Fortran 95 区間演算 プログラミングリファレンス

Forte Developer 7

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-4924-10
2002 年 6 月 , Revision A

原典： *Fortran 95 Interval Arithmetic Programming Reference*
Part No: 816-2462-10
Revision A

目次

表目次

はじめに

このマニュアルは Sun™ Forte Developer Fortran 95 コンパイラ（`f95`）における組み込みの区間データ型を解説しています。

対象読者

このマニュアルは、Fortran 言語、Solaris™ オペレーティング環境、UNIX コマンドの実用的な知識を持つプログラマを対象にしています。

内容の紹介

このマニュアルは次の章と付録から構成されています。

第 1 章では、`f95` における組み込みの区間サポートの目的を解説し、区間プログラマが `f95` における区間機能を学ぶ際に利用できるコード例を提供しています。また、`f95` を使用して区間の記述を始めるための基本的な情報も提供してます。

第 2 章では、`f95` に対する区間言語拡張を解説しています。

「用語集」には区間用語の定義が含まれています。

本書に含まれないことがら

本書は区間への入門書ではなく、また、f95 に含まれる区間の新機軸の派生内容を含むわけでもありません。予備的な区間情報を含むソースの一覧については、区間演算の README を参照してください。

関連する区間リファレンス

区間の文献は大規模であり、また増えつつあります。区間アプリケーションはいろいろな独立した分野に存在します。しかし、多くの区間書籍と雑誌論文は、新しい区間アルゴリズムを含むか、あるいは新しい区間アルゴリズムを開発している区間アナリスト用に記述されたものです。「区間の入門」というタイトルの書籍はまだ存在しません。

Sun Forte Developer f95コンパイラは区間をサポートする唯一のソースではありません。他のよく知られたソースに関心をお持ちの場合は、以下の書籍を参照することができます。

- 「IBM High Accuracy Arithmetic - Extended Scientific Computation(ACRITH-XSC)」、General Information、CG 33-6461-01 IBM Corp.（1990 年）。
- R.Klatte、U.Kulisch、M.Neaga、D.Ratz、Ch.Ullrich 著、「PASCAL-XSC Language Reference with Examples」、Springer（1991 年）。
- R.Klatte、U.Kulisch、A.Wiethoff、C.Lawo、M.Rauch 著、「C-XSC Class Library for Extended Scientific Computing」、Springer（1993 年）。
- R.Hammer、M.Hocks、U.Kulisch、D.Ratz 著、「Numerical Toolbox for Verified Computing 」、Basic Numerical Problems、Springer（1993 年）。

f95 に実装された新しい区間の基礎となる技術論文の一覧については、113 ページの「参考文献」を参照してください。これらのリファレンスのオンライン版の場所については、区間演算の README を参照してください。

オンラインリソース

Web サイトにアクセスしたり、電子メールのメーリングリストに登録することで、区間に関する情報をさらに入手できます。オンラインリソースについては、区間演算の README を参照してください。

Web サイト

区間演算の README に記載されている URL から、オンラインで詳細な出版目録と区間の FAQ を入手することができます。

電子メール

区間演算テーマを議論したり、区間演算の用い方に関する質問ができるよう、メーリングリストが用意されています。このメーリングリストにはどなたでも質問を送ることができます。このメーリングリストの申し込み方法の説明内容については、区間演算の README を参照してください。

疑わしい区間エラーを報告される場合は、次の宛先に電子メールをお送りください。

```
sun-dp-comments@Sun.Com
```

主題の行か電子メールのメッセージには、次のテキストを含めてください。

```
FORTEDEV "7.0 mm/dd/yy" INTERVAL
```

ここでは、*mm/dd/yy* は、月、日、年度を表します。

コード例

本書のすべてのコード例は次のディレクトリに含まれています。

```
/opt/SUNWspro/prod/examples/intervalmath/docExamples
```

各ファイルの名前は、ce*n*-*m*.f95 のようになっています。N はサンプルの発生する章を表し、*m* はサンプルの番号を表します。次のディレクトリには追加的な区間例が格納されています。

```
/opt/SUNWspro/prod/examples/intervalmath/general
```

書体と記号について

次の表と記述は、このマニュアルで使用している書体と記号について説明しています。

書体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% You have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	machine_name% **su** Password:
AaBbCc123 または ゴシック	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	rm *filename* と入力します。 rm ファイル名 と入力します。
『 』	参照する書名を示します。	『SPARCstorage Array ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	machinename% grep '^#define \ XV_VERSION_STRING'

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	*machine_name*%
UNIX の Bourne シェルと Korn シェル	*machine_name*$
スーパーユーザー (シェルの種類を問わない)	#

Forte Developer の開発ツールとマニュアルページへのアクセス

Forte Developer の製品コンポーネントとマニュアルページは、標準の /usr/bin/ と /usr/share/man の各ディレクトリにインストールされません。 Forte Developer のコンパイラとツールにアクセスするには、 PATH 環境変数に Forte Developer コンポーネントディレクトリを必要とします。Forte Developer マニュアルページにアクセスするには、 MANPATH 環境変数に Forte Developer マニュアルページディレクトリが必要です。

PATH 変数についての詳細は、csh(1)、sh(1)、および ksh(1) のマニュアルページを参照してください。 MANPATH 変数についての詳細は、man(1) のマニュアルページを参照してください。このリリースにアクセスするために PATH および MANPATH 変数を設定する方法の詳細は、『インストールガイド』を参照するか、システム管理者にお問い合わせください。

注 - この節に記載されている情報は Forte Developer 製品が /opt ディレクトリにインストールされていることを想定しています。Forte Developer 製品が /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

Forte Developer コンパイラとツールへのアクセス方法

PATH 環境変数を変更して Forte Developer コンパイラとツールにアクセスできるようにする必要があるかどうか判断するには以下を実行します。

▼ PATH 環境変数を設定する必要があるかどうか判断するには

1. 次のように入力して、 PATH 変数の現在値を表示します。

```
% echo $PATH
```

2. 出力内容から /opt/SUNWspro/bin/ を含むパスの文字列を検索します。

   パスがある場合は、PATH 変数は Forte Developer 開発ツールにアクセスできるように設定されています。パスがない場合は、次の指示に従って、 PATH 環境変数を設定してください。

▼ PATH 環境変数を設定して Forte Developer のコンパイラとツールにアクセスする

1. C シェルを使用している場合は、ホーム .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホーム .profile ファイルを編集します。
2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/bin

Forte Developer マニュアルページへのアクセス方法

Forte Developer マニュアルページにアクセスするために MANPATH 変数を変更する必要があるかどうかを判断するには以下を実行します。

▼ MANPATH 環境変数を設定する必要があるかどうか判断するには

1. 次のように入力して、 dbx マニュアルページを表示します。

```
% man dbx
```

2. 出力された場合、内容を確認します。

   dbx(1) マニュアルページが見つからないか、表示されたマニュアルページがインストールされたソフトウェアの現バージョンのものと異なる場合は、次の節の指示に従って MANPATH 環境変数を設定してください。

▼ MANPATH 変数を設定して Forte Developer マニュアルページにアクセスする

1. C シェルを使用している場合は、ホーム .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホーム.profile ファイルを編集します。

2. 次のパスを MANPATH 環境変数に追加します。

   /opt/SUNWspro/man

Forte Developer マニュアルへのアクセス

Forte Developer の製品マニュアルには、以下からアクセスできます。

- 製品マニュアルは、ご使用のローカルシステムまたはネットワークの製品にインストールされているマニュアルの索引から入手できます。
  /opt/SUNWspro/docs/ja/index.html

製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

- マニュアルは、docs.sun.comの Web サイトで入手できます。 次に示すマニュアルは、インストールされている製品のマニュアルの索引から入手できます (docs.sun.com Web サイトでは入手できません)。
  - 『Standard C++ Library Class Reference』
  - 『標準 C++ ライブラリ・ユーザーズガイド』
  - 『Tools.h++クラスライブラリ・リファレンスマニュアル』
  - 『Tools.h++ ユーザーズガイド』

インターネットの docs.sun.com Web サイト (http://docs.sun.com) から、サンのマニュアルを読んだり、印刷したり、購入することができます。マニュアルが見つからない場合はローカルシステムまたはネットワークの製品とともにインストールされているマニュアルの索引を参照してください。

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

アクセスできる製品マニュアル

Forte Developer 7 製品マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセス可能なマニュアルは以下の表に示す場所から参照することができます。製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

マニュアルの種類	アクセス可能な形式と格納場所
マニュアル (サードパーティ製マニュアルは除く)	形式: HTML 場所: `http://docs.sun.com`
サードパーティ製マニュアル: 『Standard C++ Library Class Reference』 『標準 C++ ライブラリ・ユーザーズガイド』 『Tools.h++ クラスライブラリ・リファレンスマニュアル』 『Tools.h++ ユーザーズガイド』	形式: HTML インストール製品について 場所: `file:/opt/SUNWspro/docs/ja/index.html` のマニュアル索引
Readme およびマニュアルページ	形式: HTML インストール製品について 場所: `file:/opt/SUNWspro/docs/ja/index.html` のマニュアル索引
リリースノート	製品 CD 内の HTML ファイル

ご意見の送付先

米国 Sun Microsystems, Inc. では、マニュアルの向上に力を注いでおり、ユーザーのご意見やご提案をお待ちしております。ご意見などがありましたら、次のアドレスまで電子メールをお送りください。

docfeedback@sun.com

第1章

f95 で区間演算を使用するには

f95 INTERVAL 型と区間演算サポート

区間演算は、数の区間を使って計算を行うためのシステムです。区間演算は常に可能なすべての結果値の組み合わせを含む区間を生成するため、区間アルゴリズムはきわめて困難な計算を実現するように開発されました。区間アプリケーションのより詳細な情報については、README の区間演算を参照してください。

区間演算の誕生以来、狭幅区間の結果を生成する区間アルゴリズムが開発され、区間言語をサポートするための構文と意味論が設計されてきました。しかし、市場で利用でき、サポートを受けられる区間コンパイラについてはあまり改善されていません。1 つの例 (M77 Minnesota FORTRAN 1977 標準バージョン第 1 版) を除いて、区間システムは、これまで、プリプロセッサ、C++ クラス、あるいは、Fortran 90 のモジュールを基につくられています。f95 で区間データのサポートをコンパイラに組み込んだ目的は次のとおりです。

- 信頼性
- 高速性
- 使いやすさ

Sun Forte Developer の Fortran 95 での区間サポートは、大幅に拡張されています。

f95 区間サポートの目的：実装品質

f95 における組み込みの区間サポートの目的は、プログラム開発者に、次のような機能を提供することで、商業ベースの区間解決ライブラリとアプリケーションの開発を活性化することです。

- 高品質の区間コード
- 狭幅区間演算結果の生成
- 迅速に実行できる区間コード
- 区間言語サポートとコンパイラ機能を含む使いやすいソフトウェア開発環境

サポートと機能は、実装品質の構成要素です。実装品質のすべての機能が実装されているわけではありません。このマニュアルでは、全体を通して、実装されていないさまざまな実装品質の説明があります。お客様からの改善提案も受け付けております。

高品質の区間コード

正しい区間サポートコンパイラは、任意の区間式の評価の結果として、可能なすべての結果値の組み合わせを含む区間を生成しなければなりません。可能なすべての結果値は、当該式の包含セット (cset) と呼ばれます。式の cset を包含するための要件として、区間演算の包含の制約が存在します。この包含の制約を満たせないことを、包含エラーと言います。警告や文書を伴わずに発生するエラーは、任意の区間計算システムで致命的なエラーとなります。このただ 1 つの制約を満たすことで、区間はきわめて優れた計算品質を提供します。

包含の制約が満たされれば、実装品質は、実行時間と区間幅の軸を持つ二次元の平面上の 1 つの点の位置で決まります。両軸上は小さい値ほど良いことになります。実行時間と区間幅間の関係はアプリケーションに依存しますが、実行時間と区間幅は、区間システム品質の明確な尺度となります。区間幅と実行時間は常に実行時に入手して利用できますから、区間アルゴリズムと実装システムの両方の正確性の測定は速度の測定と同様、難しいことではありません。

Sun Forte Developer のパフォーマンスプロファイリングツールは区間プログラムの調整に使用できます。しかし、f95 には、アルゴリズムが不要に区間幅を増大する可能性のある場所を特定するための区間固有のツールはありません。24 ページの「コード開発ツール」で解説しているように、区間の dbx と大域的なプログラム検査 (GPC) のサポートが提供されています。区間固有のコード開発とデバッグツールの追加により、優れた実装品質が得られます。

狭幅区間の生成

すべての言語やコンパイラの実装品質の基準となる項目、たとえば速度や使いやすさは区間用のツールにも適用されます。

更に、有効な区間実装システムは、常に包含の制約を満たす一方で計算区間の幅を最小化するという実装品質の基準もあります。

区間の幅が狭いほど、それは「鋭い」と表現されます。所与の浮動小数点精度について、区間幅が狭いほど、生成される区間は鋭くなります。

f95 コンパイラにより生成される区間の幅については、次のことがあてはまります。

- 個別区間は定数の鋭い近似値です。
- 個別区間の算術演算子は鋭い結果を生成します。
- 組み込みの算術関数は、通常、鋭い結果を生成します。

迅速に実行できる区間コード

コンパイラの最適化とハードウェアの命令サポートの提供により、区間演算が対応する REAL 浮動小数点よりも著しく低速になることはありません。f95 では、組み込みの区間演算子と算術関数の速度について、次のことがあてはまります。

- 算術演算は適度に高速です。
- デフォルトの区間算術関数の速度は、一般的に、対応するDOUBLE PRECISION の 2 倍未満です。KIND=4 であれば、組み込みの区間算術関数の速度は調整されません (対応するKIND=8 関数とは異なる)。このリリースでは、KIND=16 の数学関数は提供されません。しかし、その他の KIND=16 区間関数はサポートされます。
- 次の組み込み区間配列関数では、パフォーマンスが最適化されています。
  - SUM
  - PRODUCT
  - DOT_PRODUCT
  - MATMUL

使いやすい開発環境

Fortran の組み込みの区間データ型は区間コードの開発、テスト、実行に役立ちます。区間コードを分かりやすくする (読み書きしやすくする) ために、区間構文と意味論が Fortran に追加されています。最終的には、これらの機能がユーザーに受け入れられれば、将来の Fortran 規格に取り入れられることになります。

区間を Fortran への組み込みデータ型として導入することで、Fortran 言語の適用可能なすべての構文と意味論がすぐに利用できるようになります。Sun Forte Developer の Fortran 95 は、以下のような Fortran の区間固有の機能の拡張を含んでいます。

- `INTERVAL` データ型
- 区間算術演算と組み込みの数学関数は閉じた数学システムです。(つまり、ゼロによる除算やその他のゼロや無限大などの不定形式を含む、任意の可能な演算子オペランドの組み合わせに対して有効な結果を生成します。)
- 区間関係演算子の3つの種類
  - 「Certainly」(断定的な関係)
  - 「Possibly」(可能性のある関係)
  - 「Set」(集合の関係)
- `.IX.` (積集合) と `.IH.` (区間包) のような、組み込みの区間固有の演算子
- `INF`、`SUP`、`WID` などの区間固有の関数
- 区間の単数の入出力
- 式の文脈依存の区間定数
- 区間固有の混合モード (種別型パラメータ値 (KTPV) および/または型) の式の処理

これらの機能とその他の組み込み機能に関する例とより詳細な情報については、コード例 1-11 からコード例 1-14 までの例と113 ページの「組み込み区間関数」を参照してください。

第 2 章ではこれらの関数とその他の区間機能を解説しています。

`f95` 区間コードの記述

この節の例は、初めて区間に接するプログラマが基本を理解し、有用な区間コードをすぐに記述できるような内容となっています。これらの例を基にして変更したり実際に使ってみることをおすすめします。

本書のすべてのコード例は、次のディレクトリに含まれています。

```
/opt/SUNWspro/examples/intervalmath/docExamples
```

各例の名前は「ce*n-m*.f95」となっていますが、*n* は例の現れる章を、*m*は例の番号を表しています。また、次のディレクトリには、追加の区間例が含まれています。

```
/opt/SUNWspro/examples/intervalmath/general
```

コマンド行オプション

以下の f95 コマンド行マクロは、組み込みの INTERVAL データ型を認識し、区間式の処理を制御します。

- 最大幅要求区間式処理のコンパイラサポートを呼び出す。
  -xia または -xia=widestneed
- 厳密区間式処理のコンパイラサポートを呼び出す。
  -xia=strict

組み込みの INTERVAL データ型がコンパイラにより認識されるためには、f95 のコマンド行で、-xia または -xinterval を入力する必要があります。

区間と関連するすべてのコマンド行オプションについては、44 ページの「区間のコマンド行オプション」で解説しています。最大幅要求式と厳密式の処理については、40 ページの「区間演算式」で解説しています。

コード例 1-1では、f95 の区間サポートを使用した最も簡単なコマンド行呼び出しを示しています。

Hello Interval World

別途明示的な解説がない限り、すべてのコード例は、-xia コマンド行オプションを使ってコンパイルできます。-xia か -xinterval コマンド行オプションは f95 に対する区間拡張を使用するために必要です。

コード例 1-1 は、区間用の「Hello world」です。

コード例 1-1　Hello Interval World

```
math% cat ce1-1.f95
PRINT *, "[2, 3] + [4, 5] = ", [2, 3] + [4, 5]    !  1 行目
END
math% f95 -xia ce1-1.f95
math% a.out
 [2, 3] + [4, 5] =  [6.0,8.0]
```

コード例 1-1 では、並びによる出力を用いて、区間 [2, 3] と [4, 5] の名前付き合計を表示しています。

区間の宣言と初期化

区間宣言文は、REAL、INTEGER、COMPLEX の宣言がそれぞれのデータ型に対して行うのと同じ機能を区間データ項目に対して実現します。区間のデフォルト種別型パラメータ値 (KTPV) は、INTEGER のデフォルトの KTPV の 2 倍です。これにより、任意のデフォルト INTEGER を縮退したデフォルトの区間を使って正確に表現できるようになります。より詳細な情報については、12 ページの「デフォルトの種別型パラメータ値 (KTPV)」を参照してください。

コード例 1-2 は、区間変数と初期化を使用して、コード例 1-1 と同じことを実現しています。

コード例 1-2　区間変数を使用した Hello Interval World

```
math% cat ce1-2.f95
INTERVAL :: X = [2, 3], Y = [4, 5]   ! Line 1
PRINT *, "[2, 3] + [4, 5] = ", X+Y   ! Line 2
END
math% f95 -xia ce1-2.f95
math% a.out
 [2, 3] + [4, 5] =  [6.0,8.0]
```

1 行目では、変数 X と Y をデフォルト型の区間変数と宣言し、それぞれを [2, 3] と [4, 5] に初期化しています。第 2 行目では、並びによる出力を使って、名前付き区間 X と Y の合計を表示しています。

区間の入出力

区間の読み取りと書き出し用には完全なサポートが提供されます。区間と COMPLEX データ項目の読み取りと書き出しは類似しています。区間は囲み記号として丸括弧でなく角括弧を用います。f95 では、入力変換処理は入力小数値を含む 1 つの鋭い区間を構築します。その値がマシンで表現できる場合は、内部的なマシン近似値は縮退しています。その値がマシンで表現できない場合は、1-ulp (仮数部の最終位置の単位。「ulp」といいます。) の幅を持つ区間が構築されます。

区間データ項目を読み取ったり、印刷する最も簡単な方法は、並びによる入出力を使うことです。

コード例 1-3 は、ユーザーが並びによる READ と PRINT 文を使った区間演算と単数区間入出力を理解するのを支援する簡単なツールです。区間の書式化入出力については、93 ページの「入力と出力」で記述しているように、完全なサポートが提供されています。

注 - 区間の包含の制約は、入力中と出力中の両方で、丸めを使うことを必要としています。単数入力の直後で単数が出力されると、小数桁精度が失われることになります。実際に、入力値がマシン表現できない場合、入力区間幅は最大で 1-ulp だけ増えます。8 ページの「単数の入出力」と 11 ページのコード例 1-6 を参照してください。

コード例 1-3 区間の入出力

```
math% cat ce1-3.f95
   INTERVAL ::  X, Y
   INTEGER  :: IOS = 0
   PRINT *, ” Press Control/D to terminate!”
   WRITE(*, 1, ADVANCE = 'NO')
   READ(*, *, IOSTAT = IOS) X, Y
   DO WHILE (IOS >= 0)
        PRINT *, " For X =", X, ", and Y =", Y
        PRINT *, "X+Y =", X+Y
        PRINT *, "X-Y =", X-Y
        PRINT *, "X*Y =", X*Y
        PRINT *, "X/Y =", X/Y
        PRINT *, "X**Y =", X**Y
        WRITE(*, 1, ADVANCE = 'NO')
        READ(*, *, IOSTAT=IOS) X, Y
   END DO
1  FORMAT(" X, Y = ? ")
   END
```

```
%math f95 -xia ce1-3.f95
%math a.out
 Press Control/D to terminate!
 X, Y = ? [1,2] [3,4]
 For X = [1.0,2.0] , and Y = [3.0,4.0]
 X+Y = [4.0,6.0]
 X-Y = [-3.0,-1.0]
 X*Y = [3.0,8.0]
 X/Y = [0.25,0.6666666666666675]
 X**Y = [1.0,16.0]
 X, Y = ? [1,2] -inf
 For X = [1.0,2.0] , and Y = [-Inf,-1.7976931348623157E+308]
 X+Y = [-Inf,-1.7976931348623155E+308]
 X-Y = [1.7976931348623157E+308,Inf]
 X*Y = [-Inf,-1.7976931348623157E+308]
 X/Y = [-1.1125369292536012E-308,0.0E+0]
 X**Y = [0.0E+0,Inf]
 X, Y = ? ^d
```

注 - f95 は、空の区間をサポートします。空の区間は、"[empty]"と入力できます。94 ページの「外部表現」の記述と 96 ページのコード例 2-37 で示しているように、無限区間の終了点もサポートしています。

単数の入出力

区間出力を読み取る際には、区間の最大下限と最小上限を比較して一致する桁数を数えることは手間のかかる作業の一つです。たとえば、コード例 1-4 とコード例 1-5 は、いろいろな幅を持つランダムな区間データを生成します。

注 - コード例 1-4 とコード例 1-5 では、プログラムの出力だけを示しています。この出力を生成するコードは、/opt/SUNWspro/examples/intervalmath/docExamples ディレクトリの例に含まれています。

コード例 1-4　[inf,sup] 区間出力

```
%math f95 -xia ce1-4.f95
%math a.out
Press Control/D to terminate!
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,4,0
[ 0.2017321E-029, 0.2017343E-029]
```

```
[ 0.2176913E-022, 0.2179092E-022]
[-0.3602303E-006,-0.3602302E-006]
[-0.3816341E+038,-0.3816302E+038]
[-0.1011276E-039,-0.1011261E-039]
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,8,0
[ -0.3945547546440221E+035, -0.3945543600894656E+035]
[  0.5054960140922359E-270,  0.5054960140927415E-270]
[ -0.2461623589326215E-043, -0.2461623343163864E-043]
[ -0.2128913523672577E+204, -0.2128913523672576E+204]
[ -0.3765492464030608E-072, -0.3765492464030606E-072]
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,16,0
[  0.19905035325231862025624507137405 8E+055,
0.19905035325232061075974266455744 7E+055]
[ -0.27738643198941791522368251643749 3E+203,
-0.27738643198941791519594387411882 2E+203]
[  0.13258528859826547231685682138050 3E+410,
0.13258528859826547231685682270635 6E+410]
[  0.95571443664743788107172789168280 4E+351,
0.95571443664743788107172789168376 0E+351]
[ -0.22421189776882421039830699440173 2E+196,
-0.22421189776882421039830699417751 9E+196]
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: ^d
```

コード例 1-4 の出力の読みやすさをコード例 1-5 のものと比較してみてください。

コード例 1-5　単数出力

```
%math a.out
 Press Control/D to terminate!
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,4,1
     0.20173  E-029
     0.218    E-022
    -0.3602303E-006
    -0.38163  E+038
    -0.10112  E-039
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,8,1
     -0.394554          E+035
      0.505496014092    E-270
     -0.2461623         E-043
     -0.2128913523672577E+204
     -0.3765492464030607E-072
```

```
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: 5,16,1
         0.19905035325232                    E+055
        -0.2773864319894179152               E+203
         0.13258528859826547231685 6822      E+410
         0.95571443664743788107172 7891683   E+351
        -0.22421189776882421039830 6994      E+196
Enter number of intervals, KTPV (4,8,16) and 1 for single-number
output: ^d
```

区間データの読み取りと対話形式の入力は冗長になるので単数区間形式が導入されています。単数変換とは、角括弧に含まれない任意の数が、最後の表示桁に 1 単位を加算、減算して下限と上限が構築される 1 つの区間として翻訳されることです。

そこで、

2.345 = [2.344, 2.346],

2.34500 = [2.34499, 2.34501],

と

23 = [22, 24].

記号を用いると次のように表されます。

$[2.34499, 2.34501] = 2.34500 + [-1, +1]_{uld}$

ここでは、$[-1, +1]_{uld}$ は、先行する数の最後の桁に区間 [-1, +1] が追加されることを意味します。添字 uld は、「最終桁の単位」(unit in the last digit) のニーモニックです。

注 - `f95` のコード中の `INTERVAL` 文字定数を表現するためには使用しません。

縮退した区間を表すには、単一の数を角括弧で囲むことができます。たとえば次のように表されます。

[2.345] = [2.345, 2.345] = 2.345000000000.....

この変換は、単数入出力と Fortran コードの縮退したリテラルな区間定数の両方に用います。このため、0.1 をマシンが表現できない場合でも、1 つの正確な 10 進小数を入力するには `[0.1]` と入力します 。

プログラムに入力中の `[0.1,0.1]` および `[0.1]` は、点 0.1 を表しますが、単数入出力用法での `0.1` は、次のような区間を表します。

$0.1 + [-1, +1]_{uld} = [0, 0.2]$.

