Fortran: Norme 2003

Laurence Viry

Ecole d'Automne « Informatique Scientifique pour le Calcul »

2 Octobre 2008

Sommaire

Sommaire

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme
- 1977 (Fortran V) Travaille des comités X3J3/ANSI et WG5/ISO pour moderniser Fortran

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme
- 1977 (Fortran V) Travaille des comités X3J3/ANSI et WG5/ISO pour moderniser Fortran
- 1991/1992 : Norme ISO et ANSI Fortran90
  premiers compilateurs Cray et IBM en 1994
- 1997 : Norme ISO et ANSI Fortran95
  premiers compilateurs Cray et IBM RS/6000 en 1999

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme
- 1977 (Fortran V) Travaille des comités X3J3/ANSI et WG5/ISO pour moderniser Fortran
- 1991/1992 : Norme ISO et ANSI Fortran90
  premiers compilateurs Cray et IBM en 1994
- 1997 : Norme ISO et ANSI Fortran95
  premiers compilateurs Cray et IBM RS/6000 en 1999

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme
- 1977 (Fortran V) Travaille des comités X3J3/ANSI et WG5/ISO pour moderniser Fortran
- 1991/1992 : Norme ISO et ANSI Fortran90
  premiers compilateurs Cray et IBM en 1994
- 1997 : Norme ISO et ANSI Fortran95
  premiers compilateurs Cray et IBM RS/6000 en 1999

Historique

- 1954 : Création du premier langage symbolique par John Backus d'IBM FORTRAN (Mathematical FORmula TRANslating System)
- 1958 : Fortran II(IBM) Généralisation aux autres constructeurs nécessité de normalisation
- 1966 : Fortran66 (Fortran IV) première norme
- 1977 (Fortran V) Travaille des comités X3J3/ANSI et WG5/ISO pour moderniser Fortran
- 1991/1992 : Norme ISO et ANSI Fortran90
  premiers compilateurs Cray et IBM en 1994
- 1997 : Norme ISO et ANSI Fortran95
  premiers compilateurs Cray et IBM RS/6000 en 1999
- 2004 : Norme ISO et ANSI Fortran2003 /item¡6-¿ 2008 : Norme ISO et ANSI Fortran2008

Sommaire

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)
- Utilisation des noms génériques et surcharges d'opérateurs

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)
- Utilisation des noms génériques et surcharges d'opérateurs
- Pointeurs

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)
- Utilisation des noms génériques et surcharges d'opérateurs
- Pointeurs
- Allocation dynamique
- type de données définis par l'usager

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)
- Utilisation des noms génériques et surcharges d'opérateurs
- Pointeurs
- Allocation dynamique
- type de données définis par l'usager
- Extensions tableaux

Apport de Fortran 90/95

- Favorise une programmation plus structurée (module,..)
- Apporte plus de sécurité (interface explicite, données et procédure publiques ou privées)
- Utilisation des noms génériques et surcharges d'opérateurs
- Pointeurs
- Allocation dynamique
- type de données définis par l'usager
- Extensions tableaux
- ...

Sommaire

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C
- Prise en compte du standard IEEE 754 et traitement des exceptions

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C
- Prise en compte du standard IEEE 754 et traitement des exceptions
- Entrées/Sorties

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C
- Prise en compte du standard IEEE 754 et traitement des exceptions
- Entrées/Sorties
- Nouveautés concernant les types dérivés, les modules, les tableaux dynamiques

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C
- Prise en compte du standard IEEE 754 et traitement des exceptions
- Entrées/Sorties
- Nouveautés concernant les types dérivés, les modules, les tableaux dynamiques
- Support pour la programmation orientés objets

Principales nouveautés de Fortran 2003

- Outils favorisant l'intéropabilité avec C
- Prise en compte du standard IEEE 754 et traitement des exceptions
- Entrées/Sorties
- Nouveautés concernant les types dérivés, les modules, les tableaux dynamiques
- Support pour la programmation orientés objets
- Divers apports mineurs

Sommaire

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales
- Syntaxe simple permettant au compilateur et au programmeur de gérer le parallélisme

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales
- Syntaxe simple permettant au compilateur et au programmeur de gérer le parallélisme
- Permet l'écriture simple et concise d'un paralléllisme adapté aux architectures multi-core

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales
- Syntaxe simple permettant au compilateur et au programmeur de gérer le parallélisme
- Permet l'écriture simple et concise d'un paralléllisme adapté aux architectures multi-core

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales
- Syntaxe simple permettant au compilateur et au programmeur de gérer le parallélisme
- Permet l'écriture simple et concise d'un paralléllisme adapté aux architectures multi-core

Fortran 2008

Le principal apport du prochain standard(Fortran 2008) : les co-array

- Un programme contenant des co-array est repliqué, chaque copie s'exécutant sur ses données locales
- Syntaxe simple permettant au compilateur et au programmeur de gérer le parallélisme
- Permet l'écriture simple et concise d'un paralléllisme adapté aux architectures multi-core
- ...

