DES OUTILS LOGICIELS POUR L'ANALYSE DE DONNEES

Marc CSERNEL (Inria, Université Paris IX Dauphine)
INRIA
projet CLOREC
domaine de Voluceau B P. 105
78153 Le Chesnay Cedex
tel:39 63 55 10

Résumé :
Cet article décrit quelques outils logiciels destinés aux programmes FORTRAN. Ils sont basés sur l'analyse syntaxique, et portables puisqu'écrits en FORTRAN. Nous présentons, entre autres, un indentateur automatique, un générateur d'arbre d'appels, un générateur d'en-têtes de sous-programmes, un outil de repérage de chaînes de caractères. Nous proposons enfin d'éventuelles extentions de ces outils

Mots-clés : outils logiciels, FORTRAN

Des outils logiciels pour l'analyse de données

Marc CSERNEL (Inria, Université Paris IX Dauphine)
INRIA
projet CLOREC
domaine de Voluceau Bp 105
78153 Le Chesnay Cedex
tel:39 63 55 10

Parmi les nombreux ateliers logiciels développés actuellement il en existe peu qui soit développés pour les logiciels scientifiques, ou plus précisément pour ceux écrits en FORTRAN, (citons néanmoins le projet TOOLPACK [1,2]); nous avons donc été amenés à développer un certain nombre d'outils destinés à faciliter la maintenance et la documentation de programmes d'analyse de données écrits en FORTRAN. Ils ont été définis en fonctions des besoins du logiciel de classification automatique SICLA [3] et du club MODULAD[4], mais leur caractère général permet de les utiliser pour d'autres logiciels scientifiques écrits en FORTRAN. Ils présentent la particularité d'avoir été écrits en FORTRAN 77 et d'être donc portables.

Un projet comme le projet MODULAD se propose d'intégrer un certain nombre de programmes tous écrits en FORTRAN mais développés et testés dans des sites et sur des matériels différents, pour peu qu'ils satisfassent à certaines normes de programmation. Pour permettre aux différents producteurs d'effectuer leurs développements de manière coordonnée il est utile de pouvoir fournir sur chaque site de développement un ensemble d'outils identiques conduisant à une certaine uniformité de la présentation. La portabilité des outils définis est alors un atout précieux

Un certain nombre de ces outils utilisent un générateur d'analyseur syntaxique descendant déterministe assez proche des analyseurs LL(1) [5] , ils sont d'une utilisation simple et peuvent être utilisés indépendamment les uns des autres sans nécessiter un environnement spécifique.

Les outils que nous allons présenter réalisent les fonctions suivantes:

- Indentateur automatique de programme
- Aide à la documentation

- Générateur d'arbre d'appel
- Repérage des chaînes de caractères

Nous allons examiner successivement les caractéristiques de chacun des outils, puis nous donnerons quelques caractéristiques des analyseurs syntaxiques et lexicaux que nous avons générés et dont le comportement peut influer l'utilisation des outils.

I. L'indentation automatique

L'indentation des programmes permet d'accroître grandement leur lisibilité, en mettant en lumière par la présentation du texte, la structure des programmes. Nous avons donc bâti le programme INDAUTO indentateur automatique de programmes FORTRAN utilisant un analyseur syntaxique LL(1).

Cet indentateur effectue un décalage du texte vers la droite après chaque DO ou chaque IF THEN rencontré, il effectue un décalage vers la gauche après chaque étiquette correspondant à un DO et après chaque END IF. Un exemple de programme avant et après indentation est donné en annexe 1. Ce traitement qui à l'air très simple se complique considérablement lorsqu'une instruction ne tient pas sur une ligne, il convient alors de choisir une césure sans nuire à la lisibilité; une méthode simple, comme de placer la césure sur la virgule ou l'opérateur arithmétique le plus proche de la fin de la ligne est insuffisante comme le montre l'exemple suivant. Soit l'instruction FORTRAN

```
      WRITE(IMP,1000) 'VALEUR ELEMENT 1', ITAB(IMAX , IDEF + I),
     &    'VALEUR ELEMENT 2', TAB( I + J )
```

Si du fait de décalages réalisés par l'indentateur l'instruction doit être coupée au plus loin au milieu de l'élément de tableau ITAB, une césure placée sur la virgule précédant immédiatement IDEF n'entraînerait pas de faute de syntaxe, mais un résultat peu lisible:

```
                      WRITE(IMP,1000) 'VALEUR ELEMENT 1', ITAB(IMAX ,
     &                IDEF + I), 'VALEUR ELEMENT 2 ', TAB( I + J )
```

puisque la césure se trouve au milieu de la description d'un élément de tableau ; ce qui pour un outil chargé d'améliorer la lisibilité est vraiment peu souhaitable.

