POLYTECHNIQUE
MONTRÉAL

Le langage CLE-2000

A. Hébert

2020/05/29

POLYTECHNIQUE MONTRÉAL

Table des matières

Le langage CLE-2000
Règles générales
Déclarations
Variables CLE-2000
Le script fact
Appel d'un module de calcul
Instructions de transfert de variables
Procédures
Boucles et instructions conditionnelles
Le module UTL :
Le module :=
Les modules utilitaires - suite

Le langage CLE-2000

- Le noyau GANLIB5 des codes DRAGON5 et DONJON5 comporte trois composantes logicielles
  - CLE-2000: un macro-langage
  - LCM/XSM: des application programming interfaces (API) pour permettre l'utilisation de tables associatives et de listes hétérogènes (depuis ANSI C et Fortran)
  - des modules utilitaires permettant de supporter les tables associatives et les listes hétérogènes.
- CLE-2000 est utilisé pour superviser le flot de donnée entre les modules des codes DRAGON5 et DONJON5. Il permet de mettre au point les schémas de calcul et de gérer les fichiers de données.
- CLE-2000 permet de manipuler des variables (définition et utilisation) ainsi que des instructions de contrôle du flot de donnée (boucles, instructions de branchement conditionnels, etc.).
- CLE-2000 permet de manipuler les fichiers et les structures de données utilisées par les modules des codes DRAGON5 et DONJON5.
- CLE-2000 permet de contrôler la procédure d'exécution et d'appeler les modules des codes DRAGON5 et DONJON5. Une procédure CLE-2000 peut en appeler une autre.
- Pour les utilisations plus spécialisées (couplage multiphysique), il est possible d'appeler une procédure CLE-2000 depuis une fonction ANSI C.
- CLE-2000 a été conçu par le Prof. Robert Roy de l'EPM (Département de Génie Informatique et de Génie Logiciel).
- La version actuelle de CLE-2000 représente environ 6000 lignes de code ANSI C.

Règles générales

- Les fichiers script CLE-2000 (`*.x2m` ou `*.c2m`) sont des fichiers ASCII UNIX pas des fichiers DOS.
- Une ligne de code CLE-2000 ne peut dépasser 132 caractères ASCII
- Seuls les 120 premiers caractères ASCII sont significatifs
- Tous les caractères ASCII suivant un "`*`" ou un "`!`" sont considérés commentaires
- Tous les caractères ASCII entre "`(*`" et "`*)`" sont considérés commentaires
- La casse des caractères est significative
- Le caractère espace " " est le séparateur. Il est obligatoire.
- Le caractère "tab" est interdit (désactiver les tab automatiques sur l'éditeur).
- Le script principal a un suffixe "`.x2m`". Les scripts appelés ont un suffixe "`.c2m`".
- Si un script est contenu dans un fichier UNIX "`monscript.x2m`" ou "`monscript.c2m`", son nom CLE-2000 est "`monscript`". Ce nom ne peut pas comporter plus de 12 caractères.
- Une procédure CLE-2000 peut appeler une (ou plusieurs) procédure CLE-2000 (incluant elle-même) ou un (des) module des codes DRAGON5/DONJON5.
- Les variables CLE-2000 sont transmises à un module ou à une procédure par les symboles "`<<...>>`" (envoi) et "`>>...<<`" (récupération).
- Il est également possible d'échanger des fichiers et/ou des objets LCM avec un module ou avec une procédure CLE-2000.
  - Les objets LCM sont des structures de données formées de tables associatives et/ou de listes hétérogènes.
  - Les objets LCM sont gardées en mémoire ou sur fichier à accès direct (XSM)
  - Les objets LCM sont accessibles en lecture seule, création et/ou modification depuis les modules des codes DRAGON5/DONJON5.

Déclarations

Le langage CLE-2000 oblige la déclaration de tous les objets utilisés sans un script

```
REAL Pitch ;
STRING Library := "DLIBJ3_281" ;
SEQ_BINARY TRKFILN2 ;
SEQ_ASCII _TABLE1 :: FILE './MY_TABLE1.txt' ;
LINKED_LIST TABLE1 ;
XSM_FILE TABLE2 :: FILE './MY_TABLE2.xsm' ;
MODULE UTL: END: ;
PROCEDURE sub_proc ;
```

