NAGWare f95 と信頼できる、移植性の高いプログラミング

Malcolm Cohen

The Numerical Algorithms Group Ltd., Oxford

NAGWare f95を利用してどのようにエラーを検出するか。そして移植性が高く、信頼性の高いプログラムを書くか。また、最新のFortran標準からサポートされる機能と今後の計画について。

内容

1. Fortran標準:
    - •移植性の高いプログラミング
    - •近代的なプログラミング
    - •言語開発
2. NAGWare f95とFortran Builderビルダー:
    - •コンパイル時のエラー検出
    - •実行時のエラー検出
    - •Fortran 2003の機能
    - •今後の計画

移植性の高いプログラミング

移植性の高いプログラムは…

- ソースコードを変更せずに多くのシステムで実行できる。
- 更にこれらのシステムで正確な結果を出す。

移植性の高いプログラミングはライフタイムコストを削減する…

- 異なるマシン毎に異なる版を用意しなくて良い。
- メンテナンス量が減る。
- 正当であることの信頼性が高まる。

Fortran標準

- 全てのFortranコンパイラでサポートされなければならない機能
- 機能が全てのコンパイラで同じように動くよう正確な定義が行われている。

全てのコンパイラには拡張機能が搭載されているが、それを使うことは…

- 他のコンパイラで利用できるとは限らない
- 仮に利用できても同じ結果を返すとは限らない

歴史

1954 Fortranプロジェクトが IBMにて開始される

1966 Fortran 66, 初めてのプログラミング言語標準

1978 Fortran 77; 近代化が始まる

1991 Fortran 90 (メジャーな改編) 現在の近代的Fortranの基礎となる

1997 Fortran 95 (マイナーな改編)

2004 Fortran 2003 (メジャーな改編) オブジェクト指向、他

近代的なプログラミング

近代的なFortranは…

- より簡単に書ける
- より信頼性が高い (新しい機能はよりエラーを少なくする傾向にある)
- より強力
- より効率的
- 全ての主要なメーカーによりサポートされている。

近代的なFortranの主要機能 (1)

有用性

- 長い名前 (6→31文字).
- 自由形式のソース
- 近代的な制御構造
- 近代的なデータ構造
- モジュール

近代的なFortranの主要機能 (2)

機能

- 動的なメモリ確保; 特に割付け配列
- 配列式と代入
- 強力な組込み関数

近代的な制御構造

- 一般化DOループ
  (含む DO WHILE, EXIT, CYCLE);

- SELECT CASE 構文.

GOTOの必要性が減る、それにより…

- コードが読みやすくなる

- エラーが減る

近代的なデータ構造

- 構造型 (derived types) は構造体.
- 成分は配列もしくはスカラ
  組込み型 (例 Real) でもよいし構造型 (derived types)
  でもよい
- 成分はポインタでもよい

```
Type line
   Integer                 :: start(2), end(2)
   Real                    :: width
   Type(Colour)            :: colour
   Type(line),Pointer :: next_line
End Type
```

モジュール

- モジュールは名前付き定数、型定義、変数、手続きを含む事ができる。

- 定義を一回行い、他で使う。多重定義を防ぐことが可能。

- モジュール手続き呼び出しはコンパイル時にチェック可能。

USE文を使って利用する。

簡単なモジュール例

```
Module int64_module
  Integer,Parameter :: int64 = selected_int_kind(18)
Contains
  Integer(int64) Function gcd(a,b) ! Greatest Common Divisor
    Integer(int64),Intent(In) :: a,b
    ...
  End function
End Module

Program Example
  Use int64_module
  Integer(int64) x,y
  ...
  Print *,gcd(x,y)
End
```

割付け配列

- 動的に割付け
- ポインタは必要ない　–　速い
- 自動解放　–　安全
- STAT=オプションを利用したエラーハンドリング

割付け配列の例

```
Real,Allocatable :: workspace(:)
...
Allocate(workspace(n*4+10),Stat=istatus)
If (istatus==0) Then
   Call Solve_problem(...,workspace)
Else
   Print *,'Cannot allocate workspace, error code',istatus
End if
```

言語開発

- Fortran標準はしばしば改編
- 改編は常に後方互換性を確保
- ベンダーは標準に合わせて開発をし、リスクを減らす
- Fortran 2003の主な機能:
  - 割付け成分
  - IEEE 算術サポート
  - オブジェクト指向プログラミング
  - Cとの相互運用