Il convient donc pour placer convenablement la césure de faire une analyse syntaxique du texte, qui mette en lumière sa structure et qui permette son découpage de manière plus utilisable.

Dans l'exemple précédent l'analyseur reconnaîtra la séquence ITAB(IMAX ,IDEF + I) comme étant un élément de tableau et le programme d'établissement de césure essayera, dans la mesure de la place disponible, d'éviter de la découper. Avec l'exemple précédent un découpage

adéquat donnerais alors:

```
                WRITE(IMP,1000) 'VALEUR ELEMENT 1',
&               ITAB(IMAX , IDEF + I), 'VALEUR ELEMENT 2 ', TAB( I + J )
```

Ce découpage est réalisé en suivant la structure des arbres syntaxique générés par l'analyseur syntaxique, cet arbre est projeté sur le texte même de l'instruction, et seul, le placement de la césure et les décalages en début de ligne modifient le texte de l'instruction.

La manière dont s'effectue le découpage dépend donc :

- de la structure de la grammaire utilisée
- des paramètres d'indentation

La grammaire utilisée par INDAUTO peut être facilement modifiée, mais à priori ce genre de modification relève peu d'un utilisateur normal

Les paramètres modifiables sont :

- La valeur du décalage effectué après chaque DO ,IF ....
- La valeur de retrait effectué après chaque ELSE ou ELSE IF
- La longueur maximum d'un élément susceptible d'être découpé lorsqu'il est en fin de ligne.

Les valeurs des différents paramètres sont contenus dans les insertions sémantiques de la grammaire associée à INDAUTO (indauto.reg), la modification d'un de ces paramètres entraîne (pour être prise en compte) une recompilation de la grammaire.

INDAUTO permet aussi d'obtenir un ordonnancement des numéros d'étiquettes, avec deux séquences séparées :

- la première pour celles concernant le contrôle du programme
- la seconde pour celles concernant les formats d'entrée-sortie.

Les nouvelles étiquettes croissent de 10 en 10 dans chaque séquence, la première séquence commence à 10, la deuxième à 1000, les étiquettes produites sont cadrées à droite dans la zone étiquette.

Il faut remarquer que l'optique de découpage que nous avons adoptée pour INDAUTO est différente de celle adoptée par certains éditeurs syntaxiques comme MENTOR[11], ou des paragrapheurs PARADIS[12], c'est alors le texte complet des instructions qui est mis sous forme d'arbre et les blancs n'apparaissent plus. La mise en forme du texte est alors entièrement dépendante de règles définies par ailleurs, en particulier en ce qui concerne le placement des blancs, le texte est alors complètement normalisé, interdisant les éventuelles mises en page du programmeur.

Nous envisageons néanmoins d'introduire cette possibilité, afin de faciliter les recherches de chaînes sous éditeurs. En effet lorsque l'on désire rechercher sous un éditeur normal une chaîne comportant des blancs, comme par exemple:

SUBROUTINE CARAC

l'on est dépendant quant au succès de la recherche du nombre de blancs compris entre 'SUBROUTINE' et 'CARAC'. Si l'on a prévu un seul blanc dans la recherche alors que le texte source en contient plusieurs la recherche échoue. La solution habituelle consiste alors à chercher la chaîne 'CARAC', mais elle apparaîtra un nombre décourageant de fois tant dans le mot 'CARACTERE' qui peut apparaître dans les commentaires que dans les CALL éventuels. Une normalisation du nombre de blancs entre 'SUBROUTINE' et le nom du sous-programme effectuée par l'indentateur permettra d'effectuer les recherche à coup sur pour peu que le programmeur se souvienne de la normalisation adoptée.

II. La génération d'arbres d'appels

Les arbres d'appels sont particulièrement utiles lorsque l'on désire voir comment s'enchaînent les appels des sous-programmes effectué par un programme donné, que l'on cherche à optimiser , ou bien lorsque l'on désire voir combien de programmes sont touchés par la modification de l'appel d'un sous-programme donné.