注 - uld と ulp は異なります。uld は最終表示桁に対する 1 単位の加減算を行うために、暗黙に、単数入出力形式を用いて区間を構築することを意味します。ulpは内部的なマシン数に与えることのできる最小の可能なインクリメントまたはデクリメントです。

単数表示形式では、後続のゼロが意味を持ちます。より詳細な情報については、93 ページの「入力と出力」を参照してください。

区間は、伝統的な [*inf, sup*] 表示形式を使って、常に入力、表示することができます。また、角括弧で囲まれた単数は点を表します。たとえば、`[0.1]` の入力は、1/10 に翻訳されます。包含を保証するために、丸めによる方法を使って、1/10 の数を含むことがわかっている区間の近似値が構築されます。

コード例 1-6 内部データ変換を伴う文字入力

```
math% cat ce1-6.f95
INTERVAL :: X
   INTEGER  :: IOS = 0
   CHARACTER*30 BUFFER
   PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, '(A12)', IOSTAT=IOS) BUFFER
   DO WHILE (IOS >= 0)
     PRINT *, ' Your input was: ', BUFFER
     READ(BUFFER, '(Y12.16)') X
     PRINT *, "Resulting stored interval is:", X
     PRINT '(A, Y12.2)', ' Single number interval output  is:', X
     WRITE(*, 1, ADVANCE='NO')
     READ(*, '(A12)', IOSTAT=IOS) BUFFER
   END DO
1  FORMAT(" X = ? ")
   END
math% f95 -xia ce1-6.f95
math% a.out
 Press Control/D to terminate!
 X = ? 1.37
 入力値 : 1.37
 Resulting stored interval is:
[1.3599999999999998,1.3800000000000002]
 Single number interval output  is:  1.3
```

```
 X = ? 1.444
 Your input was: 1.444
 Resulting stored interval is:
[1.4429999999999998,1.4450000000000001]
 Single number interval output  is:  1.44
 X = ? ^d
```

コード例 1-6 注記：

- 角括弧内部の単数は縮退した区間を表します。
- 単数入力でマシンが表現できない数が読み取られると、10 進数から 2 進数への変換 (基数変換) が行われ、包含の制約によりその数の区間幅が 1-ulp (小数部分の最後の位置の単位) だけ強制的に増やされます。この結果を単数出力を用いて表示すると小数桁の精度が失われた形で現れます。そうではなく、単数区間入力を正確に反映するためには、11 ページのコード例 1-6 が示しているように、文字入力は内部 READ 文のデータ変換と併用されなければなりません。

区間の文と式

f95 コンパイラには以下のような区間固有の文、式、拡張が含まれます。

- 13 ページの表 1-1 で示している、INTERVAL データ型、関連命令、文がサポートされています。
- 実引数を受け入れるすべての組み込み関数は、対応する区間バージョンを持ちます。
- 区間固有の関係演算子や「集合」論関数を含む、多くの組み込みの区間固有の関数と演算子が追加されています。組み込み区間関数と区間演算子の完全なリストについては、117 ページの「組み込み関数」と 113 ページの「数学関数」を参照してください。

デフォルトの種別型パラメータ値 (KTPV)

f95 では、デフォルトの INTEGER KTPV は、KIND(0) = 4 です。縮退したデフォルトの区間を使って任意のデフォルト INTEGER を表すためには、デフォルトの区間 KTPV である KIND([0]) が、2*KIND(0) = 8 である必要があります。これは次の理由によります。

- 区間は、CDC や Cray のマシンで実行されてきたような数値演算に特化した計算の実行によく用いられます。

- 単一の算術式の評価では、累積した丸めエラー、依存、取り消しのために、区間の幅は必ず増えます。精度を上げることにより、累積した丸めエラーと取り消しの影響を軽減させることができます。依存の効果を軽減または排除するには、その他の手段が必要となります。

表 1-1　区間固有の文と式

文または式	解説
INTERVAL	デフォルトの INTERVAL 型宣言
INTERVAL(4)	KIND=4 の INTERVAL
INTERVAL(8)	KIND=8 の INTERVAL
INTERVAL(16)	KIND=16 の INTERVAL
[*a*,*b*] (1)	INTERVAL 文字定数: [*a*,*b*]
[*a*] (2)	[*a*,*a*]
INTERVAL A PARAMETER A=[*c*,*d*]	名前付き定数: A
V = *expr* (3)	値の代入
FORMAT(E, EN, ES, F, G, VE, VF, VG, VEN, VES, Y) (4)	E, EN, ES, F, G, VE, VF, VG, VEN, VES, Y 編集記述子

1. 文字 *a* と *b* は、0.1、0.2 のような小数文字定数可変部分が入ります。

2. 角括弧内部の単一の小数定数は縮退した区間定数を表します。入出力でも同様です。

3. *expr* が、INTERVAL 型の項目を含むかどうかにかかわらず、任意の Fortran 算術式を表すものとします。V = *expr* の代入文は式 *expr* を評価しその結果の値を V に代入します。-xia=strict コマンド行オプションのもとでは、混合モードの区間式は使用できません。-xia、または、-xia=widestneed オプションのもとでは、混合モード式は最大幅要求式処理を使って正しく評価されます。最大幅要求のもとでは、式評価の前に、すべての整数と浮動小数点データ項目は、V を含む式の中で使われている最大の KTPV を持つ区間を含むよう昇格されます。詳細については、14 ページの「値の代入 V = expr」を参照してください。

4. 区間の入出力サポートは、柔軟性、可読性、開発の容易性を提供するよう設計されています。最も重要な新しい編集記述子は Y ですが、この記述子は単数区間形式を使って区間を読み、表示するために用いられます。区間を処理できるすべての記述子の完全な説明については、83 ページの「入力と出力」を参照してください。

値の代入 V = *expr*

区間の代入文は、可変部分 *expr* で表される区間式の値を、区間変数、配列要素、配列、配列セクション、または構造体構成要素 V に代入します。構文は次のとおりです。

```
V = expr
```

V は、INTERVAL 型を持たなければならず、また、*expr* は任意の非 COMPLEX の数式を表します。最大幅要求式処理のもとでは、式 *expr* は、区間式である必要はありません。厳密式処理のもとでは、*expr* は、V と同じ KTPV を使った区間式でなければなりません。

混合型式の評価

混合型の区間式を有効に使用することは、明確な (理解しやすい) 算術式の記述に役立つので、重要で使いやすい機能です。

混合型の区間式は、区間コードの記述と読みを REAL コードと同じように使いやすくするためにサポートされています。区間の包含の制約は、最大幅要求または厳密のいずれかの式処理を用いた混合モード式で満たされます。

最大幅要求式処理と厳密式処理

区間定数の幅が、40 ページの「区間演算式」で解説しているように、式の文脈により動的に定義されている場合には、狭幅区間結果の計算が役立ちます。コード例 1-7 で示しているように、KTPV 混合式の中では、動的に増加する区間変数の KTPV が区間式結果の幅を減少させることもあります。

コード例 1-7　最大幅要求を用いた混合精度

```
math% cat ce1-7.f95
INTERVAL(4) :: X = [1, 2], Y = [3, 4]
INTERVAL    :: Z1, Z2

! 最大幅要求コード
Z1 = X*Y                                                ! 3 行目


! 同等の厳密コード
Z2 = INTERVAL(X, KIND=8)*INTERVAL(Y, KIND=8)     ! 4 行目
IF (Z1 .SEQ. Z2)  PRINT *, 'Check.'
END
math% f95 -xia ce1-7.f95
math% a.out
 Check.
```

3 行目では、$KTPV_{max}$ = `KIND(Z) = 8` となります。この値は、積の計算前に、`X` と `Y` の KTPV を 8 へと昇格させ、結果を `Z1` に格納するのに使用されます。

これらの手順は、4 行目の同等の厳密コードの中で明示的に示されています。

文を走査して最大 KTPV を決定し必要な昇格を実行するプロセスは、最大幅要求式処理とよばれます。40 ページの「区間演算式」を参照してください。

区間構成子の構文と意味論については、64 ページの「組み込み区間演算子の拡張」を参照してください。

(型と KTPV の) 混合モード式

KTPV とデータ型と共に、最大幅要求の原則が用いられると、(型と KTPV の) 混合モード区間式が安全かつ予測的に評価できるようになります。たとえば、16 ページのコード例 1-8 では、`X` と `Y` は区間変数なので、3 行目の `Y1` の式は 1 つの区間式です。

コード例 1-8　最大幅要求を用いた型混合

```
math% cat ce1-8.f95
INTERVAL(16) :: X = [0.1, 0.3]
INTERVAL(4)  :: Y1, Y2

! 最大幅要求コード
 Y1 = X + 0.1                                    ! 3 行目

! 同等の厳密コード
 Y2 = INTERVAL(X + [0.1_16], KIND=4)             ! 4 行目
 IF (Y1 == Y2) PRINT *, "Check."
END

math% f95 -xia ce1-8.f95
math% a.out
 Check.
```

包含を保証するためには、定数 0.1 に対する実近似値の代わりに内部的な区間が使用されなければなりません。しかし、KIND(X) = 16 なので、$KTPV_{max}$ = 16 となります。このため、X の更新には、正確な値 1/10 を含む鋭い KTPV = 16 区間である区間定数 [0.1_16] が使用されます。最後に、この結果が区間を含む KTPV = 4 へと変換され、Y に代入されます。4 行目は同等の厳密コードを含んでいます。厳密式処理のもとでは、混合型または混合 KTPV の式はどちらも許可されません。

最大幅要求式処理の論理的な手順は次のようになります。

1. 左側を含む文全体を走査して、任意の区間データ項目を求めます。

   INTERVAL 型の定数、変数、または、組み込み関数があると、式の型は INTERVAL になります。

2. 区間式を走査して、各 INTERVAL、REAL、INTEGER、定数、または変数の KTPV にもとづき、**$KTPV_{max}$** を求めます。

   注 - 整数は 2 倍の KTPV を持つ区間に変換されるので、すべての整数値は正確に表現できます。

   ---

3. すべての変数と定数を **$KTPV_{max}$** を持つ区間に昇格させます。

4. 式を評価します。

5. 左側の KTPV に一致させる必要があれば、結果をより低い KTPV へと変換します。

6. 生成された値を左側に代入します。

これらの手順は、混合モードの区間式処理が包含の制約を満たし、効果的に合理的な狭い区間結果を生成することを保証します。

最大幅要求式処理を使った混合モードの区間式の評価は、-xia コマンド行フラグを使ったデフォルトでサポートされます。-xia=strict を使うと、区間への任意の自動的な型変換と任意の区間変数の自動的な KTPV の増加を行いません。厳密モードでは、すべての区間型と精度の変換が明示的にコード化されなければなりません。

演算式

区間データ項目を含む算術式の記述は、簡単で直接的です。区間文字定数と組み込みの区間固有の関数を除けば、区間式は REAL 算術式のように見えます。特に、最大幅要求式処理では、REAL と INTEGER 変数と文字定数は、コード例 1-9 で示しているように、区間式の中のどこででも自由に使用することができます。

コード例 1-9　簡単な区間式の例

```
math% cat ce1-9.f95
INTEGER  :: N = 3
REAL     :: A = 5.0
INTERVAL :: X

X = 0.1*A/N                        ! 5 行目
PRINT *, "0.1*A/N = ", X
END

math% f95 -xia ce1-9.f95
math% a.out
 0.1*A/N =  [0.1666666666666662,0.1666666666666672]
```

5 行目で代入が行われる変数 X は 1 つの区間ですから、式 0.1 * A/N の評価の前に、次の手順が実行されます。

1. 文字定数 0.1 は、縮退した区間 [0.1] を含むデフォルトの区間変数へと変換されます。

規格に沿った区間システムの実装での要件ではありませんが、Sun Forte Developer の Fortran 95 は鋭いデータ変換を実行します。たとえば、区間の近似値 [0.1] は 1-ulp の幅となります。

2. REAL 変数 A は、縮退した区間 [5] に変換されます。

3. INTEGER 変数 N は、縮退した区間 [3] に変換されます。

式 [0.1]×[5]/[3] は、区間演算を使用して評価されます。上記手順は最大幅要求式処理の一部であり、これは、混合モードの区間式の評価時の包含の制約を満たすために必要です。14 ページの「混合型式の評価」を参照してください。

区間代入文は、代入先の変数が区間変数、配列要素、配列、配列セクション、または構造体構成要素でなければならないという 1 つの要件を満たさなければなりません。最大幅要求処理に関する詳細については、51 ページの 2.3.1 「混合モードの区間式」を参照してください。

Sun Forte Developer の Fortran 95 で実装された区間システムは閉鎖型なので、任意の区間式が有効な区間結果の生成に失敗すれば、コンパイラエラーとして表示されます。疑わしいエラーの表示方法については、24 ページの「コード開発ツール」を、既知のエラーの一覧については、30 ページの「既知の内部的なエラー」をそれぞれ参照してください。

注 - 算術的に同等の cset の区間式が常に同じ幅を持つ区間を生成するわけではありません。また、単一の区間式を評価するだけでは鋭い結果を計算できないことがよくあります。一般的に、区間の結果としての幅は、区間引数の値と式の形式に依存します。

区間の順位関係

区間の順位付けは点の順位付けよりも複雑です。2 が 3 よりも小さいことをテストするのはあいまいなことではありません。区間を使用すると、区間 [2,3] は区間 [4,5] よりも小さいことになりますが、区間 [2,3] と [3,4] の場合はどのように表現すべきなのでしょう？

区間関係演算子としては、以下の 3 つの異なるクラスが実装されています。

- 「certainly」(断定的な関係)
- 「possibly」(可能性のある関係)
- 「set」(集合の関係)

断定的な関係が *true* となるためには、オペランド区間のすべての要素がその関係を満たさなければなりません。オペランド区間の任意の要素によりその関係が満たされると、可能性のある関係は *true* となります。集合の関係は、区間を集合として取り扱います。区間関係演算子のこれらの 3 つのクラスは、両方のオペランド区間が共に縮退すると、点に関する通常の関係演算子に収束します。

3 つの演算子クラスを区別するために、通常の 2 文字で表す Fortran のニーモニックには、C、P、または、S 文字の接頭辞を付けます。`f95` では、集合演算子 `.SEQ.` と `.SNE.` は、点型のデフォルト (`.EQ.` または==、と、`.NE.` または/=) がサポートされる唯一の演算子となっています。その他のすべてのケースでは、次の例のように、関係演算子クラスが明示的に識別されなければなりません。

- `.CLT.`「断定的な関係」で小なり
- `.PLT.`「可能性のある関係」で小なり
- `.SLT.`「集合の関係」で小なり

すべての区間演算子の構文と意味論については、48 ページの「組み込み演算子」を参照してください。

次のプログラムは、「集合の関係」での等式テストを示しています。

コード例 1-10 集合等式テスト

```
math% cat ce1-10.f95
INTERVAL :: X = [2, 3], Y = [4, 5]         ! 1 行目
IF(X+Y .SEQ. [6, 8]) PRINT *, "Check."     ! 2 行目
END
math% f95 -xia ce1-10.f95
math% a.out
 Check.
```

2 行目では、集合の関係の等式テストを使って、`X+Y` が区間 [6, 8] に等しいことを検証しています。

次の記述は、2 行目のものと同等です。

```
IF(X+Y == [6, 8]) PRINT *, "Check." ! 2 行目
```

区間固有の関係演算子の結果を試してみる場合は、20 ページのコード例 1-11 と 21 ページのコード例 1-12 を使ってください。

コード例 1-11 区間関係演算子

```
math% cat ce1-11.f95
   INTERVAL ::  X, Y
   INTEGER  :: IOS = 0
   PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, *, IOSTAT=IOS) X, Y
   DO WHILE (IOS >= 0)
       PRINT *, " For X =", X, ", and Y =", Y
       PRINT *, 'X .CEQ. Y, X .PEQ. Y, X .SEQ. Y =', &
                 X .CEQ. Y, X .PEQ. Y, X .SEQ. Y
       PRINT *, 'X .CNE. Y, X .PNE. Y, X .SNE. Y =', &
                 X .CNE. Y, X .PNE. Y, X .SNE. Y
       PRINT *, 'X .CLE. Y, X .PLE. Y, X .SLE. Y =', &
                 X .CLE. Y, X .PLE. Y, X .SLE. Y
       PRINT *, 'X .CLT. Y, X .PLT. Y, X .SLT. Y =', &
                 X .CLT. Y, X .PLT. Y, X .SLT. Y
       PRINT *, 'X .CGE. Y, X .PGE. Y, X .SGE. Y =', &
                 X .CGE. Y, X .PGE. Y, X .SGE. Y
       PRINT *, 'X .CGT. Y, X .PGT. Y, X .SGT. Y =', &
                 X .CGT. Y, X .PGT. Y, X .SGT. Y
       WRITE(*, 1, ADVANCE='NO')
       READ(*, *, IOSTAT=IOS) X, Y
   END DO
1  FORMAT( " X, Y = ")
   END
math% f95 -xia ce1-11.f95
math% a.out
 Press Control/D to terminate!
 X, Y = [2] [3]
 For X = [2.0,2.0] , and Y = [3.0,3.0]
 X .CEQ. Y, X .PEQ. Y, X .SEQ. Y = F F F
 X .CNE. Y, X .PNE. Y, X .SNE. Y = T T T
 X .CLE. Y, X .PLE. Y, X .SLE. Y = T T T
 X .CLT. Y, X .PLT. Y, X .SLT. Y = T T T
 X .CGE. Y, X .PGE. Y, X .SGE. Y = F F F
 X .CGT. Y, X .PGT. Y, X .SGT. Y = F F F
 X, Y = 2 3
 For X = [1.0,3.0] , and Y = [2.0,4.0]
 X .CEQ. Y, X .PEQ. Y, X .SEQ. Y = F T F
 X .CNE. Y, X .PNE. Y, X .SNE. Y = F T T
 X .CLE. Y, X .PLE. Y, X .SLE. Y = F T T
 X .CLT. Y, X .PLT. Y, X .SLT. Y = F T T
 X .CGE. Y, X .PGE. Y, X .SGE. Y = F T F
 X .CGT. Y, X .PGT. Y, X .SGT. Y = F T F
 X, Y = ^d
```

コード例 1-12 は、表 1-2 に掲載した区間固有の演算子用法を例示しています。

表 1-2　区間固有の演算子

演算子	名前	算術記号
.IH.	区間包	$\underline{\cup}$
.IX.	積集合	$\cap$
.DJ.	素	$A \cap B = \varnothing$
.IN.	要素	$\in$
.INT.	内部	58 ページの「内部：(X .INT. Y)」を参照してください。
.PSB.	真部分集合	$\subset$
.PSP.	真超集合	$\supset$
.SB.	部分集合	$\subseteq$
.SP.	超集合	$\supseteq$

コード例 1-12 集合演算子

```
math% cat ce1-12.f95
   INTERVAL ::  X, Y
   INTEGER  :: IOS = 0
   REAL(8)  :: R = 1.5
 PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, *, IOSTAT=IOS) X, Y
   DO WHILE (IOS >= 0)
       PRINT *, " For X =", X, ", and Y =", Y
       PRINT *, 'X .IH.  Y =', X .IH. Y
       PRINT *, 'X .IX.  Y =', X .IX. Y
       PRINT *, 'X .DJ.  Y =', X .DJ. Y
       PRINT *, 'R .IN.  Y =', R .IN. Y
       PRINT *, 'X .INT. Y =', X .INT. Y
       PRINT *, 'X .PSB. Y =', X .PSB. Y
       PRINT *, 'X .PSP. Y =', X .PSP. Y
       PRINT *, 'X .SP.  Y =', X .SP. Y
       PRINT *, 'X .SB.  Y =', X .SB. Y

       WRITE(*, 1, ADVANCE='NO')
       READ(*, *, IOSTAT=IOS) X, Y
   END DO
1  FORMAT(" X, Y = ? ")
   END
math% f95 -xia ce1-12.f95
math% a.out
```

```
 Press Control/D to terminate!
 X, Y = ? [1] [2]
 For X = [1.0,1.0] , and Y = [2.0,2.0]
 X .IH.  Y = [1.0,2.0]
 X .IX.  Y = [EMPTY]
 X .DJ.  Y = T
 R .IN.  Y = F
 X .INT. Y = F
 X .PSB. Y = F
 X .PSP. Y = F
 X .SP.  Y = F
 X .SB.  Y = F
 X, Y = ? [1,2] [1,3]
 For X = [1.0,2.0] , and Y = [1.0,3.0]
 X .IH.  Y = [1.0,3.0]
 X .IX.  Y = [1.0,2.0]
 X .DJ.  Y = F
 R .IN.  Y = T
 X .INT. Y = F
 X .PSB. Y = T
 X .PSP. Y = F
 X .SP.  Y = F
 X .SB.  Y = T
 X, Y = ? ^d
```

組み込みの区間固有の関数

組み込みの区間固有の関数としてさまざまなものが提供されています。117 ページの「組み込み関数」を参照してください。コード例 1-13 を使うと、組み込みの区間固有の関数の動作を調べることができます。

コード例 1-13 組み込みの区間固有の関数

```
math% cat ce1-13.f95
   INTERVAL ::  X, Y
   PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, *, IOSTAT=IOS) X
   DO WHILE (IOS >= 0)
       PRINT *, " For X =", X
       PRINT *, 'MID(X)= ', MID(X)
       PRINT *, 'MIG(X)= ', MIG(X)
       PRINT *, 'MAG(X)= ', MAG(X)
       PRINT *, 'WID(X)= ', WID(X)
       PRINT *, 'NDIGITS(X)= ', NDIGITS(X)
       WRITE(*, 1, ADVANCE='NO')
       READ(*, *, IOSTAT=IOS) X
   END DO
```

```
1  FORMAT(" X = ?")
   END
math% f95 -xia ce1-13.f95
math% a.out
 Press Control/D to terminate!
 X = ?[1.23456,1.234567890]
 For X = [1.2345599999999998,1.2345678900000002]
 MID(X)=  1.234563945
 MIG(X)=  1.2345599999999998
 MAG(X)=  1.2345678900000001
 WID(X)=  7.890000000232433E-6
 NDIGITS(X)=  6
 X = ?[1,10]
 For X = [1.0,10.0]
 MID(X)=  5.5
 MIG(X)=  1.0
 MAG(X)=  10.0
 WID(X)=  9.0
 NDIGITS(X)=  1
 X = ? ^d
```

標準組み込み関数の区間バージョン

Fortran の REAL 引数を受け入れるすべての組み込み関数は、区間バージョンを持っています。117 ページの「組み込み関数」を参照してください。コード例 1-14 を使用すると、いくつかの組み込み関数の動作を調べることができます。

コード例 1-14 標準組み込み関数の区間バージョン

```
math% cat ce1-14.f95
   INTERVAL :: X, Y
   INTEGER  :: IOS = 0
   PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, *, IOSTAT=IOS) X
   DO WHILE (ios >= 0)
      PRINT *, "For X =", X
      PRINT *, 'ABS(X) = ', ABS(X)
      PRINT *, 'LOG(X) = ', LOG(X)
      PRINT *, 'SQRT(X)= ', SQRT(X)
      PRINT *, 'SIN(X) = ', SIN(X)
      PRINT *, 'ACOS(X)= ', ACOS(X)
      WRITE(*, 1, ADVANCE='NO')
      READ(*, *, IOSTAT=IOS) X
   END DO
1  FORMAT(" X = ?")
   END
math% f95 -xia ce1-14.f95
```

```
math% a.out
 Press Control/D to terminate!
 X = ?[1.1,1.2]
For X = [1.0999999999999998,1.2000000000000002]
 ABS(X) =  [1.0999999999999998,1.2000000000000002]
 LOG(X) =  [0.095310179804324726,0.18232155679395479]
 SQRT(X)=  [1.0488088481701514,1.0954451150103324]
 SIN(X) =  [0.89120736006143519,0.93203908596722652]
 ACOS(X)=  [EMPTY]
 X = ?[-0.5,0.5]
For X = [-0.5,0.5]
 ABS(X) =  [0.0E+0,0.5]
 LOG(X) =  [-Inf,-0.69314718055994528]
 SQRT(X)=  [0.0E+0,0.70710678118654758]
 SIN(X) =  [-0.47942553860420307,0.47942553860420307]
 ACOS(X)=  [1.0471975511965976,2.0943951023931958]
 X = ? ^d
```

コード開発ツール

区間コード開発に関する情報はオンラインで利用できます。区間の Web サイト一覧とその他のオンラインのリソースについては、区間演算の README を参照してください。

疑わしい区間エラーを報告される場合は、次の宛先に電子メールをお送りください。

```
sun-dp-comments@Sun.com
```

主題の行か本文中に次のテキストを含めてください。

```
FORTEDEV "6.0 mm/dd/yy" Interval
```

ここでは、*mm/dd/yy* は、月、日、年度を表します。

デバッグのサポート

Sun Forte Developer では、区間データ型は dbx により次に示す範囲でサポートされます。

- 各区間変数の値は print コマンドを使って表示できます。
- すべての区間変数の値は dump コマンドを使用して表示できます。
- assign コマンドを使って区間変数に新しい値を代入することができます。

- 区間データ配列を表示する手段は提供していません。
- データ型に固有の機能でなければ、すべての一般的な機能が使用できます。

dbx 機能に関する追加的な詳細については、「dbx コマンドによるデバッグ」を参照してください。

大域的なプログラム検査

Sun Forte Developer の Fortran 95 における大域的なプログラム検査 (GPC) は、1 つの区間固有のエラーとして、ユーザー供給ルーチン呼び出しでの `INTERVAL` 型の不一致を検出します。コード例 1-15 は、`INTERVAL` 型の不一致の例です。

コード例 1-15 INTERVAL 型の不一致

```
math% cat ce1-15.f95
INTERVAL X
X = [-1.0,+2.9]
PRINT *,X
CALL SUB(X)
END
SUBROUTINE SUB(Y)
INTEGER Y(2)
PRINT *,Y
END
math% f95 -xia ce1-15.f95 -Xlist

--- ce1-15.lst を参照 ---

ce1-15.f95                    （木） 5月  2 16:05:09 2002
FILE  "ce1-15.f95"
     1  ! Copyright 2001, 2002 Sun Microsystems, Inc. All Rights
        Reserved
     2
     3  INTERVAL X
     4  X = [-1.0,+2.9]
     5  PRINT *,X
     6  CALL SUB(X)
                ^
**** エラー  #325:  引数 "x" は 変数 ですが、仮引数は array です。
                 "ce1-15.f95" の 8 行目を参照してください。
     4  CALL SUB(X)
                ^
**** ERR  #560:  変数 "x" は、 integer として  行目の
              main/sub/in 中で参照されていますが、interval main
              によって 4 行目で設定されています。
     5  END
     6  SUBROUTINE SUB(Y)
     7  INTEGER Y(2)
     8  PRINT *,Y
     9  END
```

Sun Fortran ライブラリで提供される区間機能

次のライブラリには区間の組み込みルーチンがあります。

表 1-3　区間ライブラリ

ライブラリ	名前	必要なオプション
区間組み込み配列関数	`libifai`	なし
区間組み込みライブラリ	`libsunimath`	なし

コードの移植とバイナリファイル

使用上対処の必要な、制限のある古い区間 Fortran コードが存在します。言語の構文と意味論が標準化されるまでは、異なる区間コンパイラ供給者のサポートが異なることは避けらません。お客様からの最も馴染みのある区間構文と意味論を考慮したフィードバックは標準化過程に役立ちます。区間演算の `README` に一覧表示された電子メールのエイリアスに、コメントを送付することもできます。

