f77の機種依存性とそのCプリプロセッサを用いた回避方法

大阪大学 産業科学研究所 ただの学生 西松 毅
nisimatu@cmp.sanken.osaka-u.ac.jp

1 はじめに

フォートラン・コンパイラにはその文法に, いくつか, コンパイラと機種に依存した部分 (以下, 「機種依存」と書く) がある. ユーザが自分のプログラムをいくつかの異なるコンピュータでコンパイル／実行する場合, ユーザはソースコード中の機種依存部分をいちいち書き換えてコンパイル／実行しなくてはならない. これは面倒であり, またミスやバグの原因にもなりやすい.

この文章は機種依存性を回避する一方法を説明する. Cプリプロセッサを使う. Cプリプロセッサを用いることにより, ユーザは, 機種依存情報を含んだ, ただ一つのソースコードを開発／保守／管理すればよく, 異なるコンピュータでのコンパイル毎にソースコードをいちいち書き換える必要がなくなる.

なお, 本文章中に示したプログラム例は匿名FTPサーバから自由に取ってくることができる. 匿名FTPサーバの使用例を以下に示す:

```
% ftp ftp.cmp.sanken.osaka-u.ac.jp
Connected to sally.cmp.sanken.osaka-u.ac.jp.
220 sally FTP server (Version wu-2.4(1) Tue Aug 8 15:50:43 CDT 1995) ready.
Name (ftp.cmp.sanken.osaka-u.ac.jp:kate): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: kate@winslet.osaka-u.ac.jp        あなたのメイルアドレスを入力する
230- K A T A Y A M A - Y O S H I D A  L a b 's  F T P  s e r v e r
230 Guest login ok, access restrictions apply.
ftp> cd /pub/papers/NISHIMATSU.Takeshi/center.osaka-u.kiji/
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> get f77.with.cpp.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for f77.with.cpp.tar.gz (4352 bytes).
226 Transfer complete.
4352 bytes received in 0.04 seconds (107.96 Kbytes/s)
ftp> quit
221 Goodbye.
%
```

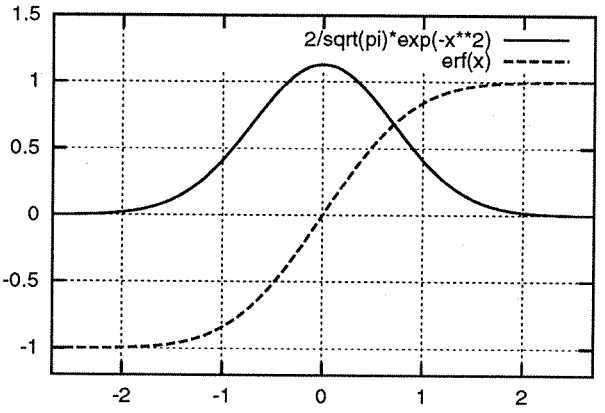

図 1: 誤差関数 $erf(x)$.

2 フォートラン・コンパイラの機種依存

フォートラン・コンパイラの機種依存の例を二, 三あげておく.

2.1 誤差関数 erf()

誤差関数 erf() は数学の関数の一つで, 参考までに書くと次のように定義される.

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{1}$$

スーパー・コンピュータのフォートラン・コンパイラの多くはこの erf() が sin() や cos() と同じように組込み関数 (intrinsic function) であり, また, erf() は総称名 (generic name) なので引数が倍精度の場合, 戻り値も倍精度になる. 一方, ワークステーションのフォートラン・コンパイラの多くは erf() が外部関数として定義され, 倍精度の戻り値がほしいときには同じく外部関数の derf() を使わなくてはならない (表 1).

