# Annexe C

# Syntaxe du pseudo code *LAF* et langage C

## Introduction

Le pseudo code LAF, variante du SCI de l'an 2000, dispose depuis 2001 d'un véritable compilateur. Cela a permis d'assouplir quelque peu l'écriture, notamment par la suppression de parenthèses pléthoriques. On a veillé à la possibilité de couper les mots, d'utiliser indifféremment les minuscules ou les majuscules, d'utiliser librement les accents.

Il faut noter que les espaces à l'intérieur des mots clés sont facultatifs et que les mots en italiques peuvent être omis :

`Fin` *du* `Traitement` ou `Fin Traitement` ou `FinTraitement` sont équivalents.

**Remarque importante :** pour économiser l'espace, j'ai supprimé les directives `#include` dans tous les programmes en C qui figurent dans la suite de cette annexe.

## C.1 Déclarations

### C.1.1 Variables globales

```
Variables Entières iVar1, iVar2
Variables Flottantes fVar3, fVar4
Début du Traitement
   iVar1 <- 1
   iVar2 <- 2
   fVar3 <- 3.0
   fVar4 <- 4.0
Fin du Traitement
```

```
int iVar1, iVar2;
double fVar3, fVar4;
int main()
{
  iVar1=1;
  iVar2=2;
  fVar3=3.0;
  fVar4=4.0;
  return 0;
}
```

### C.1.2 Variables locales

```
Début du Traitement
  Définitions Locales
    Variables Entières iVar1, iVar2
    Variables Flottantes fVar3, fVar4
  Fin des locales
  iVar1 <- 1
  iVar2 <- 2
```

```
int main()
{
  int iVar1, iVar2;
  double fVar3, fVar4;
  iVar1=1;
  iVar2=2;
  fVar3=3.0;
```

```
  fVar3 <- 3.0
  fVar4 <- 4.0
Fin du Traitement
```

```
  fVar4=4.0;
  return 0;
}
```

Les variables locales peuvent figurer dans le corps du traitement principal ou dans le code d'une procédure ou d'une fonction.

### C.1.3 Entiers

```
Variable Entière iVar
Début du Traitement
  Lire iVar
  Afficher iVar
  Si iVar == 7 Alors
    iVar <- 0
  Fin de Si
Fin du Traitement
```

```
int iVar;
int main()
{
   scanf("%d",&iVar); ViderTampon;
   printf("%i",iVar);
   if(iVar==7)
   {
      iVar=0;
   }
   return 0;
}
```

### C.1.4 Flottants ou pseudo-réels

```
Variable Flottante fVar
Début du Traitement
  Lire fVar
  Afficher fVar
  Si fVar == 7.0 Alors
    fVar <- 0.0
  Fin de Si
Fin du Traitement
```

```
double fVar;
int main()
{
  scanf("%lf",&fVar); ViderTampon;
  printf("%f",fVar);
  if(fVar==7.0)
  {
    fVar=0.0;
  }
  return 0;
}
```

### C.1.5 Caractères

```
Variable Caractère cVar
Début du Traitement
  Lire cVar
  Afficher cVar
  Si cVar == 'a' Alors
    cVar <- 'b'
  Fin de Si
Fin du Traitement
```

```
char cVar;
int main()
{
  scanf("%c",&cVar); ViderTampon;
  printf("%c",cVar);
  if(cVar=='a')
  {
    cVar='b';
  }
  return 0;
}
```

## C.1.6 Chaînes de caractères

```
Variable Chaîne sVar
Variable Chaîne #20 sDeux
Début du Traitement
  Lire sVar
  Afficher sVar
  Si sVar == "OUI" Alors
    sVar <- "oui"
  Fin de Si
Fin du Traitement
```

```
char sVar[256];
char sDeux[20];
int main()
{
  fgets(sVar,sizeof sVar,stdin);SupprimerCR(sVar);
  printf("%s",sVar);
  if(strcmp(sVar,"OUI")==0)
  {
    strcpy(sVar,"oui");
  }
  return 0;
  }
```

Par défaut, les chaînes ont une taille de 256 caractères. Il est possible de spécifier une taille plus courte en faisant suivre le mot `chaîne` de # et d'un nombre. Les chaînes de caractères ont été très peu employées dans les notes et les exemples. Elles sont traduites en C par des tableaux de caractères, dont la gestion n'est pas vraiment simple [1].