バイナリファイルでの区間の表現は、コンパイラがより狭い区間システムをサポートするかどうかにより異なります。

並列処理

このリリースでは、`-autopar` コンパイラオプションは区間算術演算を含むループ処理に効果がありません。これらのループ処理は自動的には並列処理されません。明示的な並列化指令を付したループの並列化には、`-explicitpar` コンパイラオプションを使用しなければなりません。

エラーの検出

次のコード例では、区間固有のエラーメッセージを一覧表示しています。各コード例にはエラーメッセージとエラーを生成したコードが含まれています。

コード例 1-16 無効な終了点

```
math% cat ce1-16.f95
INTERVAL :: I = [2., 1.]
END

math% f95 -xia ce1-16.f95

INTERVAL :: I = [2., 1.]
                        ^
"ce1-14.f95", Line = 1, Column = 24: エラー：区間定数の左側の終了点は右
側の終了点以下でなければなりません。

f95: 2 ソース行
f95: 1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

コード例 1-17 区間と非区間の同値

```
math% cat ce1-17.f95
INTERVAL :: I
REAL     :: R
EQUIVALENCE (I, R)
END

math% f95 -xia ce1-17.f95

EQUIVALENCE (I, R)
             ^
"ce1-15.f95", Line = 3, Column = 14: エラー : INTERVAL 実体 "I" と
REAL 実体 "R" の結合指定は許可されません。

f95: 4 ソース行
f95: 1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

コード例 1-18 異なる KTPV を持つ区間オブジェクトの同値

```
math% cat ce1-18.f95
INTERVAL(4) :: I1
INTERVAL(8) :: I2
EQUIVALENCE (I1, I2)
END

math% f95 -xia ce1-18.f95

EQUIVALENCE (I1, I2)
             ^
"ce1-16.f95", Line = 3, Column = 14: エラー : 異なる種別型パラメータを
持つ INTERVAL 実体 "I1" と "I2" の結合指定はできません。

f95: 4 ソース行
f95: 1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

コード例 1-19 厳密モードでの区間変数への REAL 式の代入

```
math% cat ce1-19.f95
INTERVAL :: X
REAL     :: R
X = R
END
math% f95 -xia=strict ce1-19.f95

X = R
  ^
"ce1-17.f95", Line = 3, Column = 3: エラー : REAL 式を INTERVAL 変数
へ代入することは許可されません。

f95: 4 ソース行
f95: 1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

コード例 1-20 厳密モードでの区間変数への区間式の代入

```
math% cat ce1-20.f95
INTERVAL     :: X
INTERVAL(16) :: y
X = Y
END
math% f95 -xia=strict ce1-20.f95

X = Y
  ^
"ce1-18.f95", Line = 3, Column = 3: エラー : INTERVAL 変数に対する
INTERVAL 式の代入はそれらが異なる種別型パラメータを持つ場合、許可されません。

f95: 4 ソース行
f95: 1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

既知の内部的なエラー

内部的なエラーが発生する可能性がある場合は常にコンパイル時の警告が出力される必要があります。最大幅要求式の処理において、このような内部エラーは発生しません。-xia=strict モードにおいて、区間 ** (整数式) 演算で整数式がオーバーフローする場合、包含の制約に違反する可能性があります。

整数オーバーフロー

数の不正確性は、通常は INTEGER 式よりもむしろ REAL 式と関係します。しかしある意味では、INTEGER 式は REAL 式よりも危険です。REAL 式がオーバーフローすると例外が提起されて IEEE 無限大が生成されます。例外はオーバーフローが発生したという警告です。無限大は、浮動小数点計算で潜在的な問題があることをユーザーに知らせます。オーバーフロー発生時にそれをトラップすることもできます。

整数式がオーバーフローすると、それらは場合によっては、反対符号の値へと暗黙のうちに変換されてしまいます。さらに、逆の操作を実行して、すべての整数操作に関し同値テストを行うことが、整数式のオーバーフローの発生時期を検出する唯一の方法です。整数定数式の場合はコンパイル中に評価され、オーバーフローが検出されて警告メッセージが出力されるため安全です。

-xia=widestneed 式の処理で、** 演算子の第 2 オペランドがオーバーフローする整数式である場合、常に正しい区間結果が返されます。ただし、-xia=strict の処理も同様であるとは必ずしも言いきれません。最大幅要求式を処理することなく特定の式に先行して区間に整数を昇格することはできないからです。** 演算子の第 2 オペランドが INTERVAL 型の変換ルーチンである場合にも、同じことが言えます。

コード例 1-21 は、最大幅要求式の処理が ** 演算子の第 2 オペランドの整数式の内側にあるすべての組み込み INTEGER 演算と関数にまで拡張されていることを示しています。これは、-xia=strict モードでは、真ではありません。

コード例 1-21 --xia=strict モードでの INTEGER オーバーフローの内部的なエラー

```
math% cat ce1-21.f95
   INTERVAL :: X = [1.5], Y = [1.5], Z = [1.5]
   INTEGER  :: I = HUGE(0)

   PRINT *, "BEFORE POW"
   PRINT *, "X = ", X
   PRINT *, "Y = ", Y
   PRINT *, "Z = ", Z
   PRINT *, "I = ", I

   X = X**(I+1)                  ! I+1 - integer overflow
   Y = Y*(Y**I)
   Z = Z**(INTERVAL(I)+INTERVAL(1))

   PRINT *, "I+1=",I,"+",1,"=",I+1

   PRINT *, "RESULTS:"
   PRINT *, "X = ", X
   PRINT *, "Y = ", Y
   PRINT *, "Z = ", Z
END

math% f95 -xia ce1-21.f95
math% a.out
 BEFORE POW
 X =  [1.5,1.5]
 Y =  [1.5,1.5]
 Z =  [1.5,1.5]
 I =  2147483647
 I+1= 2147483647 + 1 = -2147483648
 RESULTS:
 X =  [1.7976931348623157E+308,Inf]
```

```
 Y =  [1.7976931348623157E+308,Inf]
 Z =  [1.7976931348623157E+308,Inf]
math% f95 -xia=strict ce1-21.f95
math% a.out
 BEFORE POW
 X =  [1.5,1.5]
 Y =  [1.5,1.5]
 Z =  [1.5,1.5]
 I =  2147483647
 I+1= 2147483647 + 1 = -2147483648
 RESULTS:
 X =  [0.0E+0,4.9406564584124655E-324]
 Y =  [1.7976931348623157E+308,Inf]
 Z =  [1.7976931348623157E+308,Inf]
```

このコード例は、メッセージを表示しない -xia=strict モードの内部的なエラーと、-xia=widestneed モードの正しい区間結果を示しています。べき乗演算子に関する情報については、53 ページの「べき乗演算子 X**N と X**Y」を参照してください。

第2章

f95 区間リファレンス

この章は、Sun Forte Developer の Fortran 95 に実装された組み込みの INTERVAL 型の構文と意味論のリファレンスになっています。各節は任意の順序で読むことができます。

別途明示的な記述がない限り、INTERVAL データ型は他の組み込みの数値型と同じ特性を持っています。この章では REAL 型と INTERVAL 型の相違点に焦点をあてます。

コード例の中には完全でないプログラムも存在します。これらの例は、-xia コマンド行オプションを使ってコンパイルすることが暗黙に想定されています。

Fortran 拡張

INTERVAL データ型は Fortran に対する非標準の拡張です。しかし、実現可能な箇所では、実装構文と意味論は Fortran 形式に準拠しています。

文字セットの表記

Fortran 文字セットには、区間文字定数を区切るため、左右の角括弧「[...]」が追加されています。

このマニュアルの全体を通じて、別途明示的な記述がない限り、INTEGER、REAL、INTERVAL の各定数は文字定数を意味します。定数式と名前付き定数 (PARAMETERS) は常に明示的に区別されます。

表 2-1 はコードと算術に用いられる文字セットの表記を示しています。

表 2-1 書体の変換

文字セット	表記
Fortran コード	`INTERVAL :: X=[0.1,0.2]`
プログラムとコマンドへの入力	`Enter X: ? [2.3,2.4]`
コード内部の定数用の可変部分	[*a*,*b*]
スカラー演算	$x(a + b) = xa + xb$
区間演算	$X(A + B) \subseteq XA + XB$

注 - フォントの使用にはよく注意してください。異なるフォントは区間の正確な外部数学値と区間のマシン表現可能な内部的な近似値を表します。

区間定数

f95 では、区間定数は、単一の整数または角括弧で囲んだ実 10 進数 [3.5] のいずれか、あるいは、一対の整数またはコンマで区切り角括弧で囲んだ実 10 進数 [3.5 E-10, 3.6 E-10] のいずれかです。縮退した区間がマシンで表現できない場合、正確な算術値は指定された丸めを使って、内部的な制約を満たすことが明らかな内部的なマシン表現可能な区間に丸められます。

デフォルトINTEGER、デフォルト REAL または REAL(8) の両終了点を持つ 区間定数はデフォルト型区間を持ちます。

終了点の型がデフォルト INTEGER、デフォルト REAL または REAL(8) である場合、その終了点は内部的に REAL(8) 型の値へと変換されます。

終了点の型が INTEGER(8) である場合、その終了点は内部的に REAL(16) 型の値へと変換されます。

終了点の型がINTEGER(4)である場合、その終了点は内部的に REAL(8)型の値へと変換されます。

終了点の型が INTEGER(1) または INTEGER(2) である場合、その終了点は内部的に REAL(4) 型の値へと変換されます。

両方の終了点が REAL型であり、異なる KTPV を持つ場合、それらはより大きい小数点精度を持つ終了点の近似値手法を用いて内部的に表現されます。

区間定数の KTPV は、最大小数点精度を持つ部分の KTPV です。

コード例 2-1 は、さまざまな 区間定数の KTPV を示しています。

コード例 2-1　区間定数の KTPV

```
math% cat ce2-1.f95
IF(KIND([9_8, 9.0])       == 16 .AND. &
   KIND([9_8, 9_8])       == 16 .AND. &
   KIND([9_4, 9_4])       == 8  .AND. &
   KIND([9_2, 9_2])       == 4  .AND. &
   KIND([9, 9.0_16])      == 16 .AND. &
   KIND([9, 9.0])         == 8  .AND. &
   KIND([9, 9])           == 8  .AND. &
   KIND([9.0_4, 9.0_4])   == 4  .AND. &
   KIND([1.0Q0, 1.0_16]) == 16 .AND. &
   KIND([1.0_8, 1.0_4])   == 8  .AND. &
   KIND([1.0E0, 1.0Q0])   == 16 .AND. &
   KIND([1.0E0, 1])       == 8  .AND. &
   KIND([1.0Q0, 1])       == 16 ) PRINT *, 'Check'
END
math% f95 -xia ce2-1.f95
math% a.out
CHECK
```

0.1 または [0.1,0.2] のような Fortran 定数は、定数が表す外部値と内部的な近似値の 2 つの値に関連付けられます。Fortran では、定数の値はその内部的な近似値です。定数の外部値と定数の内部的な近似値とを区別する必要はありません。区間ではこれを区別する必要があります。Fortran 定数の外部値を表すには、次の表記が用いられます。

ev(0.1) = 0.1、または、ev([0.1,0.2])= [0.1, 0.2]

ev の表記は外部値を表します。

Fortran 規格に従えば、区間定数の数値は、その内部的な近似値となります。区間定数の外部値は常に明示的にそのように標識付けられます。

たとえば、区間定数 [1, 2] とその外部値 ev ([1, 2]) は、数学値 [1, 2] と同じです。しかし、ev ([0.1, 0.2]) = [0.1, 0.2] ですが、数 0.1 と 0.2 はマシンで表現できないので、[0.1, 0.2] は単なるマシンの内部的な近似値にすぎません。このため、区間定数の値 [0.1, 0.2] は、マシン内部の近似値なのです。この外部値は ev ([0.1, 0.2]) で表されます。

厳密区間式処理のもとでは、他の Fortran 数値定数の場合と同様に、区間定数の内部的な近似値は固定されます。REAL 定数の値はその内部的な近似値です。同様に、区間定数の内部的な近似値の値は定数の値として参照されます。定数の外部値 (標準 Fortran の定義概念ではありません) は、その内部的な近似値と異なることもあります。最大幅要求式処理のもとでは、区間定数の値は文脈に依存します。しかしその場合でも、strict と最大幅要求式処理の両方で、区間定数の内部的な近似値はその外部値を含まなければなりません。

任意の数学定数と同様に、区間定数の外部値は不変です。名前付き区間定数 (PARAMETER) の外部値は 1 つのプログラム単位の中では変更できません。しかし、任意の名前付き定数の場合と同様に、異なるプログラム単位間では同じ名前付き定数に異なる値を関係付けることができます。

区間は不透明ですから、区間を内部的に表現するのに必要な情報を格納する言語要件は存在しません。区間の最大下限と最小上限にアクセスするために組み込み関数が提供されています。しかし、区間定数は順位付けられた 1 対の REAL または区間定数により定義されます。定数はカンマにより区切られ、1 対の情報は角括弧で囲まれます。第 1 定数は最大下限 (infimum) であり、第 2 定数は最小上限 (supremum) です。

角括弧の中に 1 つの定数が現れる場合に限り、表現される区間は縮退して、同じ最大下限と最小上限を持つようになります。この場合、単一の小数文字定数の外部値を含むことが保証された区間の内部的な近似値が構築されます。

有効な区間は、最小上限より小さいかまたは等しい 1 つの最大下限を持たなければなりません。同様に、区間定数もまた、その最小上限より小さいかまたは等しい 1 つの最大下限を持たなければなりません。たとえば、次の部分コードは *true* と評価されなければなりません。

```
INF([0.1]) .LE. SUP([0.1])
```

コード例 2-2 は、有効な区間定数と無効な区間定数を含んでいます。

区間定数に関する追加情報については、123 ページの「参考文献」で引用している捕捉 [4] を参照してください。

コード例 2-2　有効な区間定数と無効な区間定数

```
math% cat ce2-2.f95

   INTERVAL :: X
   X=[2,3]
   X=[0.1]       ! ケース 2: 小数 1/10 を含む区間
   X=[2, ]       ! ケース 3: 無効 - 最小上限がない
   X=[3_2,2_2]   ! ケース 4: 無効 - 最大下限 > 最小上限
   X=[2_8,3_8]
   X=[2,3_8]
   X=[0.1E0_8]
   X=[2.0_16,3.0_16]
   X=[2,3.0_16]
   X=[0.1E0_16]
   END
math% f95 -xia ce2-2.f95
   X=[2, ]         ! ケース 3: 無効 - 最小上限がない
         ^
"ce2-2.f95", 行 = 4, 桁 = 13: エラー : 予期しない構文 :
" オペランドが予期されるところに "]" がありました
   X=[3_2,2_2]     ! ケース 4: 無効 - 最大下限 > 最小上限
            ^
"ce2-2.f95", 行 = 5, 桁 = 18: エラー : 区間定数の左側の終了点は、
右側の終了点以下でなければなりません。
   X=[2_16,3_16]
"ce2-2.f95", 行 = 9, 桁 = 11: エラー: 種別型パラメータの値 16 は INTEGER 型には
有効ではありません。
"ce2-2.f95", 行 = 9, 桁 = 16: エラー: 種別型パラメータの値 16 は INTEGER 型には
有効ではありません。
f95comp: 12 ソース行
f95comp: 4 個のエラー、0 個の警告、0 個の他のメッセージ , 0 個の ANSI
```

内部的な近似値

REAL 定数の内部的な近似値は定数の外部値と同じであるとは限りません。たとえば、10 進小数 0.1 は 2 進の浮動小数点数集合のメンバーではないので、この値は 0.1 に近い 2 進の浮動小数点数によって近似されます。REAL データ項目について、Fortran 規格では近似の正確性は指定されていません。区間データ項目については、区間定数を記号的に表現するのに使われる 10 進小数を使用して定義された 1 組の数学値を含むことで知られた、一対の浮動小数点数値が用いられます。たとえば、数学区間 [0.1, 0.2] は、区間定数 `[0.1,0.2]` の外部値です。

Fortran 言語に REAL 定数の正確な近似値を求める要件が存在しないのと同様に、狭い幅の区間定数を使って区間の外部値の近似値を求める言語要件も存在しません。区間定数にはその外部値を含むという要件があります。

ev(INF([0.1,0.2])) ≤ inf(ev([0.1,0.2])) = inf([0.1, 0.2])

および、

sup([0.1, 0.2]) = sup(ev([0.1,0.2])) ≤ ev(SUP([0.1,0.2])) ≤

です。

f95 の区間定数は鋭い値です。これは、実装品質の特性です。

区間文

区間宣言文は、f95 の Fortran 言語に追加された唯一の区間固有の文です。区間データ項目との対話を行う 区間宣言文と標準 Fortran 文の詳細な解説については、82 ページの「INTERVAL」を参照してください。

データ型とデータ項目

f95 のコマンド行に、-xia、または、-xinterval オプションが入力されるか、あるいは、これらのオプションが widestweed とstrict のいずれかに設定されると、INTERVAL データ型は f95 の組み込みの数値データ型として認識されます。f95 のコマンド行にいずれのオプションも入力されないか、あるいは、使用しないと設定されると、INTERVAL データ型は組み込みのデータ型とは認識されません。区間のコマンド行オプションの詳細については、44 ページの「区間のコマンド行オプション」を参照してください。

名前：INTERVAL

6 個の組み込みの Fortran データ型に、組み込み型の INTERVAL が追加されています。INTERVAL 型は、区間データ項目の内部形式が明記されていないという意味で不透明です。しかし、区間データ項目の外部形式は、区間データ項目と同じ種別型パラメータ値 (KTPV) を持つ一対の REAL データ項目です。

種別型パラメータ値 (KTPV)

区間データ項目は、最大下限と最小上限で構成される算術区間の近似値です。区間データ項目は他の数値データ項目のすべての特性を持っています。

デフォルト区間データ項目の KTPV は 8 です。KTPV の指定のないデフォルト区間データ項目のサイズは 16 バイトです。f95 のデフォルト区間データ項目のサイズは、-xtypemap、または、-r8const コマンド行オプションを使って変更することはできません。より詳細な情報については、46 ページの「-xtypemap と -r8const コマンド行オプション」を参照してください。このため、-xtypemap を使ってデフォルトの REAL と INTEGER データ項目のサイズが変更されなければ、次のようになります。

```
KIND([0]) = 2*KIND(0) = KIND(0.0_8) = 8
```

サイズと境界整列のまとめ

INTERVAL 型のサイズと境界整列は、f95 コンパイラオプションの影響を受けません。表 2-2 は、INTERVAL のサイズと境界整列を含んでいます。

表 2-2　INTERVAL のサイズと整列

データ型	バイトサイズ	境界整列
INTERVAL	16	8
INTERVAL(4)	8	4
INTERVAL(8)	16	8
INTERVAL(16)	32	16

注 - 区間配列は要素と同じ並びになります。

区間配列

区間配列は、異なる数値型のすべての配列特性を持っています。区間配列の宣言については、85 ページのコード例 2-25 を参照してください。

次の配列組み込み関数については、区間バージョンがサポートされています。

ALLOCATED()、ASSOCIATED()、CSHIFT()、DOT_PRODUCT()、EOSHIFT()、KIND()、LBOUND()、MATMUL()、MAXVAL()、MERGE()、MINVAL()、NULL()、PACK()、PRODUCT()、RESHAPE()、SHAPE()、SIZE()、SPREAD()、SUM()、TRANSPOSE()、UBOUND()、UNPACK()

MINVAL と MAXVAL 組み込み関数が区間配列に適用されると、配列の要素により処理されない区間値を返す可能性があるので、区間配列用の MINLOC() と MAXLOC() 組み込み関数は定義されていません。MAX と MIN 組み込み関数の解説については、下記の節を参照してください。

- 116 ページの「最大：MAX(X1,X2,[X3, ...])」
- 117 ページの「最小：MIN(X1,X2,[X3, ...])」

例：MINVAL((/[1,2],[3,4]/)) = [1,3]

MAXVAL(/[1,2],[3,4]/) = [2,4]

次の組み込み区間個別関数については、配列バージョンがサポートされています。

ABS()、INF()、INT()、MAG()、MAX()、MID()、MIG()、MIN()、NDIGITS()、SUP()、WID()

次の組み込み区間算術関数については、配列バージョンがサポートされています。

ACOS()、AINT()、ANINT()、ASIN()、ATAN()、ATAN2()、CEILING()、COS()、COSH()、EXP()、FLOOR()、LOG()、LOG10()、MOD()、SIGN()、SIN()、SINH()、SQRT()、TAN()、TANH()

次の区間構成子については、配列バージョンがサポートされています。

INTERVAL()、DINTERVAL()、SINTERVAL()、QINTERVAL()

区間演算式

区間算術式は、他の数値データ型と同じ算術演算子から構築されます。区間式と非区間式間の基本的な相違点は、任意の可能な区間式の結果が区間演算の包合の制約を満たす有効な区間であるということです。これと対照的に、非区間式の結果は任意の近似値であってもかまいません。

混合モードの区間式

混合モード (区間の点) 式は包含を保証するために、最大幅要求式処理が必要となります。-xia コマンド行マクロ、または、-xinterval コマンド行オプションを使用して区間サポートが呼び出された場合の式処理は、デフォルトで最大幅要求となりま

す。最大幅要求の評価が好ましくない場合は、-xia=strict、または、-xinterval=strict のオプションを使って厳密式処理を呼び出してください。厳密式処理のもとでは、混合モードの区間式はコンパイル時エラーとなります。区間と COMPLEX オペランド間の混合モード演算はサポートされていません。

最大幅要求式処理を使うと、区間式中のすべてのオペランドの KTPV は、全体を通じて最大の区間 KTPVである $KTPV_{max}$へと昇格されます。

注 - KTPV 昇格は式の評価前に実行されます。

最大幅要求式評価の保証：

- 区間の包含
- 型または精度の変換は、変換された区間に幅を追加しません。

注 - 厳密式処理を使用する必要が特別にない限り、最大幅要求式処理を導入することを強く推奨します。任意の式または式の一部では、明示的な INTEVAL 型と KTPV の変換が常に発生します。

次の例では最大幅要求式処理の動作と効果を示しています。各例には、次の 3 つのコードブロックが存在します。

- 式処理モード (最大幅要求または厳密) から独立した一般的なコード
- 最大幅要求コード
- 厳密と等価なコード

例は、次の 3 つのメッセージを伝えるように設計されています。

- 特別な環境にある場合を除き、最大幅要求式処理を使用してください。
- 最大幅式処理は有効であるが、使用したくない場合は、いつでも区間構築子を使って型と KTPV の変換を強制実行するよう上書きすることができます。
- 厳密式処理を使う場合、INTERVAL 型と精度変換は区間定数と区間構築子を使用して明示的に指定されなければなりません。

値の代入

区間の代入文は、区間のスカラー、配列要素または配列式の値を区間変数、配列要素または配列に代入します。この構文は次のとおりです。

```
V = expr
```

expr は区間演算または配列式の可変部分であり、V は区間変数、配列要素、配列、配列セクションまたは構造体構成要素です。

区間の代入を実行すると、最大幅要求または厳密式処理を使用して式が評価されます。この後で、結果の値が V に代入されます。最大幅要求式処理を用いた式の評価では次の手順が発生します。

1. すべての点 (非区間) データ項目の区間 KTPV が計算されます。

   点項目が整数であれば、結果としての区間の KTPV は整数の KTPV の 2 倍となります。その他の場合、区間の KTPV は点項目の KTPV と同じです。

2. 代入文の左側を含む式が走査され、$KTPV_{max}$ で表される最大区間 KTPV が求められます。

3. 式の評価に先立ち、区間式の中のすべての点と区間データ項目が式の評価に先立ち、$KTPV_{max}$ へと昇格されます。

4. 式の後ろの `KIND(V)` < $KTPV_{max}$ が評価されると、式の結果は KTPV = `KIND(V)` で包含区間に変換され、結果値は V に代入されます。

コード例 2-3　KIND (左側) に依存する$KTPV_{max}$

```
math% cat ce2-3.f95
INTERVAL(4)  :: X1, Y1
INTERVAL     :: X2, Y2          ! INTERVAL (8) :: X2, Y2 と同じ
INTERVAL(16) :: X3, Y3

! 最大幅要求コード
 X1 = 0.1
 X2 = 0.1
 X3 = 0.1

! 同等の厳密コード
Y1 = [0.1_4]
Y2 = [0.1_8]
Y3 = [0.1_16]

IF(X1 .SEQ. Y1)  PRINT *, "Check 1"
IF(X2 .SEQ. Y2)  PRINT *, "Check 2"
IF(X3 .SEQ. Y3)  PRINT *, "Check 3"
END

math% f95 -xia ce2-3.f95
math% a.out
 Check 1
 Check 2
 Check 3
```

注 - 最大幅要求のもとでは、代入先変数 (左側) の KTPV は、区間文のすべての項目が昇格されることになる、$KTPV_{max}$ 値の決定要素に含まれます。

コード例 2-4　混合モードの代入文

```
math% cat ce2-4.f95
INTERVAL(4) :: X1, Y1
INTERVAL(8) :: X2, Y2
REAL(8)     :: R = 0.1

! 最大幅要求コード
 X1 = R*R ! 4 行目
 X2 = X1*R ! 5 行目

! 厳密と等価なコード
 Y1 = INTERVAL((INTERVAL(R, KIND=8)*INTERVAL(R, KIND=8)), KIND=4)! 6 行目
 Y2 = INTERVAL(X1, KIND=8)*INTERVAL(R, KIND=8) ! 7 行目