La génération des arbres d'appel se fait à l'heure actuelle en deux temps:

- Dans un premier temps le texte de chaque module est examiné par le programme GENAPP qui utilise l'analyseur syntaxique, et qui à chaque rencontre d'un COMMON, d'un INCLUDE, d'un CALL ou d'un appel de fonction génère un enregistrement dans un fichier des appels. Il faut remarquer qu'une simple analyse lexicale est insuffisante pour permettre le constitution de tels fichiers; en effet si un "CALL" ou un "COMMON" peuvent être reconnus directement, les appels de fonctions nécessitent une analyse syntaxique complète. On se sert aussi de la présence de l'analyseur syntaxique pour décrire aussi les paramètres de chacun des composants. GENAPP permet suivant la volonté de l'utilisateur de créer un fichier d'appel pour chaque module, ou pour plusieurs modules.

- Dans un deuxième temps on passe à la constitution des arbres à partir des fichiers d'appels précédemment constitué en utilisant le programme ARBRE. Cette construction s'applique sur un nombre quelconque de modules. Seuls les renseignements apparaissant dans les fichiers d'appels sont utilisés.

Pour chaque programme, sous-programme ou fonction, ARBRE permet de faire apparaître:

- les sous-programmes ou fonctions utilisées
- les commons utilisés
- les fichiers include utilisés
- la liste des paramètre utilisés avec leur description (voir ci-après)

ARBRE permet aussi de faire apparaître pour les programmes principaux seulement (ou pour tous les programmes si on le désire) l'arbre complet des appels. On trouvera en annexe 2 une partie (le début et la fin) de l'arbre d'appel du programme INDAUTO.

On doit remarquer que l'arbre des appels ne sera effectivement complet que si les modules concernés ont été traités à l'aide du programme GENAPP.

Pour chaque arbre réalisé, une liste plus condensée des noms des programmes utilisés, avec associe à chaque programme son module d'origine, est fournie. Les programmes extérieurs aux modules traités y figurent avec, comme non de module, la mention "inconnu"

Une liste des programmes appelant de chacun des programmes traités peut-être obtenue.

Les outils permettant la génération des arbres d'appels servent aussi dans le cadre de l'aide à la documentation, car ils permettent aussi de produire un bref descriptif des programmes rencontré

III. L'aide à la documentation

Elle comporte deux parties différentes:

- La première partie permet au moyen des programmes GENAPP et ARBRE de générer pour chaque sous-programme ou fonction, un tableau de commentaires comprenant pour chaque paramètre formel les informations suivantes:
  - le nom du paramètre
  - son type (entier, réel, ...)
  - s'il s'agit du type character la longueur
  - s'il s'agit d'un tableau les dimensions, sinon la mention scalaire
  - le mode d'utilisation du paramètre: entrée, sortie.

  ainsi que les noms des includes et common et sous-programmes utilisés.Un exemple se trouve en annexe 3

  Ces informations sont générées par ARBRE, elles pourront être reprises par le programmeur et complétées par des commentaires complémentaires, puis insérées dans le texte du programme initial et servir d'en-tête de programme ou de sous-programme. Les renseignements générés peuvent être naturellement facilement créés manuellement, mais lorsqu'un nombre conséquent de programmes doit être traité cet outil apporte une aide non

négligeable, il permet aussi une description systématique qui met le programmeur à l'abri des oublis.

- La deuxième partie utilise le programme COMMENT et permet de créer à partir des en-têtes de chaque programme, sous-programme, fonction (conformes si possible à la description précédente) un listing de ces en-tête comprenant une table des matières triée. Ce listing permet d'obtenir facilement une description succincte mais complète d'un ensemble de programmes. Deux tables de matières, classées par ordre alphabétique ou par ordre d'apparition, permettent de retrouver les différents programmes. Nous envisageons l'introduction de mots clés permettant d'obtenir l'ensemble des sous-programmes concernant un thème donné comme par exemple la gestion des fichiers.

IV. Recherche de chaînes de caractères

La recherche des chaînes de caractères dans un logiciel présente un double intérêt :

- elle permet de vérifier la forme des messages affichés
- elle permet de repérer tous les textes affichées en vue de leur traduction ultérieure.