Fortran 2003 設計目標

全体の目標

1. Fortran 95との互換性
2. 安全で効率的

オブジェクト指向の目標

- 簡単に記述できる
- 簡単に使える
- 簡単に実装できる
- 安全に使える: 実行時ではなくコンパイル時にエラーを検出

NAGWare f95: 概略

- 世界初の Fortran 90 コンパイラ
- Fortran 95 + 多くの Fortran 2003 機能
- Fortran Builder開発環境（日本国内のみ）
- コンパイル時に多くのエラーを検出
- 標準に準拠していないプログラムを広範囲にわたってチェック
- 卓絶した実行時エラー検出

実行時エラー検出

- 標準的な検出機能: 配列添字、ヌルポインタ
- 高度な検出機能: 手続き呼出し、ぶらさがり(dangling)ポインタ、未定義変数
- メモリ割付けのトレース

手続き呼び出しチェック - 1

extraな情報が手続き参照で渡される:

- 期待される結果の型と次元数
- 引数の数
- それぞれの引数に関して…
  - 式であるかどうか
    - クラス: 標準、ポインタ、割付け、形状引継ぎ、値、多相.
  - 手続きであるかどうか
  - 型、次元数
  - 引数の数

手続き呼び出しチェック　- 2

手続き呼び出しに問題があればプログラムはエラーメッセージとともに停止されます。

無効な手続き参照-
仮引数 I の実引数がREALになっていて、INTEGERではない
致命的なエラーでプログラムが停止
file2.f90の1行目、PV内
file1.f90の23行目、Sから呼び出されている
file1.f90の7行目、MAINから呼び出されている

-C=callsオプション付きでコンパイルされた手続きは、それなしでコンパイルされたものと混在する事が可能。その場合呼び出し側と呼び出される側の両方がこのオプション付きでコンパイルされた場合にのみチェックが行われる。

ぶらさがりポインタ

1. 保存されていない局所変数を参照するポインタで、手続きから復帰した後にポインタは未定義となる。

2. 割付けられたメモリを参照するポインタで、そのメモリが解放された場合。

これらはC/C++では一般的で、さまざまなエラーの原因。コンパイラの助け無しにこれらを見つけ出す事は難しい。

-C=danglingでコンパイルされた手続きとそうでないものを混在できる。その場合チェックされるルーチンのポインタ代入のみがチェックされる。

ぶらさがりポインタ例1

```
Program Test
    Real,Pointer :: x(:,:)
    Call make_dangle
    x(10,10) = 0
Contains
    Subroutine make_dangle
        Real,Target :: y(100,200)
        x => y
    End Subroutine
End
```

ぶらさがりポインタXへの参照が行われた
- 指示先が手続き（TEST:MAKE_DANGLE）より復帰している
プログラムは致命的エラーにより停止
dangle.f90の4行目TEST内

ぶらさがりポインタ例2

```
Program dangle2
    Real,Pointer :: x(:),y(:)
    Allocate(x(100))
    y => x
    Deallocate(x)
    y = 3
End
```

ぶらさがりポインタYへの参照が行われた
- 指示先がdangle2.f90の5行目で既に解放されている
プログラムは致命的エラーにより停止

dangle2.f90の6行目DANGLE2内

未定義変数

未定義変数は…

- 一度も値を与えられていない、もしくは
- 値を既に失ったもの

これを行うには全てのプログラムを-C=undefinedオプションでコンパイルする必要がある

浮動小数点変数のみの未定義を検出する場合には-nanオプションが利用可能。こちらの方が速く部分的な利用が可能、しかしエラーメッセージはそれほどわかりやすいものではない。

未定義変数例

```
Program undef1
    Real x(100)
    Read *,n
    Read *,x(1:n)
    Print *,product(x)
End
```

```
        未定義変数Xへの参照が行われた
        プログラムは致命的エラーにより停止
        undef1.f90の5行目UNDEF1内

        ***  算術例外:      -  停止します
        undef1.f90の5行目UNDEF1内
```