- `Pitch` et `Library` sont des variables CLE-2000
- `TRKFILN2` est un fichier séquentiel binaire créé et détruit par CLE-2000 dont le nom CLE-2000 est `TRKFILN2`
- `_TABLE1` est un fichier séquentiel ascii créé par CLE-2000 dont le nom CLE-2000 est `_TABLE1` et le nom UNIX est `MY_TABLE1.txt`. Il est possible de récupérer ce fichier à la fin de l'exécution de CLE-2000 à l'aide d'un script `.save`. Le nom UNIX n'est pas limité à 12 caractères.
- `TABLE1` est un objet LCM stocké en mémoire par le noyau GANLIB5 et contenant une combinaison de tables associatives et de listes hétérogènes.
- `TABLE2` est un objet XSM stocké sur disque à accès direct par le noyau GANLIB5 et contenant une combinaison de tables associatives et de listes hétérogènes. Son nom UNIX est `MY_TABLE2.xsm`.
- `UTL:` et `END:` sont des modules de calcul appelés par CLE-2000
- `sub_proc` est un script CLE-2000 dont le nom UNIX est `sub_proc.c2m`

Variables CLE-2000

CLE-2000 permet de déclarer des variables de différent types. Par exemple:

```
REAL a b c r ;
```

Calculateur de CLE-2000

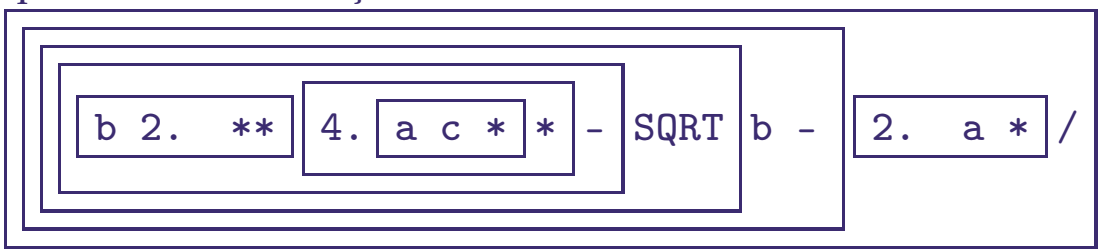
CLE-2000 possède un calculateur (commande `EVALUATE`) qui opère en notation polonaise inversée (RPN). Par exemple, pour calculer

$$r = \frac{1}{2a}\left(-b + \sqrt{b^2 - 4ac}\right)$$

on écrira

```
EVALUATE r := b 2. ** 4. a c * * - SQRT b - 2. a * / ;
```

qui s'évalue de la façon suivante:



Le script fact

Calcul récursif du factoriel d'un nombre en CLE-2000

- Le script `fact` doit être copié dans un fichier ASCII nommé `fact.c2m`.
- La signature du script `fact` est: `fact :: <<n>> >>n_fact<< ;`

```
!
! Example of a recursive procedure.
!
!  input  to  "fact": *n*
! output from "fact": *n_fact*
!
 INTEGER   n n_fact prev_fact ;
  ::  >>n<< ;
 IF n 1 = THEN
   EVALUATE n_fact := 1 ;
 ELSE
   EVALUATE n := n 1 - ;
   ! Here, "fact" calls itself
   PROCEDURE fact ;
   fact ::   <<n>>  >>prev_fact<< ;
   EVALUATE n_fact := n 1 + prev_fact * ;
 ENDIF ;
  :: <<n_fact>> ;
 QUIT " Recursive procedure *fact* XREF " .
```

Le script fact

Exécution du script `fact` en CLE-2000

```
*
* Calling the recursive "fact" procedure:
*
*  input  to  "fact": *n*
* output from "fact": *n_fact*
*
*    use to compute n!
*
 PROCEDURE fact   ;
 INTEGER   n := 8    ;
 INTEGER   n_fact    ;
 fact ::    <<n>> >>n_fact<< ;
 ECHO "FACTORIAL:" n $Bang_S "=" n_fact ;
 QUIT " Program *xfact* XREF " .
```

Résultat:

```
>|FACTORIAL: 8 ! = 40320
```

Appel d'un module de calcul

L'appel du module externe Module: s'écrit

(OutDS) := Module: (InDS) : : (data) ;