Sommaire

Documentation - Livres

- Adams, J.C., Brainerd, W.S., Hendrickson, R.A., Maine, R.E., Martin, J.T., Smith, B.T., The Fortran 2003 Handbook, the complete syntax, features and procedures, 2008, Springer Verlag, ISBN :978-1-84628-378-9
- Chapman S.J., Fortran 95/2003 For Scientists and Engineers, 2007, Mc Graw-Hill. ISBN 978-0073191577, ISBN 0073191574
- Metcalf M., Reid J. and Cohen M., Fortran 95/2003 Explained, 2004, Oxford University Press. ISBN 0-19-852693-8, ISBN 0-19-852692-X
- J. Reid The Future of Fortran. Computing in Science and Engineering, July/August 2003
- J. Reid. The new Featuresof Fortran 2003. Technical Report.
- ...

Documentation - Web

- P. Corde and H. Delouis. Les apports de Fortran 2003 (http ://www.idris.fr)
- Fortran Standards Technical Committee : http ://www.nag.co.uk/sc22wg5/.
- Fortran 90,95,2003 home page : http ://www.kcl.ac.uk/support/cit/fortran/f90home.html
- http ://www.fortranplus.co.uk/index.html : Fortran information (ressources files, compiler support for Fortran2003,...)
- Fortran Forum (ACM)
- ...

Sommaire

Standard IEEE 754 - valeurs particulières/exceptions

Valeurs particulières

le Standard IEEE 754 concerne l'arithmétique réelle flottante et le traitement des exceptions.

- **NaN** (Not a Number) : valeur associée à une expression mathématique indéterminée comme $0/0$, $0 * \infty$
- **+INF** $(+\infty)$,**-INF** $(-\infty)$
- 0^+,0^-
- dénormalisée(petite valeur)

5 classes d'exceptions

Une opération arithmétique peut produire comme résultat une valeur particulière ou bien une valeur en dehors de l'ensemble des valeurs représentables, cela génère **un événement de type exception**.

- **overflow** : valeur calculée trop grande
- **underflow** : valeur calculée trop petite(dénormalisée ou 0)
- **division par zéro**
- **opération invalide** valeur calculée égale à NaN
- **opération inexacte** : valeur calculée non représentable exactement (implique un arrondi)

Arrondis

Lorsque le valeur n'est pas représentable, une exception de type " valeur inexacte est générée", ce qui se traduit par un arrondi.

- **toward nearest**
- **toward zero**
- **toward +INF**
- **toward -INF**

Pas de valeur par défaut n'est prévu par la norme.

Fortran2003 - Standard IEEE 754

Il est possible avec Fortran2003 d'utiliser ce standard afin qu'un exécutable rencontrant une de ces exceptions ne s'arrête pas mais utilise une valeur spéciale pour la suite du calcul tout en fixant un flag pour avertir l'utilisateur.

Fortran2003 supporte d'autres aspects du standard IEEE tels que le choix du mode d'arrondi.

Les trois modules concernant l'implémentation de ces possibilités sont :

Modules IEEE

- **IEEE_EXCEPTIONS** :gestion des exceptions uniquement
- **IEEE_ARITHMETIC** : gestion des autres aspects de la norme
- **IEEE_FEATURES** : permet un contrôle sur les procédures á utiliser

Sommaire

Fonctions d'interrogation - Standard IEEE 754

Ces fonctions permettent de tester :

- Si l'environnement utilisé est conforme à la norme IEEE 754

Fonctions d'interrogation - Standard IEEE 754

Ces fonctions permettent de tester :

- Si l'environnement utilisé est conforme à la norme IEEE 754
- la classe d'une variable x(NaN,∞,...)

Fonctions d'interrogation - Standard IEEE 754

Ces fonctions permettent de tester :

- Si l'environnement utilisé est conforme à la norme IEEE 754
- la classe d'une variable x(NaN,∞,...)

Conformité de l'environnement	**classe de la variable x**
IEEE_SUPPORT_STANDARD()	IEEE_CLASS(x)
IEEE_SUPPORT_DATATYPE()	IEEE_IS_NAN(x)
IEEE_SUPPORT_DENORMAL()	IEEE_IS_FINITE(x)
IEEE_SUPPORT_INF()	IEEE_IS_NEGATIVE(x)
IEEE_SUPPORT_NAN()	IEEE_IS_NORMAL(x)
IEEE_SUPPORT_SQRT()	

Fonctions d'interrogation - Standard IEEE 754

Ces fonctions permettent de tester :

- Si l'environnement utilisé est conforme à la norme IEEE 754
- la classe d'une variable x(NaN,∞,...)