Le programme CHLANG a été développé, au départ pour faciliter la traduction des messages d'un logiciel conversationnel dans une langue étrangère, mais il sert aussi depuis à assurer par un repérage systématique la cohérence des messages trouvés dans un programme. Pour chaque module fourni en entrée, il affiche :

- chaque instruction contenant des chaînes de caractères, avec son numéro de ligne et le module d'ou elle provient
- de manière séparée toutes les chaînes de caractères contenues dans cette instruction

Le programme sous sa forme actuelle est très rustique puisqu'il se contente d'afficher les chaînes de caractères sans permettre de les modifier. En effet il n'a pas été jugé nécessaire de faire fonctionner CHLANG de manière interactive, c'est à dire de permettre à son utilisateur de modifier de manière conversationnelle les chaînes de caractères rencontrés. En effet la plupart des modifications risquent de provoquer des effets de bord, soit décalage de la présentation de messages, soit affectations des chaînes trop longues pour la variable caractère la contenant. Pour éviter ces effets intempestifs il est nécessaire de faire sur le module des modifications globales, qui ne touchent pas simplement la forme des chaînes, et rendent donc beaucoup trop lourde la possibilité de modification interactive.

V. L'analyseur

Certains des outils que nous avons présentés (INDAUTO,GENAPP) fonctionnent en utilisant un analyseur syntaxique, ces analyseur sont générés de manière adaptée à chaque programme par un générateur d'analyseur syntaxique FGRAMM développé par nos soins. Notre générateur d'analyseur génère à l'heure actuelle un analyseur déterministe descendant, il admet en entrée des grammaires proches des grammaires LL(1). Cet analyseur est composé classiquement d'un analyseur lexical et de l'analyseur syntaxique proprement dit.

Comme tous les produits de ce genre il fonctionne en deux temps:

- dans le premier temps on fournit au générateur une grammaire sous une forme appropriée (proche de la BNF); le générateur après avoir vérifié la validité de la grammaire présentée génère un analyseur capable de reconnaître un texte écrit dans un langage conforme à la grammaire entrée. Cet analyseur est écrit en FORTRAN, il faut donc le compiler et effectuer une édition de liens avant de pouvoir l'utiliser.

- dans un deuxième temps on utilise l'analyseur créé par le générateur et on lui fournit en entrée le texte à analyser. L'analyseur détecte les éventuelles erreurs de syntaxe et déclenche des actions (appels de sous-programmes) lors de la reconnaissance de certaines structures.

Dans tous les cas de figure les analyseurs que nous générons ont une différence fondamentales par rapport aux compilateurs FORTRAN classiques, ils reconnaissent le "blanc" comme un séparateur. Ceci entraîne que pour nos programme:

1000 FOR MAT (.....) ne sera pas reconnu comme l'instruction FORMAT et que GOTO100 sera reconnu comme un identificateur et non comme une instruction GO TO. Cette contrainte nous parait minime dans des programmes convenablement écrits par rapport aux complications qu'entraîne la non reconnaissance du blanc comme séparateur.

Il peut arriver que les sources des programmes fournis pour utiliser INDAUTO ou GENAPP ne correspondant pas à la grammaire de ces programmes, auquel cas un message d'erreur est envoyé par l'analyseur.

Ces messages d'erreurs peuvent être de deux types:

- ceux générés par l'analyseur lexical (première phase de l'analyse) , ces messages indiquent en général l'apparition d'un caractère non reconnu.
- ceux généré par l'analyseur syntaxique proprement dit. Ils sont presque toujours de la forme : xxx **trouvé**, yyy ou zzz **attendu** ; par exemple l'instruction :

DO I = 1,N provoquera le message d'erreur:

erreur ligne numéro xx

DO I = 1,N

Identificateur trouvé étiquette attendue

l analyse de l'instruction sera ensuite abandonnée et on passera aux instructions suivantes. Cet abandon peut être quelque fois générateur d'erreurs dans l'utilisation des logiciels Par exemple une erreur dans une instruction contenant un appel de fonction entraînera la non prise en compte par GENAPP de cet appel, de la même manière une erreur sur un DO entraînera dans INDAUTO une mauvaise indentation, néanmoins INDAUTO réalise quoiqu'il arrive l'écriture d'une instruction même non correctement analysée, son découpage en revanche, si elle tient sur plusieurs lignes, pourra être maladroit.

Il existe actuellement un grand nombre de logiciels permettant de générer de manière simple des analyseurs syntaxiques, citons en particulier le couple YACC[6] - LEX[7] de Unix, et le logiciel SYNTAX[8] qui a été développé à l'INRIA. Ces logiciels très performants présentent l'inconvénient d'être écrit dans un langage très différent du FORTRAN (PL1 ou C pour SYNTAX ou C pour YACC-LEX) et d'être liés à un système spécifique (YACC-LEX lié à UNIX). C'est pourquoi nous avons été amenés à développer un générateur d'analyseurs syntaxiques écrit en FORTRAN donc portable sur tout ordinateur susceptible d'effectuer des traitements scientifiques.