実際のソースコードでは誤差関数 erf() または derf() は次のように使われる. スーパー・コンピュータの場合は:

```
      program erfsuper
c                          1998,02,20  NISHIMATSU Takeshi
      implicit none
      double precision dp
      intrinsic erf
      dp=0.2D0
      dp=erf(dp)
      write(6,100) dp
  100 format(f17.14)
      end
```

多くのワークステーションの場合には:

```
      program erfworkstation
c                          1998,02,20  NISHIMATSU Takeshi
      implicit none
      double precision dp
      double precision derf
      external derf
      dp=0.2D0
      dp=derf(dp)
      write(6,100) dp
  100 format(f17.14)
      end
```

数は少ないが, 誤差関数を用意していない機種もあるようだ. そのような場合は誤差関数をユーザが用意しなければならない.

表 1: 機種別の誤差関数一覧. 大抵の場合, スーパーコンピュータにも特定組込み関数として `derf()` があるので, 単精度のソースコードでは `erf()` を, 倍精度のソースコードでは `derf()` を使えば機種依存を気にする必要がないといえばない.

機種	分類	外部関数	
SiliconGraphics	Workstation	`erf()`, `derf()`, `qerf()`	
DEC Alpha/OSF1	Workstation	`erf()`, `derf()`	
HP	Workstation	ない	
機種	**分類**	**総称組込み関数**	**特定組込み関数**
Linux (g77)	PC UNIX	`erf()`	`derf()`
IBM AIX	Workstation	`erf()`	`derf()`, `qerf()`
NEC SX-4	super computer	`erf()`	`derf()`
Fujitsu VPP	super computer	`erf()`	`derf()`

2.2 CPU 時間の計測

プログラムの中で, プログラムを実行開始して以降に消費した CPU 時間を計測したいことがある. 例えば, バッチジョブの制限時間ギリギリまでプログラムに計算をさせたいときや, ベンチマーク・プログラムなどでは CPU 時間の情報をプログラム中で使いたい.

大抵のフォートラン・コンパイラはCPU時間の情報をプログラム中で手に入れるための外部関数やサブルーチンを用意しているが, その仕様はまちまちで機種に依存する (表2).

表 2: 機種別の f77 の CPU 時間計測外部サブルーチン一覧表. この文章中では一番前に書いてあるものを使った.

機種	分類	CPU 時間計測外部サブルーチン
SiliconGraphics	Workstation	`etime(), mclock()`
DEC Alpha/OSF1	Workstation	`etime()`
HP	Workstation	`etime()`
	ただし, コンパイル時に `+U77` オプションが必要	
Linux	PC UNIX	`Second()`
IBM AIX	Workstation	`mclock(), etime_()`
NEC SX-4	super computer	`etime(), clock()`
Fujitsu VPP	super computer	`clockm()`

3 Cプリプロセッサ

3.1 プログラム例

まず, 上で述べた機種依存性をCプリプロセッサを用いて回避した例を示そう. 3つのファイル, `testerf.F computer.h cptime.F` にわかれている. Cプリプロセッサを利用する場合, フォートランのソースコードの拡張子は `.f` ではなく, 大文字の `.F` を使う. その理由は後で述べる.

なお, `#` で始まる行がCプリプロセッサ指令と呼ばれるものである.

testerf.F:

```
      program testerf
c                          1998,02,10  NISHIMATSU Takeshi
#include "computer.h"

      implicit none
      double precision dp(200000)
      integer i,j
      integer itime0,itime1

#ifdef EXTERNALDERF
#define ERF derf
```

```
      double precision derf
      external derf
#endif

#ifdef INTRINSICERF
#define ERF erf
      intrinsic erf
#endif

      call cptime(itime0)
      write(6,*) itime0
      do j=1,100
        do i=1,200000
          dp(i)=ERF(dble(i+j)/400000.0D0)
        end do
      end do
      write(6,100) dp(200000)
  100 format(f17.14)
      call cptime(itime1)
      write(6,*) itime1
      end
```

computer.h:

```
#ifdef __sgi
#define EXTERNALDERF
#endif

#ifdef __osf__
#define EXTERNALDERF
#endif

#ifdef __hpux
#define EXTERNALDERF
#endif

#ifdef __linux__
#define INTRINSICERF
#endif
```

```
#ifdef _AIX
#define INTRINSICERF
#endif

#ifdef SX
#define INTRINSICERF
#endif

#ifdef __uxp__
#define INTRINSICERF
#endif
```