Conformité de l'environnement	**classe de la variable x**
IEEE_SUPPORT_STANDARD()	IEEE_CLASS(x)
IEEE_SUPPORT_DATATYPE()	IEEE_IS_NAN(x)
IEEE_SUPPORT_DENORMAL()	IEEE_IS_FINITE(x)
IEEE_SUPPORT_INF()	IEEE_IS_NEGATIVE(x)
IEEE_SUPPORT_NAN()	IEEE_IS_NORMAL(x)
IEEE_SUPPORT_SQRT()	

Fournit un booléen ou une constante symbolique prédéfinie(
IEEE_SIGNALING_NAN, IEEE_NEGATIVE_NORMAL, ...)

Sommaire

Gestion du mode d'arrondi

Deux fonctions permettent de gérer le mode d'arrondi

- IEEE_GET_ROUNDING_MODE(round_value)
- IEEE_SET_ROUNDING_MODE(round_value)

round_value est une constante du type IEEE_ROUND_TYPE dont la valeur peut être :

- IEEE_NEAREST : arrondi à la valeur représetable la plus proche
- IEEE_TO_ZERO : arrondi à la valeur représentable suivante en se déplacant vers zéro
- IEEE_UP : arrondi à la valeur représentable directement supérieure
- IEEE_DOWN : arrondi à la valeur représentable directement inférieure
- IEEE_OTHER : indique que le mode d'arrondi choisi n'est pas conforme à la norme IEEE

Fonctions de gestion des exceptions

Procédures activant les flags lorsqu'ont lieu les exceptions :

- IEEE_GET_FLAG(flag,flag_value) : retourne le booléen TRUE lorsque le type d'exception flag se produit, false dans le cas inverse
- IEEE_SET_FLAG(flag,flag_value) : permet d'activer(TRUE) ou de désactiver la détection d'une exception de type flag

Example

```
use ieee_exceptions
logical : :flag_value
...
CALL IEEE_GET_FLAG(IEEE_OVERFLOW,flag_value)
if (flag_value) then
print*,"probleme d'overflow signale"
else
print*,"pas de probleme d'overflow signale"
endif
```

Fonctions de gestion des interruptions

Il est possible de donner un comportement(arrêter ou contiuer) en fonction des exceptions grâce à :

- IEEE_GET_HALTING_MODE(flag,halting) : retoune le mode d'arrêt pour une exception de type flag et met halting à TRUE si cette exception se produit
- IEEE_SET_HALTING_MODE(flag,halting) : permet de contrôler la continuation(FALSE) ou l'arrêt(TRUE) après qu'une exception de type flag se soit produite.

Example

```
USE IEEE_EXCEPTIONS
REAL : : x
...
CALL IEEE_SET_HALTING_MODE(IEEE_DIVIDE_BY_ZERO,.false.)
x = 1./0.0
! Le programme continue malgré une erreur de division par zéro
CALL IEEE_SET_HALTING_MODE(IEEE_DIVIDE_BY_ZERO,.true.)
x = 1./0.0
! Le programme s'arrête à cause d'une erreur de division par zéro
```

Sommaire

Interopérabilité avec C

- Nécessite l'utilisation d'entités communes aux deux langages(variables, fonctions,...) contraintes et restrictions

Interopérabilité avec C

- Nécessite l'utilisation d'entités communes aux deux langages(variables, fonctions,...) contraintes et restrictions
- Définition de pointeur Fortran particulier du type dérivé semi-privé C_PTR interopérable avec les pointeurs C.

Interopérabilité avec C

- Nécessite l'utilisation d'entités communes aux deux langages(variables, fonctions,...) contraintes et restrictions
- Définition de pointeur Fortran particulier du type dérivé semi-privé C_PTR interopérable avec les pointeurs C.
- L'accès au module ISO_C_BINDING (USE)permet l'utilisation d'entités interopérables avec C qui seront traitées comme telles par le compilateur Fortran

Sommaire

Interopérabilité : entités de type intrinsèque

Type(sous-type) en Fortran	**Type correspondant en C**
INTEGER(kind=C_INT)	int
INTEGER(kind=C_SHORT)	short int
INTEGER(kind=C_LONG)	long int
REAL(kind=C_FLOAT)	float
REAL(kind=C_DOUBLE)	double
CHARACTER(kind=C_CHAR)	char
...	

