Adaptation du noyau de projet sur micro-ordinateur 16 bits en pascal

Jean-Pierre Goulette

236

LI2A LABORATOIRE D'INFORMATIQUE APPLIQUEE A L'ARCHITECTURE
Ecole d'Architecture de Toulouse . Chemin du Mirail
31057 TOULOUSE Cedex . Tel:(61)40.47.28 Poste 226

ADAPTATION DU NOYAU DE PROJET SUR MICRO-ORDINATEUR 16 BITS EN PASCAL

Jean-Pierre GOULETTE

Rapport final du contrat de recherche N° 83 01287 00 223 7501 financé par le Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Secrétariat de la Recherche Architcturale

MAI 1984

LI2A. Ecole d'Architecture de Toulouse.
Chemin du Mirail. 31057 Toulouse Cedex.

CONTRAT N° 83 01287 00 223 75 01
Ministère de l'Urbanisme et du Logement, Direction de l'Architecture, Secrétariat de la Recherche Architecturale.

ADAPATION DU NOYAU DE "PROJET" SUR MICRO-ORDINATEUR 16 BITS EN PASCAL

Jean-Pierre GOULETTE

RAPPORT FINAL DE RECHERCHE

Le présent document constitue le rapport final d'une recherche remise au Secrétariat de la Recherche Architecturale en exécution du programme général de recherche mené par le Ministère de l'Urbanisme et du Logement avec le Ministère de l'Industrie et de la Recherche. Les jugements et opinions émis par les responsables de la recherche n'engagent que leurs auteurs.

Responsable scientifique
Michel LEGLISE

LI2A, Mai 1984

PRESENTATION GENERALE

La recherche portait sur l'adaptation en Pascal du noyau du logiciel de Conception Assistée par Ordinateur "Projet" (Implanté au Centre d'Informatique et de Méthodologie en Architecture). Cette projection du Fortran vers le Pascal devait s'appuyer sur les possibilités de ce dernier langage et induire certaines modifications conformes à une perspective de connexions futures avec l'intelligence artificielle. Cette adaptation a été réalisée et le logiciel résultant est implanté sur BFM 186.

Cependant il nous a semblé nécessaire de pousser l'étude un peu plus avant par une réalisation permettant l'utilisation réelle de la base de données que nous venions d'établir. Cette dernière phase a consisté en la définition et l'écriture d'une interface entre le noyau de la base et un programme d'édition graphique (PADAO).

La base et sa structure sont décrites dans le document : " 'Projet', une base de données relationnelles en Conception Assistée par Ordinateur, pour le projet d'Architecture", Marché N° 81 61156 00 223 75 01 du Plan Construction, Septembre 1982.

Le Centre d'Informatique et de Méthodologie en Architecture nous a fourni un cahier des charges complet que nous avons respecté et auquel nous avons apporté certaines améliorations ne modifiant pas la démarche de communication avec la base.

Nous rappelons brièvement ici la structure de la base de "Projet".

Le noyau de la base de données est divisé en deux blocs :

- Un noyau que nous nommerons "Noyau Relationnel" (par abréviation NR, dans la suite de ce texte pour plus de commodité) possédant quatre types de cellules (Objet, Relation, Associateur, Donnée) pouvant être associées et donner lieu à la définition d' une information d'ordre syntaxique entre les divers éléments constitutifs d'un batiment.

- Un autre noyau que nous nommerons Noyau Hiérarchique (par abréviation NH) ne possédant q'un seul type de cellules pouvant être associées par des considérations d'ordre hiérarchique. C'est un bloc à structure de liste à niveau variable permettant une décomposition simple d'un batiment et un parcours rapide entre ces éléments constitutifs. C'est dans ce bloc que nous mémoriserons de préférence les descripteurs physiques des éléments.

Nous étudierons ces deux blocs séparément.

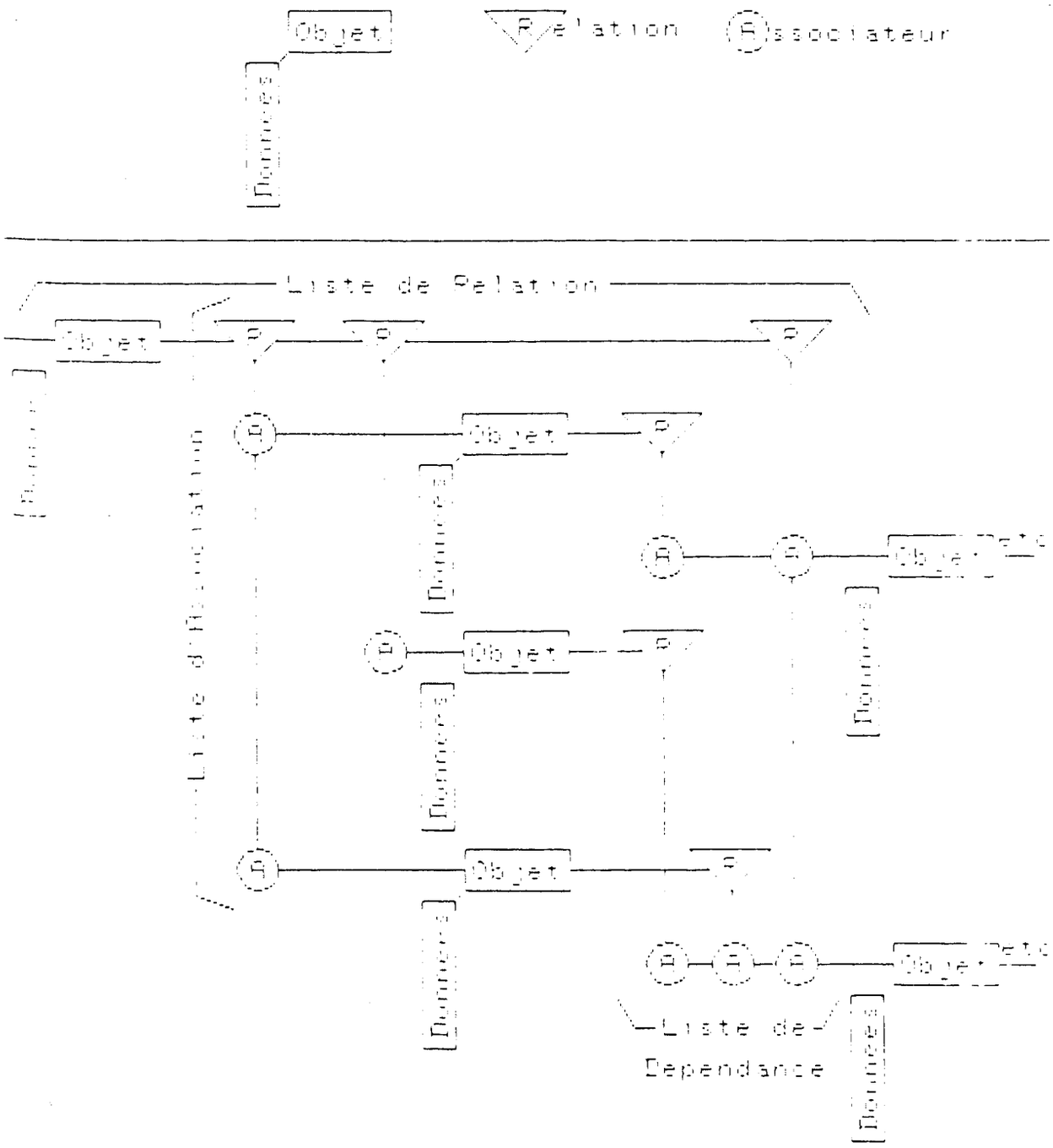
NR, LE NOYAU RELATIONNEL

Il a pour but l'établissement de listes de relation affectées à des objets. Il donne à l'utilisateur la possibilité de "tisser" (par le biais des cellules de type Objet, Relation, Associateur et des primitives de gestion présentées ci-dessous) un schéma relationnel entre ses données. Le type de ce schéma peut se présenter de la manière suivante : l'Objet, O1, pointe sur la Relation, R, qui pointe elle-même (à travers un Associateur) sur l'Objet O2. O1 et O2 sont mis en relation par R ; on peut écrire O1 R O2. Le nombre de Relations sur lequel peut pointer un Objet est infini (en théorie) ainsi que le nombres d'Associateurs affectables à chacune de ces Relations. Il y a création d'une structure de graphe présentée dans le schéma de la page suivante.

Représentation logique :

NR est constitué d'un ensemble de cellules de nature différentes possédant chacune un numéro l'identifiant d'une manière bijective à l'intérieur de l'ensemble. Ce numéro est le "contact" de NR avec le monde extérieur et l'acteur privilégié des fonctions régissant la base. Une cellule contient de plus différents paramètres décrivant une personnalisation quelconque (laissée à la disposition de l'utilisateur) et ses liens avec l'environnement.

L'ensemble de ces cellules ne réside pas en intégralité en mémoire centrale, il est "paginé" : seules les cellules intéressant les actions présentes sont accessibles directement.


Objet
Relation
Associateur
Données
Liste de Relation
Objet
Liste d'Association
Liste de
Dependance

Représentation physique :

NR est constitué d'un fichier à accès direct dont chaque enregistrement correspond à une cellule. Ce fichier est décomposé en tableaux (pages) de cinquante cellules ; seuls quatre de ces tableaux (et donc deux cents cellules) sont présents en mémoire centrale à un instant donné (ces deux valeurs paramétrant l'organisation en mémoire de NR sont déclarées en constantes en début du programme Pascal et peuvent être aisément modifiées).

L'algorithme de pagination utilisé est l'algorithme LRU (Least Recently Used) : lorsqu'un module effectue une demande d'accès à une page ne résidant pas en mémoire centrale, la page "la plus anciennement utilisée" est écartée pour laisser place à la page demandée. Cette page "écartée" est recopiée sur disque si elle a été modifiée en mémoire centrale. Il est donc associé à chaque page une variable indexant son ancieneté de consultation et un booléen forcé à "True" lorsque la page est modifiée.

L'identification d'une cellule correspond donc à son numéro d'enregistrement sur le fichier et est "traduit", à chaque appel de cellule, en un index de tableau par les fonctions de gestion interne de NR.

Le numéro d'une cellule supprimée est mémorisé dans une pile de pointeurs (adresses segmentées) de manière à être réutilisé lors d'une prochaine création de cellule. Cette pile de pointeurs est déversée sur un fichier séquentiel à la mise à jour finale de la base et sera réincorporée en mémoire centrale lors d'une prochaine initialisation.

Description des cellules :

La nature des cellules est déclarée en tête de programme dans un Type Pascal :

```
Type Nat_Cel = (Obj,Rel,Ass,Don);
```

La cellule Objet :

-Structure interne : Elle possède en premier lieu une variable (de type Nat_Cel) "attestant" qu'elle est bien un Objet et servant au décelage des erreurs pouvant survenir au cours d'une communication avec NR. Elle possède de plus un Code (formé par une chaine de trois caractères) permettant une "personnalisation" de l'objet et un Mot (un entier codé sur deux octets) qui est généralement utilisé pour mémoriser le numéro de la cellule homologue dans NH. Trois pointeurs (trois numéros de cellules donc trois entiers) décrivent ses rapports avec l'environnement.

-Rapport avec l'environnement : Trois pointeurs.
Un pointeur indiquant le numéro du successeur dans la liste de relation (une cellule de type Relation).
Un pointeur indiquant le numéro du successeur dans la liste de dépendance (une cellule de type Associateur).
Un pointeur indiquant le numéro du successeur dans la liste de données (une cellule de type Donnée).

La cellule Relation :

-Structure interne : Tout comme la cellule Objet elle possède une variable de type Nat_Cel, un Code et un Mot. Elle diffère de la cellule Objet dans ses rapports avec l'environnement.

-Rapport avec l'environnement : Trois Pointeurs.
Un pointeur indiquant le numéro du successeur dans la liste de relation (une cellule de type Relation).
Un pointeur indiquant le numéro de la la tête de liste de relations (une cellule de type Objet constituant la tête de liste de toutes les relations chaînées par le premier pointeur).
Un pointeur indiquant le numéro du successeur dans la liste d'association (une cellule de type associateur).

La cellule Associateur :

-Structure interne : Elle possède une variable de type Nat_Cel, un Code (Chaine de Trois caractères) et quatre pointeurs.

-Rapport avec l'environnement : Quatre pointeurs
Un pointeur indiquant le numéro de la tête de liste d'association (une cellule de type Relation, tête de liste de tous les Associateurs chaînés par le quatrième pointeur).
Un pointeur indiquant le numéro de la tête de liste de dépendance (une cellule de type Objet constituant le membre droit d'une relation 01 R 02).
Un pointeur indiquant le numéro du successeur dans la liste de dépendance (une cellule de type Associateur ayant même tête de liste de dépendance).
Un pointeur indiquant le numéro du successeur dans la liste d'association (une cellule de type Associateur ayant même tête de liste d'association).

La cellule Donnee :

-Structure interne : Elle possède une variable de type Nat_Cel, un Code (chaîne de trois caractères), trois Mots (trois entiers) et un pointeur.

-Rapport avec l'environnement :
Un pointeur indiquant le numéro de la cellule suucesseur dans la liste de données (une cellule de type Donnée).

Les procédures de NR

Elles se divisent en deux types :
-Les primitives de gestion interne qui gèrent la représentation physique, la cohérence, la pagination..., et qui ne seront pas détaillées ici puisqu'elles sont invisibles par l'utilisateur.
-Les procédures de communication qui assurent l'interface de la base avec le programme utilisateur et qui sont donc détaillées ci-dessous.

L'ensemble de ces procédures de communication peut être subdivisé en cinq grands groupes :
-Procédures de commandes.
-Création de cellules.
-Transfert de données.
-Parcours de la base.
-Suppression de cellules.

Les procédures de commande :

Ce sont des procédures d'initialisation et de fermeture de la base.
Un Type "Chaine8" est déclaré en tête de programme et correspond à une chaine de huit caractères.

-Function CrNR(Nom:Chaine8):Boolean;
Création d'un fichier direct de cellules de nom Nom.
Si un fichier existe déja sous le nom Nom la fonction est avortée et sa valeur est mise à "False" (ceci pour éviter la destruction d'une base existant déja sous le même nom). Si la création du fichier se passe sans problème, la fonction retourne "True". Le premier enregistrement du fichier est réservé pour mémoriser le nombre total de cellules créées ; la première cellule prendra donc "2" comme numéro.