## C.1.7 Logiques

```
Variables logiques lReussi,lTermine
Début du traitement
  lReussi<-Vrai
  lTermine<-Faux
  Si lTermine alors
    Afficher "C'est fini"
  Fin de si
Fin du traitement
```

```
int lReussi, lTermine;
int main()
{
  lReussi=1;
  lTermine=0;
  if(lTermine)
    { printf("%s","C'est fini");
  }
  return 0;
}
```

1. On notera que pour son traitement des chaînes de caractères, le pseudo code s'affranchit totalement du C, qui les ignore superbement. Par exemple, on pourra constater que les expressions

```
Si s_Var == "oui" Alors et s_Var <- "non"
```

deviennent respectivement en C

```
if(strcmp(s_Var,"non")==0) et strcpy(s_Var,"non");
```

Pour des raisons de sécurité, la lecture se fait à l'aide de la fonction `fgets`, qui conserve malheureusement le retour à la ligne à la fin de la chaîne. La macro qui suit sert à enlever ce retour de chariot. Comme il n'est pas possible de déterminer la taille d'une chaîne passée en paramètre d'une sous-routine, deux méthodes sont employées : prendre une valeur par défaut de 256 ou modifier l'instruction `Lire` pour permettre de fixer par programme la taille de cette chaîne. La procédure suivante illustre la technique et son résultat en C.

```
Code procédure prTest(Valeur chaine sPara, valeur entière iTaille)
   Lire sPara
   Lire #15 sPara
   Lire #iTaille sPara
Fin de procédure
void prTest(char *sPara,int iTaille)
{
  fgets(sPara,256,stdin);SupprimerCR(sPara);
  fgets(sPara,15,stdin);SupprimerCR(sPara);
  fgets(sPara,iTaille,stdin);SupprimerCR(sPara);
}
```

Les variables logiques servent à mémoriser certains états particuliers du programme. Elles sont utilisées par `si` et `tant que` et ne peuvent ni se lire ni s'écrire. Les variables logiques sont traduites par des entiers en C. 1 représente `Vrai` et 0 `Faux`.

### C.1.8 Tableaux

```
Tableau Entiers iTest Taille 10
Tableau Flottants fTemperature Taille 31 : 3
Début du Traitement
  Déf Locales
    Variables Entières iJour, iMoment
  Fin des locales
  Aléatoire
  Init
  Compter Avec iJour de 0 A 30
    Init
    Compter Avec iMoment de 0 A 2
      fTemperature[iJour,iMoment]
      <- HasardMax(50) - 20
    Fin de Compter
  Fin de Compter
Fin du Traitement
```

```
int iTest[10];
double fTemperature[31][3];
int main()
{
  int iJour, iMoment;
  srand(time(0));
  for(iJour=0;iJour<=30;iJour++)
  {
    for(iMoment=0;iMoment<=2;iMoment++)
    {
      fTemperature[iJour][iMoment]=rand()%(50)-20;
    }
  }
  return 0;
}
```

Les tableaux peuvent avoir plusieurs dimensions.

### C.1.9 Structures

```
/* Déclaration du type */
Déf Structure eDate
  Champ Entier iJour
  Champ Entier iMois
  Champ Entier iAnnee
Fin de structure
Déf Structure ePersonne
  Champ Chaîne sNom
  Champ Entier iAge
  Champ eDate uDateNaissance
Fin de structure
/* Déclaration d'une variable */
Variable ePersonne uMoi
Prototype Fonction ePersonne fnCoupable
    (Tableau ePersonne uSuspects)
Début du Traitement
/* Accès aux champs de la variable */
  uMoi.sNom <- "Dupont"
  uMoi.uDateNaissance.iMois <- 4
Fin du Traitement
...
```

```
typedef struct ___eDate
{
  int iJour;
  int iMois;
  int iAnnee;
}eDate;
typedef struct ___ePersonne
{
  char sNom[256];
  int iAge;
  eDate uDateNaissance;
}ePersonne;
ePersonne uMoi;
ePersonne fnCoupable(ePersonne uSuspects[])
int main()
{
  strcpy(uMoi.sNom,"Dupont");
  uMoi.uDateNaissance.iMois=4;
  return 0;
}
...
```

On distingue la définition d'un type structure, la définition d'une variable de ce type et l'utilisation d'un champ.