C ne supporte que les variables de type CHARACTER de longueur 1, les chaines de caractères sont gérées sous forme de tableaux

CHARACTER(kind=C_CHAR),DIMENSION(*) : : chaine

Interopérabilité : entités de type intrinsèque

Example

Use ISO_C_BINDING

```
  ...
  integer(kind=C_INT)        : : i     ! déclaration Fortan
  real(kind=C_DOUBLE)        : : x     ! déclaration Fortan
```

sont interopérable avec les variables i et x déclarées ainsi en C

```
  int          i ;
  double       x ;
```

Interopérabilité : caractères spéciaux

Nom	Signification en C	Valeur ASCII	Équivalent en C
C_NULL_CHAR	null character	achar(0)	\0
C_ALERT	alert	achar(7)	\a
C_BACKSPACE	backspace	achar(8)	\b
C_NEW_LINE	line feed\new line	achar(10)	\n
...			

Sommaire

Tableaux C

Un tableau est interopérable si :

- il est de type interopérable

Tableaux C

Un tableau est interopérable si :

- il est de type interopérable
- de profil explicite

Tableaux C

Un tableau est interopérable si :

- il est de type interopérable
- de profil explicite
- pour les tableaux multidimensionnés, l'ordre des indices doit être inversé

Tableaux C

Un tableau est interopérable si :

- il est de type interopérable
- de profil explicite
- pour les tableaux multidimensionnés, l'ordre des indices doit être inversé

Example

```
integer(kind=C_INT),DIMENSION(5,2 :7,*)        : : t1
real(kind=C_DOUBLE),DIMENSION(5,2 :7,10)       : : t2
```

sont interopérables avec les tableaux déclarées ainsi en C

```
int        t1[][6][5] ;
double     t2[10][6][5] ;
```

Sommaire

Interopérabilité : Variables globales C

Une variable globale C peut interopérer avec un bloc COMMON ou une variable déclarée dans un module.

- La variable globale Fortran sera déclarée avec l'attribut **BIND(C)** pour être interopérable avec une variable externe C
- Le paramètre **NAME** devra être utilisé pour le mise en correspondance des références des noms en Fortran et en C

Example

```
module variables_C
use                                : : ISO_C_BINDING
INTEGER(C_INT), BIND(C)            : : c_extern
INTEGER(C_LONG)                    : : fort_var
BIND(C,NAME='C_var')               : : fort_var
COMMON/COM/r,s
COMMON/SINGLE/ t
REAL(KIND=C_FLOAT)                 : : r,s,t
BIND(C)                            : : /COM/,/SINGLE/
end module variables_C
```

```
int                 c_extern ;
long                C_var ;
struct{float r,s ;} com ;
float               single ;
```

Sommaire

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran
- Un pointeur Fortran peut pointer sur une zone mémoire non contigue

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran
- Un pointeur Fortran peut pointer sur une zone mémoire non contigue

Le type dérivé semi-privé Fortran **C_PTR** sera utilisé pour assurer la compatibilité avec les pointeurs C. Une composante privée contient l'adresse cachée d'une cible.

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran
- Un pointeur Fortran peut pointer sur une zone mémoire non contigue

Le type dérivé semi-privé Fortran **C_PTR** sera utilisé pour assurer la compatibilité avec les pointeurs C. Une composante privée contient l'adresse cachée d'une cible.

Les manipulations relatives aux pointeurs C(C_PTR) se font via des opérateurs ou des procédures

- **C_LOX(x)** : retourne l'adresse C de x

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran
- Un pointeur Fortran peut pointer sur une zone mémoire non contigue

Le type dérivé semi-privé Fortran **C_PTR** sera utilisé pour assurer la compatibilité avec les pointeurs C. Une composante privée contient l'adresse cachée d'une cible.

Les manipulations relatives aux pointeurs C(C_PTR) se font via des opérateurs ou des procédures

- **C_LOX(x)** : retourne l'adresse C de x
- **C_F_POINTER(CPTR,FPTR[,SHAPE])** : construction d'un pointeur fortran à partir d'un pointeurC

Interopérabilité : les pointeurs C

Différences avec les pointeurs C

- On ne peut ni connaître, ni manipuler l'adresse désignée par un pointeur Fortran
- Un pointeur Fortran peut pointer sur une zone mémoire non contigue

Le type dérivé semi-privé Fortran **C_PTR** sera utilisé pour assurer la compatibilité avec les pointeurs C. Une composante privée contient l'adresse cachée d'une cible.