cptime.F:

```
c=============================================================
      SUBROUTINE CPTIME(it)
c     reports CPU time used in milli second
c=============================================================
#ifdef __sgi
c for SiliconGraphics
#define USE_ETIME
#endif

#ifdef __osf__
c for DEC Alpha/OSF1
#define USE_ETIME
#endif

#ifdef __hpux
c for HP
#define USE_ETIME
#endif

#ifdef SX
c for NEC SX-4
#define USE_ETIME
#endif

#ifdef USE_ETIME
      IMPLICIT none
```

```
      integer it
      real*4 y,tm(2),etime
      external etime
      y=etime(tm)
      it=tm(1)*1000.0
#endif

#ifdef __linux__
c for Linux (g77)
      IMPLICIT none
      integer it
      real SECONDS
      intrinsic Second
      call Second(SECONDS)
      it=SECONDS*1000.0
#endif

#ifdef _AIX
c for AIX
      IMPLICIT none
      integer it,mclock
      external mclock
      IT=mclock()*10
#endif

#ifdef __uxp__
c for Fujitsu VPP (icho.issp.u-tokyo.ac.jp)
      IMPLICIT none
      integer it
      external clockm
      call clockm(IT)
#endif

      END
```

これら3つのファイル testerf.F computer.h cptime.F をカレント・ディレクトリに置いて, SX-4 なら, f77␣-E␣p␣-o␣testerf␣testerf.F␣cptime.F とコンパイルすれば実行形式 testerf が出来上がる. Cプリプロセッサを利用する場合, フォートランのソースコードの拡張子は.f ではなく, 大文字の.F を使った方が便利である. いくつかのフォートラン・コンパイラは拡張子が.F のときにCプリプロセッサを自動的に起動するように

なっているからだ.

その他の機種上でのコンパイル方法については Makefile と 表3 とを参照してもらうことにして, 次の小節で C プリプロセッサの利用方法について説明しよう.

Makefile:

```
#   Makefile for f77.with.cpp                          -*- Makefile -*-
#                                    1998,02,16    NISHIMATSU Takeshi
###
testerf.sgi: cptime.F testerf.F computer.h
        f77 -n32 -O2 -o testerf.sgi testerf.F cptime.F

testerf.alpha: cptime.F testerf.F computer.h
        f77 -cpp -O2 -o testerf.alpha testerf.F cptime.F

testerf.hp: cptime.F testerf.F computer.h myderf.f
        f77 -O       -o testerf.hp testerf.F cptime.F myderf.f +U77

testerf.linux: cptime.F testerf.F computer.h
        g77 -O2      -o testerf.linux testerf.F cptime.F

testerf.aix: cptime.F testerf.F computer.h
        f77 -O2 -WF,-D_AIX -o testerf.aix testerf.F cptime.F

testerf.sx: cptime.F testerf.F computer.h
        f77 -E p     -o testerf.sx  testerf.F cptime.F

testerf.vpp: cptime.F testerf.F computer.h
        ln -sf testerf.F testerf.c
        fccpx -P testerf.c
        ln -sf testerf.i testerf_i.f
        ln -sf cptime.F cptime.c
        fccpx -P cptime.c
        ln -sf cptime.i cptime_i.f
        frtpx -o testerf.vpp testerf_i.f cptime_i.f
        rm -f testerf.c cptime.c testerf_i.f cptime_i.f

clean:
        rm -f hello.linux archive *.o *.aux *.log *.dvi a.out
        rm -f testerf.sgi testerf.alpha testerf.hp testerf.linux
        rm -f testerf.aix testerf.sx testerf.vpp testerf.i
        rm -f cptime.i testerf.compile.error std*.txt Q.03.*
```

この Makefile を使った Linux (g77) でのコンパイル／実行の様子は次のとおり．2 重の do loop 全体で 63270 − 9 = 63261 ミリ秒 = 63 秒 かかっていることがわかる．

```
% make testerf.linux
g77 -O2       -o testerf.linux testerf.F cptime.F
% ./testerf.linux
 9
 0.52028026452901
 63270
%
```