# C.2 Instructions

## C.2.1 instructions prédéfinies

```
Début du Traitement
  Aléatoire
  Effacer Ecran
  Afficher "Bonjour"
  Fin de Ligne
  Tab
  Attente
Fin du Traitement
```

```
int main()
{
  srand(time(0));
  printf("\e[2J\e[1;1H");
  printf("%s","Bonjour");
  printf("\n");
  printf("\t");
  getchar();
  return 0;
}
```

L'instruction `Aléatoire` a pour simple but d'initialiser le générateur aléatoire. `Tab` écrit quelques blancs pour opérer une séparation (tabulation). Les instructions `Afficher`, `Effacer écran` et `Fin de ligne` (pour saut à la ligne suivante) ont une signification évidente. L'instruction `Attente` sert à provoque un arrêt du programme avant la fermeture de la fenâtre. Pour remplacer l'instruction obsolète `Long texte`, pas toujours bien perçue, la version 2010 du pseudo-code permet d'utiliser plusieurs valeurs séparées par le caractère `&` à la suite de l'instruction `Afficher`. Notons à cet égard que la traduction en C proposée par le compilateur n'est pas toujours très élégante. Par exemple,

```
Variable entière iAge
Variable chaine sNom
Début du traitement
 Afficher "Quel est votre nom ? "
 Lire sNom
 Afficher "Quel âge avez-vous ? "
 Lire iAge
Afficher "Cher monsieur " & sNom
 & ", vous avez " & iAge & " ans."
 Attente
Fin du traitement
```

```
int iAge;
char sNom[256];
int main()
{
 printf("%s","Quel est votre nom ? ");
 fgets(sNom,sizeof sNom,stdin);SupprimerCR(sNom);
 printf("%s","Quel Ã¢ge avez-vous ? ");
 scanf("%d",&iAge); ViderTampon
 printf("%s","Cher monsieur ");
 printf("%s",sNom);
 printf("%s",", vous avez ");
 printf("%d",iAge);
 printf("%s"," ans.");
 Attente
 return 0;
}
```

De toute évidence, un programmeur C écrirait :

```
printf("Cher monsieur %s, vous avez %d ans",sNom,iAge);
```

## C.2.2 affectations

```
Variable Entière iVar
Variable chaîne sVar
Début du Traitement
  iVar <- 1
  iVar Devient iVar + 2
```

```
int iVar;
char sVar[256];
int main()
{
  iVar=1;
```

```
  Incrémenter iVar
  Décrémenter iVar
  sVar<- "Bonjour"
Fin du Traitement
```

```
  iVar=iVar+2;
  iVar++;
  iVar--;
  strcpy(sVar,"Bonjour");
  return 0;
}
```

Les opérateurs `Devient` et `<-` sont équivalents. `In|Décrémenter` ne s'applique qu'à des entiers. Rappelons que l'affectation à une chaîne fonctionne différemment en C.

### C.2.3 alternatives

```
Variable Entière iVar
Début du Traitement
  Aléatoire
  iVar <- HasardMax(5)
  Si iVar > 2 Alors
    Afficher "Plus grand que deux"
  Sinon
    Afficher "Plus petit que trois"
  Fin de Si
Fin du Traitement
```

```
int iVar;
int main()
{
  srand(time(0));
  iVar=rand()%(5);
  if(iVar>2)
  {
    printf("%s","Plus grand que deux");
  }
  else
  {
    printf("%s","Plus petit que trois");
  }
  return 0;
}
```

Rappelons que la partie `sinon` de l'instruction est facultative.

### C.2.4 choix multiple

```
Variable Entière iVar
Début du Traitement
  Aléatoire
  iVar <- HasardMax(5)
  Examen si iVar
   Vaut 0, 1
     Afficher "Petit"
   Fin de Cas
   Vaut 2
     Afficher "Deux"
   Fin de Cas
   Vaut 3
     Afficher "Trois"
   Fin de Cas
   Défaut
     Afficher "Grand"
  Fin de Cas
  Fin de Examen
Fin du Traitement
```

```
int iVar;
int main()
{
  srand(time(0));
  iVar=rand()%(5);
  switch(iVar)
  {
   case 0:
   case 1:
    printf("%s","Petit");
    break;
   case 2:
    printf("%s","Deux");
    break;
   case 3:
    printf("%s","Trois");
    break;
   default :
    printf("%s","Grand");
```

```
    break;
  }
  return 0;
}
```