Les manipulations relatives aux pointeurs C(C_PTR) se font via des opérateurs ou des procédures

- **C_LOX(x)** : retourne l'adresse C de x
- **C_F_POINTER(CPTR,FPTR[,SHAPE])** : construction d'un pointeur fortran à partir d'un pointeurC
- **C_ASSOCIATED(C_PTR1[,C_PTR2])** : teste si un pointeur C est nul ou si deux pointeurs C sont identiques.

Sommaire

Interopérabilité des types dérivés

Via l'attribut **BIND(C)**

Objet de type C_struct défini en C

```
typedef struct
{
   int m,n ;
   floaT r ;
} C_struct
```

est interopérable avec le type dérivé Fortran F_struc défini par

Objet de type F_struct défini en Fortran

```
USE,INTRINSIC : : ISO_C_BINDING
TYPE,BIND(C) : : F_struct
   INTEGER(kind=C_INT) : : m,n
   REAL(kind=C_FLOAT) : : r
END TYPE F_struct
```

Sommaire

Nouveautés syntaxiques Fortran

- Passage des arguments par copie dans la stack pour les variables déclarées avec l'attribut VALUE
- L'attribut BIND(C,...) est obligatoire à la définition d'une procédure Fortran interopérable

Nouveautés syntaxiques Fortran

- Passage des arguments par copie dans la stack pour les variables déclarées avec l'attribut VALUE
- L'attribut BIND(C,...) est obligatoire à la définition d'une procédure Fortran interopérable

L'interface de la procédure Fortran devra contenir les informations nécessaires pour définir l'interopérabilité avec un prototype de fonction C.

Fonctions C procédures Fortran

Nouveautés syntaxiques Fortran

- Passage des arguments par copie dans la stack pour les variables déclarées avec l'attribut VALUE
- L'attribut BIND(C,...) est obligatoire à la définition d'une procédure Fortran interopérable

L'interface de la procédure Fortran devra contenir les informations nécessaires pour définir l'interopérabilité avec un prototype de fonction C.

Interface de procédure Fortran

- Appel C de Fortran : appel procédurale de la fonction C et le bloc interface associé
- Appel Fortran depuis C : partie déclarative de la procédure Fortran

Fonction C appelée depuis une procédure Fortran (cours IDRIS)

Fortran passe à C deux arguments :

- un tableau passé par référence,
- une variable entière passée par valeur

Fonction C appelée depuis une procédure Fortran (cours IDRIS)

Fortran passe à C deux arguments :

- un tableau passé par référence,
- une variable entière passée par valeur

Prototype de la fonction C :

float C_Func(float *buf,int count)

Fonction C appelée depuis une procédure Fortran (cours IDRIS)

Fortran passe à C deux arguments :

- un tableau passé par référence,
- une variable entière passée par valeur

Prototype de la fonction C :
float C_Func(float *buf,int count)

Example

```
module FTN_C
  use : : ISO_C_BINDING
  interface
    function C_FUNC(array,N) BIND(C,NAME="C_Func")
      import C_INT,C_FLOAT
      implicit none
      real(kind=C_FLOAT) : : C_FUNC
      integer(kind=C_INT),dimension(*) : : array
      integer(kind=C_INT),VALUE : : N
    end function C_Func
  end interface
end module FTN_C

program p1
  use FTN_C
```

Sommaire

Passage en argument de procèdure

Il est possible de passer un tableau allouable non alloué en argument de procédure et de l'allouer dans la procédure.

Example

```
program main
   integer, dimension(*,*),allocatable : : tab_i
   ...
! Le tableau tab_i n'a pas ete alloue
   call proc(tab_i)
   ...
end program main

subroutine proc(tab)
   integer, dimension(*,*),allocatable,intent(inout) : : tab_i
   ...
   allocate(tab(100,50))
   ...
end subroutine proc
```

Sommaire

Composante allouable d'un type dérivé

Il est possible d'allouer dynamiquement une composante d'un type dérivé.

Example

```
type coord
   integer : : i,j
   double precision,dimension( :, :),allocatable : : x,y
end type coord
   ...
type(coord) : : tab_coord
   ...
imax=tab_coord%i
jmax=tab_coord%j
allocate(tab_coord%x(imax,jmax),tab_coord%y(imax,jmax))
   ...
```

Sommaire

Réallocation par affectation

A est un tableau dynamique et B est un tableau statique. En Fortran90/95, il faut s'assurer de la compatibilité des tailles des deux tableaux avant l'affectation.

Example

```
S=size(A)
if (    allocated(A)    )    then
   if (    size(A)/= S    )    deallocate(A)
endif
if (    .not.allocated(A)    )    allocate(A(S))
A=B
```

En Fortran2003 l'affectation réalloue automatiquement le tableau dynamique, l'exemple précédent devient : A=B Cette facilité simplifie considérablement la gestion des chaîne de caractères.

