,

...

...

5 POL\_RISK\_RR,

6 RISK\_CAT\_CD CHAR(15),

6 RISK\_TYP\_CD CHAR(15),

6 RISK\_VL\_CD CHAR(15),

/\*5 CHAN\_CT FIXED BIN(31),

5 POL\_CHAN\_RR(CPG01J\_I01\_CHAN\_CT\_I REFER(CPG01J\_I01.CHAN\_CT)),

6 CHAN\_CD CHAR(15), /\*Chanel code\*/

/\* 6 CHAN\_TYP\_CD CHAR(15), /\*Chanel type code\*/

5 PCG\_TARIF\_MOD,

6 POL\_TARIF\_MOD(10),

7 POL\_TARIF\_MOD\_TP\_CD CHAR(15), /\*policy tarif modification

type code\*/

7 TARIF\_MOD\_TP\_CD CHAR(15), /\*Modification tarifaire type

code\*/

7 TARIF\_MOD\_SBTP\_CD CHAR(15), /\*tarifaire subtype code \*/

7 TARIF\_MOD\_CD CHAR(15), /\*Mod. tarifaire code\*/

7 TARIF\_MOD\_AM FIXED DEC(19,8), /\*Mod. tarif. amount\*/

7 TARIF\_MOD\_VL FIXED DEC(15,4), /\*Mod Tarif. value\*/

7 TARIF\_MOD\_RT FIXED DEC(9,8), /\*Mod. tarif. rate\*/

7 TARIF\_MOD\_FL BIT(1), /\*Mod. tarif. flag\*/

5 POL\_RF,

6 PLCY\_RF\_TP\_CD CHAR(15), /\*Policy reference type code\*/

6 PLCY\_RF\_DS VAR CHAR(255), /\*P. ref. Description\*/

5 POL\_AMDT,

6 AMDT\_TP\_CD CHAR(15),

6 AMDT\_RF VAR CHAR(128),

6 AMDT\_VER\_RF CHAR(10),

6 AMDT\_STS\_CD CHAR(15),

6 AMDT\_DS VAR CHAR(255),

6 BSNS\_STA\_TS CHAR(26),

6 CRT\_TS CHAR(26),

5 POL\_SEGMT,

6 POL\_SGMT\_TP\_CD CHAR(15),/\*policy segmentation type cd\*/

6 POL\_SGMT\_CD CHAR(15), /\*Policy segmentation code\*/

5 BSNS\_STA\_TS CHAR(26), /\*Business start timestamp\*/

5 CRT\_TS CHAR(26), /\*Creation timestamp\*/

5 SBSCTN\_DT CHAR(10), /\*Subscription date\*/

5 CHAN\_CT FIXED BIN(31),

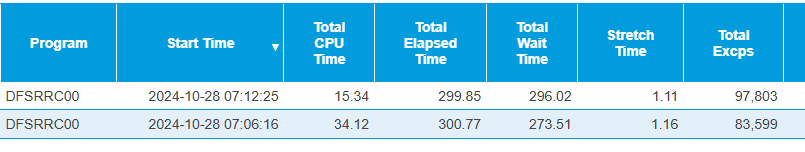
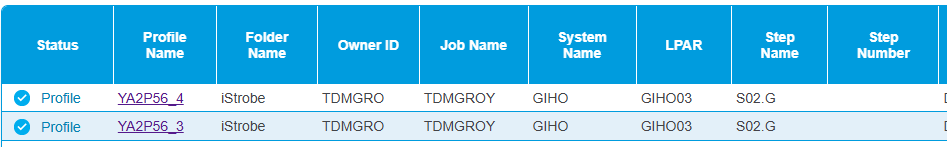
5 POL\_CHAN\_RR(CPG01J\_I01\_CHAN\_CT\_I REFER(CPG01J\_I01.CHAN\_CT)),