VI. Autres Utilisations

D'autres utilisations d'outils basés sur des analyseurs syntaxique ont été essayé, parmi toutes ces possibilités un vérificateur de normes , ou plus précisément de normes Modulad, a été tenté.

Les résultats n'ont pas été enthousiasmants:

- d'une part la réalisation d'un tel vérificateur n'est pas évidente, en particulier au niveau de la forme des instruction d'entrée sortie, des formats, ou du classement des arguments d'un appel de fonction ;
- d'autre part les quelques tests effectués (avec une vérification incomplète de la norme) sur des programmes n'ont pas entraîné de la part des auteurs des réactions extrêmement positives lors de la découverte par le vérificateur de partie de programme non conforme à la norme. Elles se sont résumées la plupart du temps à l'expression d'un 'ah oui' et ne furent suivi d'aucun autre effet.

Nous pourrions néanmoins essayer de poursuivre dans cette voie si un nombre suffisant d'auteurs était intéressé

Remarquons qu'il existe déjà des vérificateurs de normes pour le FORTRAN comme le fameux PFORT[10], néanmoins ils présentent l'inconvénient d'être non paramètrables et donc de ne pouvoir être configurés en fonction de besoins particuliers.

VII. Conclusion

Les logiciels que nous avons décrits ici ont tournés sous Multcs et tournent actuellement sous UNIX sur APOLLO, leur transport n'a pas posé de problème, mis à part les fichiers INCLUDE. Ils peuvent être obtenu auprès de l'auteur sous forme de cassette "TAR" SUN ou APOLLO ou bien sous forme de bande classique.

L'utilisation d'analyseur syntaxique écrit en FORTRAN permet de réaliser avec facilité un certain un certain nombre d'outils logiciels portable destinés à ce langage D'autre outils peuvent être envisagé comme par exemple un pré-processeur de macro tel que RATFOR[13]. Ces outils permettent au programmeur de logiciels statistiques de réaliser des programmes de meilleur qualité tout en gagnant tant au niveau du développement qu'au niveau de la maintenance, un temps substantiel

Il serait intéressant de compléter cette approche par une liaison: avec des systèmes de gestion de bases de données pour mieux gérer les bibliothèques de programmes, mais ce serait certainement au prix d'une perte de la portabilité.

Il existe pour la création d'outils logiciel d'autres approches que l'approche basée sur la syntaxe, en particulier celle faite à partir de langages orientés objet, elle nous semble moins prometteuse pour un langage tel que FORTRAN du fait de sa faible structuration.

REFERENCES

1] L. J. OSTERWEIL
Toolpack an Exeperimental Software Developement Environnment Research Project
IEE transaction on Software Engineering Vol SE-9, N° 6 1983

2] L. J. OSTERWEIL, Geoffrey CLEMM
An Extensible Toolset and Environment for the Production of Mathematical Sofware
Conférence Internationale: Outils,Méthodes et Langages adaptés au Calcul Scientifique Paris 1983

3] H. RALAMBONDRAINY
Le Système SICLA
Thèse d'état, Université Paris IX Dauphine 1986

4] MODULAD
Bochure Utilisateur V 2
INRIA 1987

5] A. V. AHO, J. D. ULLMANN
Principles of Compiler Design
Addisson-Wesley 1977

6] M.E. LESK
LEX a lexical analyzer generator
Bells Labs, Comp. Sc. Tech. Rpt 39 1975

7] S. C. JOHNSON
YACC: Yet another compiler compiler
Bell Labs, Comp. Sc. Tech. Rpt 32 1975

8] P. BOULIER
Contribution à la Construction Automatique d'Analyseurs Lexicographiques et Syntaxiques
Thèse d'état Université d'Orléans 1984

9] American National Standard Institute
Programming Language FORTRAN
ANSI X3.9 1978

10] B.G. RYDER, A.D. HALL
The PFORT Verifier
Bells Labs, Comp. Sc. Tech. Rpt 12 1975

11] B. MELESE, V. MIGOT, D. VEROVE
The Mentor-V5 documentation
INRIA Rapport Technique N° 43 1985