Sommaire

Nouveaux paramètres des instructions OPEN/READ/WRITE

- IOMSG : identifie une chaîne de caractères récupérant un message d'erreur
- ROUND : paramètre de l'instruction OPEN d'une E/S formattée permettant de contrôler le type d'arrondi(up, down, zero, nearest, compatible, processor_defined).
- SIGN :paramètre de l'instruction OPEN pour la gestion du signe + des données numériques en sortie, il peut être modifié dans l'instruction WRITE
- IOSTAT : deux fonctions élémentaires is_iostat_end et is_iostat_eor permettent de tester la valeur de l'entier référencé dans l'instruction READ

Sommaire

Sommaire

Entrées-Sorties asynchrones

Les entrées-sorties peuvent être **asynchrones**, ce qui permet au programme de continuer pendant que l'entrée-sortie est en cours.
Ce mode d'E/S est possible si :

- si le fichier externe concerné est ouvert avec le paramètre ASYNCHRONOUS='yes'
- ce paramètre sera également fourni lors de l'instruction READ/WRITE
- une synchronisation peut être demandée explicitement à l'aide de l'instruction WAIT(UNIT=...,...) (implicite à la demande d'un INQUIRE ou d'un CLOSE)
- les entités concernés par une E/S asynchrone récupèrent un nouvel attribut ASYNCHRONOUS pour indiquer au compilateur un risque en cas d'optimisation

Sommaire

Entrées-Sorties en mode stream

Le paramètre ACCES de l'instructtion OPEN admet une troisième valeur STREAM permettant d'effectuer des E/S en s'affranchissant de la notion d'enregistrement.

- le fichier est considéré comme une suite d'octets
- l'E/S est faite relativement à la position courante
- le fichier peut être formatté ou pas

Cette nouvelle méthode d'accès facilite l'échange des fichiers binaires entre Fortran et C

Sommaire

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)$\Rightarrow$ lecture_format1,lecture_format1

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)⇒ lecture_format1,lecture_format1
- GENERIC : : READ(UNFORMATTED)⇒ lecture_nonformat1,lecture_nonformat1

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)⇒ lecture_format1,lecture_format1
- GENERIC : : READ(UNFORMATTED)⇒ lecture_nonformat1,lecture_nonformat1
- GENERIC : : WRITE(FORMATTED)⇒ ecriture_format1,ecriture_format1

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)⇒ lecture_format1,lecture_format1
- GENERIC : : READ(UNFORMATTED)⇒ lecture_nonformat1,lecture_nonformat1
- GENERIC : : WRITE(FORMATTED)⇒ ecriture_format1,ecriture_format1
- GENERIC : : WRITE(UNFORMATTED)⇒ ecriture_nonformat1,ecriture_nonformat1

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)⇒ lecture_format1,lecture_format1
- GENERIC : : READ(UNFORMATTED)⇒ lecture_nonformat1,lecture_nonformat1
- GENERIC : : WRITE(FORMATTED)⇒ ecriture_format1,ecriture_format1
- GENERIC : : WRITE(UNFORMATTED)⇒ ecriture_nonformat1,ecriture_nonformat1

La définition de la procédure est insérée au sein de la définition de type

Entrées-Sorties des types dérivés

Il devient possible de contrôler les opérations d'entrée-sortie portant sur un objet de type dérivé(public ou semi-privé) via une ou plusieurs procédures.

Il y a 4 catégories de procédures :

- GENERIC : : READ(FORMATTED)⇒ lecture_format1,lecture_format1
- GENERIC : : READ(UNFORMATTED)⇒ lecture_nonformat1,lecture_nonformat1
- GENERIC : : WRITE(FORMATTED)⇒ ecriture_format1,ecriture_format1
- GENERIC : : WRITE(UNFORMATTED)⇒ ecriture_nonformat1,ecriture_nonformat1

La définition de la procédure est insérée au sein de la définition de type

Prototype des procédures

SUBROUTINE lecture_format(dtv,unit,iotype,v_list,iostat,iomsg)
SUBROUTINE ecrit_nonformat(dtv,unit,iostat,iomsg)

Entrées-Sorties des types dérivés

Example

```
module couleur_mod
   type couleur
      character(len=16) : : nom
      real,dimension(3) : : compos
      generic : : read(UNFORMATTED) ⇒ lec_binaire
   end type couleur
contains
   subroutine lec_binaire(dtv,unit,iostat,iomsg)
      type(couleur),intent(IN) : : dtv
      integer,intent(IN) : : unit
      integer,intent(OUT) : : iostat
      character(*),intent(INOUT) : : iomsg
   ...
   end subroutine lec_binaire
end module couleur_mod
```

Cette possibilité permet de bénéficier du concept d'abstraction des données dans les E/S.