- (OutDS) est une liste de fichiers et/ou d'objets LCM produits ou modifiés par le module Module:.
- (InDS) est une liste de fichiers et/ou d'objets LCM accédés en lecture seule ou modifiés par le module Module:.
- Un item accédé en modification doit apparaitre à la fois dans (OutDS) et dans (InDS).
- L'ordre des items est important (voir le guide d'utilisation).

```
LINKED_LIST MOSTELC VOLTRK LIBRARY ;
SEQ_BINARY TRKSPC ;
INTEGER an2d := 8 ;
REAL densur := 20.0 ;
...
VOLTRK TRKSPC := NXT: MOSTELC ::
    TISO <<an2d>> <<densur>> ;
LIBRARY := SHI: LIBRARY VOLTRK TRKSPC ;
```

Instructions de transfert de variables

- La commande << PLPITCH >> permet de transférer la variable PLPITCH du script CLE-2000 vers les données d'entrée du module GEO::

```
REAL PLPITCH := 1.2 ;
MOSTELC := GEO: :: CARCEL 2
  X- REFL X+ REFL MESHX 0.0 <<PLPITCH>>
  Y- REFL Y+ REFL MESHY 0.0 <<PLPITCH>>
  RADIUS 0.0 0.39306 0.45802
  SPLITR 2 1
  MIX 123;
```

- La commande >> pow2 << permet de transférer la variable pow2 calculée par le module PKINS: vers le script CLE-2000

```
REAL pow2 ;
REAL t := 0.3 ;
REAL dt := 3.0E-4 ;
FMAP := PKINS: FMAP ::
  EDIT 2
  TIME <<t>> <<dt>> PICK >>pow2<<
;
ECHO "pow2=" pow2 ;
```

Procédures

- Une procédure est un script CLE-2000, semblable à la fonction `fact` décrite précédemment, et permettant d'encapsuler des sous-ensembles validés de données.
- Une procédure dont le nom est `myproc` doit être stockée dans un fichier ascii dont le nom est `myproc.c2m` ou dans un fichier binaire dont le nom est `myproc.o2m`.
- Un fichier binaire `myproc.o2m` correspond à la version compilée de `myproc.c2m`.
- Un ensemble de fichiers `*.c2m` validés forme un schéma de calcul.

Exemple:

```
PARAMETER  FMAP :: ::: LINKED_LIST FMAP ; ;
MODULE     END: PKINI: ;
REAL dt pow0 ;
  :: >>dt<< >>pow0<< ;

FMAP := PKINI: FMAP ::
   EDIT 2
   TIME -1.0 <<dt>>
   POWER <<pow0>>
 ...
;
```

Cette procédure est stockée dans le fichier `PkinPULSE.c2m` et est appelée de la façon suivante:

```
PROCEDURE PkinPULSE ;
...
REAL powi := 67.0E-6 ; (*Reference core power (MW)*)
REAL dt := 3.0E-4 ;
FMAP := PkinPULSE FMAP :: <<dt>> <<powi>> ;
```

Boucles et instructions conditionnelles

Des ennoncés de contrôle (`WHILE DO`, `ENDWHILE`, `REPEAT UNTIL`, `IF`, `ENDIF`, etc) peuvent être insérées dans le script CLE-2000.

Exemple d'une boucle `WHILE DO`:

```
LINKED_LIST DONNEES LIBRARY TRACKSS ;
MODULE UTL: GREP: TONE: ;
REAL Tend BUend ;
REAL Fuelpwr := 36.8 ;
...
INTEGER maxstep := 4 ;
DONNEES := UTL: ::
  CREA  BURN <<maxstep>> = 0.0  9.375  18.75  37.5 ;
INTEGER istep := 0 ;
WHILE istep maxstep < DO
 EVALUATE istep := istep 1 + ;
 ECHO "istep: " istep "/" maxstep ;
 GREP: DONNEES :: GETVAL 'BURN' <<istep>> >>BUend<< ;
 EVALUATE Tend := BUend Fuelpwr / ;
  IF istep 1 = THEN
       LIBRARY := TONE: LIBRARY TRACKSS :: EDIT 3
                  MXIT 2  ARM GRMIN 52 ;
  ENDIF ;
ENDWHILE ;
```