3.2 プリプロセッサ指令

C プリプロセッサはソースコード中のプリプロセッサ指令に従ってソースコードのある部分を空白行に置換したり，ソースコード中のある単語を別の単語に置き換える．プリプロセッサ指令の詳しい説明は C プリプロセッサや C コンパイラのマニュアル (例えば文献 [Bor96]) を参照していただければよいのだが，フォートラン・コンパイラと一緒に使うときのテクニックも含めて少しだけ書いておくことにする．

通常，プリプロセッサ指令はソースコードの先頭付近に置かれるが，文法上はプログラム中のどこにでも置くことができる．フォートラン・コンパイラと一緒に使うと便利なのは次のプリプロセッサ指令である．

`#define` `#ifdef ～ #endif` `#ifndef ～ #endif` `#include` `#undef`

プリプロセッサ指令は # で始まる．# の前後には空白 (スペース文字) を置かないで，第 1 桁目に # を置き，すぐ後から文字を書き始めたほうが無難である．いくつかの C プリプロセッサは，余計なスペース文字があると変な動きをするからだ．

なお，プリプロセッサ指令を含むコードに対して C プリプロセッサを使うことを，「C プリプロセッサで処理する」，「C プリプロセッサを通す」，「プリプロセスする」とかいう．

#define マクロ識別子 <トークン>

#define 指令はマクロを定義する．この指令以降の「マクロ識別子」は<トークン>に置き換えられる．<トークン>がない場合はこの指令以降の「マクロ識別子」が消去される．マクロ識別子は大文字で書くのが慣例である．次に例を示した:

```
#define ERF derf

      dp(i)=ERF(dble(i+j)/0.4D0)
```

#ifdef 識別子 ～ #endif
#ifndef 識別子 ～ #endif

#ifdef 条件指令は「識別子」が現在定義されているときに限り #endif までの内容を有効にできる．「識別子」が定義されていないと#ifdef から #endif までは消去される．

逆に #ifndef 条件指令は「識別子」が現在定義されていないときに限り #endif までの内容が有効になる．たとえば，いま，マクロ EXTERNALDERF は定義されていて，マクロ INTRINSICERF が定義されていない状態でCプリプロセッサが

```
#ifdef EXTERNALDERF
#define ERF derf
      double precision derf
      external derf
#endif

#ifdef INTRINSICERF
#define ERF erf
      intrinsic erf
#endif
```

を処理したなら，一つめのブロックのみが有効になる．

識別子として機種ごとの predefined macro を用いれば機種ごとに異なる内容のコードをコンパイルさせることができる (表3).

#include "ヘッダ・ファイル名"

#include 指令は指定されたファイルをコードのその部分に読み込む．そのファイル (ヘッダ・ファイルと呼ばれる) はプリプロセスされる．読み込みたいファイルは二つの " で囲んで指定する．

ただし，Cプリプロセッサ指令の #include とフォートランの命令の INCLUDE を混同してはいけない．INCLUDE 命令はCプリプロセッサを通さなくても必ず実行されるし，どんな場合にも INCLUDE されるファイルの内容はCプリプロセッサで処理されることはない．よって，(#include, #define, #ifdef などの) プリプロセッサ指令を含むヘッダ・ファイルは必ず #include によってコード中に読み込まれなくてはならない．

#undef 識別子

#undef 指令はマクロ定義を解除する．すでに定義されているマクロをほかのトークンに置き換えたい場合は，#undef してから#define しなおす．そうしないとCプリプロセッサより警告を受けることになる．以前に定義が存在する可能性がある場合は，以下のようにするとよい．

```
#ifndef BLOCK_SIZE
#define BLOCK_SIZE 512
#endif
```

プリプロセッサ指令には他に #(ヌル指令), #error, #if #elif, #else, #endif, #line, #undef がある. #if のネスト (多重の#if) はやらないほうが無難である. それをうまく解釈しないCプリプロセッサがあるからだ. ネストをするにしてもインデントをしないで, かつ, すべての対象機種上でよく確かめつつコンパイルすること.