Sommaire

Fortran90/95 : vers une programmation plus modulaire

Apport des normes 90 et 95

Fortran90/95 n'est pas un langage objet mais il en possède certaines extensions :

- il facilite la définition et la manipulation de nouveaux types
- il facilite la manipulation des tableaux
- il permet la surcharge des opérateurs, la définition de fonctions génériques et l'encapsulation de types dérivés et de fonctions agissant sur ces types dans des modules
- ...

Ces deux normes permettent avant tout une plus grande modularité et plus d'encapsulation des données et des procédures.

Fortra2003 : vers une programmation orientée objet

Apport de la norme f2003

Fortran2003 introduit de nouveaux concepts qui permettent d'utiliser la puissance d'un langage orienté objet :

- les types dérivés paramétrés,
- les types dérivés étendus(hiérarchie de type) et la notion de classe
- les variables polymorphiques(de type pointeur) dont le type peut varier an cours d'exécution
- des procédures attachées à un type dérivé(pointeur de procédures) éventuellement par non générique ou par opérateur(principe des méthodes utilisées en C++).
- les types étendus héritent des composantes et des procédures des types dont ils sont issus
- classes abstraites
- ...

Sommaire

Les types dérivés paramétrés

Le paramétrage des types dérivés se fait par l'ajout de deux paramètres (identifiés par KIND et LEN) dans la définition du type.

- KIND impose le type des composantes
- LEN impose les bornes d'un tableau ou la longueur d'une chaîne de caractères.

Example

```
type coord(prec,i,j)
! on definit d'abord les parametres KIND et LEN
   integer,KIND : : prec=kind(1.0)
   integer,LEN : : i,j

! On definit ensuite les composantes
   integer : : nb_elem=i*j
   real(KIND=prec),dimension(i,j) : : x,y
end type coord

type(coord(k=4,20,30)) : : coord1 ! tableau de reels simple precision (10,10)
type(coord(4, :,30)),allocatable : : coord2 ! tableau allouable de reels simple precision
```

Extension d'un type dérivé

- l'utilisation de l'attribut extensible permet d'utiliser un type déjà défini pour lui ajouter des composantes

Example

```
type,extensible : : point2d
   real : : x,y
end type point2d

type,extends(point2d) : : point3d
   real : : z
end type point3d
type(point2d) : : pt2d ! point du plan 2D
type(point3d) : : pt3d ! point de l'espace 3D
```

Sommaire

Notion de classe

- on peut regrouper un type dérivé et les types dérivés construits à partir de lui dans une classe. **CLASS(point2d) : : point**
- on peut établir un comportement en fonction du type hérité passé en argument grâce à la structure de contrôle SELECT TYPE.

Example

```
subroutine distance(p1,p2)
   class(point2d) : : p1,p2
   real : : distance
   SELECT TYPE(p1)
! Si le point est de type point2d
      TYPE IS(point2d)
      distance = sqrt((p1%x - p2%x)^2 + (p1%y - p2%y)^2)
! Si le point est de type point3d
      TYPE IS(point3d)
      distance = sqrt((p1%x - p2%x)^2 + (p1%y - p2%y)^2 + (p1%z - p2%z)^2)
   END SELECT
end subroutine distance
```

Variable polymorphique

- C'est une variable dont le type peut varier au cours de l'exécution
- elle aura l'attribut POINTEUR ou ALLOCATABLE ou l'argument muet d'une procédure
- pour sa déclaration, on spécifie le mot-clé CLASS à la place TYPE
- le type indiqué avec le mot-clé CLASS est une type dérivé définie avec l'attribut extensible

Example

```
type,extensible point
   real : : x,y
end type point
class(point),pointer : : p
```

p pourra être associé à un objet de type point et à toutes les extensions de ce type

Pointeur polymorphique

Variable polymorphique ayant l'attribut POINTEUR, son type dynamique est celui de sa cible, il peut être définie par une association ou une allocation

Example

```
type(point2d),target                    : : p2d
type(point3d),target                    : : p3d
class(point2d),pointer                  : : ptr2d_1,ptr2d_2
class(point3d),pointer                  : : ptr3d
class(point2d),allocatable              : : point

ptr2d_1⟹ p2d                            ! Le type dynamique de ptr2d_1 est type(point2d)
ptr2d_2⟹ p3d                            ! Le type dynamique de ptr2d_2 est type(point3d)
ptr2d_2⟹ ptr2d_1                        ! Le type dynamique de ptr2d_2 est celui de ptr2D_1
ptr3d⟹ ptr2d_1                          ! Interdit
allocate(ptr2d_1)                       ! alloue un objet de type dynamique type(point2d)
allocate(type(point3d) : :ptr2d_1)      ! alloue un objet de type dynamique type(point3d)
allocate(type(point2d_coul) : :point)
```