IF((X1 == Y1)) PRINT *, "Check 1" ! 8 行目
IF((X2 == Y2)) PRINT *, "Check 2" ! 9 行目
END

math% f95 -xia ce2-4.f95
math% a.out
 Check 1
 Check 2
```

コード例 2-4 注記：

- 厳密と等価なコードは最大幅要求式処理を使用して取得した結果を再現するのに必要な手順を示しています。
- 4 行目では、KIND(R) = 8 ですが、KIND(X1) = 4 となります。包含を保証し鋭い結果を生成するために、R は式の評価前に、区間を含む $KTPV_{max} = 8$ へと変換されます。次に、その結果が区間を含む KTPV-4 へと変換され、X1 に代入されます。これらの手順は、6 行目の厳密と等価なコードでは明示的になっています。
- 5 行目では、KIND(R) = KIND(X2) = 8 となります。このため、X1 は式の評価前に KTPV-8 区間へと昇格され、その結果が X2 に代入されます。これと厳密と等価なコードを 7 行目で示しています。
- 8 行目と 9 行目の検証では、最大幅要求と厳密の結果が同じであることを確かめています。最大幅要求と厳密式処理に関するより詳細な情報については、40 ページの「区間演算式」を参照してください。

区間のコマンド行オプション

f95 コンパイラでの区間機能は次のコマンド行オプションにより起動されます。

- -xinterval=(no|widestneed|strict) コマンド行オプションは、区間処理を有効にし、許容された式評価構文を制御します。
  - 「no」は f95 の区間拡張を無効にします。
  - 「widestneed」は、オプションが指定されていない場合の -xinterval と同じ最大幅要求式処理と関数を有効にします。40 ページの「混合モードの区間式」を参照してください。
  - 「strict」はすべてのINTERVAL 型と KTPV の明示的な変換を要求するか、あるいは、それが、26 ページの「エラーの検出」で解説したように、コンパイル時エラーとなります。

- -xia=(widestneed|strict) は、INTERVAL データ型の処理を可能にし、適切な浮動小数点環境を設定するマクロです。-xia の記載がなければ (1 番目のデフォルト)、拡張は行われません。

  -xia は、次のように拡張されます。

  ```
  -xinterval=widestneed
  -ftrap=%none
  -fns=no
  -fsimple=0
  ```

  -xia=(widestneed|strict) は、次のように拡張されます。

  ```
  -xinterval=(widestneed|strict)
  -ftrap=%none
  -fns=no
  -fsimple=0
  ```

  以前に設定された -ftrap、-fns、-fsimple の値は置き換えられます。

コマンド行処理の末尾に、xinterval=(widestneed|strict) が設定され、さらに -fsimple または -fns が -fsimple=0、-fns=no 以外の値に設定されると致命的なエラーになります。

コマンド行オプションの使用時：

- コマンド行処理の最後に -ansi が設定され、また、-xinterval が widestneed と strict のいずれかに設定されると、次の警告が表示されます。
  区間データ型は非標準機能です。

- 区間演算とルーチンは開始時と終了時に丸めモードの保存と復帰を行うため、-fround = <r> (起動時に IEEE 丸めモードを有効に設定する) は、-xia と相互に作用しません。

INTERVAL 型を認識するよう起動した場合：

- 区間演算子と関数は組み込みになります。
- 組み込み区間演算子と関数の拡張には、標準の組み込み演算子と関数の拡張に課せられるのと同じ制約が課せられます。
- 組み込みの区間個別関数名が認識されます。39 ページの「区間配列」と 113 ページの「数学関数」を参照してください。

-xtypemap と -r8const コマンド行オプション

区間キーワードだけを使って宣言されたデフォルト区間変数のサイズは、-xtypemap と -r8const のコマンド行オプションを使って変更することはできません。

これらのコマンド行オプションはデフォルトの INTERVAL 型には影響を与えませんが、コード例 2-5 で示しているように、混合モードの区間式の結果を変更することができます。

コード例 2-5　混合モードの式

```
math% cat ce2-5.f95
REAL     :: R
INTERVAL :: X
R = 1.0E0 - 1.0E-15
PRINT *, 'R = ', R
X = 1.0E0 - R
PRINT *, 'X = ', X
IF (  0.0 .IN. X  ) THEN
    PRINT *, 'X contains zero'
ELSE
    PRINT *, 'X does not contain zero'
ENDIF
PRINT *, 'WID(X) = ', WID(X)
END
math% f95 -xia ce2-5.f95
math% a.out
 R =  1.0
 X =  [0.0E+0,0.0E+0]
 X contains zero
 WID(X) =  0.0E+0
math% f95 -xia -xtypemap=real:64,double:64,integer:64 ce2-5.f95
math% a.out
```

```
 R =  0.999999999999999
 X =  [9.9920072216264088E-16,9.9920072216264089E-16]
 X does not contain zero
 WID(X) =  0.0E+0
```

注 - -xtypemap は X の KTPV に影響を及ぼしませんが、X の値には影響を与えることができます。

定数式

区間の定数式は、任意の点定数式構成要素だけでなく、区間のリテラルと名前付き定数を含む場合があります。このため、各オペランドまたは引数はそれ自体が、他の定数式、定数、名前付き定数、あるいは、定数引数を使って呼び出された組み込み関数です。

コード例 2-6 定数式

```
math% cat ce2-6.f95
INTERVAL :: P, Q
! 最大幅要求コード
P = SIN([1.23])+[3.45]/[9, 11.12]

! 等価な厳密コード
Q = SIN([1.23_8])+[3.45_8]/[9.0_8, 11.12_8]
IF(P .SEQ. Q) PRINT *, 'Check'
END
math% f95 -xia ce2-6.f95
math% a.out
 Check
```

注 - 最大幅要求式のもとでは、区間定数の KTPV は区間の文脈に基づき決定されます。より詳細な情報については、13 ページの「デフォルトの種別型パラメータ値 (KTPV)」を参照してください。

区間定数の使用が許可される場合は、常に区間定数式が使用できます。

組み込み演算子

表 2-3 は、区間演算で使用できる組み込み演算子を一覧にしたものです。表 2-3 では、X と Y は区間を表します。

表 2-3　組み込み演算子

演算子	演算	式	意味
**	べき乗	X**Y	X を INTERVAL Y 乗する
		X**N	X を INTEGER N 乗する (注記1を参照)
*	乗算	X*Y	X と Y を乗ずる
/	除算	X/Y	X を Y で除する
+	加算	X+Y	X と Y を加算する
+	同一	+X	(符号なし) X と同じ
-	減算	X-Y	X から Y を減ずる
-	数値否定	-X	X を否定する
.IH.	INTERVAL包	X.IH.Y	X と Y の区間包
.IX.	積集合	X.IX.Y	X と Y の積集合

(1) N が整数式であれば、-xia=strict 式の処理において、オーバーフローにより内部的なエラーが発生する可能性があります。これは、最大幅要求式の処理の問題ではありません。strict 式の処理では、ユーザーの責任で整数のオーバーフローを防止してください。さらに詳細な情報については、29 ページの「整数オーバーフロー」を参照してください。

演算子の優先順位：

- 演算子 ** は、*、+、-、.IH.、.IX. 演算子よりも優先します。
- 演算子 * と / は、+、-、.IH.、.IX. 演算子よりも優先します。
- 演算子 + と - は、.IH. と .IX. 演算子よりも優先します。
- 演算子 .IH. と .IX. は、// 演算子よりも優先します。

区間 ** 演算子と整数指数を別にすれば、区間演算子は同じ KIND 型のパラメータ値を持つ 2 つの区間オペランドにだけ適用することができます。このため、区間演算子の結果の型と KTPV はそのオペランドの型 KTPV と同じです。

区間 ** 演算子の第 2 オペランドが整数であれば、第 1 オペランドは任意の区間 KTPV となることができます。この場合、結果は第 1 オペランドの型と KTPV を持ちます。

いくつかの区間個別演算子は点 (非区間) 用演算子がありません。表 2-4 で示すように、これらの演算子は「set」、「certainly」、「possibly」の 3 つのグループにまとめることができます。いくつかの固有の集合演算子は「certainly」または「possibly」の相当する演算子がありません。

表 2-4　組み込み区間関係演算子

set 関係演算子	.SP.	.PSP	.SB.	.PSB.	.IN.	.DJ.
	.EQ. (== と同様)	.NEQ. (/= と同様)				
	.SEQ.	.SNE.	.SLT.	.SLE.	.SGT.	.SGE.
certainly 関係演算子	.CEQ.	.CNE.	.CLT.	.CLE.	.CGT.	.CGE.
possibly 関係演算子	.PEQ.	.PNE.	.PLT.	.PLE.	.PGT.	.PGE.

組み込み区間関係演算子の先行オペランドは REAL 関係演算子の場合と同じです。

.IN. 演算子を除き、組み込み区間関係演算子は、同じ KTPV を持つ 2 つの区間オペランドにだけ適用することができます。

.IN. 演算子の第 1 オペランドは任意の INTEGER 型または REAL 型です。第 2 オペランドには任意の区間 KTPV を持つことができます。

区間関係式の結果はデフォルトの LOGICAL 種類型のパラメータを持ちます。

算術演算子 +、-、*、/

有限の REAL 区間に関する区間演算の終了点の計算公式は、すべての可能性のある点結果の集合を含むことが保証された最も狭い区間を生成する必要性からもたらされました。Ramon Moore は、これらの公式をより重要な意味で独自に開発し、また区間演算への適用に必要な分析をはじめて開発しました。より詳細な情報については、R. Moore 著『Interval Analysis』、Prentice-Hall (1966年) を参照してください。

すべての可能な値の集合は、オペランド区間の任意の要素に関する演算課題を実行することにより独自に定義されています。このため、 $op \in \{+, -, \times, \div\}$ を持つ所与の有限区間 [a,b] と [c,d] は、ゼロによる除算を除外すれば、次のようになります。

$$[a, b]\ op\ [c, d] \supseteq \{x\ op\ y\ |\ x \in [a, b]\ and\ y \in [c, d]\}$$

この公式または論理的に同じものは以下のようになります。

$$[a,b]+[c,d] = [a+c\ ,\ b+d]$$

$$[a,b]-[c,d] = [a-d\ ,\ b-c]$$

$$[a,b]\times[c,d] = [\min(a\times c, a\times d, b\times c, b\times d)\ ,\ \max(a\times c, a\times d, b\times c, b\times d)]$$

$$[a,b]/[c,d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]\ ,\ \text{if } 0 \notin [c,d]$$

生成される区間にすべての可能な値の集合が含まれることを保証するために、有限精度の算術を使った計算では有向の丸めが用いられます。

任意の区間の結果が含まなければならない値の集合は、結果を生成する演算または式の包含集合（cset）と呼ばれます。

cset は、(無限の終了点を持つ) 拡張区間とゼロによる除算を含むために、実値に関する算術演算の値に直接依存することはできません。拡張区間用の cset には、通常なら未定義の点に関する演算要件が課せられます。未定義の演算には不定形式 $1\div 0$ 、$0\times\infty$、$0\div 0$、$\infty\div\infty$ が含まれます。

包含集合の閉包単位は、特異なまたは不定な点での式の包含集合の値の識別の問題を解決します。この単位は包含集合が関数の閉包であることを示します。変域の境界上の点での関数の閉包は、すべての集積点 (limit point または accumulation point) を含みます。詳細については、「用語」と 123 ページの「参考文献」で引用した補足文献 [1]、[3]、[10]、[11] を参照してください。

式の cset に含まれる値を正当化する直観的な方法を次に示します。次の関数を考えてみましょう。

$$h(x) = \frac{1}{x}$$

$h(x_0)$ のとき $x_0 = 0$ の cset は何でしょうか。この質問に答えるために、次の関数を考えてみます。

$$f(x) = \frac{x}{x+1}$$

$f(x_0) = 0$ であれば $x_0 = 0$ となるのは明白です。しかし、次の場合はどうでしょうか。

$$g(x) = \frac{1}{1+\left(\frac{1}{x}\right)}$$

また、次の場合はどうでしょうか。

$$g(x) = \frac{1}{1 + h(x)}$$

関数 $g(x_0)$ は、$x_0 = 0$ のとき不定です。これは、$h(x_0)$ が不定であるからです。$x_0 = 0$ について $h(x_0)$ の cset は、$g(x_0) = f(x_0)$ に対する値の最小集合です。さらにこのことは、h のすべての複合関数にもあてはまります。たとえば、次の関数を考えてみましょう。

$$g'(y) = \frac{1}{1 + y} \quad ,$$

この場合、$g(x) = g'(h(x))$ になります。この例では、$\{-\infty, +\infty\}$ のときに $h(x_0)$ の cset が $\{-\infty, +\infty\}$ になることを実証できます。ここで、$\{-\infty, +\infty\}$ は、2 つの値 $-\infty$ および $+\infty$ で構成される集合を表しています。

表 2-5 から表 2-8 は、基本的な算術演算の cset を示しています。$f(x)$ で表記される式が単純にその cset を意味する記数法を採用する場合に便利です。それでは、同様に、次の関数を考えてみてください。

$$f(X) = \bigcup_{x \in X} f(x)$$

区間 X に対する f の包含集合が上式のように表される場合、閉包 ($f(x)$) は、$f(x)$ を含む鋭い区間になります。

表 2-5　加算用の包含集合：$x + y$

x + y の **cset**	**{-∞}**	**{real: y_0}**	**{+∞}**
{-∞}	{-∞}	{-∞}	$\Re^*$
{real: x_0}	{-∞}	$\{x_0 + y_0\}$	{+∞}
{+∞}	$\Re^*$	{+∞}	{+∞}

表 2-6　減算用の包含集合：$x - y$

x – y の **cset**	**{-∞}**	**{real: y_0}**	**{+∞}**
{-∞}	$\Re^*$	{-∞}	{-∞}
{real: x_0}	{+∞}	$\{x_0 - y_0\}$	{-∞}
{+∞}	{+∞}	{+∞}	$\Re^*$

表 2-7　乗算用の包含集合：$x \times y$

x × y の cset	**{-∞}**	**{real: y_0 < 0}**	**{0}**	**{real: y_0 > 0}**	**{+∞}**
{-∞}	{+∞}	{+∞}	$\Re^*$	{-∞}	{-∞}
{real: x_0 < 0}	{+∞}	$\{x \times y\}$	{0}	$\{x \times y\}$	{-∞}
{0}	$\Re^*$	{0}	{0}	{0}	$\Re^*$
{real: x_0 > 0}	{-∞}	$x \times y$	{0}	$x \times y$	{+∞}
{+∞}	{-∞}	{-∞}	$\Re^*$	{+∞}	{+∞}

表 2-8　除算用の包含集合：$x \div y$

x ÷ y の cset	**{-∞}**	**{real: y_0 < 0}**	**{0}**	**{real: y_0 > 0}**	**{+∞}**
{-∞}	[0, +∞]	{+∞}	{-∞, +∞}	{-∞}	[-∞, 0]
{real: $x_0 \neq 0$}	{0}	$\{x \div y\}$	{-∞, +∞}	$\{x \div y\}$	{0}
{0}	{0}	{0}	$\Re^*$	{0}	{0}
{+∞}	[-∞, 0]	{-∞}	{-∞, +∞}	{+∞}	[0, +∞]

表の中のすべての入力は集合として示されています。結果は、集合または区間として示されています。必要に応じて cset が暗黙に包含されるという理解に基づいて、$(-\infty) + (+\infty) = -\infty$ 、$(-\infty) + y = -\infty$ 、$(-\infty) + (+\infty) = \Re^*$ などの慣行の表記が使用されています。

1 つのケースとして、ゼロによる除算では、結果は区間でなく、集合$\{-\infty, +\infty\}$ です。この場合、区間算術の包含の制約に反しない現在のシステムでの最も狭い区間は、区間 $[-\infty, +\infty] = \Re^*$ となります。

符号の変更は期待とおりの結果を生成します。

これらの結果を区間終了点の計算用の公式に組み込むために必要なのは、同様に丸め方向に符号化される必要な終了点を識別することだけです。↓ を使って (-∞ に対する) 丸め切り下げを表し、↑ を使って (+∞に対する) 丸め切り上げを表すと、以下のようになります。

$\downarrow (+\infty) \div (+\infty) = 0$ and $\uparrow (+\infty) \div (+\infty) = +\infty$

$\downarrow 0 \times (+\infty) = -\infty$ and $\uparrow 0 \times (+\infty) = +\infty$

同様に、$\mathtt{hull}(\{-\infty, +\infty\}) = [-\infty, +\infty]$ なので、

$\downarrow x \div 0 = -\infty$ and $\uparrow x \div 0 = +\infty$

となります。

最後に、Fortran では空の区間は文字列 [empty] で表され、集合代数の ∅ で表される空の集合と同じ特性を持ちます。空の区間に関する任意の算術演算は空の区間結果を生成します。空の区間用法の追加情報については、123 ページの「参考文献」で引用した補足文献 [6]、[7] を参照してください。

f95 は、これらの結果を用いて、「単純な」閉じた区間システムを実装しています。すべての算術演算と関数は常に有効な区間結果を生成しますから、システムは閉じています。123 ページの「参考文献」で引用した補足文献 [2]、[8] を参照してください。

べき乗演算子 X**N と X**Y

べき乗演算子は整数指数 (X**N) と共に使うことも、連続する指数 (X**Y) と共に使うこともできます。べき乗演算子は連続する指数と共に用いると、4 つの算術演算子に類似した不定の形式を持ちます。

整数指数のケースでは、X^n の囲みが含まなければならないすべての値の集合は、次のようになります。

$$\{z \mid z \in x^n \text{ and } x \in X\}$$

単調性は、整数べき乗関数の鋭い区間の囲みを構築するのに使用できます。$n = 0$ のとき、X^n の cset を表す X^n は、すべての $x \in [-\infty, +\infty]$ について 1 になり、すべての n について $\varnothing^n = \varnothing$ になります。

連続する指数のケースでは、X**Yの区間の囲みが含まなければならないすべての値の集合は次のようになります。

$$\exp(Y(\ln(X))) = \{z \mid z \in \exp(y(\ln(x))), y \in Y_0, x \in X_0\}$$

ここでは、cset(exp(y ln(x)), {(Y_0, X_0)}) は、式 exp(y ln(x)) の包含集合です。関数 exp(Y(ln(X))) は、$x \geq 0$ の値だけを考慮する必要のあることを明示し、これは、Fortran での REAL 引数を持つ X**Y の定義と一貫性があります。

区間引数が空であるか、または、$x < 0$であれば、この結果は空になります。これは、Fortran での X**Y の点バージョンと一貫性があります。

表 2-9 は、exp(y(ln(x))) のすべての特異点と不定形式についての内部的な集合を表しています。

表 2-9　exp(y(ln(x)))

x_0	y_0	**exp(y(ln(x)))**
0	$y_0 < 0$	+∞
1	-∞	[0,+∞]
1	+∞	[0,+∞]
+∞	0	[0,+∞]
0	0	[0,+∞]

表 2-9 の結果は、以下の 2 つの方法で求めることができます。

- 式が未定義の x_0 と y_0 の値について、合成式の包閉、exp(y(ln(x))) を直接計算します。
- 包含集合評価理論を使って、包含集合内部の値の集合を囲みます。

ほとんどの合成では、2 つ目のオプションを使う方が簡単です。十分な条件が満たされない場合、合成の囲みはその包閉の合成から計算することができます。つまり、各下位式の包閉を使って式全体の包閉を計算します。存在するケースでは次のようになります。

$$\exp(y(\ln(x))) = \exp(y_0 \times \ln(x_0)).$$

つまり、左側の式の cset は右側の cset の構成と同等です

常にあてはまるケースは次のようになります。

$$\exp(y(\ln(x))) \subseteq \exp(y_0 \times \ln(x_0))$$

これは区間演算が区間にどのように機能するか正確に表しているという点に留意してください。ln とexp 関数に必要な包閉は以下のとおりです。

$$\begin{aligned} \ln(0) &= -\infty \\ \ln(+\infty) &= +\infty \\ \exp(-\infty) &= 0 \\ \exp(+\infty) &= +\infty \end{aligned}$$

包閉合成の等式に必要な条件は、式が単一用途式 (SUE = *single-use expression*) でなければならないことです。つまり、各独立変数は式の中で 1 回だけ現れることができることを意味します。

存在するケースでは、式は明らかに 1 つの SUE となります。

表 2-9 の項目は、ln とexp 関数の包閉に関する表 2-7 での基本乗算の包含集合用法の直接の結果です。たとえば、$x_0 = 1$ と $y_0 = -\infty$ を使うと、$\ln(x_0) = 0$ となります。52 ページの表 2-7 での値 -∞ と0 に関する乗算の包閉について、結果は [-∞, +∞] となります。最後に、表 2-9 での 2 番目の項目は、exp([-∞, +∞])= [0, +∞] となります。残りの項目も同じステップを使って取得できます。exp(*y*(ln(*x*))) の包含集合から直接導くことで、これらと同じ結果が得られます。今度は、任意の式の包閉合成等式の十分な条件が明らかにされていません。しかし、それでも以下のようになります。

- 包含集合評価理論は、包閉でなく包閉合成の計算から、包含の失敗が決して生成されないことを保証します。
- 式は、包閉合成の等式が true となる SUE でなければなりません。

依存減算演算子

依存減算演算子 `.DSUB.` を使用すると、直前の区間加算のいずれかのオペランドを回復できます。

2 つの区間変数は、一方の区間変数がもう一方の区間変数に適用される区間算術演算の結果である場合に依存します。たとえば、`X = A + B` のとき、`X` は `A` と `B` の両方に依存します。依存区間減算は、`X` から `A` または `B` を回復する際に、より狭い区間を生成します。

区間定数が区間演算の結果でないため依存できないので、依存演算を区間定数に適用することはできません。区間定数に依存演算を適用すると、コンパイル時にエラーが発生します。

X.DSUB.A の結果は、に示すように、B が X = A + B の場合の囲みを戻します。

表 2-10 X と *A* の値が異なる場合の X.DSUB.A の結果

	A = [EMPTY]	**A =** 有限区間	**A = [-inf, inf]**
X = [EMPTY]	[-inf, inf]	[EMPTY]	[EMPTY]
X = Finite Interval	[-inf, inf]	Finite interval[1]	[-inf, inf]
X = [-inf, inf]	[-inf, inf]	[-inf, inf]	[-inf, inf]

1. 戻される有限区間は、X = A + B の場合、必ず B を包含していなければならない。

.

集合理論演算子

f95 は以下のような、2 つの区間の区間包と積集合を判定するための集合理論演算子をサポートしています。

包： $X \cup Y$、または (X.IH.Y)

解説： 2 つの区間の区間包です。区間包はオペランド区間のすべての要素を含む最小の区間です。

算術と演算の定義：

$$X.IH.Y \equiv [\inf(X \cup Y), \sup(X \cup Y)]$$

$$= \begin{cases} Y, \text{ if } X = \varnothing, \\ X, \text{ if } Y = \varnothing, \text{ and} \\ [\min(\underline{x}, \underline{y}), \max(\bar{x}, \bar{y})], \text{ otherwise.} \end{cases}$$

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： X と同じです。

積集合： $X \cap Y$、または (X.IX.Y)

解説： 2 つの区間の積集合です。

算術と演算の定義：

$$X\ .IX.\ Y \equiv \{z \mid z \in X \text{ and } z \in Y\}$$

$$= \begin{cases} \varnothing, \text{ if } (X = \varnothing) \text{ or } (Y = \varnothing) \text{ or } (\min(\bar{x}, \bar{y}) < \max(\underline{x}, \underline{y})) \\ [\max(\underline{x}, \underline{y}), \min(\bar{x}, \bar{y})], \text{ otherwise.} \end{cases}$$

引数： X と Y は、同じKTPVを持つ区間でなければなりません。

結果型： X と同じです。

集合の関係

f95 は区間をサポートするために拡張された下記のような集合関係演算子を提供しています。

素：$X \cap Y = \varnothing$、または (X.DJ.Y)

解説： 2 つの区間が素であるかどうかをテストします。

算術と演算の定義：

$$X\ .DJ.\ Y \equiv (X = \varnothing) \text{ or } (Y = \varnothing) \text{ or } ((X \neq \varnothing) \text{ and } (Y \neq \varnothing) \text{ and } (\forall x \in X, \forall y \in Y : x \neq y))$$

$$= (X = \varnothing) \text{ or } (Y = \varnothing) \text{ or } ((X \neq \varnothing) \text{ and } (Y \neq \varnothing) \text{ and } ((\bar{y} < \underline{x}) \text{ or } (\bar{x} < \underline{y})))$$

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

要素：$r \in Y$、または (R.IN.Y)

解説： 数 R が区間 Y の要素であるかどうかをテストします。

算術と演算の定義：

$$r \in Y \equiv (\exists\ y \in Y\ :\ y = r)$$
$$= (Y \neq \emptyset)\ \text{and}\ (\underline{y} \leq r)\ \text{and}\ (r \leq \bar{y})$$

引数： R の型は INTEGER または REAL であり、Y の型は INTERVAL です。

結果型： デフォルトの論理スカラーです。

次の注釈は r ∈ Y 集合関係に関するものです。

- 最大幅要求式処理のもとでは、異なる KTPV を持つ R と Y は、それらの評価方法に影響しません。最大幅要求式処理は Y に適用されますが、R の評価には適用されません。評価後、要素包含テストが行われる前に、Y または R の KTPV の昇格が行われます。
- 厳密式評価のもとでは、R と Y は同じ KTPV を持たなければなりません。
- R が NaN (非数値) であれば、R .IN. Y は無条件に *false* となります。
- Y が空であれば、R .IN. Y は無条件に *false* となります。

内部：(X .INT. Y)

解説： X が Y の内部かどうかをテストします。

位相空間での集合の内部は、すべての開いた部分集合の和集合です。

区間について、X .INT. Y (Y の内部における X) は X が Y の 1 つの部分集合であり、下記の両方の関係が false となることを意味します。

- inf(Y) ∈ X、あるいは、Fortran では、INF(Y) .IN. X です。
- sup(Y) ∈ X、あるいは、Fortran では、SUP(Y) .IN. X です。

∅ ∉ ∅ ですが、[empty] .INT. [empty] = true である点に注意してください。

空の集合は開いているので、それ自体の 1 つの部分集合となります。

算術と演算の定義：

$$X\ .\text{INT}.\ Y \equiv (X = \emptyset)\ \text{or}$$
$$((X \neq \emptyset)\ \text{and}\ (Y \neq \emptyset)\ \text{and}\ (\forall x \in X,\ \exists y' \in Y,\ \exists y'' \in Y\ :\ y' < x < y''))$$
$$= (X = \emptyset)\ \text{or}\ ((X \neq \emptyset)\ \text{and}\ (Y \neq \emptyset)\ \text{and}\ (\underline{y} < \underline{x})\ \text{and}\ (\bar{x} < \bar{y}))$$

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

真部分集合：$X \subset Y$ または（X.PSB.Y）

解説： X が Y の真部分集合であるかどうかをテストします。

算術と演算の定義：

$$X.\mathrm{PSB}.Y \equiv (X \subseteq Y) \text{ and } (X \neq Y)$$

$$= ((X = \emptyset) \text{ and } (Y \neq \emptyset)) \text{ or}$$
$$(X \neq \emptyset) \text{ and } (Y \neq \emptyset) \text{ and } (\underline{y} < \underline{x}) \text{ and } (\bar{x} < \bar{y}) \text{ or}$$
$$(\underline{y} < \underline{x}) \text{ and } (\bar{x} \leq \bar{y})$$

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

真超集合：$X \supset Y$、または（X.PSP.Y）

解説： $X \leftrightarrow Y$ を持つ真超集合を参照してください。

部分集合：$X \subseteq Y$、または（X.SB.Y）

解説： X が Y の部分集合であるかどうかをテストします。

算術と演算の定義：

$$X.\mathrm{SB}.Y \equiv (X = \emptyset) \text{ or}$$
$$((X \neq \emptyset) \text{ and } (Y \neq \emptyset) \text{ and } (\forall x \in X, \exists y' \in Y, \exists y'' \in Y : y' \leq x \leq y''))$$

$$= (X = \emptyset) \text{ or } ((X \neq \emptyset) \text{ and } (Y \neq \emptyset) \text{ and } (\underline{y} \leq \underline{x}) \text{ and } (\bar{x} \leq \bar{y}))$$

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

超集合： $X \supseteq Y$、または（X.SP.Y）

解説： X↔Y を持つ超集合を参照してください。

関係演算子

.*qop*.で表される組み込みの区間関係演算子は、次の連結により構成されます。

- ピリオドによる区切り記号
- 演算子接頭辞、*q* ∈ {C, P, S} (C、P、S はそれぞれ「certainly」、「possibly」、「set」を表す)
- Fortran 関係演算子接尾辞、*op* ∈ {LT, LE, EQ, NE, GT, GE}

.SEQ. と.SNE. の代わりに、デフォルト演算子の.EQ.(または、==) と.NE. (または、/=) が受け入れられます。コードのあいまいさを取り除くために、他のすべての区間関係演算子は、接頭辞を指定しなければなりません。

すべての区間関係演算子は同じ優先順位を持ちます。算術演算子は関係演算子よりも高い優先順位を持ちます。

区間関係式は 2 つのオペランドが最初に評価され、その後で、2 つの式の値が比較されることにより評価されます。指定された関係を持つ場合は結果が true となり、それ以外は false となります。

最大幅要求式処理が呼び出されると、区間関係演算子の両方の区間オペランド式に適用されます。

「*nop*」を演算子 *op* の補数とした場合、「certainly」と「possibly」演算子は次のように関係付けられます。

.C*op*. ≡ .NOT.(.P*nop*.)

.P*op*. ≡ .NOT.(.C*nop*.)

注 - この「certainly」と「possibly」演算子の間の同一性は *op* ∈ {EQ, NE} であれば無条件に、それ以外の場合は両方のオペランドが空の場合にだけ成り立ちます。逆に、*op* ∈ {LT, LE, GT, GE} であり、どちらかのオペランドが空である場合は、同一性は成り立ちません。

---

2 つのオペランドのどちらも空でない場合を前提に、表 2-11 は、次の形式のすべての区間関係演算子の Fortran 演算定義を含んでいます。

[$\underline{x}$,$\overline{x}$].*qop*.[$\underline{y}$,$\overline{y}$]

最初の桁は接頭辞の値を含み、最初の行は演算子接尾辞の値を含んでいます。表に示した条件が満たされると結果は true となります。

表 2-11 区間順位関係の演算定義

	LT.	LE.	EQ.	GE.	GT.	NE.
. S	$\underline{x} < \underline{y}$ and $\_$ $x < y$	$\underline{x} \le \underline{y}$ and $\overline{x} \le \overline{y}$	$\underline{x} = \underline{y}$ and $\_$ $x = y$	$\underline{x} \ge \underline{y}$ and $\overline{x} \ge \overline{y}$	$\underline{x} > \underline{y}$ and $\_$ $x > y$	$\underline{x} \ne \underline{y}$ or $\overline{x} \ne \overline{y}$
. C	$\_$ $x < \underline{y}$	$\overline{x} \le \underline{y}$	$\overline{y} \le \underline{x}$ and $\overline{x} \le \underline{y}$	$\underline{x} \ge \overline{y}$	$\_$ $\underline{x} > y$	$\overline{\underline{x} > y}$ or $\_$ $\underline{y} > x$
. P	$\_$ $\underline{x} < y$	$\underline{x} \le \overline{y}$	$\underline{x} \le \overline{y}$ and $\underline{y} \le \overline{x}$	$\overline{x} \ge \underline{y}$	$\_$ $x > \underline{y}$	$\overline{y > \underline{x}}$ or $x > \underline{y}$

コード例 2-7　関係演算子