メモリ割付けのトレース

-mtrace オプションはメモリ割付けと解放をトレースします。
f95mcheckプログラムと共にメモリリークを検出します。

```
Program memory_leak
   Real,Pointer :: x(:,:)
   Allocate(x(10,20)) ! Leak
   x = 0
   Allocate(x(3,4))
   Deallocate(x)
   Allocate(x(5,6)) ! Leak
   Allocate(x(20,30))
   x = 3
   Deallocate(x)
End
```

メモリ割付のトレース

出力そのまま

[Allocated item 1 (size 1025) = Z’2E0008’]
[Allocated item 2 (size 1025) = Z’2E0418’]
[Allocated item 3 (size 1025) = Z’2E0828’]
[Allocated item 4 (size 800) at line 3 of memleak.f90 = Z’2F0008’]
[Allocated item 5 (size 48) at line 5 of memleak.f90 = Z’2F0330’]
[Deallocated item 5 (size 48, at Z’2F0330’) at line 6 of memleak.f90]
[Allocated item 6 (size 120) at line 7 of memleak.f90 = Z’2F0368’]
[Allocated item 7 (size 2400) at line 8 of memleak.f90 = Z’2F03E8’]
[Deallocated item 7 (size 2400, at Z’2F03E8’) at line 10 of memleak.f90]
[Deallocated item 2 (size 1025, at Z’2E0418’)]
[Deallocated item 3 (size 1025, at Z’2E0828’)]
[Deallocated item 1 (size 1025, at Z’2E0008’)]

f95mcheck の出力

7 allocations
***MEMORY LEAK:
LEAK: Allocation 4 (size 800) = Z’2F0008’ at line 3 of memleak.f90
LEAK: Allocation 6 (size 120) = Z’2F0368’ at line 7 of memleak.f90

Fortran 2003 の機能：サポート済みのもの

- 割付け成分
- IEEE 算術サポート
- オブジェクト指向

割付け成分

- 配列成分の動的サイズ
- ポインタ成分よりも効率的
- ポインタ成分よりも安全– 自動解放

```
Type matrix
    Real,Allocatable :: value(:,:)
End type
...
Type(matrix) x
...
Allocate(x%value(100,200))
...
```

IEEE 算術サポート

- IEEE 例外処理(例 overflow, underflow).
- IEEE 操作 (例 remainder, nextafter)
- IEEE 問合せ関数(例 IEEE_IS_NAN).
- 丸めモードの制御
- 停止モードの制御

```
Use ieee_arithmetic
...
z = x/y
If (ieee_is_nan(z)) Stop 'Result is Not a Number'
```

基本オブジェクト指向機能

既に利用可能:

- 型拡張(単一継承).
- 多相変数
- 型選択

基本オブジェクト指向まとめ

- “型拡張”は既にあるものを拡張して新しい型をつくる。新しい型は古い型の成分を継承する。

- 多相変数は動的に異なる型を持つことができる。常に仮引数、ポインタ、割付けのいずれかである。

- 型選択は多相変数の動的な型を検出し、拡張成分への直接アクセスが可能である。

NAGWare f95 今後の計画

- 英語版Fortran Builder
- Fortran 2003 機能を更にサポート
- パフォーマンス向上
- エラー検出を更に

Fortran 2003 の機能: 次期更新

- Cとの相互運用
- ストリームI/Oと他のI/O機能アップ
- 新しい組込み関数とモジュール
- オブジェクト指向を更に
- その他多くの追加

Linuxの次期更新を開始しました。

高度なオブジェクト指向機能

まもなく:

- クローニング
- 型結合手続き
- 一般手続きと算術子

これら全てが次期更新に含まれる。

まとめ

- Fortran標準は移植性の高いプログラミングを可能とする。
- Fortranの新しい機能は信頼性を高める。
- NAGWare f95は他の追従をゆるさないエラー検出機能が備わっている。
- NAGWare f95は現在、最新のFortran標準に準拠させつつある。

リソース

今回の資料:

http://www.nag-j.co.jp/~malcolm/May2006-J.pdf

近代的なFortranプログラミングに関する資料:

http://www.nag-j.co.jp/~malcolm/Modern-Fortran-J.pdf

Fortran 2003についての資料 (英語):

http://www.nag-j.co.jp/~malcolm/F2003-Illustrated.pdf

リファレンス本 (英語): “Fortran 95/2003 Explained”
by Metcalf, Reid and Cohen