-Function InitFNR(Nom:Chaine8):Boolean;
Ouverture d'un fichier direct de cellules de nom Nom.
Si le fichier de nom Nom n'est pas présent sur disque la fonction est avortée et sa valeur est mise à "False". Si l'ouverture du fichier se passe sans problème, elle est suivie d'initialisations diverses pour les procédures internes à NR (en particulier pour la gestion des cellules vides, s'il y en a) et la fonction retourne "True".

-Procedure FinNR;
Fermeture du fichier direct de cellules ouvert par une des deux fonctions précédentes.

Les procédures de création de cellules.

Un Type Chaine3 est déclaré en tête de programme et correspond à une chaine de trois caractères, ainsi qu'un type TabDon correspondant à un tableau de trois entiers.

La seule erreur pouvant survenir est le dépassement de la valeur entière maximale représentable par la machine pour la numérotation de la cellule ("overflow" du fichier). Cette erreur est "trapée" par le programme qui affecte, dans ce cas, la valeur zéro à la fonction.

La valeur du Code communiqué comme paramètre de la fonction ne peut être (pour des raisons que nous verrons au chapitre concernant les procédures de parcours de la base) la chaine de caractères : '0' (zéro, de type caractère). Si un tel Code est passé en paramètre d'une fonction de création, la fonction avorte et prend la valeur zéro.

-Function NewObj(C:Chaine3;Mot:Integer):Integer;

Création d'une cellule Objet isolé de code C et de mot Mot. La fonction retourne le numéro de la cellule créée.

-Function NewRel(NcObj:Integer;C:Chaine;Mot:Integer):Integer;

Création d'une cellule Relation de code C et de mot Mot. Cette cellule est placée en tête de liste de relation de l'Objet NcObj avec une liste d'association vide. La fonction retourne le numéro de la cellule créée.

-Function NewAss(NcObj,NcRel:Integer;C:Chaine3):Integer;

Création d'une cellule Associateur de code C. Cette cellule est placée en tête de liste de dépendance de l'Objet NcObj et en tête de liste d'association de la Relation NcRel. La fonction retourne le numéro de la cellule créée.

-Function SuccAss(NcObj,NcAss:Integer;C:Chaine3):Integer;

Création d'une cellule Associateur de code C. Cette cellule est placée en tête de liste de dépendance de l'Objet NcObj et sur la liste d'association à laquelle appartient NcAss, à la suite de cette dernière. La fonction retourne le numéro de la cellule créée.

-Function NewDat(NcObj:Integer;C:Chaine3;Itab:TabDon):Integer;

Création d'une cellule Donnée de code C. Cette cellule est "remplie" avec les valeurs du tableau Itab et est placée en fin de liste de donnée de l'Objet NcObj. La fonction retourne le numéro de la cellule créée.

Les procédures de transfert de données:

Dans ces procédures, un numéro de cellule est communiqué à NR. Si ce numéro correspond à un cellule valide la fonction est executée normalement (avec certaines restrictions dues à la nécessaire compatibilité entre le type de la cellule et l'opération demandée). Dans le cas contraire (cellule inexistante ou supprimée) la fonction retourne "une valeur nulle" (que ce soit, suivant le cas, un entier, un tableau ou une chaine de caractères) et un message d'erreur est envoyé à l'écran. Nous avons pris cette liberté, l'expérience nous ayant montré que la présence de ces messages facilitait grandement la mise au point des interfaces avec les programmes utilisateur (ces messages ne devant plus apparaître dès que que le programme appelant est correct). Nous avons donc préféré cette méthode à celle qui aurait consisté à écrire systématiquement des fonctions qui retourneraient un code d'erreur.

-Function LirDat(NcD:Integer):TabDon;
Cette fonction retourne le tableau de données contenu dans la cellule NcD. Si la cellule n'est pas du type Donnée, un message ("LECTURE IMPOSSIBLE") est envoyé à l'écran et la fonction retourne un tableau de trois zéros.

-Function Lircod(Nc:Integer):Chaine3;
Cette fonction retourne le code de la cellule de numéro Nc. Aucune restriction n'est faite sur le type de la cellule.

-Function LirMot(Nc:Integer):Integer;
Cette fonction retourne le Mot de la cellule de numéro Nc. Si la cellule est du type Associateur (ne contenant pas de Mot), un message ("LECTURE IMPOSSIBLE") est envoyé à l'écran et la fonction prend la valeur nulle.

-Procedure ModDat(NcD:Integer;Ndata:TabDon);
Cette procédure remplace le tableau de donnéés de la cellule de numéro NcD par le tableau Ndata. Si la cellule n'est pas du type Donnée, un message ("MODIFICATION IMPOSSIBLE") est envoyé à l'écran.

-Procedure ModCod(Nc:Integer;C:Chaine3);
Cette procédure remplace le Code de la cellule de numéro Nc par la valeur de C. Si C a pour valeur '0', la procédure avorte et un message ("MODIFICATION IMPOSSIBLE") est envoyé à l'écran.

-Procedure ModMot(Nc,Info:Integer);
Cette procédure remplace le Mot de la cellule de numéro Nc par la valeur contenue dans Info. Si la cellule est du type Associateur (ne contenant pas de Mot), un message ("MODIFICATION IMPOSSIBLE") est envoyé à l'écran et la procédure avorte.

Les procédures de parcours de la base :

-Function NxtDep(Ncoa:Integer;C:Chaine3):Integer;
Si C est différent de '0', cette fonction parcourt la liste de dépendance de la cellule Ncoa (Objet ou Associateur), à partir de la cellule Ncoa (non comprise) jusqu'au premier associateur ayant le code C dont elle retourne le numéro. Si aucun Associateur de la liste n'a le code C la fonction retourne la valeur zéro.
Si C est égal à '0', la fonction retourne le numéro de la cellule Associateur suivant la cellule Ncoa sur la liste de dépendance (ou retourne zéro si Ncoa est fin de liste).
Si la cellule de numéro Ncoa n'est ni de type Objet, ni de type Associateur, la fonction retourne zéro.

-Fonction NxtAss(Ncra:Integer;C:Chaine3):Integer;
Si C est différent de '0', cette fonction parcourt la liste d'association de la cellule Ncra (Relation ou Associateur), à partir de la cellule Ncra (non comprise) jusqu'au premier Associateur ayant le code C dont elle retourne le numéro. Si aucun Associateur de la liste n'a le code C la fonction retourne la valeur zéro.
Si C est égal à '0', la fonction retourne le numéro de la cellule Associateur suivant la cellule Ncra sur la liste d 'association (ou retourne zéro si Ncra est fin de liste).
Si la cellule de numéro Ncra n'est ni du type Relation, ni du type Associateur, la fonction retourne zéro.

-Function NxtRel(Ncro:Integer;C:Chaine3):Integer;
Si C est différent de '0', cette fonction parcourt la liste de relation de la cellule Ncro (Objet ou Relation), à partir de la cellule Ncro (non comprise) jusqu'a la première Relation ayant le code C dont elle retourne le numéro. Si aucune Relation de la liste n'a le code C la fonction retourne zéro.
Si C est égal à '0', la fonction retourne le numéro de la cellule Relation suivant Ncro sur la liste de relation (ou retourne zéro si Ncro est fin de liste).
Si la cellule de numéro Ncro n'est ni du type Relation, ni du type Objet, la fonction retourne zéro.

-Function NxtDat(Ncod:Integer;C:Chaine3):Integer;

Si C est différent de '0', cette fonction parcourt la liste de données de la cellule Ncod (Objet ou Donnée), à partir de la cellule Ncod (non comprise) jusqu'a la première Donnée ayant le code C dont elle retourne le numéro. Si aucune Donnée de la liste n'a le code C la fonction retourne zéro.

Si C est égal à '0', la fonction retourne le numéro de la cellule Donnée suivant Ncod sur la liste de données (ou retourne zéro si Ncod est fin de liste).

Si la cellule de numéro Ncod n'est ni du type Objet, ni du type Donnée, la fonction retourne zéro.

-Function LhdObj(Ncra:Integer):Integer;

Cette fonction retourne le numéro de l'Objet tête de liste de relation si la cellule Ncra est de type Relation, ou le numéro de l'Objet tête de liste de dépendance si la cellule Ncra est de type Associateur.

Si la cellule de numéro Ncra n'est ni de type Relation, ni de type Associateur, la fonction retourne zéro.

-Function LhdRel(Nca:Integer):Integer;

Cette fonction retourne le numéro de la Relation tête de liste d'association contenant l'Associateur Nca.

Si la cellule de numéro Nca n'est pas du type Associateur, la fonction retourne zéro.

Les procédures de suppression de cellules :

Ces procédures libèrent la place occupée par une cellule d'un type particulier (une procédure pour chaque type) et insère son numéro dans la pile des pointeurs de cellules vides.

-Procedure SupDon(NcObj,Ncd:Integer);
Cette procédure supprime la cellule Donnée de numéro Ncd appartenant à la liste de données de l'Objet NcObj.
Si la cellule de Numero Ncd n'est pas du type Donnée, ou si la cellule de numéro NcOBj n'est pas du type Objet, la procédure avorte.

-Procedure Supdat(Nco:Integer);
Cette procédure supprime toutes les cellules de la liste de données de l'Objet Nco. Si la cellule de numéro Nco n'est pas du type Objet, la procédure avorte.

-Procedure SupAss(Nca:Integer);
Cette procédure supprime la cellule Associateur de numéro Nca et met à jour les listes de dépendance et d'association auquelles elle appartenait.
Si la cellule de numéro Nca n'est pas du type Associateur, la procédure avorte.

-Procedure SupRel(Ncr:Integer);
Cette procédure supprime la cellule Relation de numéro Ncr, supprime la liste d'association dont elle était la tête et met à jour la liste de relation à laquelle elle appartenait.
Si la cellule de numéro Ncr n'est pas de type Relation, la procédure avorte.

-Procedure SupObj(Nco:Integer);
Cette procédure supprime la cellule Objet de numéro Nco ainsi que ses listes de données, de relation et de dépendances (dont tous les éléments constitutifs sont supprimés).
Si la cellule de numéro Nco n'est pas du type Objet, la procédure avorte.

--==*==--

Nous proposons en appendice à l'utilisateur de NR, la lecture d'un "listing " comportant toutes les procédures de communication externes, ainsi qu'une explication rapide des primitives de gestion interne. Celui-ci sera donc à même de mieux appréhender les manipulations effectuées sur les cellules au sein même de NR.

NH, LE NOYAU HIERARCHIQUE

NH permet la mise en hiérarchie des divers éléments constitutifs d'un bâtiment à partir d'une structure de liste à niveau variable (Exemple : bâtiment est tête de liste des étages, eux-mêmes sont tête de liste des pièces, elles-mêmes étant tête de liste des facettes composant les pièces ... etc).

Représentation logique :

NH est constitué d'un ensemble de cellules de même nature possèdant chacune un numéro identificateur, contact de NH avec le monde extérieur. Un bâtiment y est décomposé en une "super" tête de liste (la première cellule créée) suivie d'une sous-liste. Une sous-liste est un ensemble de cellules possédant une tête de liste commune et dont chaque élément peut être tête d'une autre sous-liste (définition récursive permettant un niveau de profondeur indéfini).

Chaque cellule possède différents paramètres :

Trois pointeurs décrivant ses rapports avec l'environnement (les autres cellules).

Un ensemble de valeurs (Données).

NH, de par la simplicité de sa syntaxe constructive, permet un parcours rapide à l'intérieur des données du bâtiment et sera donc le lieu privilégié de mémorisation des descripteurs physiques des éléments.

Représentation physique :

NH est constitué d'un fichier à accès direct dont les caractéristiques de pagination, d'indexation des cellules et de gestion des cellules vides sont identiques à celles de NR (se reporter page 4).

De plus, la longueur du tableau de données affecté à chaque cellule de NH n'est plus fixée à la création de la base, mais peut varier suivant chaque cellule. Ces données sont en effet gérées par allocation dynamique et mémorisées dans une file de pointeurs dont chaque élément peut être un tableau de trois nombres entiers ou une chaîne de cinq caractères (il a semblé en effet utile de pouvoir identifier ces données à l'intérieur même de NH plutôt que par réference à une information extérieure décodant chaque donnée en fonction de sa position dans le tableau, comme il a été établi dans les programmes-source de la version Fortran dont nous disposions).

Pour permettre une augmentation, à volonté, des données de chaque cellule sans "écraser" celles de la cellule suivante, il a été nécessaire d'affecter à chaque page de NH son propre fichier de données, géré par le programme et invisible à l'utilisateur. Chaque page peut être amenée en mémoire centrale indépendamment de ses données, accélérant ainsi la vitesse d'exécution des fonctions de parcours de la base où seule la lecture des pointeurs d'une cellule est nécessaire.

Description de la cellule :

-Structure interne : Une cellule de NH ne possède pas de paramètre identificateur "in corpo" ; elle est "personnalisée" par sa position à l'intérieur de l'ensemble des listes. Son pointeur sur les données est un pointeur physique (une adresse segmentée) dont l'allocation ne relève pas de la procédure standard "New" (qui n'affecte que la pile du segment par défaut), mais d'une procédure interne à NH (disponible pour l'utilisateur) dont nous donnerons les caractéristiques au chapitre suivant.

-Rapport avec l'environnement : Trois pointeurs.
Un pointeur sur la cellule suivante dans la liste de même niveau.
Un pointeur sur la cellule précédente (qui peut être la tête de liste).
Un pointeur vers la sous-liste dont elle est la tête.

Les procédures de NH

Elles se divisent en deux types :
-Les primitives de gestion interne qui gèrent la représentation physique, la cohérence, la pagination..., et qui ne seront pas détaillées ici puisqu'elles sont invisibles par l'utilisateur.
-Les procédures de communication qui assurent l'interface de la base avec le programme utilisateur et qui sont donc détaillées ci-dessous.

L'ensemble de ces procédures de communication peut être subdivisé en cinq grands groupes :
-Procédures de commandes.
-Création de cellules.
-Transfert de données.
-Parcours de la base.
-Suppression de cellules.