```
math% cat ce2-7.f95
INTERVAL :: X = [1.0, 3.0], Y = [2.0, 4.0], Z
INTEGER  :: V = 4, W = 5
LOGICAL  :: L1, L2, L3, L4
REAL :: R

L1 = (X == X) .AND. (Y .SEQ. Y)
L2 = X .SLT. Y

! 最大幅要求コード
Z  = W
L3 = W .CEQ. Z
L4 = X-Y .PLT. V-W
IF( L1 .AND. L2 .AND. L3 .AND. L4) PRINT *, 'Check1'

! 同等の厳密コード (L3 と L4 への代入用 )
L3 = INTERVAL(W, KIND=8) .CEQ. Z
L4 = X-Y  .PLT. INTERVAL(V, KIND=8)-INTERVAL(W, KIND=8)
IF(L3 .AND. L4) PRINT *, 'Check2'
END
math% f95 -xia ce2-7.f95
math% a.out
 Check1
 Check2
```

コード例 2-7 注記：

- 区間はそれ自体に対して等しく、デフォルトの.EQ.(または、==) 演算子は.SEQ.と同じなので、L1 は true となります。
- (INF(X).LT.INF(Y)).AND.(SUP(X).LT.SUP(Y)) が true なので、L2 は true となります。
- 最大幅要求で W を区間 [5,5] に昇格します。また、2 つの区間は 4 つのすべての終了点が等しい場合に限り、「断定的な関係」で等しくなるので、L3 は true となります。
- 区間式 X-Y と V-W の評価により、それぞれ、区間 [-3,1] と [-1,1] が生成されるので、L4 は true となります。このため、式 (INF(X-Y) .LT. SUP(V-W)) は、true となります。

集合関係演算子

次のような関係を持つ、2 つの点 x と y 間の「断定的な」順位関係について、

op ∈ {LT, LE, EQ, GE, GT} and

op ∈ { <, ≤, =, ≥, > }

対応する 2 つの空でない区間 X と Y 間の「集合の関係」.S*op*.の数学定義は、次のようになります。

X .S*op*. Y ≡ (∀x ∈ X, ∃y ∈ Y : x op y) and (∀y ∈ Y, ∃x ∈ X : x op y).

x と y の 2 点間の関係 ≠ について、対応する 2 つの空でない区間 *X* と Y 間の「集合の関係」.SNE.は、次のようになります。

X .SNE. Y ≡ (∃ x ∈ X, ∀y ∈ Y : x ≠ y) or (∃ y ∈ Y, ∀x ∈ X : x ≠ y).

空の区間は、後続の各関係の中で明示的に考慮されます。どの場合も次の型規則に従います。

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

「Certainly」(断定的な) 関係演算子

「断定的な」関係演算子は、オペランド区間のすべての要素について前提となる関係が true であれば、true となります。たとえば、すべての x ∈ [a, b] と y ∈ [c, d] について、x < y であれば、[a,b] .CLT. [c,d] は true となります。これは、$b < c$ と同じです。

次のような関係を持つ、2 つの点 x と y 間の「断定的な」順位関係について、

op ∈ {LT, LE, EQ, GE, GT} and

op ∈ { <, ≤, =, ≥, > }

対応する 2 つの区間 X と Y 間の「断定的な」true の関係.C*op*.は、次のようになります。

X .C*op*. Y ≡ (X ≠ ∅) and (Y ≠ ∅) and (∀x ∈ X, ∀y ∈ Y : x op y)

否定的な、「断定的に」等しくない関係を除いて、「断定的な」関係のどちらかのオペランドが空であれば、その結果は false となります。この 1 つの例外である「断定的に」等しくない関係 .CNE. は、この場合に true となります。

それぞれの「断定的な」関係演算子では、次の型規則に従います。

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

「Possibly」(可能性のある) 関係演算子

「可能性のある」関係演算子は、オペランド区間の任意の要素が前提となる関係を満たせば、true となります。たとえば、$x < y$ であるような、$x \in [a, b]$ と $y \in [c, d]$ が存在すれば、`[a,b] .PLT. [c,d]` は true となります。これは、$a < d$ と同じです。

次のような関係を持つ、2 つの点 x と y 間の「肯定的な」順位関係について、

op ∈ {LT, LE, EQ, GE, GT} and

op ∈ { <, ≤, =, ≥, > }

対応する 2 つの区間 *X* と *Y* 間の「可能性のある」true の関係 *.Pop.* は、次のように定義されます。

$$X \,.Pop.\, Y \equiv (X \neq \emptyset) \text{ and } (Y \neq \emptyset) \text{ and } (\exists x \in X, \exists y \in Y : x \,\mathrm{op}\, y)$$

空の区間が「可能性のある」関係のオペランドであれば、結果は false となります。この 1 つの例外である、否定的な「可能性のある」等しくない関係 `.PNE.` は、この場合に true となります。

それぞれの「可能性のある」関係演算子では、以下の型規則に従います。

引数： X と Y は、同じ KTPV を持つ区間でなければなりません。

結果型： デフォルトの論理スカラーです。

組み込み区間演算子の拡張

ユーザーの提供する演算子インタフェースブロックの `INTERFACE` 文の中で指定された演算子が組み込みの区間演算子 (たとえば `.IH.`) である場合、組み込みの区間演算子の 1 つの拡張が作成されます。

組み込みの区間演算子を拡張するユーザーが提供する演算子関数は、オペランドのデータ型についてその演算子が事前定義されている場合、拡張することができません。

以下のリストのような引数の組み合わせについては、組み込みの区間演算子 +、-、*、/、.IH.、.IX.、** が事前定義されており、ユーザーが拡張することはできません。

(任意の INTERVAL 型、任意の INTERVAL 型)

(任意の INTERVAL 型、任意の REAL または INTEGER 型)

(任意の REAL または INTEGER 型、任意の INTERVAL 型)

整数指数を持つ区間演算子 ** は事前定義されており、次のような引数の組み合わせとしてユーザーが拡張することはできません。

(任意の INTERVAL 型、任意の INTEGER 型)

.IN. を除き、区間関係演算子は以下のリストのような引数の組み合わせとして事前定義されていますから、ユーザーが拡張することはできません。

(任意の INTERVAL 型、任意の INTERVAL 型)

(任意の INTERVAL 型、任意の REAL または INTEGER 型)

(任意の REAL または INTEGER 型、任意の INTERVAL 型)

区間関係演算子 .IN. は事前定義されており、次のような引数の組み合わせとして、ユーザーが拡張することはできません。

(任意の REAL または INTEGER 型、任意の INTERVAL 型)

コード例 2-8 の定義では、.IH. が (LOGICAL, INTERVAL(16)) オペランド用には事前定義されていないので、S1 と S2 インタフェースは正しい記述です。

コード例 2-8 区間の .IH. 演算子の拡張

```
math% cat ce2-8.f95
MODULE M
INTERFACE OPERATOR (.IH.)
    MODULE PROCEDURE S1
    MODULE PROCEDURE S2
END INTERFACE
CONTAINS
REAL FUNCTION S1(L, Y)
LOGICAL, INTENT(IN)       ::  L
INTERVAL(16), INTENT(IN) ::  Y
    S1 = 1.0
END FUNCTION S1
```

```
INTERVAL FUNCTION S2(R1, R2)
REAL, INTENT(IN) ::  R1
REAL, INTENT(IN) ::  R2
    S2 = [2.0]
END FUNCTION S2
END MODULE M

PROGRAM TEST
USE M
INTERVAL(16) :: X = [1, 2]
LOGICAL      :: L = .TRUE.
REAL         :: R = 0.1
PRINT *, 'L  .IH. X  = ', L  .IH. X
PRINT *, 'R1 .IH. R2 =', R1 .IH. R2
END PROGRAM TEST

math% f95 -xia ce2-8.f95
math% a.out
 L  .IH. X  =  1.0
 R1 .IH. R2 = [2.0,2.0]
```

コード例 2-9 の + 演算子の拡張は、(INTERVAL, INTERVAL) 型のオペランド用に事前定義されている組み込みの区間 (+) 演算子の定義を変更しようとしているので正しくありません。

コード例 2-9　組み込みの区間の (+) 演算子用法と衝突するユーザー定義のインタフェース

```
math% cat ce2-9.f95
MODULE M1
INTERFACE OPERATOR (+)
    MODULE PROCEDURE S4
END INTERFACE
CONTAINS
REAL FUNCTION S4(X, Y)
INTERVAL, INTENT(IN) ::  X
INTERVAL, INTENT(IN) ::  Y
    S4 = 4.0
END FUNCTION S4
END MODULE M1

PROGRAM TEST
USE M1
INTERVAL :: X = [1.0], Y = [2.0]
PRINT *, 'X + Y = ', X + Y
END PROGRAM TEST
```

```
math% f95 -xia ce2-9.f95

MODULE M1
       ^
"ce2-9.f95", 行 = 1, 桁 = 8: エラー：コンパイラがモジュール "M" でエラー
を検出しました。このモジュールにはモジュール情報ファイルは作成されません。

    MODULE PROCEDURE S4
                     ^
"ce2-9.f95", 行 = 9, 桁 = 22: エラー：この個別引用仕様 "S4" は、"+" の組
み込み使用と衝突しています。

USE M1
    ^
"ce2-9.f95", 行 = 20, 桁 = 5: エラー：モジュール "M1" にはコンパイラエ
ラーがあるため、USE 文を通してこのモジュールから獲得された宣言は不十分な可能
性があります。

f90comp: 23 ソース行
f90comp: 3 個のエラー 、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

コード例 2-10 では、.IH. は (INTERVAL(4),INTERVAL(8)) のオペランド用に事前定義されているので、以下の S1 インタフェースは正しくありません。

コード例 2-10 組み込みの.IH.用法と衝突するユーザー定義のインタフェース

```
math% cat ce2-10.f95
MODULE M
INTERFACE OPERATOR (.IH.)
    MODULE PROCEDURE S1
END INTERFACE
CONTAINS
INTERVAL FUNCTION S1(X, Y)
INTERVAL(4), INTENT(IN) ::  X
INTERVAL(8), INTENT(IN) ::  Y
    S1 = [1.0]
END FUNCTION S1
END MODULE M

PROGRAM TEST
USE M
INTERVAL(4) :: X = [1.0]
INTERVAL(8) :: Y = [2.0]
PRINT *, 'X .IH. Y = ', X .IH. Y
END PROGRAM TEST
math% f95 -xia ce2-10.f95

MODULE M
       ^
"ce2-10.f95", Line = 1, Column = 8: エラー：コンパイラがモジュール "M"
でエラーを検出しました。このモジュールにはモジュール情報ファイルは作成されま
せん。

    MODULE PROCEDURE S1
                     ^
"ce2-10.f95", Line = 3, Column = 22: エラー：この個別引用仕様 "S1"
は、"ih" の組み込み使用と衝突しています。

USE M
    ^
"ce2-10.f95", Line = 14, Column = 5:  エラー：モジュール "M" にはコン
パイラエラーがあるため、USE 文を通してこのモジュールから獲得された宣言は不十
分な可能性があります。

f95: コンパイル時間 0.190000 SECONDS
f95: 最大フィールド 4135778 10 進ワード
f95: 18 ソース行
f95: 3 個のエラー , 0 個の警告 , 0 個の他のメッセージ , 0 個の ANSI
```

組み込みの区間演算子を拡張する演算子関数の引数の数は、コード例 2-11 で示しているように、組み込みの演算子に必要なオペランド数と一致しなければなりません。

コード例 2-11 事前定義された区間演算子の引数の数の間違った変更

```
math% cat ce2-11.f95
MODULE M
INTERFACE OPERATOR (.IH.)
    MODULE PROCEDURE S1
END INTERFACE
CONTAINS
REAL FUNCTION S1(R)
REAL, INTENT(IN) :: R
    S1 = 1.0
END FUNCTION S1
END MODULE M

PROGRAM TEST
USE M
REAL :: R = 0.1
PRINT *, ' .IH. R = ', .IH. R
END PROGRAM TEST
math% f95 -xia ce2-11.f95

MODULE M
       ^
"ce2-11.f95", Line = 1, Column = 8: エラー:コンパイラがモジュール "M"
でエラーを検出しました。このモジュールにはモジュール情報ファイルは作成されま
せん。

    MODULE PROCEDURE S1
                     ^
"ce2-11.f95", Line = 3, Column = 22: エラー:個別引用仕様 "S1" は利用
者定義 2 項演算子の引用仕様宣言の内部にあるときは、ちょうど 2 個の仮引数をも
たなければなりません。

USE M
    ^
"ce2-11.f95", Line = 13, Column = 5: エラー:モジュール "M" にはコンパ
イラエラーがあるため、USE 文を通してこのモジュールから獲得された宣言は不十分
な可能性があります。

PRINT *, ' .IH. R = ', .IH. R
                       ^
"ce2-11.f95", Line = 15, Column = 24: エラー:予期しない構文:
"operand" が予期されるところに "." がありました。

f95: 16 ソース行
f95: 4 個のエラー , 0 個の警告 , 0 個の他のメッセージ , 0 個の ANSI
```

組み込みの区間二項演算子は、1 つの INTERVAL 引数をとる単項演算子関数を使用して拡張することはできません。

コード例 2-12 では、「+」は区間オペランド用に定義済みなので、S1 インタフェースは正しくありません。

コード例 2-12 組み込みの単項「+」用法と衝突するユーザー定義のインタフェース

```
math% cat ce2-12.f95
MODULE M
INTERFACE OPERATOR (+)
    MODULE PROCEDURE S1
END INTERFACE
CONTAINS
REAL FUNCTION S1(X)
    INTERVAL, INTENT(IN) :: X
    S1 = 1.0
END FUNCTION S1
END MODULE M

PROGRAM TEST
USE M
INTERVAL :: X = 0.1
PRINT *, ' + X = ', + X
END PROGRAM TEST

math% f95 -xia ce2-12.f95

MODULE M
       ^
"ce2-12.f95", Line = 1, Column = 8: エラー:コンパイラがモジュール "M"
でエラーを検出しました。このモジュールにはモジュール情報ファイルは作成されま
せん。

    MODULE PROCEDURE S1
                     ^
"ce2-12.f95", Line = 3, Column = 22: エラー:この個別引用仕様 "S1"
は、"+" の組み込み使用と衝突しています。

USE M
    ^
"ce2-12.f95", Line = 13, Column = 5: エラー:モジュール "M" にはコンパ
イラエラーがあるため、USE 文を通してこのモジュールから獲得された宣言は不十分
な可能性があります。

f95: 16 ソース行
f95: 3 個のエラー , 0 個の警告 , 0 個の他のメッセージ , 0 個の ANSI
```

一般的なインタフェースブロックの中では、INTERFACE 文の中で指定した一般的な名前が組み込みの区間サブプログラムの名前であれば、特定のユーザー定義サブプログラムは組み込みサブプログラムの定義済みの意味を拡張します。

同じ一般名を持つサブプログラムへのすべての参照は、あいまいであってはなりません。

組み込みのサブプログラムは、そのインタフェース定義もまた一般的なインタフェースブロックで指定された特定の組み込みサブプログラムの 1 つの集まりとして扱われます。

コード例 2-13 組み込み区間関数 WID の正しい拡張

```
math% cat ce2-13.f95
MODULE M
INTERFACE WID
    MODULE PROCEDURE S1
    MODULE PROCEDURE S2
END INTERFACE
CONTAINS
REAL FUNCTION S1(X)
REAL, INTENT(IN) :: X
    S1 = 1.0
END FUNCTION S1
INTERVAL FUNCTION S2(X, Y)
INTERVAL, INTENT(IN) :: X
INTERVAL, INTENT(IN) :: Y
    S2 = [2.0]
END FUNCTION S2
END MODULE M

PROGRAM TEST
USE M
INTERVAL :: X = [1, 2], Y = [3, 4]
REAL     :: R = 0.1
PRINT *, WID(R)
PRINT *, WID(X, Y)

END PROGRAM TEST
math% f95 -xia ce2-13.f95
math% a.out
 1.0
 [2.0,2.0]
```

コード例 2-14 は正しいコードです。

コード例 2-14 組み込み区間関数 ABS の正しい拡張

```
math% cat ce2-14.f95
MODULE M
INTERFACE ABS
    MODULE PROCEDURE S1
END INTERFACE
CONTAINS
INTERVAL FUNCTION S1(X)
INTERVAL, INTENT(IN) :: X
    S1 = [-1.0]
END FUNCTION S1
END MODULE M
PROGRAM TEST
USE M
INTERVAL :: X = [1, 2]
PRINT *, ABS(X)

END PROGRAM TEST
math% f95 -xia ce2-14.f95
math% a.out
 [-1.0,-1.0]
```

コード例 2-15 は正しいコードです。

コード例 2-15 組み込み区間関数 MIN の正しい拡張

```
math% cat ce2-15.f95
MODULE M
INTERFACE MIN
    MODULE PROCEDURE S1
END INTERFACE
CONTAINS
INTERVAL FUNCTION S1(X, Y)
    INTERVAL(4), INTENT(IN) :: X
    INTERVAL(8), INTENT(IN) :: Y
    S1 = [-1.0]
END FUNCTION S1
END MODULE M

PROGRAM TEST
USE M
INTERVAL(4) :: X = [1, 2]
INTERVAL(8) :: Y = [3, 4]
REAL        :: R = 0.1
PRINT *, MIN(X, Y)
END PROGRAM TEST
math% f95 -xia ce2-15.f95
math% a.out
 [-1.0,-1.0]
```

最大幅要求の評価を持つ拡張演算子

コード例 2-16 は、組み込みの区間演算子の事前定義バージョンと拡張バージョンを呼び出す場合の、最大幅要求式処理がどのように発生するかを示しています。

コード例 2-16 組み込み区間演算子の事前定義バージョンを呼び出す場合の最大幅要求式の処理

```
math% cat ce2-16.f95
MODULE M
INTERFACE OPERATOR (.IH.)
    MODULE PROCEDURE S4
END INTERFACE
CONTAINS
INTERVAL FUNCTION S4(X, Y)
    COMPLEX, INTENT(IN) :: X
    COMPLEX, INTENT(IN) :: Y
    S4 = [0]
END FUNCTION S4
END MODULE M
USE M
INTERVAL :: X = [1.0]
REAL     :: R = 1.0
COMPLEX  :: C = (1.0, 0.0)
X = (R-0.1).IH.(R-0.2)   ! 両方の引数が最大幅要求で、
                         ! 組み込みの区間演算子 .IH. が呼び出される。

X = X .IH. (R+R)         ! 両方の引数が最大幅要求で、
                         ! 組み込みの区間演算子 .IH. が呼び出される。

X = X .IH. (R+R+X)       ! 第 2 引数が最大幅要求で、
                         ! 組み込みの区間演算子 .IH. が呼び出される。

X = (R+R) .IH. (R+R+X)   ! 両方の引数が最大幅要求で、
                         ! 組み込みの区間演算子 .IH. が呼び出される。

X = C .IH. (C+R)         ! 最大幅要求なしで、s4 が呼び出される。
END

math% f95 -xia ce2-16.f95
math% a.out
```

コード例 2-17 は、ユーザー定義の演算子を呼び出す場合に最大幅要求式の処理がどのように発生するか示しています。

コード例 2-17 ユーザー定義演算子を呼び出す場合の最大幅要求式の処理

```
math% cat ce2-17.f95
MODULE M
INTERFACE OPERATOR (.AA.)
    MODULE PROCEDURE S1
    MODULE PROCEDURE S2
```

```
END INTERFACE
CONTAINS
INTERVAL FUNCTION S1(X, Y)
INTERVAL, INTENT(IN) :: X
REAL, INTENT(IN)     :: Y
    PRINT *, 'S1 is invoked.'
    S1 = [1.0]
END FUNCTION S1
INTERVAL FUNCTION S2(X, Y)
INTERVAL, INTENT(IN) :: X
INTERVAL, INTENT(IN) :: Y
    PRINT *, 'S2 is invoked.'
    S2 = [2.0]
END FUNCTION S2
END MODULE M
USE M
INTERVAL :: X = [1.0]
REAL     :: R = 1.0
X = X .AA. R+R     ! S1 is invoked
X = X .AA. X       ! S2 is invoked
END


math% f95 -xia ce2-17.f95

    MODULE PROCEDURE S1
                     ^
"ce2-17.f95", 行 = 5, 桁 = 22: 警告：最大幅要求の評価は、ユーザー定義の引
数に適用されません。

USE M
    ^
"ce2-17.f95", 行 = 22, 桁 = 5: 警告：最大幅要求の評価は、ユーザー定義の引
数に適用されません。

f90comp: 28 ソース行
f90comp: 0 個のエラー , 2 個の警告 , 0 個の他のメッセージ , 0 個の ANSI
math% a.out
 S1 が呼び出されました。
 S2 が呼び出されました。
```

INTERVAL (X [,Y, KIND])

解説： INTERVAL 型へと変換します。

クラス： 要素別処理関数

引数：

X は、INTEGER、REAL、または、INTERVAL 型です。

Y (オプション) は INTEGER または REAL 型です。X が INTERVAL 型であれば、Y は指定してはいけません。

KIND (オプション) はスカラー INTEGER の初期値式です。

結果の特性： INTERVAL

KIND が存在する場合は、結果の KTPV の決定にその値が使われます。それ以外は、結果の KTPV はデフォルトで使われる区間の KTPV と同じです。

包含：

X が区間の場合は包含が保証されます。たとえば、次の場合、

```
INTERVAL(16):: X
```

INTERVAL (X, KIND=4) の結果には INTERVAL X が含まれます。

しかし、REAL(8) :: X, Y であれば、INTERVAL(X,Y, KIND=4) の結果は内部的な区間 X .IH. Y を含むとは限りません。この理由は、X と Y が REAL 式でもよく、それらの値は保証されないからです。

INTERVAL 構成子は、必ずしも同じ終了点を持つ INTERVAL 文字定数の値を含むわけではありません。たとえば、INTERVAL(1.1,1.3) は必ずしも外部値 ev([1.1, 1.3]) = [1.1, 1.3] を含むわけではありません。その理由は、REAL 定数の内部値が未知の正確さを持つ近似値であるからです。

常に 2 つの REAL 値を含む区間を構築するためには、78 ページのコード例 2-18 で示しているように、区間包演算子 .IH. を使います。

結果値： 区間の結果値は 1 つの有効な区間です。

Y が存在せず、X が区間である場合、INTERVAL(X[,KIND]) は X を含む 1 つの区間であり、INTERVAL(X[,KIND]) は左右の終了点 [XL,XU] を持つ 1 つの区間となります。

この場合、

XL = REAL(INF(X) [,KIND]) は丸め切り下げにより、XL .LE. INF(X) となり、

また、

XU = REAL(SUP(X) [,KIND]) は丸め切り上げにより、XU .GE. SUP(X) となります。

X と Y が共に存在する (このため、区間ではない) 場合、INTERVAL(X,Y[,KIND]) は左右の終了点がそれぞれ REAL(X[,KIND]) と REAL(Y[,KIND]) に等しい終了点を持つ区間となります。