12] P. DESCHAMP, P. BOULLIER
PARADIS un système de paragraphage dirigé par la syntaxe
INRIA Rapport de Recherche N° 455 1985

13] B W KERNIGHAN
RATFOR, a preprocessor for a rational FORTRAN
Bells Labs, Comp. Sc. Tech. Rpt 55 1975

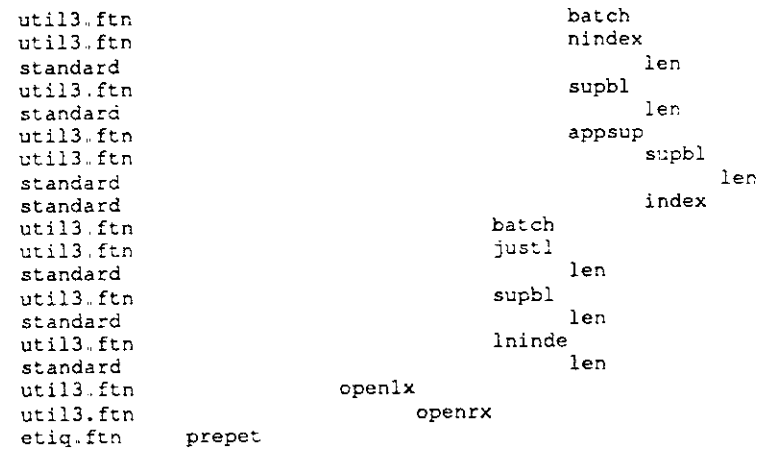
AVANT INDENTAIION

```
      subroutine affnap(nompro,iprog,iprog1,nomfic,idebpr,nbprog,nofic,
     &   maxfic, maxpro,nb)
      integer iprog(maxpro,5),iprog1(nb),nofic,maxpro,nb,nbprog,idepr
      character *(*) nomfic(maxfic),nompro(maxpro)
      character *10 nom(3),fic(3)
            integer k
      k = 0
      write(*,1000)
      if (nb.gt.5) then
      do 10 i = idebpr + 1,nbprog
      do 20 j = 1,nb
      if (iprog1(j).eq.i) then
      go to 30
            end if
   20    continue
         if (k.ge.3) then
         write(*,1010)(nom(j),fic(j),j=1,3)
         k = 0
         end if
         if (iprog1(j).eq.9) then
         if (nompro(i)(1:5).ne.'Main:') then
         k = k + 1
         nom(k) = nompro(i)
         nompro(i) = ' '
         fic(k) = nomfic(iprog(i,nofic))
         else if(iprog1(j).eq.5) then
           k = k + 3
         end if
         end if
   30    continue
   10    continue
         end if
         nb = 0
 1010    format(3(a8,' (',a11,') * '))
 1000    format(//'           LISTE DES SOUS-PROGRAMMES NON UTILISES')
         return
         end
```