Nous rajouterons à ces groupes une procédure d'allocation de pointeur de données :

-Procedure NewDNH(Vars P:PointDon);
Un type "PointDon" est déclaré en tête de programme :

```
Type TabDon = Array(.1..3.) of Integer;
     Dons =     Record
                Case Ca : Char of
                'C' : (Mot : Lstring(5));
                     (* chaine de cinq caractéres *)
                'D' : (Tdon : TabDon)
                     (* Le tableau de données *)
                End;
     PointDon = Ads of Donnee;    (* Adresse segmentée *)
     Donnee =  Record
                Don : Dons;
                Case Boolean of
                True : (Pdon : PointDon);
                     (* pointeur de donnée *)
                False : (Nc : Integer)
                     (* le numéro de la cellule *)
                End;
```

P de type PointDon est initialisé à la valeur "Nil" par l'assignation P.S:=0; (pointe sur le segment zéro).

Une Donnée possède, un pointeur (sur la donnée suivante) quand elle réside en mémoire centrale, ou le numéro de la cellule auquelle elle appartient quand elle est sur fichier (article à champ variable). Les procédures internes à NH se chargent des différentes gestions (lecture sur fichier, chaînage, écriture) qui deviennent transparentes pour l'utilisateur. Celui-ci a à sa disposition la procédure NewDNH lui permettant l'allocation d'une adresse dans toute la mémoire disponible pour un pointeur de type PointDon qu'il "chargera" par une variable de type Dons.

Le chaînage et le "remplissage" des différents pointeurs de données d'une cellule pourra d'ailleurs être effectué par une même procédure du type :

```
Procedure Insere(Don_a_inserer:Dons;Vars Racine:Pointdon);
Begin
If Racine.S=0 Then  Begin
                    NewDNH(Racine);
                    Racine^.Don:=Don_a_inserer;
                    Racine^.Pdon.S:=0
                    End
              Else Insere(Don_a_inserer,Racine^.Pdon)
End;
```

Il suffit alors de passer "Racine" (racine de la liste des pointeurs de données) comme paramètre d'une des procédures de NH (décrites plus bas).

Les procédures de commande :

Ce sont des procédures d'initialisation et de fermeture de la base.

Un type "Chaine8" est déclaré en tête de programme et correspond à une chaîne de huit caractères.

-Function Crlst(Nom:Chaine8):Boolean;
Création d'un fichier direct de cellules de nom Nom. Si un fichier existe déjà sous le nom Nom la fonction avorte et sa valeur est mise à "False" (ceci pour éviter la destruction d'une base existant sous le même nom). Si la création du fichier se passe sans problème, la fonction retourne "True". Le premier enregistrement est réservé pour mémoriser le nombre total de cellules créées ; la première cellule prendra donc "2" comme numéro.

-Function Initl(Nom:Chaine8):Boolean;
Ouverture d'un fichier direct de cellules de nom Nom.
Si le fichier de nom Nom n'est pas présent sur disque la fonction avorte et sa valeur est mise à "False". Si l'ouverture du fichier se passe sans problème, elle est suivie d'initialisations diverses pour les procédures internes à NH (en particulier pour la gestion des cellules vides, s'il y en a) et la fonction retourne "True".

-Procedure Finlst;
Fermeture du fichier direct de cellules NH ouvert par une des deux fonctions précédentes.

Les procédures de création de cellules :

-Function Premc(P:PointDon):Integer;
Création de la première cellule de la liste principale à laquelle est affecté le pointeur de Donnée P (racine de la file de données). La fonction donne en retour le numéro de la cellule créée.

-Fonction Intet(Nct:Integer;P:PointDon):Integer;
Création d'une cellule dans la liste dont la cellule de numéro Nct est la tête. Cette cellule est placée "juste après" la cellule Nct (elle devient donc la première cellule de la sous-liste de Nct). La fonction donne en retour le numéro de la cellule créée.

-Function Newcq(Nct:Integer;P:PointDon):Integer;
Création d'une cellule dans la liste dont la cellule de numéro Nct est la tête. Cette cellule est placée en fin de la sous-liste de la cellule Nct. La fonction donne en retour le numéro de la cellule créée.

-Function Incel(Nc:Integer;P:PointDon):Integer;
Création d'une cellule dans la liste de même niveau que celle de numéro Nc (la cellule Nc et la cellule créée ont donc la même tête de liste). Cette cellule est placée à la suite de la cellule Nc. La fonction donne en retour la numéro de la cellule créée.

Les procédures de parcours de liste :

-Function Nxtet(Nct:Integer):Integer;
Cette fonction donne le numéro de la première cellule dans la liste dont la tête est la cellule de numéro Nct.

-Function Nxcel(Nc:Integer):Integer;
Cette fonction donne le numéro de la cellule successeur de la cellule de numéro Nc dans le liste de même niveau.

-Function Prcel(Nc:Integer):Integer;
Cette fonction donne le numéro de la cellule qui précède la cellule de numéro Nc.
Une variable, Ic, de type Booléen, déclarée en tête de programme dans NH, permet de connaitre le rapport existant entre la cellule de numéro Nc et la cellule précèdente. L'utilisateur de NH pourra donc déclarer une variable externe, Ic, de type Booléen :
Var (.Extern.) Ic:Boolean;
et tester ensuite la valeur de Ic :
-Ic=False --> La cellule précédent Nc est sur la liste de même niveau que Nc,
-Ic=True --> La cellule précédent Nc est tête de la liste de Nc.

-Function Tetli(Nc:Integer):Integer;
Cette fonction donne le numéro de la cellule tête de la liste contenant la cellule de numéro Nc. La tête de liste de la premiére cellule créée est zéro (Tetli(2) renvoie 0), tout autre cellule possède une tête de liste valide.

Les procédures de transfert de données:

-Function Lircl(Nc:Integer):PointDon;
Cette fonction donne le pointeur de données de la cellule Nc. Ce pointeur est la racine de la pile de pointeurs de données de la cellule Nc (qu'il conviendra donc de "dépiler").

-Procédure Modif(Nc:Integer;P:PointDon);
Cette procédure substitue le pointeur P au pointeur Pdon (pointeur de données) de la cellule Nc. Il constituera donc la nouvelle racine de la pile de données de la cellule Nc (l'ancienne pile de données est "disposée" en mémoire centrale par des procédures internes à NH).

La procédure de suppression de cellules :

Procedure Supcl(Nc:Integer);
Cette procédure supprime la cellule Nc et éventuellement la liste dont elle est la tête ainsi que toutes les listes de niveau inférieur. Le numéro, Nc, et les numéros des cellules libérées (sous-liste) sont insérés dans la pile des cellules vides pour être réutilisés lors d'une prochaine création de cellule.

--==*==--

Nous proposons en appendice à l'utilisateur de NH, la lecture d'un "listing " comportant toutes les procédures de gestion externes, ainsi qu'une explication rapide des primitives de gestion internes. Celui-ci sera donc à même de mieux appréhender les manipulations effectées sur les cellules au sein même de NH.

CONSIDERATIONS GENERALES SUR LA PROGRAMMATION

La première tentative d'utiliser les pointeurs Pascal pour l'indexation des cellules s'est heurtée à une incompatibilité entre la nécessité d'obtenir une indexation constante des cellules (leur numéro qui ne peut varier pour une bonne cohérence de la base) et les méthodes d'allocation dynamique où l'indexation est fonction de la position en mémoire centrale de la cellule. Cette incompatibilité, cumulée à la non-transportabilité du stockage sur fichier des pointeurds Pascal nous a fait opter pour une numérotation des cellules par indexation dans un tableau (ou page).

S'il s'est révélé inadéquat d'indexer les cellules de NH et de NR par des pointeurs physiques l'allocation de variables pour les primitives internes (non visibles par l'utilisateur) s'effectue dynamiquement par adressage segmenté. Ceci nous a permis d'optimiser la vitesse d'exécution des fonctions de gestion de la base. Les procédures de "manipulation" d'adresses segmentées écrites, dans un premier temps, de manière récursive (pour la simplicité conceptuelle), ont été réécrites de manière déclarative pour tenir compte des risques d'overflow de la pile de récursivité (risque minime, mais que nous ne voulions pas ignorer).

Le langage utilisé est MS-Pascal de Microsoft.

NH et NR ont été "mis à l'épreuve" par une série de tests systématiques qui ont prouvé leur bon fonctionnement.

NH et NR sont disponibles séparement ou regroupés en un même module (suppression des primitives internes communes) et sont implantés, sur BFM 186 (micro 16 bits à base de 8086, 256 Ko de mémoire centrale, 2 x 1,2 Mo sur disque, sous MS-DOS), au Laboratoire d'Informatique Appliquée à l'Architecture.

UN EXEMPLE D'UTILISATION DE LA BASE

Nous terminerons ce rapport par l'analyse d'un "test" un peu plus complexe qui permettra, nous l'espérons, de fournir au lecteur une approche des possibilités d'utilisation de la base.

Nous avons établi, à partir des modules NH, NR et d'un éditeur graphique 2D (PADAO (1), développé au sein de LI2A), une interface permettant la manipulation d'objets graphiques. La structure de la base ayant été expliquée plus haut, nous donnerons, tout d'abord, un bref aperçu du programme d'édition graphique.

Les objets manipulés par PADAO :

PADAO (Programme d'Assistance au Dessin d'Architecte sur Ordinateur), dans sa version de base (sans liaison avec NH-NR), permet la création et la manipulation (par divers utilitaires de dessin) d'objets graphiques en deux dimensions.

Notre étude, entraînant le développement de PADAO, a eu comme point de départ la théorie d'un Architecte du début du dix-neuvième siècle, JNL Durand (2), définissant un "catalogue" d'éléments architecturaux avec leur règles de syntaxe, et des règles de composition globales concernant la formation de bâtiments (en d'autres termes, une démarche de conception).

Notre premier objectif, création d'un catalogue d'éléments architecturaux définis en deux dimensions et stockés sur disque, nous a amené à développer différents utilitaires de saisie graphique surtout orientés vers l'interactivité et l'ergonomie (deux principes dont nous regrettions souvent l'absence dans les programmes de CAAO).

Nous avons, ensuite, implémenté des primitives de manipulation des éléments (duplication, translation, changement d'échelle, déformation, réunion de deux éléments pour n'en former qu'un seul à nouveau manipulable, élimination...), permettant un processus de conception conforme à celui exposé par Durand.

Au niveau physique, un élément de PADAO peut revêtir deux formes :

-En mémoire centrale : Il est constitué d'un ensemble d'arbres binaires et de piles de pointeurs (nous avons accordé beaucoup d'importance à la rapidité d'exécution des utilitaires) contenant les coordonnées des objets graphiques décrivant l'élément (horizontales, verticales, diagonales, arcs de cercle).

-Sur disque : Il est mémorisé dans un fichier dont chaque primitive graphique est identifiable et réexploitable lors d'un transfert en mémoire centrale.

De l'intérêt d'une interface PADAO-NR-NH :

Un dessin est donc constitué, sur PADAO, par appel (ou création) d'éléments du catalogue et manipulation de ces éléments en vue de la constitution d'une façade ou d'un plan de bâtiment (suivant la représentation des éléments, bien entendu).

L'élément du catalogue peut alors être considéré comme un Type (architectural) et l'élément du dessin comme une occurence du Type. Il devient donc intéressant tant au niveau conceptuel (Architecture) qu'au niveau annalyse (Informatique) de représenter ce dessin non pas en tant qu'ensemble d'éléments mais bien plutôt en tant qu'ensemble d'occurences de Types définis par ailleurs. Ceci est assuré par NH à partir d'une décomposition hiérarchique du dessin en éléments constitutifs.

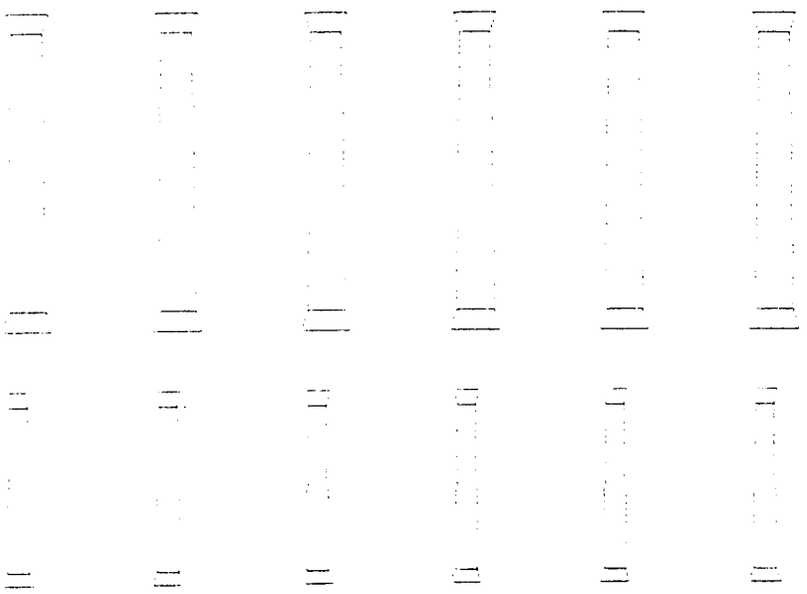
Nous avons voulu de plus, après l'élaboration de la base, réaliser une liaison PADAO-NR pour pouvoir en tester (dans un premier temps) puis en attester (dans un second temps) l'efficacité. Ceci est chose faite. Notre volonté était certes modeste : établir une relation d'échelle entre deux éléments ; nous précisons qu' il s'agit là, bien entendu, d'une échelle géométrique et non d'une échelle architecturale (nous nous excusons de l'"hérésie" auprés de la Théorie de Philippe Boudon (3), mais la complexité de notre travail -et surtout le manque de temps- nous imposait des directives draconiennes). Cette relation peut avoir deux effets (il y a en fait deux cellules "Relation" exploitées dans NR, -voir dessin page suivante-) :

-Le changement d'échelle d'un élément implique (réellement sur l'écran) une même opération sur d'autre éléments sans modifier leur positionnement sur le dessin.