注 - このケースでは、有向の丸めは指定されません。このため、包含は提供されません。

以下の 2 つのケースでは [-inf,inf] が返されます。

- X と Y が共に存在し、Y が X より小さい場合。
- X または Y かその両方が算術整数または実数を表さない (たとえば、1 つまたは両方の実引数が NaN である) 場合。

最大幅要求のスコープ制限

組み込みの 区間構成子関数は以下の 2 つの用途で用いられます。

- INTERVAL 変数または式の KTPV 変換を行うため。
- 混合モード 区間式の評価から非区間式を隔離するため。

所与の非区間 (REAL または INTEGER) 式 *EXP* について、次のコードは、

```
INTERVAL Y
REAL R
R = EXP
Y = R
```

次のコードと同じです。

```
INTERVAL Y
Y = INTERVAL(EXP)
```

これは、次のコードとは異なります。

```
INTERVAL Y
Y = EXP
```

後のコードは、*EXP* を 1 つの区間式として評価することになります。最初の 2 つの部分コードでは、式 *EXP* は非区間式として評価され、その結果が縮退区間の構築に使われています。

2 つの引数 EXP_1 と EXP_2 を使えば、区間(EXP_1, EXP_2)は両方の式を最大幅要求式処理から隔離し、その式の非区間評価結果と同じ終了点を持つ 1 つの区間を構築します。

KIND パラメータを含めると、結果の KTPV を制御できるようになります。これは多くの場合、明示的な KTPV 変換が必要な -strict 式処理のもとで必要です。

非区間引数を持つ組み込みの区間関数の扱いには注意してください。区間の包含が必要な場合は、78 ページのコード例 2-18 で示しているように、区間包演算子 .IH. を使ってください。

区間構成子は INTERVAL と REAL または INTERGER 式間の境界として動作します。この境界の非 INTERVAL 側では正確性 (このため、さらに包含も) の保証を強制することができません。

コード例 2-18 .IH. 演算子を使用した包含

```
math% cat ce2-18.f95
REAL(16) :: A, B
INTERVAL :: X1, X2
PRINT *, "Press Control/D to terminate!"
WRITE(*, 1, ADVANCE='NO')
READ(*, *, IOSTAT=IOS) A, B
DO WHILE (IOS >= 0)
    PRINT *, " FOR A =", A, ", AND B =", B

    ! 最大幅要求コード
     X1 = A .IH. B

    ! 同等の厳密コード
    X2 = INTERVAL(INTERVAL(A, KIND=16) .IH. INTERVAL(B, KIND=16))
    IF (X1 .SEQ. X2)  PRINT *, 'Check.'
    PRINT *, 'X1 = ', X1
    WRITE(*, 1, ADVANCE='NO')
    READ(*, *, IOSTAT=IOS)  A, B
END DO
1  FORMAT(" A, B = ")
END
math% f95 -xia ce2-18.f95
math% a.out
 Control/D to terminate!
 A, B = 1.3 1.7
 FOR A = 1.3 , AND B = 1.7
```

```
  X1 =  [1.2999999999999998,1.7000000000000002]
  A, B = 0.0  0.2
  FOR A = 0.0E+0 , AND B = 0.2
  X1 =  [0.0E+0,0.20000000000000002]
  A, B = ^d
```

組み込みの区間構成子関数の使い方の詳細については、75 ページの「INTERVAL (X [,Y, KIND])」を参照してください。

組み込みの区間構成子関数の KTPV 個別名

表 2-12 に示しているように、組み込みの区間構成子関数は、オプションの KIND パラメータを使用しない KTPV 個別名を使用して呼び出すことができます。

表 2-12 組み込みの区間構成子関数用の KTPV 個別式

KTPV個別名	結果
DINTERVAL(X[,Y])	INTERVAL(X[,Y], KIND = 8)、または、INTERVAL(X[,Y])
SINTERVAL(X[,Y])	INTERVAL(X[,Y], KIND = 4)
QINTERVAL(X[,Y])	INTERVAL(X[,Y], KIND = 16)

組み込み区間構成子関数の変換例

この節の 3 つの例は、組み込みの区間構築子を使って、REAL から INTERVAL 型データ項目に変換する方法を示しています。コード例 2-19 は、区間構築子の REAL 式引数が REAL 演算を使って評価されるので最大要求幅式の評価から隔離されることを示しています。

コード例 2-19 区間変換

```
math% cat ce2-19.f95
REAL        :: R = 0.1, S = 0.2, T = 0.3
REAL(8)     :: R8 = 0.1D0, T1, T2
INTERVAL(4) :: X, Y
INTERVAL(8) :: DX, DY
R = 0.1
Y  = INTERVAL(R, R, KIND=4)
X  = INTERVAL(0.1, KIND=4) ! 7 行目
IF ( X == Y ) PRINT *, 'Check1'
X  = INTERVAL(0.1, 0.1, KIND=4) ! 10 行目
IF ( X == Y ) PRINT *, 'Check2'
T1 = R+S
```

```
T2 = T+R8
DY = INTERVAL(T1, T2)
DX = INTERVAL(R+S, T+R8) ! 15 行目
IF ( DX == DY ) PRINT *, 'Check3'
DX = INTERVAL(Y, KIND=8) ! 17 行目
IF (Y .CEQ. INTERVAL(0.1, 0.1, KIND=8)) PRINT *, 'Check4'
END

math% f95 -xia ce2-19.f95
math% a.out
 Check1
 Check2
 Check3
 Check4
```

コード例 2-19 注記：

- 7、10 行目：区間 X には、実定数 0.1 の内部表現と同じ両終了点を持つ縮退した区間が代入されます。
- 15 行目：区間 DX には、R+S と T+R8 のそれぞれの REAL 式の結果と同じ左右の終了点を持つ 1 つの区間が代入されます。
- 17 行目：区間 Y は、KTPV-8 を含む区間へと変換されます。

コード例 2-20 は、区間構成子を使って、Y の終了点が所与の区間、X の要素とならない、可能な最小区間 Y を構築する方法を示しています。

コード例 2-20 所与の実数を含む狭い区間を作成する

```
math% cat ce2-20.f95
INTERVAL :: X = [10.E-10,11.E+10]
INTERVAL :: Y
Y = INTERVAL(-TINY(INF(X)), TINY(INF(X))) + X
PRINT *, X .INT. Y
END
math% f95 -xia ce2-20
math% a.out
  T
```

所与の区間 X について、条件 X .INT. Y を満たす鋭い区間 Y が構築されます。内部集合関係に関する情報については、58 ページの「内部：(X .INT. Y)」を参照してください。

81 ページのコード例 2-21 は、区間構成子がどのような場合に区間 [-INF,INF] と [MAX_FLOAT,INF] を返すかを示しています。

コード例 2-21 INTERVAL(NaN)

```
math% cat ce2-21.f95
REAL :: R = 0., S = 0.
T = R/S                          ! 2 行目
PRINT *, T
PRINT *, INTERVAL(T, S)          ! 4 行目
PRINT *, INTERVAL(T, T)          ! 5 行目
PRINT *, INTERVAL(2., 1.)        ! 6 行目
PRINT *, INTERVAL(1./R)          ! 7 行目
END

math% f95 -xia ce2-21.f95
math% a.out
 NaN
 [-Inf,Inf]
 [-Inf,Inf]
 [-Inf,Inf]
 [1.7976931348623157E+308,Inf]
```

コード例 2-21 注記：

- 2 行目：変数 T には NaN の値が代入されます。
- 4、5 行目：区間構成子の 1 つの引数は NaN であり、結果は区間 [-INF, INF] です。
- 6 行目：無効な区間 [2,1] の代わりに、区間 [-INF, INF] が構築されます。
- 7 行目：[MAX_FLOAT,INF] が構築されます。この区間には区間 [INF,INF] が含まれます。内部表現のための区間選択の議論については、123 ページの「参考文献」で引用した補足文献 [8] を参照してください。

組み込みの一般区間関数用の個別名

組み込みの一般区間関数用の f95 個別名は、末尾が組み込み関数の一般名となり、V で始まり、その後ろに、INTERVAL(4)、INTERVAL(8)、INTERVAL(16) 型の引数用にそれぞれ S、D、または Q が続きます。

f95 では、INTERVAL(16) データ型用には次の個別名組み込み関数だけがサポートされています。

VQABS、VQAINT、VQANINT、VQINF、VQSUP、VQMID、VQMAG、VQMIG、VQISEMPTY

非区間プログラムでの名前空間の衝突を回避するために、コマンド行オプションによる方法でのみ個別名が利用できるようになっています。

- -xinterval
- -xinterval=strict、または、-xinterval=widestneed
- マクロ-xia、-xia=strict、または、-xia=widestneed

より詳細な情報については、44 ページの「区間のコマンド行オプション」を参照してください。

サポートされるすべての組み込み関数は個別名を持ちます。たとえば、表 2-13 では、ABS 組み込み関数の区間バージョンの名前を一覧表示しています。

表 2-13 組み込みの区間 ABS 関数用の固有の名前

個別名	引数	結果
VSABS	INTERVAL(4)	INTERVAL(4)
VDABS	INTERVAL(8)	INTERVAL(8)
VQABS	INTERVAL(16)	INTERVAL(16)

これ以外の個別名組み込み関数は、117 ページの「組み込み関数」に一覧表示されています。

INTERVAL

この節では f95 により認識される INTERVAL 文を解説します。ここでは、考えられる制約と例を交えて、各文の構文と解説を示します。

型の宣言

INTERVAL 名前付き定数、変数、関数の結果を宣言するには、INTERVAL 文を使用します。INTERVAL は標準の数値型宣言文と同じ構文と意味論を持つ組み込みの数値型宣言文です。INTERVAL 文を使った用法では、他の数値型宣言用法に存在するのと同じ指定が利用できます。

解説：宣言は、INTERVAL、INTERVAL(4)、INTERVAL(8)、INTERVAL(16)のいずれかにすることができます。

INTERVAL

次のような宣言では、

```
INTERVAL :: W
```

変数 W は、デフォルトの 8 の区間 KTPV を持ち、16 バイトの連続するメモリーを占有します。Sun Forte Developer の Fortran 95 では、デフォルトの区間 KTPV は、-xtypemap または -r8const のような任意のコマンド行オプションにより変更されることはありません。

INTERVAL は構造型名としては使用できません。たとえば、コード例 2-22 のコードは正しくありません。

コード例 2-22　　間違った構造型：INTERVAL

```
math% cat ce2-22.f95
TYPE INTERVAL
    REAL :: INF, SUP
END TYPE INTERVAL

END
math% f95 -xia ce2-22.f95

TYPE INTERVAL
     ^
"ce2-22.f95", Line = 1, Column = 6: ERROR: 構造型の型名は、組み込み型
INTERVAL の名前と同じであってはなりません。

f95: 5 ソース行
f95: 1 E1 個のエラー、0 個の警告、0 個の他のメッセージ、0 個の ANSI
```

n ∈{4, 8, 16} 用の INTERVAL(*n*)

次のような宣言では、

```
INTERVAL(n) :: W
```

変数 W は、KTPV = *n* の KTPV を持ち、*2n* バイトの連続するメモリーを占有します。

84 ページのコード例 2-23 は、異なる KTPV を持つ区間変数の宣言を示しています。最大幅要求値と厳密値の整列も示しています。

コード例 2-23 異なる KTPV を持つ区間の宣言

```
math% cat ce2-23.f95
INTERVAL(4)  :: X1, X2
INTERVAL(8)  :: Y1, Y2
INTERVAL(16) :: Z1, Z2
REAL(8)      :: D = 1.2345

! 最大幅要求コード
 X1 = D
 Y1 = D
 Z1 = D


! 同等の厳密コード
X2 = INTERVAL(INTERVAL(D, KIND=8), KIND=4)
Y2 = INTERVAL(D, KIND=8)
Z2 = INTERVAL(D, KIND=16)

IF (X1 == X2) PRINT *, 'Check1'
IF (Y1 == Y2) PRINT *, 'Check2'
IF (Z1 == Z2) PRINT *, 'Check3'
END

math% f95 -xia ce2-23.f95
math% a.out
 Check1
 Check2
 Check3
```

コード例 2-24 は、区間変数の宣言と初期化について示しています。区間定数を別の方法で表現するには、34 ページの「区間定数」を参照してください。

コード例 2-24 区間変数の宣言と初期化

```
math% cat ce2-24.f95
INTERVAL :: U = [1, 9.1_8], V = [4.1]

! 最大幅要求コード
INTERVAL :: W1 = 0.1_16

! 同等の厳密コード
INTERVAL :: W2 = [0.1_16]

PRINT *, U, V
IF (W1 .SEQ. W2) PRINT *, 'Check'
END

math% f95 -xia ce2-24.f95
math% a.out
 [1.0,9.1000000000000015] [4.0999999999999996,4.1000000000000006]
 検証
```

任意の初期化を伴う宣言文の中では、データ式の型が記号名の型と一致しない場合、型変換が実行されます。

コード例 2-25 区間配列の宣言

```
math% cat ce2-25.f95
INTERVAL(4) :: R(5), S(5)
INTERVAL :: U(5), V(5)
INTERVAL(16) :: X(5), Y(5)
END
math% f95 -xia ce2-25.f95
math% a.out
```

DATA 文

構文

区間変数を含む DATA 文の構文は、区間変数が区間定数を用いて初期化される点を除けば、他の数値データ型のものと同じです。

コード例 2-26 区間変数を含む DATA 文

```
math% cat ce2-26.f95
INTERVAL X
DATA X/[1,2]/
END

math% f95 -xia ce2-26.f95
math% a.out
```

EQUIVALENCE 文

任意の区間変数または配列は、次の制限付きで EQUIVALENCE 文の中に現れてもかまいません。つまり、結合対応が区間変数または配列を含む場合、結合対応内部のすべてのオブジェクトは、78 ページのコード例 2-18 で示しているように、同じ型を持たなければなりません。これは区間固有の制約ではなく、Fortran 規格の制約です。

FORMAT 文

構文

区間用の反復可能な編集記述子は次のとおりです。

D ∈ {E, EN, ES, G}

である場合、

F*w.d*、VF*w.d*、D*w.d*、VD*w.d*、D*w.d*E*e*、VD*w.d*E*e*、Y*w.d*、Y*w.d*E*e*

w と *e* は、非ゼロの符号なし整数定数を、*d* は符号なし整数定数を表します。

編集記述子を使って区間データを処理するための記述子の指定方法については、93 ページの「入力と出力」を参照してください。また、非区間データを用いた標準編集記述子の動作については、Fortran のリファレンスマニュアルを参照してください。

すべての標準 Fortran の編集記述子は区間を受け入れます。区間専用バージョンの標準 E、F、G の編集記述子には接頭辞として V を付けることができます。

コード例 2-27 で示しているように、区間データを読み込みまたは書き出しする場合、反復不能の編集記述子を変更する必要はありません。

コード例 2-27 反復不能の編集記述子の例

```
math% cat ce2-27.f95
INTERVAL :: X = [-1.3, 1.3]
WRITE(*, '(SP, VF20.5)') X
WRITE(*, '(SS, VF20.5)') X
END
math% f95 -xia ce2-27.f95
math% a.out
 [-1.30001,+1.30001]
 [-1.30001, 1.30001]
```

解説

反復可能編集記述子

反復可能な編集記述子、E、F、EN、ES、G、VE、VEN、VES、VF、VG、Y は、区間データの編集方法を指定します。

コード例 2-28 は、区間固有の編集記述子の例を含んでいます。

コード例 2-28 区間固有の編集記述子を使った FORMAT 文

```
math% cat ce2-28.f95

10   FORMAT(VE22.4E4)
20   FORMAT(VEN22.4)
30   FORMAT(VES25.5)
40   FORMAT(VF25.5)
50   FORMAT(VG25.5)
60   FORMAT(VG22.4E4)
70   FORMAT(Y25.5)

END

math% f95 -xia ce2-28.f95
math% a.out
```

追加的な例については、93 ページの「入力と出力」を参照してください。

FUNCTION (外部)

コード例 2-29 で示しているように、区間外部関数と非区間外部関数との間には、関数と引数の定義の中で INTERVAL 型 (INTERVAL、INTERVAL(4)、INTERVAL(8)、または、INTERVAL(16)) を使用する点を除けば、他に違いはありません。

コード例 2-29 デフォルトの区間関数

```
math% cat ce2-29.f95
PROGRAM ce2_29
INTERVAL :: X, Y
EXTERNAL SQR
INTERVAL :: SQR

Y = [4.0]
X = SQR(Y)
print *, "X = ", X
print *, "KIND(X) =", KIND(X)
END

INTERVAL FUNCTION SQR (A)           ! 1 行目
INTERVAL :: A
SQR = A**2
RETURN
END

math% f95 -xia ce2-29.f95
math% a.out
 X =  [16.0,16.0]
 KIND(X) = 8
```

1 行目のデフォルトの INTERVAL は、コード例 2-30 で示しているように、明示的な表現にすることができます。

コード例 2-30 明示的な INTERVAL(16) 関数宣言

```
math% cat ce2-30.f95
PROGRAM ce2_30
INTERVAL(16) :: X, Y
EXTERNAL SQR
INTERVAL(16) :: SQR

Y = [4.0]
X = SQR(Y)
print *, "X = ", X
print *, "KIND(X) =", KIND(X)
END

INTERVAL(16) FUNCTION SQR (A)          ! 1 行目
INTERVAL(16) :: A
SQR = A**2
RETURN
END
math% f95 -xia ce2-30.f95
math% a.out
 X =  [16.0,16.0]
 KIND(X) = 16
```

IMPLICIT 属性

区間名のデフォルト型を指定するには、IMPLICIT 属性を使用してください。

```
IMPLICIT INTERVAL (8) (V)
```

INTRINSIC 文

実際の引数として引き渡せるようにするために組み込みの関数を宣言するには、INTRINSIC 文を使用してください。

コード例 2-31 組み込みの関数宣言

```
INTRINSIC VDSIN, VDCOS, VSSIN, VSCOS
X = CALC(VDSIN, VDCOS, VSSIN, VSCOS)
```

注 - INTRINSIC 文では、一般関数の個別名が使用され、実際の引数として引き渡されなければなりません。81 ページの「組み込みの一般区間関数用の個別名」と 117 ページの「組み込み関数」を参照してください。

次の組み込みの区間関数は、一般関数ですから、実際の引数として引き渡すことはできません。

NDIGITS、INTERVAL

NAMELIST 文

NAMELIST 文は区間をサポートします。

コード例 2-32 NAMELIST での INTERVAL

```
CHARACTER(8) :: NAME
CHARACTER(4) :: COLOR
INTEGER      :: AGE
INTERVAL(4)  :: HEIGHT
INTERVAL(4)  :: WEIGHT
NAMELIST /DOG/ NAME, COLOR, AGE, WEIGHT, HEIGHT
```

PARAMETER 属性

PARAMETER 属性は、区間の初期化結果を名前付き定数 (PARAMETER) に代入するのに使用します。

構文

PARAMETER (*p* = *e* [, *p* = *expr*]...)

p 区間英字名

expr 区間定数式

= *e* の値を記号名 *p* に代入する

解説

記号名 *p* と定数式 *expr* は共に INTERVAL 型を持たなければなりません。

定数式では、整数の累乗に対する累乗法は許されます。

最大幅要求式処理のもとでは、区間の名前付き定数定義の中で、混合モードの区間式の評価がサポートされます。定数式の型が名前付き定数の型と一致しない場合は、最大幅要求式処理のもとで型変換が実行されます。

注 - f95 では、非区間定数式は後続の混合モード区間式の中での使用を考慮せずにコンパイル時に評価されます。これらの式は最大幅要求式処理のスコープ外にあります。このため、非区間の名前付き定数の値の設定に用いられる点式の値を含むための要件は存在しません。混合モードの区間式の中で非区間の名前付き定数が現れる度に、ユーザーが気付くようコンパイル時の警告メッセージが出力されます。Fortran 規格で定義された名前付き定数はより適切には、読み取り専用変数と呼ばれます。読み取り専用変数に関連付けられた外部値は存在しません。

標準 Fortran 95 では、名前付き定数は、区間定数の最大下限と最小上限の表現には使用できません。この制約がこのリリースで強制されないのは、既知のエラーです。

コード例 2-33 非区間の PARAMETER 属性での定数式

```
math% cat ce2-33.f95
REAL(4), PARAMETER      :: R4 = 0.1
INTERVAL(4), PARAMETER  :: I4  = 0.1
INTERVAL(16), PARAMETER :: I16 = 0.1
INTERVAL                :: XR, XI
XR = R4
XI = I4
IF ((.NOT.(XR.SP.I16)).AND. (XI.SP.I16)) PRINT *, 'Check.'
END
math% f95 -xia ce2-33.f95
math% a.out
 Check.
```

注 - XR は、1/10 を含みませんが、XI の場合は含みます。

Fortran 95 形式の POINTER

ポインタを使用して区間にアクセスすることができます。

コード例 2-34 区間ポインタ

```
math% cat ce2-34.f95
INTERVAL, POINTER :: PX
INTERVAL, TARGET  :: X
X = [0.1,0.3]
PX => X
PRINT*, X
PRINT*, PX
END
math% f95 -xia ce2-34.f95
math% a.out
 [0.09999999999999991,0.30000000000000005]
 [0.09999999999999991,0.30000000000000005]
```

パラメータ付き区間式の宣言と評価には文関数を使用することができます。その場合、非区間文関数の制約が適用されます。

コード例 2-35 区間文関数

```
math% cat ce2-35.f95
INTERVAL :: X, F
F(X) = SIN(X)**2 + COS(X)**2
IF(1 .IN. F([0.5])) PRINT *, 'Check'
END
math% f95 -xia ce2-35.f95
math% a.out
 Check
```

型宣言文

型宣言文は、変数並びの中の変数のデータ型を指定します。オプションとして、型宣言文は配列の次元を指定し、値を使用して初期化します。

構文

構文は、型が INTERVAL、INTERVAL(4)、INTERVAL(8)、INTERVAL(16)の INTERVAL 型指定子のいずれかである点を除けば、非区間の数値データ型の場合と同じです。

解説

型宣言文の特性は、INTERVAL 型についても、他の数値データ型の場合と同じです。

制約

非 INTERVAL の数値型の場合と同じです。

コード例 2-36 INTERVAL の型宣言文

```
math% cat ce2-36.f95

INTERVAL     :: I,J = [0.0]
INTERVAL(16) :: K = [0.1, 0.2_16]
INTERVAL(16) :: L = [0.1]

END
math% f95 -xia ce2-36.f95
math% a.out
```

例の注記：

- J は、[0.0] に初期化されます。
- K は、[0.1, 0.2] を含む区間に初期化されます。
- L は、[0.1] を含む区間に初期化されます。

WRITE 文

WRITE 文は、非 INTERVAL 型の変数が処理されるのと同じ方法で、区間変数を受け入れて入力/出力の並びを処理します。定義済みの区間編集記述子を使えば、区間データの書式化記述を行うことができます。変数群要素並びの WRITE 文は区間をサポートします。

READ 文

READ 文は、非 INTERVAL 型の変数が処理されるのと同じ方法で、区間変数を受け入れて入力/出力の並びを処理します。

入力と出力

区間入力/出力を実行するプロセスは他の非区間データ型の場合と同じです。

外部表現

x が、並びによる、または、書式化された入力/出力を用いて読み書きできる外部 (小数) 数であるものとします。内部的な近似値と外部値との区別に関しては、33 ページの「Fortran 拡張」以下の各節を参照してください。このような数は外部区間または終了点のいずれかの表現に使えます。外部区間には、次に示す 3 つの表示可能な形式があります。

- `[X_inf, X_sup]` は、算術区間 $[\underline{x}, \bar{x}]$ を表します。
- `[X]` は、縮退した算術区間 $[x, x]$ または [x] を表します。
- `X` は、非縮退算術区間 $[x] + [-1,+1]_{uld}$ (最終桁の単位) を表します。この形式は、単数表現であり、ここでは区間の構築に最終小数桁が使われます (`Y` 編集記述子を参照してください)。この形式では後続のゼロが重要な意味を持ちます。このため、`0.10` は区間 `[0.09, 0.11]` を表し、`100E-1` は区間 `[9.9, 10.1]` を表し、また、`0.10000000` は区間 `[0.99999999, 0.100000001]` を表します。

正または負の不定区間終了点は、マイナスまたはオプションのプラス記号の接頭辞を持つ大文字と小文字を区別しない文字列 `INF` または `INFINITY` としての入力/出力です。

空の区間は、角括弧 `[...]` で囲まれた大文字と小文字を区別しない文字列 `EMPTY` としての入力/出力です。文字列 `EMPTY` は、その前後を空白とすることができます。

11 ページのコード例 1-6 を使えば、拡張区間を試すことができます。

さらに詳細については、49 ページの「算術演算子 +、-、*、/」を参照してください。

入力

入力時点では、任意の外部区間 *X* またはその構成要素である *X_inf* と *X_sup* を、D*w.d* 編集記述子が受け入れる任意の方法で書式化することができます。そこで、*input-field*、$input\text{-}field_1$、$input\text{-}field_2$ が、それぞれ、D$w'.d$、D$w_1.d$、D$w_2.d$ 編集記述子用の有効な入力フィールドであるものとします。

w は区間入力フィールドの幅であるものとします。入力時に、*w* はゼロより大きくなければいけません。すべての区間編集記述子は、以下の 3 つの形式のどれか 1 つの形式で区間データの入力を受け入れます。

- [*input-field1, input-field2*]、このケースでは、$w_1 + w_2 = w - 3$、または、$w = w_1 + w_2 + 3$ です。

- [*input-field*]、このケースでは、*w′* = *w* -2、または、*w* = *w′* +2 です。
- *input-field*、このケースでは、*w′* = *w* です。

最初の形式 (角括弧で囲まれ、コンマで区切られた 2 つの数) は、馴染みのある [*inf, sup*] 表現です。

2 番目の形式 (角括弧で囲まれた単一の数) は点、または縮退した区間を表します。

3 番目の形式 (角括弧なし) は区間の単数形式であり、ここでは区間幅の決定に最終表示桁が用いられます。101 ページの「Y 編集記述子を用いた単数編集」を参照してください。さらに詳細な情報については、M.Schulte、V.Zelov、G.W.Walster、D.Chiriaev 著、「Single-Number Interval I/O」、『Developments in Reliable Computing』、T. Schulte他 (Kluwer 1999年) を参照してください。