## C.2.5 répétitions à test initial

```
Variable Entière iVar
Début du Traitement
  Initialisation
    iVar <- 1
  Tant que
    iVar < 11
  Répéter
    Afficher iVar
    Fin de Ligne
    Moteur Incrémenter iVar
  Fin de Boucle
Fin du Traitement
```

```
int iVar;
int main()
{
  iVar=1;
  while(iVar<11)
  {
    printf("%i",iVar);
    printf("\n");
    /* Moteur : */
    iVar++;
  }
  return 0;
}
```

## C.2.6 répétitions à test final

```
Variable Entiére iVar
Début du Traitement
  Initialisation
    iVar <- 1
  Répéter
    Afficher iVar
    Fin de Ligne
    Moteur Incrémenter iVar
  Boucler tant que iVar <> 11
Fin du Traitement
```

```
int iVar;
int main()
{
  iVar=1;
  do
  {
    printf("%i",iVar);
    printf("\n");
    /* Moteur : */
    iVar++;
  }
  while(iVar!=11);
  return 0;
}
```

## C.2.7 répétitions avec compteur

### Boucles croissantes :

```
Variable Entière iVar
Début du Traitement
  Initialisation
  Compter Avec iVar De 1 A 10 Croissant
    Afficher iVar
    Fin de Ligne
  Fin de Compter
Fin du Traitement
```

```
int iVar;
int main()
{
  for(iVar=1;iVar<=10;iVar++)
  {
    printf("%i",iVar);
    printf("\n");
  }
```

```
  return 0;
}
```

**Boucles décroissantes :**

```
Variable Entière iVar
Début du Traitement
  Initialisation
  Compter Avec iVar De 10 A 1 Décroissant
    Afficher iVar
    Fin de Ligne
  Fin de Compter
Fin du Traitement
```

```
int iVar;
int main()
{
  for(iVar=10;iVar>=1;iVar--)
  {
    printf("%i",iVar);
    printf("\n");
  }
  return 0;
}
/
```

## C.3 Divers

### C.3.1 Fonctions prédéfinies

```
Début du Traitement
  Déf Locales
    Variable Flottante fTest
    Variable Entière iVal
  Fin des locales
  fTest <- Pi
  /* Nombre compris entre 0 et 9 inclus */
  iVal <- HasardMax(10)
  fTest <- Rac(iVal)
  fTest <- Sin(Pi/2)
  fTest <- Cos(Pi/4)
  fTest <- Tan(Pi/3)
Fin du Traitement
```

```
int main()
{
  double fTest;
  int iVal;
  /* Nombre compris entre 0 et 9 inclus */
  fTest=3.14159265358979323846;
  iVal=rand()%(10);
  fTest=sqrt(iVal);
  fTest=sin(3.14159265358979323846/2);
  fTest=cos(3.14159265358979323846/4);
  fTest=tan(3.14159265358979323846/3);
return 0;
}
```

### C.3.2 Procédures prédéfinies

Les fonctions `Lire` et `Afficher` ont été vues plus haut. Les instructions `Effacer Ecran`, *`Afficher`* `Message`, `Fin` *`de`* `Ligne`, `Tab` et `Aléatoire` peuvent également être considérées comme des procédures prédéfinies.

### C.3.3 Manipulation des chaînes [2]

Le langage C n'est pas favorable à la manipulation simple des chaînes de caractères. Deux fonctions et une commande permettent de petits traitements : `Longueur()` renvoie le nombre de caractères d'une chaîne, `SousChaîne()` renvoie un caractère d'une chaîne dont on spécifie la position. `SousChaîne()` peut également figurer à gauche d'une instruction `Devient`