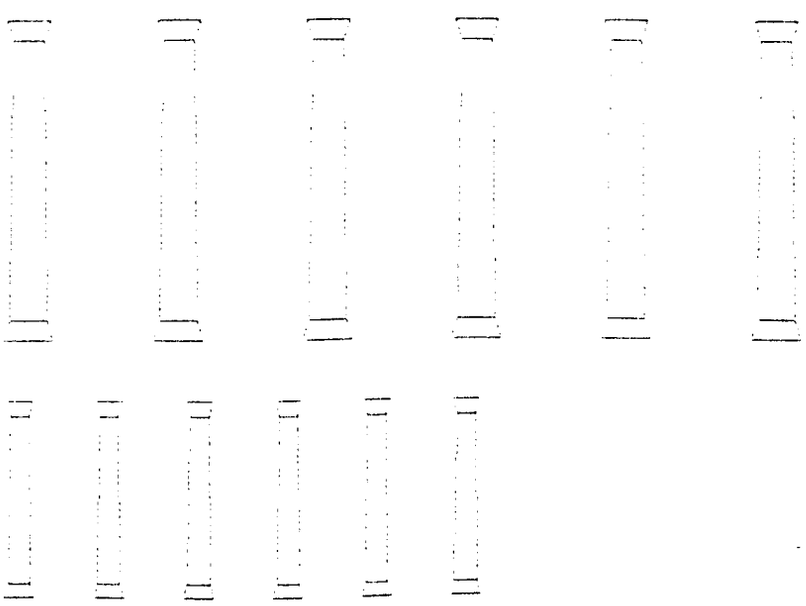
-Le changement d'échelle d'un élément implique la même opération sur d'autres éléments, et entraîne de surcroît leur déplacement (Exemple : le changement d'échelle de deux colonnes entraîne le changement d'échelle de leur entraxe et donc leur déplacement relatif). Il devient alors possible de définir des compositions stables d'éléments telles une colonnade ou une arcade (conformément à la syntaxe architecturale).

Pour tester l'utilité de la structure du noyau de la base, il fallait utiliser NR et NH depuis PADAO, sans modifier ni NR, ni NH, ni PADAO. D'où la nécessité d'écrire une interface entre les deux bases de données communiquant par les protocoles définis dans les chapitres précédents.

-Changement d'échelle effectuée sur un ensemble de colonnes reliées par la relation "Echelle simple" (nous présentons la colonnade avant et après l'activation de la primitive) :



-Changement d'échelle effectuée sur un ensemble de colonnes reliées par la relation "Echelle + Déplacement relatif" :



Réalisation de l'interface

Considérations générales :

A chaque élément de PADAO (qu'il soit créé directement ou amené sur l'écran à partir du catalogue) correspond une cellule dans NH et une cellule dans NR (nous verrons plus loin qu'il peut y avoir un cas particulier).

Nous devons mémoriser les points suivants :

-Index de l'élément dans le catalogue (si l'élément ne figure pas encore au catalogue, son index est créé et communiqué à la base au moment de la sauvegarde du dessin).

-Paramètres de translation de l'élément par rapport à la position sur l'écran de l'élément type.

-Paramètres de changement d'échelle de l'élément par rapport à l'échelle de l'élément type.

-les relations d'échelle qu'il entretient avec d'autres éléments.

-Le cas échéant, les "sous-éléments" issus de la décomposition de l'élément (Exemple : l'élément colonne qu'on peut être amené à décomposer en base + fût + chapiteau, bien qu'il soit généralement plus commode de manipuler l'élément colonne comme un tout).

Ces différents points constituent des "tâches" à soumettre à la base de données.

La répartition des "tâches" :

-Dans NH :

La tête de liste générale (première cellule créée) est assimilée (conceptuellement) au dessin dans sa globalité et n'est pas utilisée comme stockage de paramètres.

La première sous-liste (liste principale) est assimilée à l'ensemble des éléments directement manipulables par PADAO. Chaque cellule possède une file de deux pointeurs de données dans lesquels sont mémorisés :

-L'index de l'élément dans le catalogue (nous verrons plus loin le cas de l'élément composé).
-La translation en X effectuée sur l'élément.
-La translation en Y effectuée sur l'élément.
-Le changement d'échelle subi par l'élément.
-Le numéro de la cellule homologue dans NR.

Une cellule quelconque de cette liste peut être tête de liste de "sous-élément" . Un tel cas peut se présenter lorsqu'un élément est défini par association de plusieurs autres éléments (cas de colonne ou de la colonnade cités plus haut). Ce type de manipulation sur PADAO a, dans NH, les conséquences suivantes (nous prendrons un exemple pour être plus explicite, le cas "colonne1 + colonne2 + ... + colonneN devient colonnade") :

-Création d'une cellule (homologue de la colonnade) dans la liste principale. L'index de cette cellule est mis à -1 (l'élément ne figure pas, tel quel, au catalogue).

-Création d'une sous-liste pour cette cellule. Cette sous-liste est formée de N cellules initialisées avec les paramètres actuels des cellules homologues des N colonnes.

-Suppression des cellules homologues des N colonnes dans la liste principale.

-Communication du numéro de la "cellule colonnade" à PADAO (elle devient l'interlocuteur de PADAO avec la base pour l'ensemble des N colonnes, en particulier pour le stockage de certains paramètres de manipulation; nous verrons plus loin le cas particulier du changement d'échelle).

Ceci nous permet de maintenir une bijection entre "l'écran de PADAO" et la liste principale de NH tout en permettant à chaque élément d'être décomposable et d'offrir ses "sous-éléments" aux primitives de manipulation.

A son tour, une décomposition d'élément (si elle est possible, c'est à dire si l'élément est composé) se présente comme suit (nous prendrons le même exemple) :

-Lecture des paramètres de la cellule homologue de colonnade et passage de ces paramètres aux N cellules (des N colonnes) de la sous-liste (mise à jour des tranformations effectuées sur l'ensemble des éléments pour chacun d'entre eux).

-Création de N cellules dans la liste principale. Ces N cellules sont initialisées avec les paramètres actuels des cellules (de la sous-liste) homologues des N colonnes.

-Suppression de la cellule colonnade de la liste principale (et donc suppression de sa sous-liste ; cf la procédure Supcl).

Dans NR :

Les rapports que les éléments de PADAO entretiennent avec NR diffèrent quelque peu de ceux que nous venons de décrire pour NH. Pour NR, le regroupement de plusieurs éléments pour n'en former qu'un seul, n'entraîne pas de formation ou de suppression de cellules ; ainsi l'élément colonnade n'a aucun homologue dans NR. De plus NR possède deux éléments privilégiés n'ayant pas de correspondance dans PADAO. Ces deux éléments (deux Objets que nous nommerons O1, O2) sont créés à l'ouverture de la base et deviendront, respectivement, le support des relations "Echelle simple" et le support des relations "Echelle + Déplacement relatif des éléments" (cf plus haut la description des deux relations).

Une primitive de PADAO permet d'établir explicitement la première relation (Echelle simple) entre deux éléments. les actions correspondantes dans NR peuvent alors se décomposer en trois cas :

-Aucun élément ne possède déja de relation d'échelle ("Echelle simple") : il y a alors création d'une cellule Relation attachée à O1 (insérée en tête de la liste des Relations de celui-ci), puis création de deux cellules Associateurs (une pour chaque élément) "attachées" à la Relation nouvellement créée.

-Un élément possède une relation d'échelle (avec d'autres éléments) : il n'y pas de création de cellule Relation. L'élément ne possédant pas de relation d'échelle est alors inséré dans la liste d'association contenant l'autre (par création d'une cellule Associateur). La relation "Echelle simple" est donc transitive (ce qui est, pour le moins, logique).

-Les deux éléments possédent une relation d'échelle. Il y a alors concaténation des deux listes de relation : une cellule Relation est conservée et devient tête de la liste de tous les éléments concernés, l'autre cellule Relation est supprimée (ainsi que les cellules Associateurs de sa liste d'association, cf la procédure SupRel).

La seconde relation, "Echelle + Déplacement relatif des éléments", est induite implicitement dans NR lors du regroupement de plusieurs éléments pour n'en former qu'un seul. Les mécanismes d'établissement d'un tel schéma sont identiques à ceux décrits précédemment (l'Objet tête de liste de relation est, cette fois-ci, 02).

Il s'est révélé nécessaire d'établir cette relation entre les cellules des "sous-éléments" ("sous-éléments" dont l'intégrité de l'ensemble est pourtant déjà identifiable dans NH) pour pouvoir réaliser des changements d'échelle corrects d'un élément composé.

Contrairement à l'activation d'une translation, pour un élément composé, l'activation de la primitive de changement d'échelle ne doit pas fournir des paramètres de manipulation identiques à chaque cellule de "sous-élément". En effet le déplacement relatif de chacun de ces derniers (pour maintenir l'intégrité globale de l'élément) est particulier (par définition puisqu'il est relatif). Il faudra donc communiquer à la base des paramétres spécifiques pour chaque cellule de "sous-élément" (et non pas des paramètres globaux pour la cellule de l'élément composé) . (Nous demandons au lecteur de bien vouloir excuser les nombreuse "redites" qui nous semblent nécessaires à une meilleure compréhension).

PADAO doit donc être en mesure de "voir" chaque cellule de "sous-élément". Ceci entraîne donc une "décomposition" dans NH (voir plus haut les mécanismes de décomposition) et le positionnement des cellules homologues de chaque "sous-élément" dans la liste principale (pour établir un passage de paramètres direct entre elles et PADAO). Celui-ci retrouve donc la "trace de la globalité" de l'élément composé dans NR (ainsi que la Relation "Echelle simple" qu'il entretient avec d'autres éléments) et peut ainsi maintenir une cohérence entre la manipulation effectuée et la base de données.

Une fois la primitive de changement d'échelle abandonnée (quand l'utilisateur juge, après une ou plusieurs tentatives, que la manipulation effectuée sur l'élément est convenable), la composition (et donc l'existence de l'élément composé) est induite implicitement dans NH (tout le processus reste donc transparent pour l'utilisateur).

PADAO, dans sa nouvelle version, possède donc ce que nous nommerons (sûrement un peu inconsidérémment), un "semblant" d'aide à la conception (plus exactement une assistance à la manipulation d'éléments). Il est de plus capable, lors du rappel à l'écran d'un dessin stocké antérieurement, de présenter à l'utilisateur des conditions de manipulation identiques à celles qu'il avait définies (en termes du langage Lisp nous dirons qu'il y a conservation de l'univers). L'individualité des éléments est préservée ainsi que le schéma relationnel et les possibilités de décomposition.

Une duplication d'un élément sur l'écran entraîne la duplication des schémas Relationnel et Hiérarchique dans la base.

Une Composition (Architecturale cette fois-ci) n'est plus figée comme un tout, lors de la sauvegarde sur disque, mais peut-être aisément remodelable.

Le logiciel PADAO est implanté, sur BFM 186, au Laboratoire d'Informatique Appliquée à l'Architecture.

CONCLUSION GENERALE

Nous espérons avoir exposé une annalyse assez précise des modes et des possibilités d'exploitation de l'ensemble de la base de données, ainsi qu'une étude réelle des spécificités de chaque noyau.

Nous souhaitons que le futur utilisateur de l'ensemble "NR + NH" trouve, dans ce texte, un quelconque secours au difficile problème de conceptualisation des logiciels destinés à l'Architecture.

Quant à nous, après avoir apprécié les qualités de souplesse et de modularité du noyau NR+NH, nous allons poursuivre "l'interfaçage" d'utilitaires communiquant entre eux à travers ce noyau.

--==*==--

Jean-Pierre Goulette

LI2A, Mai 1984.

APPENDICES

Les programmes qui sont retranscrits ne détaillent que les procédures externes de communication entre NR ou NH et le monde extérieur.

Nous publions par ailleurs, dans ce listing, les noms des primitives de gestion interne des deux noyaux pour que l'utilisateur se fasse une idée de la totalité de la structure. Les listings ne représentent donc pas le source complet des modules NR et NH.