最大下限が内部的に表現できない場合は、同じものよりも小さいことがわかっている内部的な近似値へと丸め切り下げが行われます。最小下限が内部的に表現できない場合は、入力値と同じものよりも大きいことがわかっている内部的な近似値へと丸め切り上げが行われます。縮退した区間が内部的に表現できない場合は、丸め切り下げまたは丸め切り上げにより、入力値と同じものを含むことがわかっている内部的な INTERVAL 近似値が形成されます。

並びによる入力

入力の並び項目が区間である場合、入力記録内部の対応する要素は外部の区間またはヌル値でなければなりません。

外部の区間値は、区間、REAL または INTEGER 文字定数と同じ形式を持つことができます。区間の値が角括弧 [...] で囲まれない REAL または INTEGER 文字定数の形式を持つ場合、外部の区間は単数区間表現 ([x] + $[-1,1]_{uld}$ －最終桁での単位) を使って翻訳されます。

[*inf, sup*] 入力形式を使う場合は、最大下限 (infimum) とカンマの間、または、カンマと最小上限 (supremum) の間で記録終了が発生する可能性があります。

2 つの連続するコンマにより指定されたヌル値は、対応する区間の並び項目が変更されないことを意味します。

注 - 区間の最大下限または最小上限用にヌル値を使用してはいけません。

コード例 2-37 並び入力/出力コード

```
math% cat ce2-37.f95
INTERVAL, DIMENSION(6) :: X
INTEGER I
DO I = LBOUND(X, 1), UBOUND(X, 1)
    READ(*, *) X(I)
    WRITE(*, *) X(I)
END DO
END
math% f95 -xia ce2-37.f95
math% a.out
1.234500
 [1.2344989999999997,1.2345010000000001]
[1.2345]
 [1.2344999999999999,1.2345000000000002]
[-inf,2]
 [-Inf,2.0]
[-inf]
 [-Inf,-1.7976931348623157E+308]
[EMPTY]
 [EMPTY]
[1.2345,1.23456]
 [1.2344999999999999,1.2345600000000002]
```

書式付き入力/出力

次に、区間編集記述子を示します。

- E*w.d*E*e*
- EN*w.d*
- ES*w.d*
- F*w.d*
- G*w.d*E*e*
- VE*w.d*E*e*
- VEN*w.d*E*e*
- VES*w.d*E*e*
- VF*w.d*
- VG*w.d*E
- Y*w.d*E*e*

区間編集記述子は以下のような指定を行います。

- *w* は、フィールドが占有する位置の数を指定します。

- *d* は、小数点の右側の桁数を指定します。
- E*e* は指数フィールドの幅を指定します。

w と *d* パラメータは必ず使用しなければなりません。E*e* は、オプションです。

w と *d* 指定子は必ず存在しなければならず、以下の制約に従わなければなりません。

- *e* ＞ 0 であること。
- F 編集記述子を使う場合は、*w* ≧ 0、F 以外のすべての編集記述子を使う場合は、*w* ＞ 0であること。

入力動作

書式付き区間入力の入力動作は、すべての場合で格納された内部的な近似値が入力文字列で表される外部値を含むという点を除けば、他の数値データ型の場合と同じです。このため、包含では、区間終了点の丸めが必要になることがあります。任意の入力区間文字を input_string、これに対応する外部値 ev を (*input_string*)、入力変換後の結果としての内部的な近似値を X とすると、次のようになります。

ev(*input_string*) ⊆ X

入力過程では、すべての区間編集記述子は同じ意味論を持ちます。パラメータ *w* の値は、外部の区間を含むフィールド幅であり、*e* の値は無視されます。

出力動作

書式付き区間出力の出力動作は、すべての場合で出力文字列の算術値が出力並びの内部的なデータ項目算術値を含まなければならない点を除けば、他のデータ型の場合と同じです。このため、包含では区間終了点の丸めが必要になることもあります。任意の内部的な区間 X が与えられると、これに対応する出力文字 output_string と外部値 ev(*output_string*) は、次に関連付けられます。

X ⊆ ev(*output_string*)

出力過程では、異なる編集記述子を用いると、区間出力並び項目の区間値が異なる形式を使用して表示されるようになります。しかし、包含の制約は次のことを要求します。

ev(*input_string*) ⊆ X ⊆ ev(*output_string*)

書式付き入力

96 ページの「書式付き入力/出力」に掲載したすべての区間編集記述子について、書式化出力の動作は同じです。94 ページの「入力」で解説しているすべての入力が受け入れられます。

入力フィールドが小数点を含む場合、*d* の値は無視されます。入力フィールドで小数点が省略されている場合、*d* は入力値の小数点の位置を表します。つまり、入力値は整数として読み取られて、$10^{(-d)}$ が乗ぜられます。

コード例 2-38 入力値の小数点は書式指定子に優先する

```
math% cat ce2-38.f95
INTERVAL :: X, Y
READ(*, '(F10.4)') X
READ(*, '(F10.4)') Y
WRITE(*, *)'12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, E19.6)') X
WRITE(*, '(1X, E19.6)') Y
END
math% f95 -xia ce2-38.f95
math% a.out
[.1234]
[1234]
 12345678901234567890123456789012345678901234567890-position
      0.123400E+000
      0.123400E+000
```

コード例 2-39 区間のすべての編集記述子は単数の入力を受け入れる

```
math% cat ce2-39.f95
INTERVAL, DIMENSION(9) :: X
INTEGER                :: I
READ(*, '(Y25.3)')   X(1)
READ(*, '(E25.3)')   X(2)
READ(*, '(F25.3)')   X(3)
READ(*, '(G25.3) ')  X(4)
READ(*, '(VE25.3)')  X(5)
READ(*, '(VEN25.3)') X(6)
READ(*, '(VES25.3)') X(7)
READ(*, '(VF25.3)')  X(8)
READ(*, '(VG25.3)')  X(9)
```

コード例 2-39 区間のすべての編集記述子は単数の入力を受け入れる (続き)

```
DO I = LBOUND(X, 1), UBOUND(X, 1)
    PRINT *, X(I)
END DO
END
%math f95 -xia ce2-39.f95
%math a.out
1.23
1.23
1.23
1.23
1.23
1.23
1.23
1.23
1.23
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
 [1.2199999999999999,1.2400000000000003]
```

空白の編集 (BZ)

後続のゼロは単数の区間入力では意味を持つので、区間リスト項目の入力のために空白を処理すると、BZ 制御編集記述子は無視されます。

コード例 2-40 BZ 記述子

```
math% cat ce2-40.f95
INTERVAL :: X
REAL(4)  :: R
READ(*, '(BZ, F40.6 )') X
READ(*, '(BZ, F40.6 )') R
WRITE(*, '(VF40.3)')    X
WRITE(*, '(F40.3)')     R
END
math% f95 -xia ce2-40.f95
math% a.out
[.9998   ]
    .9998
 [             0.999,             1.000]
                                   1.000
```

桁移動数 (P)

Y、VE、VEN、VES、VF、VG 記述子用の桁移動数と、区間に適用された場合の F、E、EN、ES、G 編集記述子用の桁移動数は P 編集記述子で変更できます。P 編集記述子は、区間の終了点を REAL 値の場合と同様の方法で桁移動します。

書式付き出力

区間に適用された、F、E、EN、ES および G 編集記述子は、F または G 編集記述子が用いられると出力フィールドが F 編集記述子を使用して書式化されるようになるという点を除けば、Y 編集記述子と同じ意味を持ちます。E 編集記述子が使用される場合は、出力フィールドは対応する E 、EN または ES 編集記述子により記述された形式を常に持つようになります。

書式付き区間出力は次のような特性を持ちます。

- 正の区間終了点はオプションのプラス記号で始まります。
- 負の終了点は常に先行するマイナス記号で始まります。
- ゼロの区間終了点は先行するプラスまたはマイナス記号で始まることはありません。
- VF、VE VEN、VES および VG 編集記述子は区間の [*inf*, *sup*] 形式の書式化を提供します。
- Y 編集記述子は単数区間出力を生成します。

- VF、VE、VEN、VES、VG あるいは Y 編集記述子に一致する出力の並び項目が区間以外の任意の型である場合は、出力フィールド全体がアスタリスク (「*」) で満たされます。
- VF、VE、VEN、VES、VG 編集記述子での出力フィールドの幅 *w* が偶数の場合、そのフィールドは 1 つの先行する空白文字で満たされ、出力フィールド幅には *w*-1 が使われます。

表 2-14 は、出力に関する指数フィールド、*e* のデフォルト値を示しています。

表 2-14 出力編集記述子の指数フィールドデフォルト値

編集記述子	**INTERVAL(4)**	**INTERVAL(8)**	**INTERVAL(16)**
Y, E, EN, ES, G	3	3	3
VE, VEN, VES, VG	3	3	3

Y 編集記述子を用いた単数編集

Y 編集記述子は単数形式での拡張区間値を書式化します。

外部区間値が縮退していない場合、出力形式は REAL または INTEGER の文字定数 (角括弧 [...] で囲まない X) の場合と同じです。外部値は縮退していない算術区間 $[x] + [-1,1]_{uld}$ に翻訳されます。

Y 編集記述子の一般形式は次のとおりです。

Y*w.d*E*e*

d 指定子は、有効桁の表示用に割り当てられた場所の数を設定します。しかし、実際に表示される桁数は、*w* の値と外部区間の幅に依存して、*d* より多いことも、少ないこともあります。

e 指定子 (存在すれば) は、指数用に確保された出力下位フィールドの場所の数を定義します。

e 指定子が存在すると、出力フィールドは (F 編集記述子とは反対に) E 編集記述子により記述された形式を持つようになります。

単数区間表現は、[*inf*, *sup*] 表現よりも正確性に劣ることがしばしばあります。これは、特に区間または区間の単数表現がゼロまたは無限大を含んでいる場合にあてはまります。

たとえば、[-15, +75] の単数表現の外部値は、ev(`[0E2]`) = [-100, +100]です。[1, ∞] の単数表現の外部値は、ev(`[0E+inf]`) = [−∞, +∞] です。

これらの場合、内部的な近似値のより狭い外部表現を生成するために、*w* 文字の入力フィールド内部の最大表示可能な有効桁数を表示する *d′* ≧ 1と共に、`VG`*w.d′*`E`*e* 編集記述子が使われます。

コード例 2-41 `Y` [*inf*, *sup*] 形式の出力

```
math% cat ce2-41.f95
INTERVAL :: X = [-1, 10]
INTERVAL :: Y = [1, 6]
WRITE(*, '(Y20.5)') X
WRITE(*, '(Y20.5)') Y
END
math% f95 -xia ce2-41.f95
math% a.out
 [-1.      ,0.1E+002]
 [1.0      ,6.0     ]
```

w 文字出力フィールド内部の縮退した区間の表示が可能であれば、単数の出力文字列は通常の角括弧 [と] で囲まれ、結果が点であることを示します。

十分なフィールド幅があれば、より大きい有効桁数を表示できるかどうかに応じて、`E` または `F` 編集記述子が使用されます。`E` と `F` 編集記述子を使用した表示桁数が同じであれば、`F` 編集記述子が使用されることになります。

コード例 2-42 Y*w.d* 出力

```
cat math% cat ce2-42.f95
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1x, F20.6)') [1.2345678, 1.23456789]
WRITE(*, '(1x, F20.6)') [1.234567, 1.2345678]
WRITE(*, '(1x, F20.6)') [1.23456, 1.234567]
WRITE(*, '(1x, F20.6)') [1.2345, 1.23456]
WRITE(*, '(1x, F20.6)') [1.5111, 1.5112]
WRITE(*, '(1x, F20.6)') [1.511, 1.512]
WRITE(*, '(1x, F20.6)') [1.51, 1.52]
WRITE(*, '(1x, F20.6)') [1.5, 1.5]
END
math% f95 -xia ce2-42.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
       1.2345679
       1.234567
       1.23456
       1.2345
       1.511
       1.51
       1.5
  [    1.50000000000]
```

区間幅が増えると、単数表現で表示される桁数は減少します。区間が縮退していると、残りのすべての位置がゼロで埋められ、縮退した区間の値が正確に表される場合は角括弧が追加されます。

組み込みの関数 NDIGITS (122 ページの表 2-22 を参照してください) は、単数形式を使った区間変数または配列の書き出しに必要な最大上位桁数を返します。

コード例 2-43 NDIGITS 組み込み関数を使用した Y*w.d* 出力

```
math% cat ce2-43.f95
INTEGER :: I, ND, T, D, DIM
PARAMETER(D=5)       ! デフォルトの桁数
PARAMETER(DIM=8)
INTERVAL, DIMENSION(DIM) :: X
CHARACTER(20) :: FMT
X = (/ [1.2345678, 1.23456789], &
  [1.234567, 1.2345678], &
  [1.23456, 1.234567], &
  [1.2345, 1.23456], &
  [1.5111, 1.5112], &
  [1.511, 1.512], &
```

```
   [1.51, 1.52], &
   [1.5]/)
ND=0
DO I=1, DIM
    T = NDIGITS(X(I))
    IF(T == EPHUGE(T)) THEN ! 区間は縮退している
        ND = MAX(ND, D)
    ELSE
        ND = MAX( ND, T )
    ENDIF
END DO

WRITE(FMT, '(A2, I2, A1, I1, A1)') '(E', 10+ND, '.', ND, ')'

DO I=1, DIM
    WRITE(*, FMT) X(I)
END DO
END
math% f95 -xia ce2-43.f95
math% a.out
  0.12345679E+001
  0.1234567 E+001
  0.123456  E+001
  0.12345   E+001
  0.1511    E+001
  0.151     E+001
  0.15      E+001
[ 0.15000000E+001]
```

読みやすくするため、小数点は常に出力フィールドの右側から数えて、$p = e + d + 4$の位置に配置されます。

コード例 2-44 {Y, F, E, ,EN,ES,G}*w.d* 出力では、*d* は上位桁の最小値を表示設定する

```
math% cat ce2-44.f95
INTERVAL :: X = [1.2345678, 1.23456789]
INTERVAL :: Y = [1.5]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, F20.5)') X
WRITE(*, '(1X, F20.5)') Y
WRITE(*, '(1X, E20.5)') X
WRITE(*, '(1X, E20.5)') Y
WRITE(*, '(1X, G20.5)') X
WRITE(*, '(1X, G20.5)') Y
WRITE(*, '(1X, Y20.5)') X
WRITE(*, '(1X, Y20.5)') Y
END
math% f95 -xia ce2-44.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
        1.2345679
 [      1.5000000000]
        0.12345E+001
 [      0.15000E+001]
        1.2345679
 [      1.5000000000]
        1.2345679
 [      1.5000000000]
```

指数部の数字の数は、オプションの *e* 指定子で指定します。指数部の数が指定される場合、*w* は少なくとも、*d* + *e* + 7 でなければなりません。

コード例 2-45 Y*w.d*E*e* 出力 (*e* 指定子の用法)

```
math% cat ce2-45.f95
INTERVAL :: X = [1.2345, 1.2346]
INTERVAL :: Y = [3.4567, 3.4568]
INTERVAL :: Z = [1.5]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, Y19.5E4)') X
WRITE(*, '(1X, Y19.5E4)') Y
WRITE(*, '(1X, Y19.5E4)') Z
WRITE(*, '(1X, Y19.5E3)') X
WRITE(*, '(1X, Y19.5E3)') Y
WRITE(*, '(1X, Y19.5E3)') Z
END
math% f95 -xia ce2-45.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
      0.1234 E+0001
      0.3456 E+0001
 [    0.15000E+0001]
       0.1234 E+001
       0.3456 E+001
 [     0.15000E+001]
```

E EN、ES 編集記述子

E、EN、ES 編集記述子は、Y 編集記述子の単数 E、EN、ES 形式を使用して、区間データ項目を書式化します。

一般形式は次のとおりです。

E*w.d*E*e*

EN*w.d*E*e*

ES*w.d*E*e*

コード例 2-46 E*w.d*E*e*、EN*w.d*E*e*、ES*w.d*E*e* 編集記述子

```
math% cat ce2-46.f95
INTERVAL :: X = [1.2345678, 1.23456789]
INTERVAL :: Y = [1.5]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, E20.5)')   X
WRITE(*, '(1X, E20.5E3)') X
WRITE(*, '(1X, E20.5E3)') Y
WRITE(*, '(1X, E20.5E4)') X
```

```
WRITE(*, '(1X, E20.5E2)') X
END
math% f95 -xia ce2-46.f95
math% a.out
 1234567890123456789012345678901234567890-position
        0.12345E+001
        0.12345E+001
 [      0.15000E+001]
       0.12345E+0001
         0.12345E+01
```

F 編集記述子

F 編集記述子は、区間の Y 編集記述子の F 形式だけを使って、区間データ項目を書式化します。この一般形式は次のとおりです。

F*w.d*

F 記述子を使うと、*d* を指定する場合よりも上位の桁を表示できるようになります。表示されない桁に対応する位置は空白で埋められます。

コード例 2-47 F*w.d* 編集記述子

```
math% cat ce2-47.f95
INTERVAL :: X = [1.2345678, 1.23456789]
INTERVAL :: Y = [2.0]
WRITE(*, *) '1234567890123456789012345678901234567890-position'
WRITE(*, '(1X, F20.4)') X
WRITE(*, '(1X, E20.4)') X
WRITE(*, '(1X, F20.4)') Y
WRITE(*, '(1X, E20.4)') Y
END
math% f95 -xia ce2-47.f95
math% a.out
 1234567890123456789012345678901234567890-position
         1.2345679
         0.1234E+001
 [       2.000000000]
 [       0.2000E+001]
```

G 編集記述子

G 編集記述子は、単数 E または Y 編集記述子の F 形式を使って、区間データ項目を書式化します。この一般形式は次のとおりです。

G*w.d*E*e*

コード例 2-48 G*w.d*E*e* 編集記述子

```
math% cat ce2-48.f95
INTERVAL :: X = [1.2345678, 1.23456789]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, G20.4)')   X
WRITE(*, '(1X, G20.4E3)') X
END
math% f95 -xia ce2-48.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
         1.2345679
         0.1234E+001
```

注 - F 記述子に従って区間の終了点が出力できない場合、G 編集記述子は E 記述子を使用します。

VE 編集記述子

VE 編集記述子の一般形式は次のとおりです。

VE*w.d*E*e*

Xd が、E*w′.d* 編集記述子を使用した有効な外部値であるものとします。VE 編集記述子は、区間データ項目を次の形式で出力します。

[X_inf,X_sup]、ただし、*w′* = (*w*-3)/2

外部値 X_inf と X_sup は、それぞれ、区間出力並び項目の最大下限と最小上限に関する下限と上限です。

コード例 2-49 VE の出力

```
math% cat ce2-49.f95
INTERVAL :: X = [1.2345Q45, 1.2346Q45]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, VE25.3)')   X
WRITE(*, '(1X, VE33.4E4)') X
END
math% f95 -xia ce2-49.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
 [ 0.123E+046, 0.124E+046]
 [   0.1234E+0046,   0.1235E+0046]
```

VEN 編集記述子

VEN 編集記述子の一般形式は次のとおりです。

VEN*w*.*d*E*e*

X_inf と X_sup が、EN*w′*.*d* 編集記述子を使って表示される有効な外部値であるものとします。VEN 編集記述子は、区間データ項目を次の形式で出力します。

[X_inf,X_sup]、ただし、*w′* = (*w*-3)/2

外部値 X_inf と X_sup は、それぞれ、区間出力並び項目の最大下限と最小上限です。

コード例 2-50 VEN の出力

```
math% cat ce2-50.f95
INTERVAL :: X = [1024.82]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, VEN25.3)') X
WRITE(*, '(1X, VEN33.4E4)') X
END
math% f95 -xia ce2-50.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
 [ 1.024E+003, 1.025E+003]
 [   1.0248E+0003,   1.0249E+0003]
```

VES 編集記述子

VES 編集記述子の一般形式は次のとおりです。

VES*w.dEe*

X_inf と X_supが、ES*w′.d* 編集記述子を使用した有効な外部値であるものとします。VES 編集記述子は、区間データ項目を次の形式で出力します。

[X_inf,X_sup]、ただし、*w′* = (*w*-3)/2

外部値 X_inf と X_sup は、それぞれ、区間出力並び項目の最大下限と最小上限です。

コード例 2-51 VES の出力

```
math% cat ce2-51.f95
INTERVAL :: X = [21.234]
WRITE(*, *) '12345678901234567890123456789012345678901234567890-position'
WRITE(*, '(1X, VES25.3)')   X
WRITE(*, '(1X, VES33.4E4)') X
END
math% f95 -xia ce2-51.f95
math% a.out
 12345678901234567890123456789012345678901234567890-position
 [ 2.123E+001, 2.124E+001]
 [   2.1233E+0001,   2.1235E+0001]
```

VF 編集記述子

X_inf と X_supが、F*w′.d* 編集記述子を使用して表示される有効な外部値であるものとします。VF 編集記述子は、区間データ項目を次の形式で出力します。

[X_inf,X_sup]、ただし、*w′* = (*w*-3)/2

外部値 X_inf と X_sup は、それぞれ、区間出力並び項目の最大下限と最小上限です。

コード例 2-52 VF 出力編集

```
math% cat ce2-52.f95
INTERVAL :: X = [1.2345, 1.2346], Y = [1.2345E11, 1.2346E11]
WRITE(*, *) '1234567890123456789012345678901234567890-position'
WRITE(*, '(1X, VF25.3)') X
WRITE(*, '(1X, VF25.3)') Y
END
math% f95 -xia ce2-52.f95
math% a.out
 1234567890123456789012345678901234567890-position
 [      1.234,      1.235]
 [***********,***********]
```

注 - 区間編集記述子に従って区間終了点が出力できない場合は、アスタリスク (「*」) が表示されます。

VG 編集記述子

区間の出力では、G 編集記述子が区間の終了点出力の書式化に使われる点を除けば、VG 編集は VE 編集または VF 編集と同じです。

コード例 2-53 VG の出力

```
math% cat ce2-53.f95
INTERVAL :: X = [1.2345, 1.2346], Y = [1.2345E11, 1.2346E11]
WRITE(*, *) '1234567890123456789012345678901234567890-position'
WRITE(*, '(1X, VG25.3)') X
WRITE(*, '(1X, VG25.3)') Y
END
math% f95 -xia ce2-53.f95
math% a.out
 1234567890123456789012345678901234567890-position
 [  1.23     ,  1.24     ]
 [ 0.123E+012, 0.124E+012]
```

注 - F 記述子に従って区間の終了点が出力できない場合、VG 編集記述子は E 記述子を使用します。例えば、[0.9999, ******] などです。

書式なし入力/出力

書式なし入力/出力は、データの内部的な表現を変更せずに、メモリー位置との間でデータの転送に用います。区間を使う場合には、入力/出力に関する外側の丸めが回避できるので、書式なし入力/出力は特に重要です。

注 - 書式なし入力/出力は書式なし区間データの読み書きのためだけに使ってください。将来のリリースとのバイナリファイルの互換性は保証されません。
書式なし入力/出力は区間データ項目が透明であるという事実に依存しています。

並びによる出力

左右の終了点用の `REAL` 定数は、`F` または `E` 編集記述子を使用して生成されます。|*x*| が、出力区間終了点の絶対値であるものとします。すると、

$$10^{d_1} \le |x| \le 10^{d_2}$$

であれば、`0PF`*w.d* 編集記述子を使用して終了点が生成されます。それ以外の場合は、`1PE`*w.d*`E`*e* 記述子が使用されます。`f95` では、d_1 = -2、かつ、d_2 = +8 です。

`f95` での区間データ項目の出力については、*d* と *e* 用の値は、同じ KTPV を持つ `REAL` 型用のものと同じです。*w* の値は、96 ページのコード例 2-37 が示しているように、2 つの `REAL` と角括弧「`[`」、「`]`」とコンマの 3 つの追加文字の調整を行います。

単数入力/出力と基本変換

単数区間入力はその直後に出力が続くと、実際に基数変換が、格納された入力区間幅を 1 または 2-ulp だけ増加させるため、小数桁の正確性が失われた形で現れます。たとえば、1.37 の入力の直後に表示を行うと、1.3 が出力されることになります。96 ページの「書式付き入力/出力」を参照してください。

11 ページのコード例 1-6 で示しているように、プログラムは、入力値と (入力文字列を有効な内部的な近似値に変換するための) 内部的な読み取りとを正確に反映するために、文字入力/出力を使用しなければなりません。

組み込み区間関数

この節には f95 の組み込みの区間関数の特性定義が含まれています。

複数の KTPV を持つ引数を受け入れる一般的な組み込みの区間関数は、一般的な名前と KTPV 個別名の両方を持ちます。組み込みの関数がその KTPV 個別名の名前を使用して呼び出される場合、引数はそれに見合った KTPV を持たなければなりません。

注 - f95 では、いくつかの KTPV-16 個別組み込み関数は提供されていません。これは、実装上の未解決の問題です。

複数の区間データ項目 (たとえば、SIGN(A,B)) を受け入れる関数を使用した場合は、すべてのデータ引数は同じ KTPV を持たなければなりません。最大幅要求式処理のもとでは、この制約への準拠が自動的に行われます。厳密式処理を使う場合は、組み込み関数の引数に関する型と KTPV の制限を開発者の責任で守らなければなりません。異なる KTPV の引数を使用すると実行時エラーが発生します。

数学関数

この節では、区間引数を受け入れる型変換関数、三角関数、その他の関数を列挙します。区間 $[\underline{x}, \bar{x}]$ での記号 $\underline{x}$ と $\bar{x}$ は、それぞれ、順位付けられた要素として、最大下限と最小上限を表すために用いられます。点 (非区間) 関数定義内部では、小文字の *x* と *y* は、REAL または INTEGER 値を表すために用いられます。

区間引数 *X* の関数 *f* の評価時には、区間結果 *f*(*X*) は、次のように包含集合、set, *f*(*x*) の囲みの中に入っていなければなりません。

$$f(X) = \bigcup_{x \in X} f(x)$$

n 個の変数の関数についても同様の結果が当てはまります。区間 $[\underline{x}, \bar{x}]$ が *f* の変域の外側の値を含む場合に含まれなければならない値の包含集合の決定のしかたについては、123 ページの「参考文献」で引用した補足文献 [1] の中で解説されています。そこでの結果は、定義の境界上であるいは定義の変域の外側で評価される場合に、関数が生成できる値の集合を決定するために必要です。この包含集合と呼ばれる値の集合は、関数の引数または演算子オペランドの値のが何であるかにかかわらず有効な結果を返す区間システムを定義する上での鍵となります。このため、f95 の任意の組み込みの INTERVAL 関数に関し、引数の制約は存在しません。