6 CHAN\_CD CHAR(15), /\*Chanel code\*/

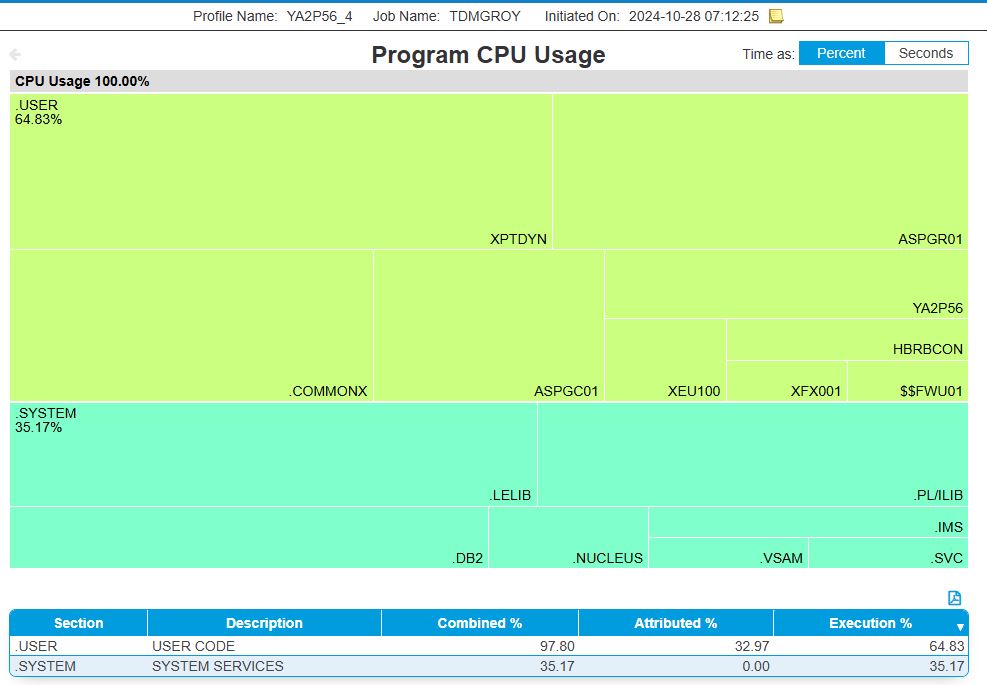
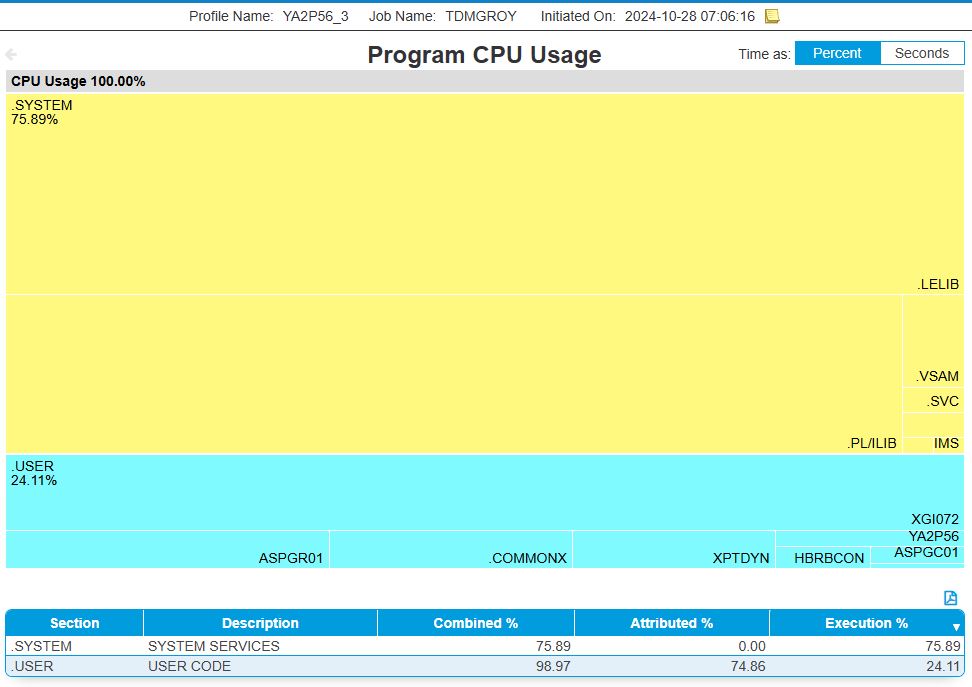
6 CHAN\_TYP\_CD CHAR(15); /\*Chanel type code\*/

*Analyse Strobe  :*

On passe d’un temps de « 34.12 » à « 15.34 », ce qui correspond à un facteur de 2



Plus en détail, avec la 1ère version de l’interface, on voit que le système prend 76% du temps CPU et 24% pour le Programme. Et avec la 2ième version de l’interface, c’est 35% Pour le PGM et 35 pour le Systéme.



*Conclusion* :

Dans une interface Mixte, si la partie Dynamique est au milieu de la partie Fixe, le Système doit recalculer « à l’exécution » l’adresse des variables situées après la Partie Dynamique.

## Variables pré-processeur

**Eviter au maximum l’utilisation de variables pré-processeur**

**Consigne**

Enlever l’utilisation des variables préprocesseur dans le code

DO I = 1 TO $NBELEMT;

. . .

END ;

$NBELEMT a servi à la définition de tableaux au niveau des pages et de la spa et justifie tout-à-fait son existence ; par contre son utilisation dans le programme doit être évitée, comme suit :

*Partie déclaration :*

DCL NBELEMT BIN FIXED (31);

*Dans la racine,*

NBELEMT = $NBELEMT;

*La boucle devient donc*

DO I = 1 TO NBELEMT;

. . .

END ;

Ce mode de travail est aussi utile pour tester, au lieu de remplir des combos avec 50 éléments ou d’encoder une multitude d’éléments pour tester un scrolling, …, il suffit de modifier la valeur de cette nouvelle variable,

*Dans la racine,*

NBELEMT = $NBELEMT;

NBMAXPG = $NBMAXPG ;

NBELEMT = 3 ; /\* pour test combo remplie \*/

NBMAXPG = 5 ; /\* max 5 éléments par page : test scrolling \*/

Une telle modification au niveau de la variable préprocesseur aurait été TRES lourde.

## Relire le code

**Recommandation**

Bien que cela paraisse fastidieux, relire au moins une fois son code permet de mettre en évidence :

* certaines « fautes de frappe », mais qui ne génèrent pas d’erreurs de compilation :

LTTYPBI = ‘abcd’ ;

CDTYPRI = 2 ; 🡨 à la relecture, la question se posera inévitablement, cdtypri ou cdtypbi (alors que les 2 existent)

LTDSCBI = ‘efgh’ ;

CDDSCBI = 7 ;

* de se rendre compte immédiatement qu’un paragraphe est dur à comprendre, et dans ce cas le réécrire directement
* de détecter des erreurs de logique

et finalement de gagner beaucoup plus de temps que celui passé à la re-lecture.