L'obtention du code exécutable des deux modules est soumis à des accords conjoints entre le Secrétariat de la Recherche Architecturale, le CIMA et LI2A.

```
        (*************************************************************)
        (* Ce module comporte des procédures ou fonctions            *)
        (* "publiques" (qui sont celles du cahier des charges        *)
        (* de NR) et des procédures ou fonctions internes            *)
        (* nécessaires à la mise en oeuvre des premières.            *)
        (* Ces procédures ou fonction "publiques" sont formées       *)
        (* de 2 parties :                                            *)
        (*       _Un dépistage d'erreurs dans la communication       *)
        (*       de valeurs par des programmes extérieurs et qui     *)
        (*       n'a donc plus d'utilité quand ces programmes        *)
        (*       sont "debuggés" (excepté pour les manipulations     *)
        (*       de fichier).                                        *)
        (*       _Le corps de la procédure.                          *)
        (*************************************************************)

Module NR;
Const LimPNR = 50; (* Nombre de cellules dans un tableau (ou page) *)
      NbPNR = 4;   (* Nombre de pages en mémoire centrale *)
      Limem=1600;  (* Test pour Memavl<LImite inférieur MEMoire *)

Type PagNR = 0..LimPNR;          (* Index des cellules *)
     NbrePNR = 1..NbPNR;         (* Index des tableaux *)
     TabDon = Array(.1..3.) of Integer;
     Chaine3 = Lstring(3);
     Chaine12 = Lstring(12);
     Pointvid = Ads of Vcel;     (* Pointeur de cellule vide *)
     Vcel =     Record           (* Cellule vide : *)
                W : Integer;     (* son numéro *)
                P : Pointvid     (* pointeur sur la suivante *)
                End;
Nat_Cel = (Obj,Rel,Ass,Don);     (* Nature des Cellules *)
Cellule = Record                 (* Constitution d'une cellule : *)
        Nat : Nat_Cel;           (* sa nature *)
        Code : Chaine3;          (* son code *)
        P1,P2,P3,P4 : Integer    (* pointeur sur d'autres cellules *)
        End;
TabNR = Array(.PagNR.) of Cellule;       (* Tableau de cellules *)
FichierNR = File of Cellule;             (* Fichier de cellules *)

Var PageNR:Array(.NbrePNR.) of TabNR;
        (* Les tableaux en mémoire centrale *)
    DebNR:Array(.NbrePNR.) of Integer;
        (* Le début de chaque tableau *)
    EcrNR:Array(.NbrePNR.) of Boolean;
        (* Le tableau a-t-il ètè modifié ? *)
    EtaNR:Array(.NbrePNR.) of Integer;
        (* Premier...dernier tableau utilisé *)

    FichNR:FichierNR;  (* Le fichier de cellules *)
    NumNR:Integer;     (* le nombre total de cellules *)
    RacVidNR:Pointvid; (* Racine des pointeurs de cellules vides *)
    FVidNR:File of Integer; (* Fichier de cellules vides *)
```

```
        (*************************************************************)
        (* Procédures MS-Pascal.                                     *)

Function Memavl:Word;Extern; (* Mémoire disponible *)
Function Getmqq(Wants:Word):Adsmem;Extern;
    (* Demande d'allocation d'un bloc supplémentaire *)
Procedure  Dismqq(Block:Adsmem);Extern;  (* Abandon d'un bloc *)



        (*************************************************************)
        (* Procédure interne. Alloue une nouvelle adresse pour       *)
        (* une cellule vide. Active Getmqq.                          *)

Procedure NewV(Vars P:Pointvid);
Cette  Procédure alloue une nouvelle adresse segmentée (codée sur
4 octets) pour le pointeur P (la Procédure New de Pascal standard
n'alloue  q'une  adresse codée sur 2 octets dans le  segment  par
défaut).

        (*************************************************************)
        (* Procédure interne. Dispose une adresse de cellule         *)
        (* vide. Active Dismqq.                                      *)

Procedure DispV(Vars P:Pointvid);


        (*************************************************************)
        (* Fonction interne. Sans commentaire.                       *)

Function Inf(W1,W2:Integer):Integer;


        (*************************************************************)
        (* Procédure interne. Ecrit un message d'erreur d'après      *)
        (* la valeur de Ioer (<>0).                                  *)

Procedure Erreur(Ioer:Word);
Les  erreurs  pouvant  advenir au cours d'une  lecture  ou  d'une
écriture sur disque ne sont pas fatales et sont "trapées" par  le
programme.  Cette  procédure affiche une message correspondant au
type de l'erreur survenue.


        (*************************************************************)
        (* Procédure interne. Ecrit la page sur le fichier des       *)
        (* cellules si elle a ete modifiee en mémoire centrale       *)

Procedure PutPNR(Nt:Integer);
```

```
        (*************************************************************)
        (* Fonction interne. "Balance" une nouvelle page du          *)
        (* fichier en mémoire centrale à la place de la page la      *)
        (* plus anciennement consultée (qu'elle écrit sur            *)
        (* fichier par un appel à PutPNR). Communique en retour     *)
        (* l'index de cette nouvelle page.                           *)

Function GetPNR(Nc:Integer):Integer;


        (*************************************************************)
        (* Fonction interne. Donne l'indice de la cellule de         *)
        (* numéro Nc dans le tableau d'index Nt.                     *)

Function IndNR(Nc,Nt:Integer):Integer;


        (*************************************************************)
        (* Procédure interne. Initialise les tableaux. Elle est      *)
        (* appelée à la création ou à l'ouverture d'un fichier       *)

Procedure InitNR;


        (*************************************************************)
        (* Fonction interne. Donne l'index du tableau où se          *)
        (* trouve la cellule de numéro Nc. Si ce tableau n'est       *)
        (* pas en mémoire centrale, il est appelé par                *)
        (* l'activation de GetPNR.                                   *)

Function PresNR(Nc:Integer):Integer;


        (*************************************************************)
        (* Procédure interne. Met à jour le tableau EtaNR            *)
        (* indexant l'ancienneté de consultation des pages.          *)

Procedure DerNR(Nt:Integer);


        (*************************************************************)
        (* Fonction interne. Donne la cellule de numéro Nc et        *)
        (* met à jour le tableau d'ancienneté de consultation.       *)

Function GetCNR(Nc:Integer):Cellule;
```

```
        (*************************************************************)
        (* Procédure interne. Affecte la nature Nature, le code *)
        (* C, les pointeurs A, A2, A3, A4 à la cellule de       *)
        (* numéro Nc en fonction de sa position en mémoire      *)
        (* centrale. Si la cellule n'est pas en mémoire         *)
        (* centrale, elle appelée par l'activation de PresNR.   *)
        (* Met à jour les tableaux d'écriture et de             *)
        (* consultation.                                        *)

Procedure AffNR(Nc:Integer;Nature:Nat_Cel;C:Chaine3;A,A2,A3,A4:Integer);


        (*************************************************************)
        (* Procédure interne. Corrige le champ (pointeur)       *)
        (* Champs de la cellule Nc avec la valeur W. Si la      *)
        (* cellule n'est pas en mémoire centrale, elle est      *)
        (* appelée par l'activation de PresNR. Met à jour les   *)
        (* tableaux d'écriture et de consultation.              *)

Procedure CorNR(Nc,W,Champs:Integer);


        (*************************************************************)
        (* Fonction interne. Dépistage d'erreur pour les        *)
        (* programmes exploitant NR. Retourne la valeur True    *)
        (* si la cellule Nc n'existe pas ou fait partie des     *)
        (* cellules vides.                                      *)

Function InvNR(Nc:Integer):Boolean;


        (*************************************************************)
        (* Procédure interne. Insère le numéro de cellule Nc    *)
        (* en tête de pile des cellules vides. Cette procédure  *)
        (* est appelée à chaque suppression de cellule et permet*)
        (* de récupérer le "trou" de numéro Nc pour une         *)
        (* prochaine création de cellule.                       *)

Procedure InsVNR(Nc:Integer);
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'un fichier direct         *)
        (* de cellules NR, de nom Nomfic (communiqué par des         *)
        (* programmes extérieurs). Vérifie que ce fichier            *)
        (* n'existe pas. Réserve la premiere cellule (non            *)
        (* valide) pour mémoriser le nombre total de cellules.       *)
        (* Initialisations diverses. Si un fichier existe déja       *)
        (* sous le même nom la création est avortée (le              *)
        (* fichier est conservé) et CrNR retourne False.             *)

Function CrNR(Nomfic:Chaine8):Boolean;
Var Nomf:Chaine12;

Begin
Nomf:=Nomfic;
Assign(FichNR,Nomfic);
FichNR.Mode:=Direct;FichNR.Trap:=True;FichNR.Errs:=0;
Reset(FichNR);
If FichNR.Errs=0 Then
        Begin
        CrNr:=False;
        Close(FichNR)
        End
Else    Begin
        CrNR:=True;
        FichNR.Errs:=0;
        Rewrite(FichNR);
        If FichNR.Errs<>0 Then
                Begin
                Erreur(FichNR.Errs);FichNR.Errs:=0
                End
        Else    Begin
                NumNR:=1;FichNR^.P1:=NumNR;Put(FichNR);
                If FichNR.Errs<>0 Then
                        Begin
                        Erreur(FichNR.Errs);FichNR.Errs:=0
                        End;
                Concat(Nomf,'.vid');
                Assign(FVidNR,Nomf);
                RacVidNR.S:=0;InitNR
                End
        End;
FichNR.Trap:=False
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Ouvre (s'il existe) un fichier       *)
        (* direct de cellules NR de nom Nomfic (communiqué par       *)
        (* des programmes extérieurs). Ouvre (s'il existe) le        *)
        (* fichier de cellules vides et l'insère dans la pile        *)
        (* des cellules vides. Si le fichier n'existe pas            *)
        (* InitFNR renvoie False                                     *)

Function InitFNR(Nomfic:Chaine8):Boolean;
Var Nomf:Chaine12;

Begin
Nomf:=Nomfic;Assign(FichNR,Nomfic);
FichNR.Mode:=Direct;FichNR.Trap:=True;FichNR.Errs:=0;
Reset(FichNR);FichNR.Trap:=False;
If FichNR.Errs<>0 Then
        Begin
        InitFNR:=False;
        FichNR.Errs:=0
        End
Else    Begin
        InitFNR:=True;
        NumNR:=FichNR^.P1;
        RacVidNR.S:=0;
        Concat(Nomf,'.vid');
        Assign(FVidNR,Nomf);
        FVidNR.Trap:=True;FVidNR.Errs:=0;
        Reset(FVidNR);
        If FVidNR.Errs=0 Then
                Begin
                While Not Eof(FVidNR) Do
                                Begin
                                InsVNR(FVidNR^);
                                Get(FVidNR)
                                End;
                Close(FVidNR)
                End;
        FVidNR.Errs:=0;FVidNR.Trap:=False;
        InitNR
        End;
FVidNR.Trap:=False
End;
```

```
        (*************************************************************)
        (* Procédure interne. "Déverse " la pile de numéros de       *)
        (* cellules vides dans le fichier de cellules vides.         *)

Procedure InsFcNR(Vars Po:Pointvid);


        (*************************************************************)
        (* Procédure "PUBLIQUE". Ferme un fichier de cellules        *)
        (* NR ainsi que son fichier de cellules vides (s'il y        *)
        (* en a). S'il n'y a plus de celules vides, ce fichier       *)
        (* (de cellule vides) est effacé (s'il existait).            *)

Procedure FinNR;
Var I:Integer;

Begin
For I:=1 To NbPNR Do PutPNR(I);
FichNR.Trap:=True;FichNR.Errs:=0;Seek(FichNR,1);
If FichNR.Errs<>0 Then
        Begin
        Erreur(FichNR.Errs);FichNR.Errs:=0
        End
Else    Begin
        FichNR^.P1:=NumNR;Put(FichNR);
        Close(FichNR);NumNR:=0;
        If FichNR.Errs<>0 Then
                Begin
                Erreur(FichNR.Errs);FichNR.Errs:=0
                End;
        FVidNR.Trap:=True;FVidNR.Errs:=0;
        If RacVidNR.S<>0 Then
                        Begin
                        Rewrite(FVidNR);
                        If FVidNR.Errs<>0 Then
                                Begin
                                Erreur(FVidNR.Errs);FVidNR.Errs:=0
                                End
                        Else    Begin
                                InsFcNR(RacVidNR);
                                Close(FVidNR)
                                End
                        End
        Else    Begin
                Reset(FVidNR);
                If FVidNR.errs=0 Then Discard(FVidNR);
                FVidNR.Errs:=0
                End;
        FVidNR.Trap:=False
        End;
FichNR.Trap:=False
End;
```

```
        (*************************************************************)
        (* Fonction interne. Alloue un numéro pour une nouvelle      *)
        (*  cellule soit :                                           *)
        (*         _En le faisant pointer sur une cellule vide       *)
        (*          si RacVidNR<>Nil.                                *)
        (*         _En Incrémentant NumNR.                           *)
        (* Remplit cette cellule avec les valeurs Nat, C,N1,         *)
        (* N2, N3, N4 par l'activation de AffNR. Donne en            *)
        (* retour le numéro de la cellule créée.                     *)

Function NouvNR(Nat:Nat_Cel;C:Chaine3;N1,N2,N3,N4:Integer):Integer;


        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'une cellule Objet         *)
        (* isolée par activation de NouvNR. Donne en retour le       *)
        (* numéro de l'objet créé. Les valeurs C et Mot sont         *)
        (* communiquées par des programmes extérieurs.               *)

Function NewObj(C:Chaine3;Mot:Integer):Integer;
Begin
If C(.1.)='0' Then NewObj:=0
                Else NewObj:=NouvNR(Obj,C,Mot,0,0,0)
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'une cellule Relation      *)
        (* par activation de NouvNR. Les champs sont                 *)
        (* communiqués par des programmes extérieurs.                *)
        (* Place cette Relation en tête de la liste des              *)
        (* Relations de l'Objet NcObj (corrige P2 dans NcObj)        *)
        (* avec une liste d'Association vide.                        *)
        (* Donne en retour le numéro de la Relation créée.           *)

Function NewRel(NcObj:Integer;C:Chaine3;Mot:Integer):Integer;
Var Pos:Integer;

Begin
If (C(.1.)='0') or (InvNR(NcObj)) Then Newrel:=0
        Else If GetCNR(NcObj).Nat<>Obj Then Newrel:=0
                Else    Begin
                        Pos:=NouvNR(Rel,C,Mot,GetCNR(NcObj).P2,NcObj,0);
                        CorNR(NcObj,Pos,2);
                        NewRel:=Pos
                        End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'une cellule               *)
        (* Associateur par activation de NouvNR. Les champs          *)
        (* sont communiqués par des programmes extérieurs.           *)