Sommaire

Pointeur de procédure

Objectif Appeler une procédure pour effectuer un traitement dont la nature dépend de type dynamique d'un objet

Mise en oeuvre au moyen de **procédures attachées à un type dérivé**(*type-bound procedures*)

- La procédure récupère en entrée l'objet à l'origine de l'appel
- elles peuvent faire l'objet d'une surcharge lors d'extensions du type

Procédure attachée par nom

```
module mat
   type matrix(k,n,m)
      integer,KIND : : k ;integer,LEN : : n,m
      real(KIND=k),dimension(n,m) : : A
   contains
      procedure(),pointer : : calcul ⟹ mon_calcul
      GENERIC : : max ⟹ max_4,max_8
   end type matrix
contains
   subroutine mon_calcul(b,x,y)
      type(matrix),intent(inout) : : b
      real,intent(in) : : x,y
   end subroutine mon_calcul
   real(kind=4) function max_4(this)
      matrix(k=4,n=*,m=*),intent(in) : : this
   end function max_4
   real(kind=8) function max_8(this)
      matrix(k=8,n=*,m=*),intent(in) : : this
   end function max_8
end module mat
```

Procédure attachée par nom (suite)

```
program prog
   type(matrix(k=4,n=10,m=20)) : : obj1
   type(matrix(k=8,n=10,m=20)) : : obj2
   real : : x1,y1
   call obj1%mon_calcul(x1,y1)
   max4=obj1%max() ;    max8=obj2%max()
end program prog
```

L'attachement pourra se faire par :

- un pointeur de procédure,
- un opérateur
- avec l'attribut FINAL, la procédure s'exécutera lorsque l'objet cessera d'exister

```
type matrix
   ...
contains
   GENERIC : : OPERATEUR(+) ⟹ add4,add8
   GENERIC : : ASSIGNMENT(=) ⟹ affect4,affect8
   FINAL : : finalise4,finalise8
end type matrix
```

Sommaire

Héritage d'une procédure

Un type étendu d'un type extensible héritera à la fois des composantes mais aussi des procédures *type-bound*

```
module point
   type,public : : point2d
      real : : x,y
   contains
      procedure,pass : : affichage⟹ affichage_2d
   end type
   end module point
module point_coul
   use point
   type,public,extends(point2d) : : point2d_coul
      real,dimension(3) : : compos_rvb
   end type
end module point_coul
prog prog
   use pointcoul
   type(point2d_coul) : : pcoul
   call pcoul%affichage(« Voici mes ccordonnees »)
end program prog
```

Héritage d'une procédure

Il sera également possible d'étendre ou de surcharger les procédures de *type-bounds*

Example

```
module pointext
   use point
   type,public,extends(pont2d) : : point3d
      real : : z
   contains
      procedure,pass : : affichage ⟹ affichage_3d
   end type
contains
   subroutine affichage_3d(this,texte)
   ...
   end subroutine affichage_3d
end module pointext
```

Sommaire

Type abstrait

On peut définir des types abstraits qui sont des types qui servent de base à de futures extensions.

```
module numerique
  type,abstract : :mon_type_num
  contains
    procedure(func_oper),DEFERRED : :add
    procedure(func_oper),DEFERRED : :mult
    procedure(sub_oper),DEFERRED : :affect
    generic,public : :operator(+) ⟹ add
    generic,public : :operator(*) ⟹ mult
    generic,public : :assignment(=) ⟹ affect
  end type : :mon_type_num
  abstract interface
    function func_oper(a,b) result(r)
      class(mon_type_num),intent(in) : :a,b
      class(mon_type_num),allocatable : :a,b
    end function func_oper
    subroutine sub_oper(a,b)
      class(mon_type_num),intent(inout) : :a,b
      class(mon_type_num),intent(in) : :b
    end subroutine sub_oper
  end interface
end module numerique
```

Type personnel héritant d'un type abstrait

```
module entier
   use numerique
   type,extents(mon_type_num) : :mon_entier
      inter,private : :valeur
   contains
      procedure : :add ⟹ add_mon_entier
      procedure : :mult ⟹ mult_mon_entier
      procedure : :affect ⟹ affect_mon_entier
   end type mon_type
contains
! Definition des procedures :
end module entier
program prog
   use entier
   type(mon_entier) : : int1,int2,int3
   ...
   int1=int1*(int2+int3)
end program prog
```