APRES INDENTATION

```
      subroutine affnap(nompro,iprog,iprog1,nomfic,idebpr,nbprog,nofic,
     & maxfic, maxpro,nb)
      integer iprog(maxpro,5),iprog1(nb),nofic,maxpro,nb,nbprog,idepr
      character *(*) nomfic(maxfic),nompro(maxpro)
      character *10 nom(3),fic(3)
      integer k
      k = 0
      write(*,1010)
      if (nb.gt.5) then
          do 30 i = idebpr + 1,nbprog
              do 10 j = 1,nb
                  if (iprog1(j).eq.i) then
                      go to 20
                  end if
   10         continue
              if (k.ge.3) then
                  write(*,1000)(nom(j),fic(j),j=1,3)
                  k = 0
              end if
              if (iprog1(j).eq.9) then
                  if (nompro(i)(1:5).ne.'Main:') then
                      k = k + 1
                      nom(k) = nompro(i)
                      nompro(i) = ' '
                      fic(k) = nomfic(iprog(i,nofic))
                   else if(iprog1(j).eq.5) then
                      k = k + 3
                  end if
              end if
   20         continue
   30     continue
      end if
      nb = 0
 1000 format(3(a8,' (',a11,') * '))
 1010 format(//'           LISTE DES SOUS-PROGRAMMES NON UTILISES')
      return
      end
```

```
*****************************************************************************
programme Main:indauto      fichier   indauto.ftn

   appel les programmes :
openlx    lirgra    initot    finarb    opficf    prepet    analis
opficl

   utilise les common:
cetat     etat      sem

   utilise les includes:
etat.inc  sem.inc

                  ARBRE D'APPEL

 util3.ftn    openlx
 util3.ftn         openrx
 util2.ftn    lirgra
 util3.ftn    initot
 util3.ftn         apollo
 analigra.f   finarb
 usem.ftn          wind
 util2.ftn              wrchai
 util3.ftn                   lninde
 standard                         len
 util2.ftn              inssou
 util2.ftn              wbarb
 opficf.ftn   opficf
 util2.ftn         suffic
 util3.ftn              supbl
 standard                    len
 util3.ftn              carac
 util3.ftn                   r240
 util3.ftn                        lninde
 standard                              len
 util3.ftn                   batch
 util3.ftn                   nindex
 standard                         len
 util3.ftn                   supbl
 standard                         len
 util3.ftn                   appsup
 util3.ftn                        supbl
 standard                              len
 standard                         index
 util3.ftn              lninde
 standard                    len
 util2.ftn              opdel
 util3.ftn                   ouinon
 util3.ftn                        repons
 standard                              len
 util3.ftn                             carac
 util3.ftn                                   r240
 util3.ftn                                        lninde
 standard                                              len
 util3.ftn                                   batch
 util3.ftn                                   nindex
 standard                                         len
 util3.ftn                                   supbl
 standard                                         len
 util3.ftn                                   appsup
 util3.ftn                                        supbl
 standard                                              len
 standard                                         index
 util3.ftn                                   batch
 util3.ftn                                   justl
 standard                                         len
 util3.ftn                                   supbl
 standard                                         len
 util3.ftn                                   lninde
 standard                                         len
 util3.ftn                   openlx
 util3.ftn                        openrx
 util3.ftn              openlx
 util3.ftn                   openrx
 standard          index
 util3.ftn         lninde
 standard               len
 util2.ftn         opdel
 util3.ftn               ouinon
 util3.ftn                    repons
 standard                          len
 util3.ftn                         carac
 util3.ftn                               r240
 util3.ftn                                    lninde
 standard                                          len
```



Ensemble des programmes appelles

openlx	(util3.ftn)	*	openrx	(util3.ftn)	*	lirgra	(util2.ftn)	*			
initot	(util3.ftn)	*	apollo	(util3.ftn)	*	finarb	(analigra.f)	*			
wind	(usem.ftn)	*	wrchai	(util2.ftn)	*	lninde	(util3.ftn)	*			
len	(standard)	*	inssou	(util2.ftn)	*	wbarb	(util2.ftn)	*			
opficf	(opficf.ftn)	*	suffic	(util2.ftn)	*	supbl	(util3.ftn)	*			
carac	(util3.ftn)	*	r240	(util3.ftn)	*	batch	(util3.ftn)	*			
nindex	(util3.ftn)	*	appsup	(util3.ftn)	*	index	(standard)	*			
opdel	(util2.ftn)	*	ouinon	(util3.ftn)	*	repons	(util3.ftn)	*			
justl	(util3.ftn)	*	prepet	(etiq.ftn)	*	lirins	(analigra.f)	*			
lirfor	(analigra.f)	*	endinc	(util2.ftn)	*	numer	(inconnu)	*			
analex	(analex3.ft)	*	lex1	(analex3.ft)	*	itrans	(analex3.ft)	*			
ichar	(standard)	*	lex2	(usem.ftn)	*	nuvar	(usem.ftn)	*			
numpla	(usem.ftn)	*	mod	(standard)	*	numter	(analex3.ft)	*			
justr	(util3.ftn)	*	placet	(etiq.ftn)	*	analis	(analigra.f)	*			
delvar	(analex3.ft)	*	entier	(util3.ftn)	*	r80	(util3.ftn)	*			
chaent	(util3.ftn)	*	maxent	(util3.ftn)	*	tstent	(util3.ftn)	*			
analy	(analigra.f)	*	automa	(analigra.f)	*	finins	(analigra.f)	*			
okprem	(analigra.f)	*	crarbr	(analigra.f)	*	seman	(indauto.ft)	*			
tseti	(etiq.ftn)	*	warn	(util2.ftn)	*	decsp	(usem.ftn)	*			
insvar	(usem.ftn)	*	includ	(usem.ftn)	*	getsar	(util2.ftn)	*			
deccom	(usem.ftn)	*	vereti	(etiq.ftn)	*	appel	(usem.ftn)	*			
etido	(etiq.ftn)	*	eti	(etiq.ftn)	*	numida	(util2.ftn)	*			
narbel	(util2.ftn)	*	wtrace	(util2.ftn)	*	erreur	(util2.ftn)	*			
cpterm	(util2.ftn)	*	uptree	(analigra.f)	*	apres	(usem.ftn)	*			
setmod	(usem.ftn)	*	edicom	(analigra.f)	*	opficl	(opficf.ftn)	*			