        (* Place cette cellule en tête de la liste des               *)
        (* dépendances de l'Objet NcObj (corrige P3 dans             *)
        (* NcObj).                                                   *)
        (* Place cette cellule en tête de la liste des               *)
        (* Associateurs de la Relation NcRel (corrige P4 dans        *)
        (* NcRel).                                                   *)
        (* Donne en retour le numéro de l'Associateur crée.          *)

Function NewAss(NcObj,NcRel:Integer;C:Chaine3):Integer;
Var Pos:Integer;

Begin
If (C(.1.)='0') or (InvNR(NcObj)) or (InvNR(NcRel)) Then NewAss:=0
Else If (GetCNR(Ncobj).Nat<>Obj) or (GetCNR(NcRel).Nat<>Rel) Then
                                                        NewAss:=0
Else    Begin
      Pos:=NouvNR(Ass,C,NcRel,NcObj,GetCNR(NcObj).P3,GetCNR(NcRel).P4);
        CorNR(NcObj,Pos,3);CorNR(NcRel,Pos,4);NewAss:=Pos
        End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'une cellule               *)
        (* Associateur par activation de NouvNR. Les champs          *)
        (* sont communiqués par des programmes extérieurs.           *)
        (* Place cette cellule en tête de la liste des               *)
        (* dépendances de l'objet NcObj (corrige P3 dans             *)
        (* NcObj).                                                   *)
        (* Place cette cellule sur la liste d'association à          *)
        (* laquelle appartient la cellule NcAss, à la suite de       *)
        (* cette cellule (corrige P4 dans NcAss).                    *)
        (* Donne en retour le numéro de l'Associateur crée.          *)

Function SuccAss(NcObj,NcAss,:Integer;C:Chaine3):Integer;
Var Pos:Integer;
    Cel:Cellule;

Begin
If (C(.1.)='0') or (InvNR(NcObj)) Or (InvNR(NcAss)) Then Succass:=0
Else If (GetCNR(NcObj).Nat<>Obj) or (GetCNR(Ncass).Nat<>Ass) Then
                                                        Succass:=0
        Else    Begin
                Cel:=GetCNR(NcAss);
                Pos:=NouvNR(Ass,C,Cel.P1,NcObj,GetCNR(NcObj).P3,Cel.P4);
                CorNR(NcObj,Pos,3);CorNR(NcAss,Pos,4);SuccAss:=Pos
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'une cellule Donnée        *)
        (* associée (en fin de liste des Données) à l'Objet          *)
        (* NcObj par activation de NouvNR. Les champs sont           *)
        (* communiqués par des programmes extérieurs.                *)
        (* Corrige le pointeur P4 dans la cellule précedente         *)
        (* (qui peut être l'Objet lui-meme).                         *)
        (* Donne en retour le numéro de la Donnée créée.             *)

Function NewDat(NcObj:Integer;C:Chaine3;Itab:TabDon):Integer;
Var Pos,I:Integer;
    Provi:Cellule;

Begin
If (C(.1.)='0') or (InvNR(NcObj)) Then NewDat:=0
Else If GetCNR(NcObj).Nat<>Obj Then NewDat:=0
        Else    Begin
                Pos:=NouvNR(Don,C,Itab(.1.),Itab(.2.),Itab(.3.),0);
                I:=NcObj;Provi:=GetCNR(I);
                While Provi.P4<>0 Do    Begin
                                        I:=Provi.P4;
                                        Provi:=GetCNR(I)
                                        End;
                CorNR(I,Pos,4);NewDat:=Pos
                End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Retourne, aux programmes             *)
        (* extérieurs, la valeur des Données contenues dans la       *)
        (* cellule Ncd (par activation de GetCNR).                   *)

Function Lirdat(Ncd:Integer):TabDon;
Var T:TabDon;
    Cel:Cellule;

Procedure Erreur;
Begin
Writeln('LECTURE IMPOSSIBLE');
Lirdat(.1.):=0;Lirdat(.2.):=0;Lirdat(.3.):=0
End;

Begin
If InvNR(Ncd) Then Erreur
Else If GetCNR(Ncd).Nat<>Don Then Erreur
        Else    Begin
                Cel:=GetCNR(Ncd);
                T(.1.):=Cel.P1;T(.2.):=Cel.P2;T(.3.):=Cel.P3;
                Lirdat:=T
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Retourne, aux programmes             *)
        (* extérieurs, le Code de la cellule Nc (par activation      *)
        (* de GetCNR).                                               *)

Function Lircod(Nc:Integer):Chaine3;
Begin
If InvNR(Nc) Then Begin Writeln('LECTURE IMPOSSIBLE');Lircod:='0' End
        Else Lircod:=GetCNR(Nc).Code
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Retourne, aux programmes             *)
        (* extérieurs, le Mot de la cellule Nc (par activation       *)
        (* de GetCNR).                                               *)

Function Lirmot(Nc:Integer):Integer;

Procedure Erreur;
Begin
Writeln('LECTURE IMPOSSIBLE');Lirmot:=0
End;

Begin
If InvNR(Nc) Then Erreur
Else If GetCNR(Nc).Nat=Ass Then Erreur
        Else Lirmot:=GetCNR(Nc).P1
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Modifie le Code de la cellule       *)
        (* Nc avec la valeur C communiquée par des programmes        *)
        (* extérieurs. Met à jour les tableaux d'écriture et         *)
        (* de consultation.                                          *)

Procedure Modcod(Nc:Integer;C:Chaine3);
Var Nt:Integer;

Begin
If (C(.1.)='0') or (InvNR(Nc)) Then Writeln('MODIFICATION IMPOSSIBLE')
        Else    Begin
                Nt:=PresNR(Nc);
                PageNR(.Nt.)(.IndNR(Nc,Nt).).Code:=C;
                EcrNR(.Nt.):=True;DerNR(Nt)
                End
End;
```

```
        (*************************************************************)
        (* Procédure "PUBLIQUE". Modifie le Mot de la cellule        *)
        (* Nc avec la valeur Info communiquée par des                *)
        (* programmes extérieurs (par activation de CorNR).          *)

Procedure Modmot(Nc,Info:Integer);
Begin
If InvNR(Nc) Then Writeln('MODIFICATION IMPOSSIBLE')
Else If GetCNR(Nc).Nat=Ass Then Writeln('MODIFICATION IMPOSSIBLE')
        Else CorNR(Nc,Info,1)
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Remplace les Données de la          *)
        (* cellule Ncd par le contenu de Ndata communiqué par        *)
        (* des programmes extérieurs (par activation de              *)
        (* CorNR).                                                   *)

Procedure Moddat(Ncd:Integer;Ndata:TabDon);
Var i:Integer;

Begin
If InvNR(Ncd) Then Writeln('MODIFICATION IMPOSSIBLE')
Else If GetCNR(Ncd).Nat<>Don Then Writeln('MODIF. IMPOSSIBLE')
        Else For I:=1 To 3 Do CorNR(Ncd,Ndata(.I.),I)
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE".
        (* Si C<>'0' --> Parcours de l'anneau de dépendance à        *)
        (* partir de la cellule Ncoa (Objet ou Associateur) non      *)
        (* comprise jusqu'au 1° Associateur ayant le code            *)
        (* cherché ou jusqu'à la fin de liste.                       *)
        (* Si C='0' --> Retour du numéro de la cellule               *)
        (* Associateur suivant Ncoa (Objet ou Associateur) sur       *)
        (* l'anneau de dépendance (ou retour de 0 si Ncoa est        *)
        (* fin de liste).                                            *)
        (* Ces parcours se font par activation de GetCNR.            *)

Function Nxtdep(Ncoa:Integer;C:Chaine3):Integer;
Var Provi:Integer;
Begin
If InvNR(Ncoa) Then     Nxtdep:=0
Else If (GetCNR(Ncoa).Nat<>Obj) And (GetCNR(Ncoa).Nat<>Ass) Then
                                                    Nxtdep:=0
        Else    Begin
                Provi:=GetCNR(Ncoa).P3;
                If C(.1.)<>'0' Then
      While Provi<>0 And Then GetCNR(Provi).Code<>C Do
                                        Provi:=GetCNR(Provi).P3;
                Nxtdep:=Provi
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE".                                      *)
        (* Si C<>'0' --> Parcours de la liste d'association à        *)
        (* partir de la cellule Ncra (Relation ou Associateur)       *)
        (* non comprise jusqu'au 1° Associateur ayant le code        *)
        (* cherché ou jusqu'à la fin de liste.                       *)
        (* Si C='0' --> Retour de la cellule Associateur             *)
        (* suivant Ncra (Relation ou Associateur) sur la liste       *)
        (* d'association (ou retour de 0 si Ncra fin de liste)       *)
        (* Ces parcours se font par activation de GetCNR.            *)

Function Nxtass(Ncra:Integer;C:Chaine3):Integer;
Var Provi:Integer;

Begin
If InvNR(Ncra) Then Nxtass:=0
Else If (GetCNR(Ncra).Nat<>Rel) And (GetCNR(Ncra).Nat<>Ass) Then
                                                    Nxtass:=0
        Else    Begin
                Provi:=GetCNR(Ncra).P4;
                If C(.1.)<>'0' Then
   While Provi<>0 And Then GetCNR(Provi).Code<>C Do
                                        Provi:=GetCNR(Provi).P4;
                Nxtass:=Provi
                End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE".                                      *)
        (* Si C<>'0' --> Parcours de la liste de relation à          *)
        (* partir de la cellule Ncro (Objet ou Relation) non         *)
        (* comprise jusqu'à la 1° Relation ayant le code             *)
        (* cherché ou jusqu'à la fin de liste.                       *)
        (* Si C='0' --> Retour du numéro de la cellule               *)
        (* Relation suivant Ncro (Objet ou Relation) sur la          *)
        (* liste de relation (ou retour de 0 si Ncro est la          *)
        (* fin de liste).                                            *)
        (* Ces parcours se font par activation de GetCNR.            *)

Function Nxtrel(Ncro:Integer;C:Chaine3):Integer;
Var Provi:Integer;
Begin
If InvNR(Ncro) Then Nxtrel:=0
Else If (GetCNR(Ncro).Nat<>Rel) And (GetCNR(Ncro).Nat<>Obj) Then
                                                    Nxtrel:=0
        Else    Begin
                Provi:=GetCNR(Ncro).P2;
                If C(.1.)<>'0' Then
   While Provi<>0 And Then GetCNR(Provi).Code<>C Do
                                        Provi:=GetCNR(Provi).P2;
               Nxtrel:=Provi
               End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE".                                      *)
        (* Si C<>'0' --> Parcours de la liste de donnée à            *)
        (* partir de la cellule Ncod (Objet ou Donnée) non           *)
        (* comprise jusqu'à la 1° Donnée ayant le code cherché       *)
        (* ou jusqu'à la fin de liste.                               *)
        (* Si C='0' --> Retour du numéro de la cellule Donnée        *)
        (* suivant Ncod (Objet ou Donnée) sur la liste de            *)
        (* donnée (ou retour de 0 si Ncod fin de liste).             *)
        (* Ces parcours se font par l'activation de GetCNR.          *)

Function Nxtdat(Ncod:Integer;C:Chaine3):Integer;
Var Provi:Integer;

Begin
If InvNR(Ncod) Then Nxtdat:=0
Else If (GetCNR(Ncod).Nat<>Obj) And (GetCNR(Ncod).Nat<>Don) Then
                                                    Nxtdat:=0
        Else    Begin
                Provi:=GetCNR(Ncod).P4;
                If C(.1.)<>'0' Then
   While Provi<>0 And Then GetCNR(Provi).Code<>C Do
                                        Provi:=GetCNR(Provi).P4;
                Nxtdat:=Provi
                End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Retourne le numéro de l'Objet        *)
        (* tête de liste de relation si Ncra est une Relation,       *)
        (* ou l'Objet tête de liste de dépendance si Ncra est        *)
        (* un associateur (par activation de GetCNR).                *)

Function Lhdobj(Ncra:Integer):Integer;
Var Cel:Cellule;

Begin
If InvNR(Ncra) Then Lhdobj:=0
Else If (GetCNR(Ncra).Nat<>Rel) And (GetCNR(Ncra).Nat<>Ass) Then
                                                    Lhdobj:=0
        Else    Begin
                Cel:=GetCNR(Ncra);
                Case Cel.Nat Of
                        Rel : Lhdobj:=Cel.P3;
                        Ass : Lhdobj:=Cel.P2
                        End
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Retourne le numéro de la             *)
        (* Relation tête de liste d'association contenant            *)
        (* l'Associateur Nca (par activation de GetCNR).             *)

Function Lhdrel(Nca:Integer):Integer;
Begin
If InvNR(Nca) Then Lhdrel:=0
Else If GetCNR(Nca).Nat<>Ass Then Lhdrel:=0
        Else Lhdrel:=GetCNR(Nca).P1
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime la cellule de Donnée       *)
        (* Ncd appartenant à la liste de Donnée de Ncobj (par        *)
        (* activation de InsVNR).                                    *)
        (* Corrige le pointeur P4 de la cellule précedente.          *)

Procedure Supdon(NcObj,Ncd:Integer);
Var Provi:Integer;
    Cel:Cellule;

Begin
If (InvNR(NcObj)) or (InvNR(Ncd)) Then Return;
If (GetCNR(NcObj).Nat<>Obj) or (GetCNR(Ncd).Nat<>Don) Then Return;
Provi:=Ncobj;Cel:=GetCNR(Provi);
While Cel.P4<>Ncd And Then Cel.P4<>0 Do Begin
                                        Provi:=Cel.P4;Cel:=GetCNR(Provi)
                                        End;
CorNR(Provi,GetCNR(Cel.P4).P4,4);
InsVNR(Ncd)
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime toutes les cellules        *)
        (* données associées à l'Objet Nco (par activation de        *)
        (* Supdon).                                                  *)

Procedure Supdat(Nco:Integer);
Begin
If InvNR(Nco) Then Return;
If GetCNR(Nco).Nat<>Obj Then Return;
While GetCNR(Nco).P4<>0 Do Supdon(Nco,GetCNR(Nco).P4)
End;
```

```
        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime la cellule                 *)
        (* Associateur Nca (par activation de InsVNR) et met à       *)
        (* jour les 2 listes auxquelles il appartenait.              *)

Procedure Supass(Nca:Integer);
Var Provi:Integer;
    Cel:Cellule;

Begin
If InvNR(Nca) Then Return;
If GetCNR(Nca).Nat<>Ass Then Return;
Provi:=GetCNR(Nca).P1;Cel:=GetCNR(Provi);
While Cel.P4<>Nca Do    Begin
                        Provi:=Cel.P4;Cel:=GetCNR(Provi)
                        End;
Cel:=GetCNR(Nca);
CorNR(Provi,Cel.P4,4);
Provi:=Cel.P2;Cel:=GetCNR(Provi);
While Cel.P3<>Nca Do    Begin
                        Provi:=Cel.P3;Cel:=GetCNR(Provi)
                        End;
CorNR(Provi,GetCNR(Nca).P3,3);
InsVNR(Nca)
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime la Relation Ncrel ;        *)
        (* si la liste d' association de cette relation n'est        *)
        (* pas vide, tous les Associateurs de la liste sont          *)
        (* supprimés.                                                *)

Procedure Suprel(Ncr:Integer);
Var Provi:Integer;
    Cel:Cellule;

Begin
If InvNR(Ncr) Then Return;
If GetCNR(Ncr).Nat<>Rel Then Return;
While GetCNR(Ncr).P4<>0 Do Supass(GetCNR(Ncr).P4);
Provi:=GetCNR(Ncr).P3;Cel:=GetCNR(Provi);
While Cel.P2<>Ncr Do    Begin
                        Provi:=Cel.P2;Cel:=GetCNR(Provi)
                        End;
CorNR(Provi,GetCNR(Ncr).P2,2);
InsVNR(Ncr)
End;
```

```
        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime la cellule Objet Nco       *)
        (* ainsi que ses listes de donnée, de relation, de           *)
        (* dépendance (si elles existent).                           *)

Procedure Supobj(Nco:Integer);
Begin
If InvNR(Nco) Then Return;
If GetCNR(Nco).Nat<>Obj Then Return;
Supdat(Nco);
While GetCNR(Nco).P2<>0 Do Suprel(GetCNR(Nco).P2);
While GetCNR(Nco).P3<>0 Do Supass(GetCNR(Nco).P3);
InsVNR(Nco)
End;

        (*************************************************************)

End.

        (*************************************************************)
```