Boucles et instructions conditionnelles

Exemple d'une boucle `REPEAT UNTIL`:

```
REAL DeltaP := 0.0 ;
REAL  PLPITCH ;
REPEAT
  Dimensions :: <<DeltaP>> >>PLPITCH<< ;
  MOSTELC := GEO: :: CARCEL 2
  X- REFL X+ REFL MESHX 0.0 <<PLPITCH>>
  Y- REFL Y+ REFL MESHY 0.0 <<PLPITCH>>
  RADIUS 0.0 0.39306 0.45802 SPLITR 2 1 MIX 123;
  VOLTRK TRKSPC := NXT: MOSTELC :: TISO 12 20.0 ;
  LIBRARY := SHI: LIBRARY VOLTRK TRKSPC ;
  CP := ASM: LIBRARY VOLTRK TRKSPC ;
  FLUX := FLU: CP LIBRARY VOLTRK :: TYPE K ;
  MOSTELC VOLTRK TRKSPC CP FLUX := DELETE:
  MOSTELC VOLTRK TRKSPC CP FLUX ;
  EVALUATE DeltaP := DeltaP 0.1 + ;
UNTIL PLPITCH 2.0 > ;
```

où la procédure `Dimensions` s'écrit

```
REAL PitchAdd Pitch ;
REAL PitchRef := 1.26209 ;
:: >>PitchAdd<< ;
Pitch := PitchRef PitchAdd + ;
:: <<pitch>> ;
```

Le module UTL :

Le noyau GANLIB5 comporte des modules utilitaires permettant de supporter les tables associatives et les listes hétérogènes. La documentation des modules utilitaires se trouve à la Sect. 5 de la Ref. suivante:

A user guide for Dragon version5

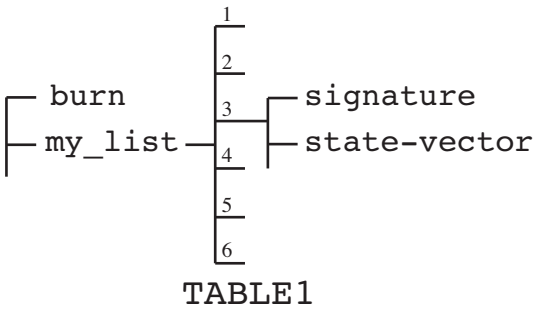
Le module UTL: permer d'effectuer diverses opérations utilitaires sur une une table associative ou sur un fichier XSM.

[*NAME1* :=] UTL: [*NAME1*] ::
[EDIT *iprint*]
[DIR] [VAL]
[[STEP { UP *NOMDIR* | AT *index* | DOWN | ROOT } [NEW { DICT | LIST *nsize* }]]]
[[IMPR { *BLOCK* | *index* } { *ileni* | * }]]
[[CREA { *BLOCK* | *index* } [*ilenc1*] *ilenc2* = { (*valc*(i),i=*ilenc1*,*ilenc2*) | (*ivalc*(i),i=*ilenc1*,*ilenc2*)
| (*hvalc*(i),i=*ilenc1*,*ilenc2*) | (*dvalc*(i),i=*ilenc1*,*ilenc2*) }]]
[[DEL *BLOCK*]]
[[MULT { *BLOCK* | *index* } *flott*]]
[[{ COPY | STAT { REL | ABS } | ADD } *NOMREF NOMALT*]]
[DUMP] ;

Exemple:

```
INTEGER maxstep := 12 ;
INTEGER maxstta := 4 ;
TABLE1 := UTL: ::
  CREA burn <<maxstep>> =
   0.0     9.375  18.75   37.5    75.0   150.0
 325.0   500.0   750.0  1000.0  1500.0  2000.0
  STEP UP my_list NEW LIST 6 STEP AT 3 NEW DICT
  CREA signature 3 = 'L_LI' 'BRAR' 'Y   '
  CREA state-vector <<maxstta>> = 10 11 12 13 ;

UTL: TABLE1 :: DIR DUMP ;
```



Le module :=

Le module égalité (:=) permet de copier un objet (LCM, XSM, fichier binaire ou ASCII sur un autre.

NAME1 := [*NAME1*] *NAME2*
[:: [EDIT *iprint*] [{ OLD | SAP}] [[STEP { UP *NOMDIR* | AT *index* }]]] ;

Exemple:
Copie d'une branche de la table associative TABLE1 dans TABLE2

```
TABLE2 := TABLE1 :: STEP UP 'my_list' STEP AT 3 ;
```

Il est également possible d'ajouter de l'information dans une table associative (ou un fichier XSM) existante

```
TABLE2 := TABLE2 TABLE1 :: STEP UP 'my_list' STEP AT 3 ;
```