この小節の説明から想像できると思うが, プリプロセッサ指令を多用するとソースコードが読みにくくなる. Cプリプロセッサの利用は最終手段であるとわきまえて, プリプロセッサ指令をなるべく使わないプログラミングが肝要である.

4 まとめ

以上, 見てきたように, Cプリプロセッサを使うことによって, フォートラン・コンパイラの機種依存を回避することができ, いくつかの機種につきただ一つの (プリプロセッサ指令が書き込まれた) プログラムを保守／管理すればよくなることがわかった.

機種依存の回避方法としてCプリプロセッサの他に autoconf というものがあることを紹介しておこう. autoconf はCプリプロセッサと相反するものではなく, Cプリプロセッサと組み合わせて使うことで威力を発揮する. 詳しくは文献 [Shi95] を参照のこと.

- フォートラン・コンパイラと一緒につかって便利なプリプロセッサ指令は: #define, #ifdef ～ #endif, #ifndef ～ #endif, #include, #undef.
- #ifdef のマクロ識別子として機種ごとの predefined macro を用いれば, 機種ごとに異なる内容のコードをコンパイルさせることができる.
- Cプリプロセッサを使うときには, フォートランのプログラムの拡張子は.F とする.
- プリプロセッサ指令を書くときは # の前後に空白 (スペース文字) を置かない方が無難. すなわち, # は第 1 桁目に書いた方が無難.
- プリプロセッサ指令の if 文のネストはしないほうが無難.
- 無闇やたらとCプリプロセッサを使わない. Cプリプロセッサは最終手段である.

参考文献

[Bor96] ボーランド株式会社: Borland C++ 5.0 プログラマーズガイド (市販の C コンパイラのオンラインマニュアル) (ボーランド株式会社, 東京都渋谷区, 1996).

[FSF97] Free Software Foundation: info of g77 (GNU Fortran のオンラインマニュアル) (Free Software Foundation, Inc., Boston, USA, 1997).

[Shi95] 島慶一: autoconf (1), UNIX MAGAZINE **10**, 141 (1995), 7 月号 連載 UNIX 知恵袋 9.

表 3: 機種別の C プリプロセッサの predefined macros, および foo.F というファイル名の FORTRAN のコードをまず C プリプロセッサ (cpp) に処理させて, その後にコンパイルして foo という実行形式を作る方法. たいていの C プリプロセッサには, 機種ごとに異なるいくつかの predefined macros がある. ユーザは predefined macros を, その名のとおり, ソースコードの中や C プリプロセッサの -D オプションで定義せずに使うことができる. predefined macros はどの機種が使われているのかを (ソースコードに) 知らせる働きをする.

機種	predefined macros	C プリプロセッサの起動
SiliconGraphics	__sgi	f77 -o foo foo.F
	mips, __mips	(sgi の f77 はデフォルトで cpp を起動する.)
DEC Alpha/OSF1	__osf__, __alpha	f77 -cpp -o foo foo.F
HP	__hpux	f77 -o foo foo.F
	(HP の f77 は拡張子が .F のときに cpp を自動的に起動する.)	
Linux	__linux__, linux	g77 -o foo foo.F
	__linux	
	(Linux の g77 は拡張子が .F のときに cpp を自動的に起動する.)	
IBM AIX	_AIX	f77 -o foo foo.F
(IBM の f77 は拡張子が .F のときに cpp を自動的に起動する. ただし, predefined symbol _AIX は foo.F の中で #include するヘッダファイル *.h についてのみ適用されるようだ. よって, "f77 -WF,-D_AIX -o foo foo.F" とコンパイルするのが無難である.)		
NEC SX-4	SX	f77 -E p -o foo foo.F
Fujitsu VPP	__uxp__	cp foo.F foo.c
		fccpx -P foo.c
		cp foo.i foo_i.f
		frtpx -o foo foo_i.f