LISIE DES SOUS-PROGRAMMES NON UTILISES

tstlg	(usem.ftn)	*	decpp	(usem.ftn)	*	affvar	(usem.ftn)	*			
afpar	(usem.ftn)	*	genlis	(usem.ftn)	*	alpha	(analc.ftn)	*			
specar	(analc.ftn)	*	carfor	(analc.ftn)	*	insfor	(analc.ftn)	*			
isent	(analc.ftn)	*	isiden	(analc.ftn)	*	ispar	(analc.ftn)	*			
isreg	(analc.ftn)	*	isnum	(analc.ftn)	*	inser	(util2.ftn)	*			
ichacro	(util2.ftn)	*	arbele	(util2.ftn)	*	kconv	(util2.ftn)	*			
numidf	(util2.ftn)	*	reelma	(util3.ftn)	*	tstrel	(util3.ftn)	*			
charel	(util3.ftn)	*	reel	(util3.ftn)	*	mreel	(util3.ftn)	*			
mentie	(util3.ftn)	*	inter	(util3.ftn)	*	mindex	(util3.ftn)	*			

annexe_3

IEXTE PRODUII PAR ENIEI POUR IE SP "affnap'

```
**************************************************************************
 programme affnap          fichier   arbr.ftn

     parametres utilises:
* nofic   integer         scal      e
* idebpr  default         scal      e
* iprog   integer         maxpro,5  e
* maxpro  integer         scal      e
* nb      integer         scal      e/s
* maxfic  default         scal      e
* nompro  charact (*)     maxpro    e/s
* nbprog  integer         scal      e
* iprog1  integer         nb        e
* nomfic  charact (*)     maxfic    e


  IE SOUS-PROGRAMME 'affnap' INDENIE AVEC INIEGRAIION DE L'EN IEIE

      subroutine affnap(nompro,iprog,iprog1,nomfic,idebpr,nbprog,nofic,
     & maxfic, maxpro,nb)
**************************************************************************
*    sous-programme d'affichage des sous-programme non utilise
*
* nofic   integer         scal      e    place de la colonne de num de
*                                        fichier
* idebpr  default         scal      e    numero du premier programme
* iprog   integer         maxpro,5  e    tableau des descriprions de
*                                        programmes
* maxpro  integer         scal      e    nbmax de programmes
* nb      integer         scal      e    nb de prog utilise
* maxfic  default         scal      e    nbmax de fichier
* nompro  charact  *(*)   maxpro    e/s  nom des programmes
* nbprog  integer         scal      e    nb programmes utiles
* iprog1  integer         nb        e/s  num des programmes utilises
* nomfic  charact  *(*)   maxfic    e    nom des fichier
*
**************************************************************************
      subroutine affnap(nompro,iprog,iprog1,nomfic,idebpr,nbprog,nofic,
     & maxfic, maxpro,nb)
      integer iprog(maxpro,5),iprog1(nb),nofic,maxpro,nb,nbprog,idepr
      character *(*) nomfic(maxfic),nompro(maxpro)
      character *10 nom(3),fic(3)
      integer k
      k = 0
      write(*,1010)
      if (nb.gt.5) then
          do 30 i = idebpr + 1,nbprog
              do 10 j = 1,nb
                  if (iprog1(j).eq.i) then
                      go to 20
                  end if
   10         continue
              if (k.ge.3) then
                  write(*,1000)(nom(j),fic(j),j=1,3)
                  k = 0
              end if
              if (iprog1(j).eq.9) then
                  if (nompro(i)(1:5).ne.'Main:') then
                      k = k + 1
                      nom(k) = nompro(i)
                      nompro(i) = ' '
                      fic(k) = nomfic(iprog(i,nofic))
                   else if(iprog1(j).eq.5) then
                      k = k + 3
                  end if
              end if
   20         continue
   30     continue
      end if
      nb = 0
 1000 format(3(a8,' (',a11,') * '))
 1010 format(///'           LISIE DES SOUS-PROGRAMMES NON UIILISES')
      return
      end
```