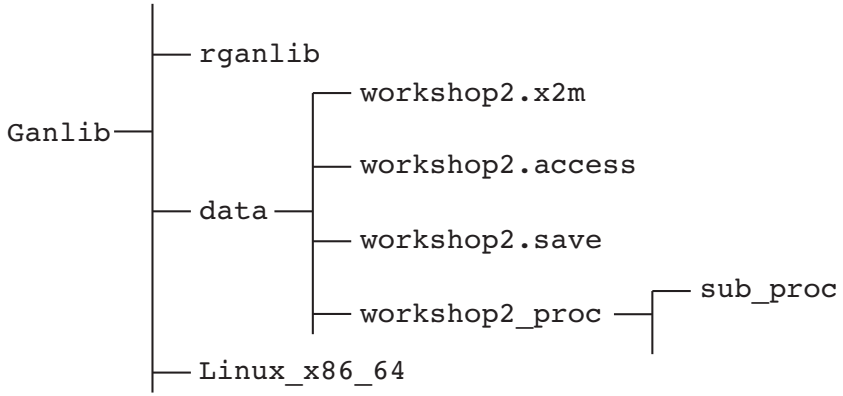
Le module égalité permet également de sérialiser (ou de dé-sérialiser) un objet LCM ou XSM (c'est-à-dire de le transformer en fichier séquentiel binaire ou ASCII.

```
SEQ_ASCII _TABLE1 :: FILE './MY_TABLE1.txt' ;
_TABLE1 := TABLE1 ;
```

Le fichier MY_TABLE1.txt peut être récupéré à la fin de l'exécution de CLE-2000 et être lu par un éditeur de texte.

Le module :=

L'exécution du cas test `workshop2.x2m` par le noyau GANLIB5 demande la préparation de la structure UNIX suivante:



`workshop2.x2m` Fichier CLE-2000 principal

`workshop2.access` Script Bourne exécuté avant CLE-2000 et permettant de récupŕer les fichiers requis pour l'exécution. Doit être déclaré exécutable par la commande `chmod 755 workshop2.access`. Non requis par cet exemple

`workshop2.save` Script Bourne exécuté après CLE-2000 et permettant de sauver en lieu sûr les fichiers produits par le calcul. Doit être déclaré exécutable par la commande `chmod 755 workshop2.save`. Par défaut, seul le fichier `workshope.result` est conservé. Tous les autres fichiers non-récupérés par le script `.save` sont détruits.

`workshop2_proc` Répertoire UNIX contenant tous les scripts CLE-2000 (procédures) utilisés par le fichier CLE-2000 principal ainsi que certains fichiers de données spécifiques au cas traité.

Le module :=

L'exécution est lancée par

```
cd Ganlib/
.\rganlib workshop2.x2m
```

Le résultat du code se retrouve dans le répertoire `Linux_x86_64` ou `Darwin_x86_64`.

```
cd Ganlib/Linux_x86_64/
cat MY_TABLE1.txt
```

On obtient la sérialisation de `TABLE1`:

```
->       1      12       2      12                                  <-
burn
  0.00000000E+00  9.37500000E+00  1.87500000E+01  3.75000000E+01  7.50000000E+01
  1.50000000E+02  3.25000000E+02  5.00000000E+02  7.50000000E+02  1.00000000E+03
  1.50000000E+03  2.00000000E+03
->       1      12      10       6                                  <-
my_list
->       2       0      99       0                                  <-   00000001
->       2       0      99       0                                  <-   00000002
->       2       0       0      -1                                  <-   00000003
->       3      12       3       3                                  <-
signature
         4         4         4
L_LIBRARY
->       3      12       1       4                                  <-
state-vector
        10        11        12        13
->      -3       0       0       0                                  <-
->       2       0      99       0                                  <-   00000004
->       2       0      99       0                                  <-   00000005
->       2       0      99       0                                  <-   00000006
->      -1       0       0       0                                  <-
```

Les modules utilitaires - suite

`DELETE:`	Suppression d'objets LCM ou XSM
`BACKUP:`	Regroupement de plusieurs objets LCM ou XSM sur un archive LCM ou XSM
`RECOVER:`	Extraction de plusieurs objets LCM ou XSM d'un archive LCM ou XSM
`ADD:`	Addition de deux objets LCM ou XSM
`MPX:`	Multiplication d'un objet LCM ou XSM par une variable CLE-2000
`STAT:`	Comparaison de deux objets LCM ou XSM
`GREP:`	Extraction d'une variable CLE-2000 de l'information contenue dans un objet LCM ou XSM
`FINDO:`	Recherche de la racine d'une équation non-linéaire par la méthode de Brent
`ABORT:`	Arret de l'exécution en cas d'erreur
`END:`	Fin normale de l'exécution d'un script.