Récapitulatif des procédures à déclarer en "external" pour l'utilisation de NR :

```
Function CrNR(Nom:Chaine8):Boolean;
Function InitfNR(Nom:Chaine8):Boolean;
Procedure FinNR;
Function NewObj(C:Chaine3;Mot:Integer):Integer;
Function NewRel(NcObj:Integer;C:Chaine3;Mot:Integer):Integer;
Function NewAss(NcObj,NcRel:Integer;C:Chaine3):Integer;
Function SuccAss(NcObj,NcAss:Integer;C:Chaine3):Integer;
Function NewDat(NcObj:Integer;C:Chaine3;Itab:TabDon):Integer;
Function Lirdat(Ncd:Integer):TabDon;
Function Lircod(Nc:Integer):Integer;
Function Lirmot(Nc:Integer):Integer;
Procedure Modcod(Nc:Integer;C:Chaine3);
Procedure Modmot(Nc,Info:Integer);
Procedure Moddat(Nc:Integer;Ndata:TabDon);
Function Nxtdep(Ncoa:Integer;C:Chaine3):Integer;
Function Nxtass(Ncra:Integer;C:Chaine3):Integer;
Function Nxtrel(Ncro:Integer;C:Chaine3):Integer;
Function Nxtdat(Ncod:Integer;C:Chaine3):Integer;
Function LhdObj(Ncra:Integer):Integer;
Function LhdRel(Nca:Integer):Integer;
Procedure SupDon(NcObj,Ncd:Integer);
Procedure SupDat(Nco:Integer);
Procedure SupAss(Nca:Integer);
Procedure SupRel(Ncr:Integer);
Procedure SupObj(Nco:Integer);
```

```
(*************************************************************)
(* Ce module comporte des procédures ou fonctions            *)
(* "publiques" (qui sont celles du cahier des charges        *)
(* de NH) et des procédures ou fonctions internes            *)
(* nécessaires à la mise en oeuvre des premières.            *)
(* Ces procédures ou fonction "publiques" sont formées       *)
(* de 2 parties :                                            *)
(*       _Un dépistage d'erreurs dans la communication       *)
(*       de valeurs par des programmes extérieurs et qui     *)
(*       n'a donc plus d'utilité quand ces programmes        *)
(*       sont "debuggés" (exepté pour les manipulations      *)
(*       de fichier).                                        *)
(*       _Le corps de la procédure.                          *)
(*************************************************************)

Module NH;

Const LimPNH = 50; (* Nombre de cellules dans un tableau *)
      NbPNH = 4;   (* Nombre de tableaux en mémoire centrale *)
      Cent = 100;  (* Sans commentaire *)
      Dix = 10;
      Limem=1600;  (* Test pour Memavl<LImite inférieur MEMoire *)

Type Chaine3 = Lstring(3);
     PagNH = 0..LimPNH;  (* Index des cellules *)
     NbrePNH = 1..NbPNH; (* Index des tableaux (ou pages) *)
     TabDon = Array(.1..3.) of Integer; (* Tableau de données *)
     Dons =      Record  (* Constitution d'un article Donnée *)
                 Case Ca : Char of (* Pour différencier les champs *)
                 'C' : (Mot:Lstring(5)); (* Chaine de 5 caractères *)
                 'D' : (Tdon:Tabdon)     (* Le tableau *)
                 End;
     Pointdon = Ads of Donnee;           (* Pointeur de Donnée *)
     Donnee =   Record (* Donnée en mémoire ou sur fichier *)
                Don : Dons;              (* la Donnée *)
                Case Boolean of
                True:(Pdon:Pointdon);    (* en mémoire centrale *)
                False:(Nc:Integer)       (* sur fichier *)
                End;
     CelNH =    Record                   (* Cellule NH sur fichier *)
                Psui,Pinf,Ppre : Integer  (* ses pointeurs *)
                End;
     Element =  Record (* Cellule NH en mémoire centrale *)
                Cel:CelNH;       (* la cellule sur fichier *)
                Pdon:Pointdon (* le pointeur sur les données *)
                End;
     TabNH = Array(.PagNH.) of Element;
                      (* Une page en mémoire centrale *)
     FichierNH = File of CelNH; (* Fichier des cellules *)
     Chaine12 = Lstring(12);
```

```
      Pointvid = Ads of Vcel;     (* Pointeur de cellule vide *)
      Vcel =     Record                    (* Une cellule vide *)
                 W : Integer;              (* son numéro *)
                 P : Pointvid              (* pointeur sur la suivante *)
                 End;
      Donfich = File of Donnee; (* Fichier des données d'une page *)

Var (.Public.)  Ic:Boolean;   (* Prédécesseur tête de liste ? *)

Var PageNH:Array(.NbrePNH.) of TabNH;
                 (* Tableau des pages en mémoire *)
    DebNH:Array(.NbrePNH.) of Integer;
                 (* Début de chaque page *)
    EcrNH:Array(.NbrePNH.) of Boolean;
                 (* Page modifiée en mémoire ? *)
    EtaNH:Array(.NbrePNH.) of Integer;
                 (* Ancienneté de consultation *)
    Ecridon:Array(.NbrePNH.) of Boolean;
                 (* Données modifiées ? *)
    Presdon:Array(.NbrePNH.) of Boolean;
                 (* Données en mémoire centrale ? *)

    Fdon:Donfich;                  (* Fichier de Données *)
    FichNH:FichierNH;              (* Fichier de Cellules *)
    NumNH:Integer;                 (* Nombre total de cellules *)
    RacVidNH:Pointvid;  (* Racine pointeurs cellules vides *)
    FVidNH:File of Integer;        (* Fichier de cellules vides *)
    Nomf:Chaine12;                 (* Nom de fichier *)

        (*************************************************************)
        (* Procédures MS_Pascal.                                     *)

Function Memavl:Word;Extern; (* Mémoire disponible *)
Function Getmqq(Wants:Word):Adsmem;Extern;
    (* Demande d'allocation d'un bloc supplémentaire *)
Procedure Dismqq(Block:Adsmem);Extern; (* Abandon d'un bloc *)

        (*************************************************************)
        (* Procédure interne. Alloue une nouvelle adresse pour       *)
        (* une cellule vide. Active Getmqq.                          *)

Procedure NewV(Vars P:Pointvid);
Allocation d'une adresse pour un pointeur de cellule vide dans les
256k de la machine.


        (*************************************************************)
        (* Procédure interne. Dispose une adresse de cellule         *)
        (* vide. Active Dismqq.                                      *)

Procedure DispV(Vars P:Pointvid);
```

```
        (*************************************************************)
        (* Procédure "PUBLIQUE". Alloue une nouvelle adresse         *)
        (* pour une cellule de données. Active Getmqq.               *)

Procedure NewDNH(Vars P:Pointdon);
Var adres:Adsmem;
Begin
Adres:=Getmqq(Sizeof(P^));P:=Adres
End;

        (*************************************************************)
        (* Procédure interne. Dispose une adresse de cellule         *)
        (* de données. Active Dismqq.                                *)

Procedure DispDNH(Vars P:Pointdon);


        (*************************************************************)
        (* Fonction interne. Retourne le numéro du fichier de        *)
        (* données d'une page d'index Nt.                            *)

Function Numfich(Nt:Integer):Integer;


        (*************************************************************)
        (* Fonction interne. Sans commentaire.                       *)

Function Inf(W1,W2:Integer):Integer;


        (*************************************************************)
        (* Fonction interne. Retourne le nom du fichier de           *)
        (* données d'une page d'index Nt (Active Numfich).           *)

Function Nomfdon(Nt:Integer):Chaine12;


        (*************************************************************)
        (* Procédure interne. Dispose les pointeurs de données.      *)

Procedure Relache(Vars Po:Pointdon);


        (*************************************************************)
        (* Procédure interne. Ecrit un message d'erreur d'après      *)
        (* la valeur de Ioer (<>0).                                  *)

Procedure Erreur(Ioer:Word);
Ecriture d'un message à partir du code d'une erreur survenue lors
d'une lecture ou d'une écriture sur disque.
```

```
        (*************************************************************)
        (* Procédure interne. Ecrit les données d'une page sur       *)
        (* disque. Dispose les pointeurs utilisés (Active            *)
        (* Relache). Active Nomfdon. Met à jour les tableaux         *)
        (* d'écriture et de présence des données.                    *)

Procedure Putdon(Nt:Integer);


        (*************************************************************)
        (* Procédure interne. Ecrit la page sur le fichier des       *)
        (* cellules si elle a été modifiée en mémoire centrale.      *)
        (* Met à jour le tableau d'écriture des pages. Active        *)
        (* Putdon si les données ont été modifiées, ou active        *)
        (* directement Relache dans le cas contraire. Met à          *)
        (* jour les tableaux de présence et d'écriture Données.      *)

Procedure PutPNH(Nt:Integer);


        (*************************************************************)
        (* Fonction interne. "Balance" une nouvelle page du          *)
        (* fichier en mémoire centrale à la place de la page la      *)
        (* plus anciennement consultée (qu'elle écrit sur            *)
        (* fichier par un appel à PutPNH). Communique en retour      *)
        (* l'index de cette nouvelle page. Met à Nil les             *)
        (* pointeurs de données.                                     *)

Function GetPNH(Nc:Integer):Integer;


        (*************************************************************)
        (* Fonction interne. Donne l'indice de la cellule de         *)
        (* numéro Nc dans le tableau d'index Nt.                     *)

Function IndNH(Nc,Nt:Integer):Integer;


        (*************************************************************)
        (* Procédure interne. Assigne à chaque cellule de la         *)
        (* page Nt ses données. Active Nomfdon. Met à jour le        *)
        (* tableau de présence de Données.                           *)

Procedure Getdon(Nt:Integer);


        (*************************************************************)
        (* Procédure interne. Initialise les tableaux. Elle est      *)
        (* appelée à la création ou à l'ouverture d'un fichier       *)

Procedure InitNH;
```

```
        (*************************************************************)
        (* Fonction interne. Donne l'index du tableau où se          *)
        (* trouve la cellule de numéro Nc. Si ce tableau n'est       *)
        (* pas en mémoire centrale, il est appelé par                *)
        (* l'activation de GetPNH.                                   *)

Function PresNH(Nc:Integer):Integer;

        (*************************************************************)
        (* Procédure interne. Met à jour le tableau EtaNH            *)
        (* indexant l'ancienneté de consultation des pages.          *)

Procedure DerNH(Nt:Integer);

        (*************************************************************)
        (* Fonction interne. Donne la cellule de numéro Nc et        *)
        (* met à jour le tableau d'ancienneté de consultation.       *)

Function GetCNH(Nc:Integer):Element;

        (*************************************************************)
        (* Procédure interne. Affecte les pointeurs (tableau)        *)
        (* W1, W2, W3 et le pointeur (Pascal) P (données) à la       *)
        (* cellule de numéro Nc en fonction de sa position en        *)
        (* mémoire centrale. Si la cellule n'est pas en mémoire      *)
        (* centrale, elle est appelée par l'activation de            *)
        (* PresNH. Met à jour les tableaux d'écriture et de          *)
        (* consultation.                                             *)

Procedure AffNH(Nc,W1,W2,W3:Integer;P:Pointdon);

        (*************************************************************)
        (* Fonction interne. Alloue un numéro pour une nouvelle      *)
        (*  cellule soit :                                           *)
        (*         _En le faisant pointer sur une cellule vide       *)
        (*          si RacVidNH<>Nil.                                *)
        (*         _En Incrémentant NumNH.                           *)
        (* Remplit cette cellule avec les valeurs W1, W2, W3, P      *)
        (* par l'activation de AffNH. Donne en retour le             *)
        (* numéro de la cellule créée.                               *)

Function NouvNH(W1,W2,W3:Integer;P:Pointdon):Integer;
```

```
        (*************************************************************)
        (* Procédure interne. Insère le numéro de cellule Nc         *)
        (* en tête de pile des cellules. Cette Procédure est         *)
        (* appelée à chaque suppression de cellule et permet         *)
        (* de récuperer le "trou" de numéro Nc pour une              *)
        (* prochaine création de cellule.                            *)

Procedure InsVNH(Nc:Integer);


        (*************************************************************)
        (* Fonction "PUBLIQUE". Création d'un fichier direct         *)
        (* de cellules NH, de nom Nomfic (communiqué par des         *)
        (* programmes extérieurs). Vérifie que ce fichier            *)
        (* n'existe pas. Réserve la première cellule (non            *)
        (* valide) pour mémoriser le nombre total de cellules.       *)
        (* Initialisations diverses. Si le fichier existe déjà       *)
        (* la fonction avorte et renvoie False.                      *)

Function Crlst(Nomfic:Chaine8):Boolean;
Begin
Nomf:=Nomfic;
Assign(FichNH,Nomfic);
FichNH.Mode:=Direct;FichNH.Trap:=True;FichNH.Errs:=0;
Reset(FichNH);
If FichNH.Errs=0 Then
        Begin
        Crlst:=False;Close(FichNH)
        End
Else    Begin
        Crlst:=True;
        FichNH.Errs:=0;
        Rewrite(FichNH);
        If FichNH.Errs<>0 Then
                Begin
                Erreur(FichNH.Errs);FichNH.Errs:=0
                End
        Else    Begin
                NumNH:=1;FichNH^.Psui:=NumNH;Put(FichNH);
                If FichNH.Errs<>0 Then
                        Begin
                        Erreur(FichNH.Errs);FichNH.Errs:=0
                        End;
                Concat(Nomf,'.vid');
                Assign(FVidNH,Nomf);Nomf:=Nomfic;
                RacVidNH.S:=0;InitNH
                End
        End;
FichNH.Trap:=False
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Ouvre (s'il existe) un fichier       *)
        (* direct de cellules NH de nom Nomfic (communiqué par       *)
        (* des programmes extérieurs). Ouvre (s'il existe) le        *)
        (* fichier de cellules vides et l'insère dans la pile        *)
        (* des cellules vides. Si le fichier n'existe pas, la        *)
        (* fonction avorte et renvoie False.                         *)

Function Initl(Nomfic:Chaine8):Boolean;
Begin
Nomf:=Nomfic;Assign(FichNH,Nomfic);
FichNH.Mode:=Direct;FichNH.Trap:=True;FichNH.Errs:=0;
Reset(FichNH);
If FichNH.Errs<>0 Then
        Begin
        Initl:=False;FichNH.Errs:=0
        End
Else    Begin
        Initl:=True;
        NumNH:=FichNH^.Psui;
        RacVidNH.S:=0;
        Concat(Nomf,'.vid');Assign(FVidNH,Nomf);
        FVidNH.Trap:=True;FVidNH.Errs:=0;
        Reset(FVidNH);
        If FVidNH.Errs=0 Then
                Begin
                While Not Eof(FVidNH) Do
                                Begin
                                InsVNH(FVidNH^);
                                Get(FVidNH)
                                End;
                Close(FVidNH)
                End;
        FVidNH.Errs:=0;FVidNH.Trap:=False;
        Nomf:=Nomfic;InitNH
        End;
FichNH.Trap:=False
End;

        (*************************************************************)
        (* Procédure interne. "Déverse" la pile de numéros de        *)
        (* cellules vides dans le fichier de cellules vides.         *)

Procedure InsFcNH(Vars Po:Pointvid);
```

```
        (*************************************************************)
        (* Procédure "PUBLIQUE". Ferme un fichier de cellules        *)
        (* NH ainsi que son fichier de cellules vides (s'il y        *)
        (* en a). S'il n'y a plus de celules vides, ce fichier       *)
        (* (de cellule vides) est effacé (s'il existait).            *)

Procedure FinIst;
Var I:Integer;

Begin
For I:=1 To NbPNH Do PutPNH(I);
FichNH.trap:=True;FichNH.Errs:=0;
Seek(FichNH,1);
If FichNH.Errs<>0 Then
        Begin
        Erreur(FichNH.Errs);FichNH.Errs:=0
        End
Else    Begin
        FichNH^.Psui:=NumNH;Put(FichNH);
        Close(FichNH);NumNH:=0;
        If FichNH.Errs<>0 Then
                Begin
                Erreur(FichNH.Errs);FichNH.Errs:=0
                End;
        FVidNH.Trap:=True;FVidNH.Errs:=0;
        If RacVidNH.S<>0 Then
                        Begin
                        Rewrite(FVidNH);
                        If FVidNH.Errs<>0 Then
                                Begin
                                Erreur(FVidNH.Errs);FVidNH.Errs:=0
                                End
                        Else    Begin
                                InsFcNH(RacVidNH);
                                Close(FVidNH);
                                If FVidNH.Errs<>0 Then
                                        Begin
                                        Erreur(FVidNH.Errs);
                                        FVidNH.Errs:=0
                                        End
                                End
                        End
                Else    Begin
                        Reset(FVidNH);
                        If FVidNH.errs=0 Then Discard(FVidNH);
                        FVidNH.errs:=0
                        End;
        FVidNH.Trap:=false
        End;
FichNH.Trap:=False
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Création de la premiere cellule      *)
        (* de la liste principale. Active NouvNH. Donne en           *)
        (* retour le numéro de la cellule créée. P est               *)
        (* communiqué par programme extérieur.                       *)

Function Premc(P:Pointdon):Integer;
Begin
Premc:=NouvNH(0,0,0,P)
End;

        (*************************************************************)
        (* Fonction interne. Corrige le champ (pointeur)             *)
        (* Champs de la cellule Nc avec la valeur W. Si la           *)
        (* cellule n'est pas en mémoire centrale, elle est           *)
        (* appelée par l'activation de PresNH. Met à jour les        *)
        (* tableaux d'écriture et de consultation. Donne en          *)
        (* retour la valeur du pointeur modifié.                     *)

Function CorNH(Nc,W,Champs:Integer):Integer;

        (*************************************************************)
        (* Fonction interne. Dépistage d'erreur pour les             *)
        (* programmes exploitant NH. Retourne la valeur VRAI         *)
        (* si la cellule Nc n'existe pas ou fait partie des          *)
        (* cellules vides.                                           *)

Function InvNH(Nc:Integer):Boolean;


        (*************************************************************)
        (* Fonction "PUBLIQUE". Insère une cellule dans la           *)
        (* liste dont Nct est la tête, juste après Nct. Active       *)
        (* NouvNH. Met à jour le pointeur Inf de Nct et le           *)
        (* pointeur Pre de la cellule dont elle prend la place,      *)
        (* si elle existe (active CorNH). Retoune le numéro          *)
        (* de la cellule créée. Nct et P sont communiqués par        *)
        (* programme extérieur.                                      *)

Function Intet(Nct:Integer;P:Pointdon):Integer;
Var Pos:Integer;

Begin
If InvNH(Nct) Then Intet:=0
        Else    Begin
                Pos:=NouvNH(GetCNH(Nct).Cel.Pinf,0,Nct,P);
                Intet:=Pos;
                Pos:=CorNH(CorNH(Nct,Pos,2),Pos,3)
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Insère une cellule dans la           *)
        (* liste dont Nct est la tête, en fin de liste. Active       *)
        (* NouvNH. Met à jour le pointeur Sui de la cellule          *)
        (* précédente (active CorNH). Retourne le numéro de          *)
        (* la cellule créée. Nct et P sont communiqués par           *)
        (* programme extérieur.                                      *)

Function Newcq(Nct:Integer;P:Pointdon):Integer;
Var W,Pos:Integer;
    Cellule:Element;

Begin
If InvNH(Nct) Then Newcq:=0
        Else    Begin
                W:=GetCNH(Nct).Cel.Pinf;
                If W=0 Then     Begin
                                Pos:=NouvNH(0,0,Nct,P);
                                Newcq:=Pos;Pos:=CorNH(Nct,Pos,2)
                                End
                       Else     Begin
                                Cellule:=GetCNH(W);
                                While Cellule.Cel.Psui<>0 Do
                                                Begin
                                                W:=Cellule.Cel.Psui;
                                                Cellule:=GetCNH(W)
                                                End;
                                Pos:=NouvNH(0,0,W,P);
                                Newcq:=Pos;Pos:=CorNH(W,Pos,1)
                                End
                End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Insère une cellule dans la           *)
        (* liste de même niveau que la cellule Nc, à la suite        *)
        (* de celle-ci. Active NouvNH. Met à jour le pointeur        *)
        (* Sui de Nc et le pointeur Pre de la cellule qui            *)
        (* suivait Nc, si elle existe (active CorNH).                *)
        (* Donne en retour le numéro de la cellule créée. Nc et      *)
        (* P sont communiqués par programme extérieur.               *)

Function Incel(Nc:Integer;P:Pointdon):Integer;
Var Pos:Integer;

Begin
If InvNH(Nc) Then Incel:=0
        Else    Begin
                Pos:=NouvNH(GetCNH(Nc).Cel.Psui,0,Nc,P);
                Incel:=Pos;
                Pos:=CorNH(CorNH(Nc,Pos,1),Pos,3)
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Donne le numero de la première       *)
        (* cellule dans la liste dont la tête est la cellule de      *)
        (* numéro Nct. Active GetCNH. Nct est communiqué par         *)
        (* programme extérieur.                                      *)

Function Nxtet(Nct:Integer):Integer;
Begin
If InvNH(Nct) Then Nxtet:=0
        Else Nxtet:=GetCNH(Nct).Cel.Pinf
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Donne le numéro de la cellule        *)
        (* successeur de la cellule Nc dans la liste de même         *)
        (* niveau. Active GetCNH. Nc est communiqué par              *)
        (* programme extérieur.                                      *)

Function Nxcel(Nc:Integer):Integer;
Begin
If InvNH(Nc) Then Nxcel:=0
        Else Nxcel:=GetCNH(Nc).Cel.Psui
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Donne le numéro N de la cellule      *)
        (* qui précède la cellule Nc. Active GetCNH.                 *)
        (* Ic est Vrai (True) si N est tête de la liste de Nc.       *)
        (*        Faux (False) sinon.                                *)
        (* Nc est communiqué par programme extérieur.                *)

Function Prcel(Nc:Integer):Integer;
Var W:Integer;

Begin
If InvNH(Nc) Then       Begin
                        Prcel:=0;Ic:=False
                        End
        Else    Begin
                W:=GetCNH(Nc).Cel.Ppre;
                If GetCNH(W).Cel.Pinf=Nc Then Ic:=True
                                         Else Ic:=False;

                Prcel:=W
                End
End;
```

```
        (*************************************************************)
        (* Fonction "PUBLIQUE". Donne le numéro de la cellule    *)
        (* tête de la liste où se trouve la cellule Nc. Active   *)
        (* Prcel. Nc est communiqué par programme extérieur.     *)

Function Tetli(Nc:Integer):Integer;
Var W:Integer;
Begin
If InvNH(Nc) Then Tetli:=0
        Else    Begin
                W:=Prcel(Nc);
                If W<>0 Then While Not Ic Do W:=Prcel(W);
                Tetli:=W
                End
End;

        (*************************************************************)
        (* Fonction "PUBLIQUE". Donne le pointeur de données de  *)
        (* la cellule Nc. Active Getdon (s'il y a lieu) et       *)
        (* GetCNH. Nc est communiqué par programme extérieur.    *)

Function Lircl(Nc:Integer):Pointdon;
Begin
If InvNH(Nc) Then       Begin
                        Writeln('LECTURE D''UNE CELLULE NON VALIDE');
                        Lircl.S:=0
                        End
        Else    Begin
                If Not Presdon(.PresNH(Nc).) Then Getdon(PresNH(Nc));
                Lircl:=GetCNH(Nc).Pdon
                End
End;

        (*************************************************************)
        (* Procédure "PUBLIQUE". Substitue le pointeur P au      *)
        (* pointeur Pdon dans la cellule Nc. Met à jour les      *)
        (* tableaux d'écriture et consultation. Relache Pdon.    *)
        (* Nc et P sont communiqués par programme extérieur.     *)

Procedure Modif(Nc:Integer;P:Pointdon);
Var Nt:Integer;
Begin
If InvNH(Nc) Then Begin
                Writeln('MODIFICATION D''UNE CELLULE NON VALIDE');
                DispDNH(P) End
        Else    Begin
                Nt:=PresNH(Nc);
                If Not Presdon(.Nt.) Then Getdon(Nt);
                Relache(PageNH(.Nt.)(.IndNH(Nc,Nt).).Pdon);
                With PageNH(.Nt.)(.IndNH(Nc,Nt).) Do Pdon:=P;
                DerNH(Nt);Ecridon(.Nt.):=True
                End
End;
```

```
        (*************************************************************)
        (* Procédure interne. Corrige les cellules précédente    *)
        (* et suivante de Nc. Relache le pointeur Pdon de la     *)
        (* cellule Nc. Insère Nc dans les cellules vides. Met à  *)
        (* jour Ecridon.                                         *)

Procedure Libcl(Nc:Integer);


        (*************************************************************)
        (* Procédure "PUBLIQUE". Supprime la cellule Nc ainsi    *)
        (* que la liste dont elle est la tête et toutes les      *)
        (* listes de niveau inférieur. Cette Procédure est une   *)
        (* boucle de parcours des listes (la récursivité n'est   *)
        (* pas utilisée pour éviter les risques d'overflow de    *)
        (* la pile de recursivité). Active Libcl pour chaque     *)
        (* cellule. Nc est communiqué par programme extérieur.   *)

Procedure Supcl(Nc:Integer);
Var Nc1,Nc2:Integer;

Begin
If InvNH(Nc) Then Writeln('SUPPRESSION D''UNE CELLULE NON VALIDE')
        Else    Repeat
                        Nc1:=Nc;
                        Repeat
                                Nc2:=GetCNH(Nc1).Cel.Pinf;
                                If  Nc2<>0 Then Nc1:=Nc2
                        Until Nc2=0;
                        Libcl(Nc1)
                Until Nc1=Nc
End;

        (*************************************************************)

End.

        (*************************************************************)
```