2. Ces instructions ne sont pas décrites dans le cours.

et autorise alors la modification du caractère spécifié. Le programme suivant affiche les caractères d'un mot en les séparant, puis remplace la première lettre par une majuscule.

```
Variable chaine sTest
variable entière iI
Début du traitement
  Initialisation
    sTest <- "bonjour"
  Compter avec iI de 0 à longueur(sTest)
    Afficher SousChaine(sTest,iI) tab
  fin de compter
  SousChaine(sTest,0)<-'B'
  Afficher sTest
Fin du traitement
```

```
char sTest[256];
int iI;
int main()
{
  strcpy(sTest,"bonjour");
  for(iI=0;iI<=strlen(sTest);iI++)
  {
    printf("%c",sTest[iI]);
    printf("\t");
  }
  sTest[0]='B';
  printf("%s",sTest);
  return 0;
}
```

### C.3.4 Commentaires

Le texte placé entre /* et */ est ignoré par le compilateur et placé tel quel dans le programme objet en C. La position exacte peut varier quelque peu par rapport au source. C'est dû au fait que l'analyseur peut avoir commencé à écrire du code C au moment où il rencontre le commentaire. Il est conseillé de placer les commentaires sur des lignes séparées.

### C.3.5 Inclusion d'instructions C

Il est possible de placer une ou plusieurs instructions C encadrées par <* et *> là où peut figurer une instruction normale. L'instruction n'est pas analysée et peut donc donner lieu à une erreur de compilation par `gcc`.

```
Variable Entière iVar
Début du Traitement
   Lire iVar
   <* printf("%d\n",iVar); *>
Fin du Traitement
```

On peut également utiliser du code C pour initialiser un tableau.

```
Tableau Entier iTab Taille 4 <* = {1,2,3,4} *>
```

## C.4 Procédures et fonctions

### C.4.1 Définition des prototypes

```
Prototype Fonction Entière fnCarre
   (Valeur Entière iNombre)
Prototype Procedure prAfficherEuros
   (Valeur Entière iFrancs)
```

```
int fnCarre(int iNombre);
void prAfficherEuros(int iFrancs);
```

Le prototype reprend tout l'en-tête de la définition d'une fonction ou d'une procédure, précédé du terme `Prototype` :

### C.4.2 Définition des corps de procédures et fonctions

```
Code Fonction Entière fnCarre
    (Valeur Entière iNombre)
  Renvoyer iNombre*iNombre
Fin de Fonction
Code Procedure prAfficherEuros
    (Valeur Entière iFrancs)
  Afficher iFrancs / 40.3399
Fin de Procédure
```

```
int fnCarre(int iNombre)
{
  return iNombre*iNombre;
}
void prAfficherEuros(int iFrancs)
{
  printf("%f",iFrancs/40.3399);
}
```

### C.4.3 Appel des sous-routines

```
Début du Traitement
  Afficher fnCarre(2)
  prAfficherEuros(10)
Fin du Traitement
```

```
int main()
{
  printf("%i",fnCarre(2));
  prAfficherEuros(10);
  return 0;
}
```

### C.4.4 Passage des paramètres

Les paramètres passés par valeur sont précédés du mot `Valeur` :

```
Prototype Fonction Entière fnCarre
(Valeur Entière iNombre)
```

```
int fnCarre(int iNombre);
```

Les paramètres passés par adresse (ou par référence) sont précédés du mot `Adresse`. En C, leur nom est prédédé de `*`.

```
Prototype Procédure prInverser  (Adresse Entier
iN1, Adresse Entier iN2)
```

```
void prInverser(int *iN1,int *iN2);
```

La définition de paramètres par adresse suppose l'emploi de pointeurs en C, ce qui rend l'écriture de la sous-routine nettement plus difficile.

```
Code Procédure prInverser
(Adresse Entier iN1, Adresse Entier iN2)
  Déf Locales
    Variable Entière iTemp
  Fin des locales
  iTemp <- iN1
  iN1 <- iN2
  iN2 <- iTemp
Fin de Procédure
```

```
void prInverser(int *iN1,int *iN2)
{
  int iTemp;
  iTemp=*iN1;
  *iN1=*iN2;
  *iN2=iTemp;
}
```

De même, l'appel d'une sous-routine avec des paramètres passés par adresse suppose l'utilisation d'un opérateur en C.

```
Début du traitement
  Déf locales
    Variables entières iV1, iV2
  fin des locales
  iV1 <- 1
  iV2 <- 143
  prInverser(iV1,iV2)
Fin du traitement
```

```
int main()
{
  int iV1, iV2;
  iV1=1;
  iV2=143;
  prInverser(&iV1,&iV2);
  return 0;
}
```