Récapitulatif des procédures à déclarer en "External" pour l'utilisation de NH :

```
Procedure NewDNH(Vars P:PointDon);
Function Crlst(Nom:Chaine8):Boolean;
Function Initl(Nom:Chaine8):Boolean;
Procedure Finlst;
Function Premc(P:PointDon):Integer;
Function Intet(Nct:Integer;P:PointDon):Integer;
Function Newcq(Nct:Integer;P:PointDon):Integer;
Function Incel(Nc:Integer;P:PointDon):Integer;
Function Nxtet(Nct:Integer):Integer;
Function Nxcel(Nc:Integer):Integer;
Function Prcel(Nc:Integer):Integer;
Function Tetli(Nc:Integer):Integer;
Function Lircl(Nc:Integer):PointDon;
Procedure Modif(Nc:Integer;P:PointDon);
Procedure Supcl(Nc:Integer);
```

BIBLIOGRAPHIE

Index des renvois :

Page 26 (3) : Philippe Boudon, Sur l'espace architectural. Ed. Dunod, Paris 1971.
Et Architecture d'Aujourd'hui N° 220, Avril 1982 (cf "L'architecture n'est pas géométrie").

Page 24 (2) : JNL Durand, Précis des leçons d'Architecture donnée à l'école royale polytechnique. Edition Verlag Dr. Alfons Uhl, Nördlingen 1981.

Page 24 (1) : JP Goulette, Padao, note interne LI2A. JPG 83/1, Novembre 1983.

Ouvrages consultés :

Meyer B. et Baudoin C., Méthodes de programmation, Ed. Eyrolles, Paris 1980.

Microsoft Corporation, Microsoft Pascal for MS-DOS, 1983.

Projet, Une base de données relationnelles en conception assistée par ordinateur, pour le projet d'architecture. Groupe d'étude et de recherche en informatique appliquée, Septembre 1982. Centre d'Informatique et de Méthodologie en Architecture, 9 rue Barbanègre, 75019 PARIS.

N. Wirth, Algorithms + data structures = programs, Ed. Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1976.

Table des matières