積集合関数 DIVIX による除算

関数 DIVIX は、区間 C と交差する区間除算演算 (A/B) の結果として区間の囲みを返します。

A にゼロが含まれている場合、区間除算演算 (A/B) の数学的な結果は、2 つの互いに素の区間の共用体になります。共用体の中の各区間は、現在実装されている区間算術演算システム内で表すことができます。DIVIX 関数は、これらの区間の 1 つまたは両方を算出するのに便利です。

逆正接の組み込み関数 ATAN2(Y,X)

この節では、逆正接組み込み関数の追加的な情報を提供します。詳細については、123 ページの「参考文献」で引用した補足文献 [9] を参照してください。

解説： 一対の区間に対する逆正接組み込み関数の区間の囲みです。

数学定義：

$$\tan2(Y,\ X) \supseteq \bigcup_{\substack{x \in X \\ y \in Y}} \left\{ \theta \;\middle|\; \begin{array}{l} h\sin\theta = y_0 \\ h\cos\theta = x_0 \\ h = (x_0^2 + y_0^2)^{1/2} \end{array} \right.$$

クラス： 要素別処理関数です。

特別な値： 表 2-15 とコード例 2-54 は ATAN2 の不定形式を示しています。

表 2-15 ATAN2 の不定形式

y_0	x_0	$\{\sin\theta \mid h\sin\theta = y_0\}$	$\{\cos\theta \mid h\cos\theta = x_0\}$	$\{\theta \mid h = (x_0^2 + y_0^2)^{1/2}\}$
0	0	[-1, 1]	[-1, 1]	$[-\pi,\ \pi]$
+∞	+∞	[0, 1]	[0, 1]	$[0,\ \frac{\pi}{2}]$
+∞	-∞	[0, 1]	[-1, 0]	$[\frac{\pi}{2},\ \pi]$
-∞	-∞	[-1, 0]	[-1, 0]	$[-\pi,\ \frac{-\pi}{2}]$
-∞	+∞	[-1, 0]	[0, 1]	$[\frac{-\pi}{2},\ 0]$

コード例 2-54 ATAN2 の不定形式

```
math% cat ce2-54.f95

   INTERVAL :: X, Y
   INTEGER  :: IOS = 0
   PRINT *, "Press Control/D to terminate!"
   WRITE(*, 1, ADVANCE='NO')
   READ(*, *, IOSTAT=IOS) Y, X
   DO WHILE (IOS >= 0)
      PRINT *,  "For Y =", Y, "For X =", X
      PRINT *, 'ATAN2(Y,X) = ', ATAN2(Y,X)
      WRITE(*, 1, ADVANCE='NO')
      READ(*, *, IOSTAT=IOS) Y, X
   END DO
1  FORMAT("Y, X = ?")
   END
math% f95 -xia ce2-54.f95
math% a.out
 Press Control/D to terminate!
Y, X = ?[0] [0]
For Y = [0.0E+0,0.0E+0] For X = [0.0E+0,0.0E+0]
 ATAN2(Y,X) =  [-3.1415926535897936,3.1415926535897936]
Y, X = ?inf inf
For Y = [1.7976931348623157E+308,Inf] For X =
[1.7976931348623157E+308,Inf]
 ATAN2(Y,X) =  [0.0E+0,1.5707963267948968]
Y, X = ?inf -inf
For Y = [1.7976931348623157E+308,Inf] For X =
[-Inf,-1.7976931348623157E+308]
 ATAN2(Y,X) =  [1.5707963267948965,3.1415926535897936]
Y, X = ?-inf +inf
For Y = [-Inf,-1.7976931348623157E+308] For X =
[1.7976931348623157E+308,Inf]
 ATAN2(Y,X) =  [-1.5707963267948968,0.0E+0]
Y, X = ?-inf -inf
For Y = [-Inf,-1.7976931348623157E+308] For X =
[-Inf,-1.7976931348623157E+308]
 ATAN2(Y,X) =  [-3.1415926535897936,-1.5707963267948965]
Y, X = ? ^d
```

引数： Y は INTERVAL 型です。X は Y と同じ型と KIND のパラメータです。

結果特性： 引数に同じ。

結果値： 区間の結果値は、指定した区間に対する 1 つの囲みです。理想的な囲みは、記述されたものと同じ数学的な区間を含む最小幅の区間です。

1 つまたは両方の引数が空であれば、結果は空になります。

x < 0、かつ、0 ∈ Y の場合に、鋭い区間の囲み (Θで表される) を得るためには、すべての返される可能性のある区間角度の集合を一意に定義する次の方法を用いてください。

$$-\pi < m(\Theta) \leq \pi$$

この選択を次の選択と合わせれば、

$$0 \leq w(\Theta) \leq 2\pi$$

ATAN2(Y, X)が必ず含まなければならない区間角度 Θ の一意の定義となります。

表 2-16 は、鋭い区間角度の生成に必要な制限を満たすアルゴリズムでの Θ の終了点の計算に使用する REAL ATAN2 関数のテスト内容と引数を示しています。最初の 2 つの桁は区別するケースを定義しています。3 つ目の桁は区間 Θ の中点 *m* (Θ)の可能な値の範囲を含みます。最後の 2 つの桁は、REAL ATAN2 組み込み関数を使って、Θ の終了点が計算される方法を示しています。包含を保証するためには有向の丸めが使われなければなりません。

表 2-16 REAL ATAN2 関数のテスト内容と引数

Y	*X*	*m(Θ)*	$\underline{\theta}$	$\overline{\theta}$
$-\underline{y} < \overline{y}$	$\overline{x} < 0$	$\frac{\pi}{2} < m(\Theta) < \pi$	$ATAN2(\overline{y}, \overline{x})$	$ATAN2(\underline{y}, \overline{x}) + 2\pi$
$-\underline{y} = \overline{y}$	$\overline{x} < 0$	$m(\Theta) = \pi$	$ATAN2(\overline{y}, \overline{x})$	$2\pi - \underline{\theta}$
$\overline{y} < -\underline{y}$	$\overline{x} < 0$	$-\pi < m(\Theta) < \frac{-\pi}{2}$	$ATAN2(\overline{y}, \overline{x}) - 2\pi$	$ATAN2(\underline{y}, \overline{x})$

最大：MAX(X1,X2,[X3, ...])

解説： 最大範囲です。

max(X_1, ..., X_n) に対する包含集合は次のとおりです。

$$\{z \mid z = \max(x_1, \ldots, x_n), x_i \in X_i\} = [\sup(\mathrm{hull}(\underline{x_1}, \ldots, \underline{x_n})), \ \sup(\mathrm{hull}(\overline{x_1}, \ldots, \overline{x_n}))]$$

MAX 組み込み関数の実装は次の式を満たさなければなりません。

$$\mathrm{MAX(X1,X2,[X3, ...])} \supseteq \{\max(X_1, \ldots, X_n)\}$$

クラス： 要素別処理関数です。

引数： 引数は INTERVAL 型であり、同じ型と KIND 型のパラメータを持ちます。

結果特性： 結果特性は INTERVAL 型です。kind 型パラメータは引数と同じ型です。

最小：MIN(X1,X2,[X3, ...])

解説： 最小範囲です。

$\min(X_1, ..., X_n)$ に対する包含集合は次のとおりです。

$$\{z \mid z = \min(x_1, ..., x_n), x_i \in X_i\} = [\inf(\mathrm{hull}(\underline{x_1}, ..., \underline{x_n})),\ \inf(\mathrm{hull}(\overline{x_1}, ..., \overline{x_n}))]$$

MIN 組み込み関数の実装は次の式を満たさなければなりません。

$$\mathtt{MIN(X1,X2,[X3, ...])} \supseteq \{\min(X_1, ..., X_n)\}$$

クラス： 要素別処理関数です。

引数： 引数は INTERVAL 型であり、同じ型と KIND 型のパラメータを持ちます。

結果特性： 結果は INTERVAL 型です。 kind 型パラメータは引数と同じ型です。

組み込み関数

表 2-19 から表 2-22 では、区間引数を受け入れる組み込み関数の特性を一覧表示しています。表 2-17 では、これらの表の区間組み込み関数の特性項目を一覧表示しています。

表 2-17 各区間組み込み関数の特性項目

特性項目	解説
組み込み関数	関数の処理内容
定義	数学上の定義
引数の数	関数が受け入れる引数の数
一般名	関数の一般名
個別名	関数の個別名
引数の型	各固有の名前に関連付けられたデータ型
関数の型	個別引数データ型に対する戻り値のデータ型

区間組み込み関数には、KTPV4、8、16 のバージョンが定義されています。対応する個別組み込み関数名は VS、VD、VQ で始まり、それぞれ、inter**V**al **S**ingle、inter**V**al **D**ouble、inter**V**al **Q**uad を表します。各個別 REAL 組み込み関数に対応する区間組み込み関数が存在し、VSSIN() と VDSIN() のように、これらの関数には、VS、VD、VQ の接頭辞が付きます。不定形式は可能ですから、と 114 ページの「逆正接の組み込み関数 ATAN2(Y,X)」には、X**Y と ATAN2 関数の特別な値が含まれています。その他の組み込み関数ではこのような取り扱いの必要はありません。

表 2-18 組み込みの区間型変換関数

変換先	引数の数	総称名	引数の型	関数の型
INTERVAL	1、2 または 3	INTERVAL	INTERVAL	INTERVAL
			INTERVAL(4)	INTERVAL
			INTERVAL(8)	INTERVAL
			INTEGER	INTERVAL
			REAL	INTERVAL
			REAL(8)	INTERVAL
			REAL(16)	INTERVAL
INTERVAL(4)	1 または 2	SINTERVAL	INTERVAL	INTERVAL(4)
			INTERVAL(4)	INTERVAL(4)
			INTERVAL(8)	INTERVAL(4)
			INTEGER	INTERVAL(4)
			REAL	INTERVAL(4)
			REAL(8)	INTERVAL(4)
			REAL(16)	INTERVAL(4)
INTERVAL(8)	1 または 2	DINTERVAL	INTERVAL	INTERVAL(8)
			INTERVAL(4)	INTERVAL(8)
			INTERVAL(8)	INTERVAL(8)
			INTEGER	INTERVAL(8)
			REAL	INTERVAL(8)
			REAL(8)	INTERVAL(8)
			REAL(16)	INTERVAL(8)
INTERVAL(16)	1 または 2	QINTERVAL	INTERVAL	INTERVAL(16)
			INTERVAL(4)	INTERVAL(16)
			INTERVAL(8)	INTERVAL(16)
			INTERVAL(16)	INTERVAL(16)
			INTEGER	INTERVAL(16)
			REAL	INTERVAL(16)
			REAL(8)	INTERVAL(16)

表 2-19 組み込みの区間演算関数

組み込み関数	非区間での定義	引数の数	総称名	個別名	引数の型	関数の型
絶対値	|*a*|	1	ABS	VDABS VSABS VQABS	INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTERVAL(8) INTERVAL(4) INTERVAL(16)
切り上げ (注記 1 を参照)	int(*a*)	1	AINT	VDINT VSINT VQINT	INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTERVAL(8) INTERVAL(4) INTERVAL(16)
最近似値整数	*a* ≥ *0* の場合は int(*a* + *.5*) *a* < *0* の場合は int(*a* - *.5*)	1	ANINT	VDNINT VSNINT VQNINT	INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTERVAL(8) INTERVAL(4) INTERVAL(16)
剰余	*a*-*b*(int(*a*/*b*))	2	MOD	VDMOD VSMOD	INTERVAL(8) INTERVAL(4)	INTERVAL(8) INTERVAL(4)
符号転送 (注記 2 を参照)	|*a*| sgn(*b*)	2	SIGN	VDSIGN VSSIGN	INTERVAL(8) INTERVAL(4)	INTERVAL(8) INTERVAL(4)
最大値の選択 (注記 3 を参照)	max(*a*,*b*,...)	≥2	MAX	MAX	INTERVAL	INTERVAL
最小値の選択 (注記 3 を参照)	min(*a*,*b*,...)	≥2	MIN	MIN	INTERVAL	INTERVAL
下限	floor(*A*)	1	FLOOR		INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTEGER INTEGER INTEGER
上限	ceiling(*A*)	1	CEILING		INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTEGER INTEGER INTEGER
精度	precision(*A*)	1	PRECISION		INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTEGER INTEGER INTEGER
範囲	range(*A*)	1	RANGE		INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTEGER INTEGER INTEGER

(1) a > 0 の場合は int(*a*) = floor(*a*)、a < 0 の場合は ceiling(*a*)

(2) signum 関数は、*a* < 0 の場合 sgn(*a*) = -1、*a* < 0の場合 +1、*a* = 0 の場合 0 となります。

(3) MIN と MAX 組み込み関数は、すべての引数が空でなければ、空の区間引数を無視します。すべての引数が空の場合は空の区間が返されます。

表 2-20 組み込みの区間三角関数

組み込み関数	非区間での定義	引数の数	総称名	個別名	引数の型	関数の型
正弦	sin(*a*)	1	SIN	VDSIN	INTERVAL(8)	INTERVAL(8)
				VSSIN	INTERVAL(4)	INTERVAL(4)
余弦	cos(*a*)	1	COS	VDCOS	INTERVAL(8)	INTERVAL(8)
				VSCOS	INTERVAL(4)	INTERVAL(4)
正接	tan(*a*)	1	TAN	VDTAN	INTERVAL(8)	INTERVAL(8)
				VSTAN	INTERVAL(4)	INTERVAL(4)
逆正弦	arcsin(*a*)	1	ASIN	VDASIN	INTERVAL(8)	INTERVAL(8)
				VSASIN	INTERVAL(4)	INTERVAL(4)
逆余弦	arccos(*a*)	1	ACOS	VDACOS	INTERVAL(8)	INTERVAL(8)
				VSACOS	INTERVAL(4)	INTERVAL(4)
逆正接	arctan(*a*)	1	ATAN	VDATAN	INTERVAL(8)	INTERVAL(8)
				VSATAN	INTERVAL(4)	INTERVAL(4)
逆正接 (注記 1 を参照)	arctan(*a*/*b*)	2	ATAN2	VDATAN2	INTERVAL(8)	INTERVAL(8)
				VSATAN2	INTERVAL(4)	INTERVAL(4)
双曲正弦	sinh(*a*)	1	SINH	VDSINH	INTERVAL(8)	INTERVAL(8)
				VSSINH	INTERVAL(4)	INTERVAL(4)
双曲余弦	cosh(*a*)	1	COSH	VDCOSH	INTERVAL(8)	INTERVAL(8)
				VSCOSH	INTERVAL(4)	INTERVAL(4)
双曲正接	tanh(*a*)	1	TANH	VDTANH	INTERVAL(8)	INTERVAL(8)
				VSTANH	INTERVAL(4)	INTERVAL(4)

(1) $a = h \sin\theta$、$b = h \cos\theta$、かつ、$h^2 = a^2 + b^2$ であれば、arctan(a/b) = θ となります。

表 2-21 その他の組み込みの区間数学関数

組み込み関数	非区間での定義	引数の数	総称名	個別名	引数の型	関数の型
平方根 (注記 1 を参照)	exp{ln(*a*)/2}	1	SQRT	VDSQRT	INTERVAL(8)	INTERVAL(8)
				VSSQRT	INTERVAL(4)	INTERVAL(4)
指数	exp(*a*)	1	EXP	VDEXP	INTERVAL	INTERVAL(8)
				VSEXP	INTERVAL(4)	INTERVAL(4)
自然対数	ln(*a*)	1	LOG	VDLOG	INTERVAL(8)	INTERVAL(8)
				VSLOG	INTERVAL(4)	INTERVAL(4)
常用対数	log(*a*)	1	LOG10	VDLOG10	INTERVAL(8)	INTERVAL(8)
				VSLOG10	INTERVAL(4)	INTERVAL(4)

(1) sqrt(*a*) は複数の値を持ちます。正と負の両方の平方根を含むためには、適切な区間の囲みが必要です。
SQRT 組み込み関数を次のように定義するとこの問題を除去できます。

$$\exp\left\{\frac{\ln a}{2}\right\}$$

表 2-22 区間組み込み関数

組み込み関数	定義	引数の数	総称名	個別名	引数の型	関数の型
Infimum	inf([*a*, *b*]) = *a*	1	INF	VDINF VSINF VQINF	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
Supremum	sup([*a*, *b*]) = *b*	1	SUP	VDSUP VSSUP VQSUP	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
幅	w([*a*, *b*]) = *b* - *a*	1	WID	VDWID VSWID VQWID	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
中点	mid([*a*, *b*]) = (*a* + *b*)/2	1	MID	VDMID VSMID VQMID	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
マグニチュード (注記 1 を参照)	max(|*a*|) ∈ *A*	1	MAG	VDMAG VSMAG VQMAG	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
ミグニチュード (注記 2 を参照)	min(|*a*|) ∈ *A*	1	MIG	VDMIG VSMIG VQMIG	INTERVAL(8) INTERVAL(4) INTERVAL(16)	REAL(8) REAL(4) REAL(16)
空区間の検査	*A* が空なら *true*	1	ISEMPTY	VDISEMPTY VSISEMPTY VQISEMPTY	INTERVAL(8) INTERVAL(4) INTERVAL(16)	LOGICAL LOGICAL LOGICAL
積集合による除算	(A/B) ∩ C		DIVIX	VDDIVIX VSDIVIX VQDIVIX	INTERVAL(8) INTERVAL(4) INTERVAL(16)	INTERVAL(8) INTERVAL(4) INTERVAL(16)
桁数 (注記 3 を参照)	Y 編集記述子を使った最大桁数	1	NDIGITS		INTERVAL INTERVAL(4) INTERVAL(16)	INTEGER INTEGER INTEGER

(1) mag([*a*, *b*]) = max(\|*a*\|,\|*b*\|)
(2) *a* > 0 または *b* < 0 であれば、mig([*a*, *b*]) = min(\|*a*\|,\|*b*\|)、その他の場合は、0
(3) 特別なケース： NDIGITS([-inf, +inf]) = NDIGITS([EMPTY]) = 0

乱数サブルーチン

RANDOM_NUMBER (HERVEST) は、区間変数 HERVEST を介して、1 つの擬似ランダム区間 [*a*, *b*]、または $0 \le a \le 1$ および $a \le b \le 1$ の範囲で一様分布から擬似ランダム区間の配列を返します。

参考文献

下記の技術報告書がオンラインで利用できます。これらのファイルの所在については、区間演算の README を参照してください。

1. G.W.Walster、E.R.Hansen、J.D.Pryce著、『Extended Real Intervals and the Topological Closure of Extended Real Relations』、技術報告書、Sun Microsystems (2000 年 2 月)。
2. G.William Walster著、『Empty Intervals』、技術報告書、Sun Microsystems (1998 年 4 月)。
3. G.William Walster著、『Closed Interval Systems』、技術報告書、Sun Microsystems (1999 年 8 月)。
4. G.William Walster著、『Literal Interval Constants』、技術報告書、Sun Microsystems (1999 年 8 月)。
5. G.William Walster著、『Widest-need Interval Expression Evaluation』、技術報告書、Sun Microsystems (1999 年 8 月)。
6. G.William Walster著、『Compiler Support of Interval Arithmetic With Inline Code Generation ans Nonstop Exception Handling』、技術報告書、Sun Microsystems (2000 年 2 月)。
7. G.William Walster著、『Finding Roots on the Edge of a Function's Domain』、技術報告書、Sun Microsystems (2000 年 2 月)。
8. G.William Walster著、『Implementing the 'Simple' Closed Interval System』、技術報告書、Sun Microsystems (2000 年 2 月)。
9. G.William Walster著、『Interval Angles and the Fortran ATAN2 Intrinsic Function』、技術報告書、Sun Microsystems (2000 年 2 月)。

10. G.William Walster著、『The 'Simple' Closed Interval System』、技術報告書、Sun Microsystems (2000 年 2 月)。

11. G.William Walster、Margaret S. Bierman著、『Interval Arithmetic in Forte Developer Fortran』、技術報告書、Sun Microsystems (2000 年 2 月)。

用語集

certainly true 関係演算子 (certainly true relational operator)

関係演算子: certainly true (relational operators: certainly true) を参照してください。

ev(literal_constant)

ev (*literal_constant*) の表記はリテラル定数文字列により定義された外部値を表すのに使用します。たとえば、定数 0.1 はマシン表現できないので、0.1 の内部的な近似値が使われなければならないという事実に関係なく、ev(0.1) = 1/10 となります。

f(set)

f(*set*) という表記は、一組の引数群に対して評価される式の包含集合を記号的に表すために使用します。たとえば、式 *f*(*x*, *y*) = *xy* について、区間式 [0] × [+∞] が満たさなければならない包含の制約は、次のように表されます。

[0] × [+∞] ⊇ = [-∞, +∞].

INTERVAL 固有の関数 (INTERVAL-specific function)

`f95` では、INTERVAL 固有の関数は標準 Fortran 関数の区間バージョンではない、区間関数です。たとえば、`WID`、`MID`、`INF`、`SUP` は `INTERVAL` 固有の関数です。

KTPV (kind type parameter value)

種別型パラメータ値 (KTPV) を参照してください。

$KTPV_{max}$

区間式の最大幅要求式処理のもとでは、すべての区間は式の中の任意のデータ項目の KTPV の最大値へと変換されます。この最大値には、$KTPV_{max}$ という名前が与えられています。

KTPV混合の区間式 (mixed-KTPV INTERVAL expression)

KTPV 混合の区間式は異なる KTPV を持つ定数と (または) 変数を含みます。たとえば、[1_4] + [0.2_8] はKTPV 混合の区間式です。最大幅要求式処理のもとでは、KTPV 混合の区間式が許されますが、厳密式処理のもとでは許されません。

possibly true 関係演算子 (possibly true relational operators)

関係演算子: possibly true (relational operators: possibly true) を参照してください。

値の代入 (value assignment)

Fortran では、代入文は式の値を値の代入演算子 = の右側で計算し、左側の変数、配列要素、または、配列に格納します。

外部値 (external value)

Fortran リテラル定数の外部値は、リテラル定数の文字列により定義された数学的な値です。リテラル定数の外部値は定数の内部的な近似値と同じであるとは限りません。後者は、Fortran 標準ではリテラル定数の唯一の定義された値です。ev(literal_constant) を参照してください。

外部表現
(external
representation)

Fortran データ項目の外部表現は、入力データ変換中での定義に使用される文字列、または、出力データ変換後の表示に使用される文字列のことです。

拡張区間
(extended interval)

拡張区間の用語は、その終了点が -∞ と +∞ を含む実数に拡張できる区間を意味します。完全な意味では、空の区間もまた拡張実区間の集合に含まれます。

下限
(lower bound)

最大下限 (infumum(plural, infima)) を参照してください。

空の区間
(empty interval)

空の区間 [*empty*] は、メンバーを持たない区間です。空の区間は 2 つの互いに素の区間の積集合として、自然に発生します。たとえば、[2, 3]∩[4, 5] = [*empty*] となります。

空の集合
(empty set)

空の集合 ∅ は、メンバーを持たない集合です。空の集合は 2 つの互いに素の集合の積集合として自然に発生します。たとえば、{2, 3} ∩ {4, 5} = ∅ となります。

仮数 (mantissa)

科学的記数法で記述された1 つの数は、仮数、または、有効数字と指数、または、10 のべき乗で構成されます。Fortranの E 編集記述子は数を仮数形式、または、有効数字と指数形式、または、10 のべき乗形式で表示します。

関係演算子: certainly true (relational operators: certainly true)

certainly true 関係演算子としては、{.CLT.,.CLE.,.CEQ.,.CNE.,.CGE.,.CGT.} があります。certainly true 関係演算子は、オペランド区間のすべての要素について、課題となる関係が true であれば、true となります。つまり、すべての x ∈ [a, b] and y ∈ [c, d] について、*x* .op. *y* = *true* であれば、[*a*, *b*] .Cop. [*c*, *d*] = *true* となります。

たとえば、(*b* < *c*)が *true* であれば、[*a*, *b*] .CLT. [*c*, *d*] は *true* となります。

関係演算子: possibly true (relational operators: possibly true)

possibly true 関係演算子としては、{.PLT., .PLE., .PEQ., .PNE., .PGE., .PGT.} があります。possibly true 関係演算子は、オペランド区間の任意の要素について、課題となる関係が true であれば、true となります。たとえば、(a < d) が true で あれば、[*a*, *b*] .PLT. [*c*, *d*] if *a* < *d* は true となります。

関係演算子：set (relational operators: set)

set 関係演算子としては、{.SLT., .SLE., .SEQ., .SNE., .SGE., .SGT.} があります。集合関係演算子は、その区間の終了点について、課題となる関係が true であれば、true となります。たとえば、(*a* = *c*) が true であり、(*b* = *d*)も *true* であれば、[*a*, *b*] .SEQ. [*c*, *d*] は true となります。

基数変換 (radix conversion)

基数変換とは、外部 10 進数と内部 2 進数間相互の変換プロセスのことです。基数変換は書式化された並びによる入力/出力の中で行われます。同じ数は 2 進数システムと 10 進数システムで常に表現可能なわけではありませんから、包含を保証するためには、基数変換中の有向の丸めが要件となります。

狭幅区間
(narrow-width interval)

区間 [*a*, *b*] が値 v ∈ [a, b] の近似値であるものとします。w[*a*, *b*] = *b* - *a* が小さければ、[*a*, *b*] は狭幅区間であることになります。区間 [*a*, *b*]の幅が狭ければ狭いほど、[*a*, *b*] はより正確に V に近似します。**鋭い区間結果** (sharp interval result)) も参照してください。

区間アルゴリズム
(interval algorithm)

区間アルゴリズムは、区間結果の計算に使用される一連の演算です。

区間演算
(interval arithmetic)

区間演算は区間を使った計算に使用される算術システムです。

区間近似値
(internal approximation)

Fortran では、リテラル定数の区間近似値はマシンで表現可能な値です。Fortran 標準には、内部的な近似値の正確性の要件が存在しません。

区間定数
(INTERVAL constant)

区間定数は閉じた補正集合です。対の数 $a \le b$ により定義された $[a, b] = \{z \mid a \le z \le b\}$ です。

区間定数の外部値
(INTERVAL constant's external value)

区間定数の外部値は、区間定数の文字列により定義された数学的な値です。**外部値** (external value)を参照してください。

区間定数の内部的な近似値
(INTERVAL constant's internal approximation)

`f95` では、区間定数の内部的な近似値は、定数の外部値の鋭い内部的な近似値です。このため、この近似値は定数の外部値を含む最も狭いマシン表現可能な区間となります。

区間幅 (interval width)

区間幅は、$w([a, b]) = b - a$ のように表されます。

区間包 (interval hull)

1 対の区間 = [x̄, x̲] ?and_rom? Y = [ȳ, に関する区間包演算子 ∪ は、X と Y の両方を含む最小の区間です (?inf?(X ∪ Y), ?sup?(X ∪ Y) とも表される) 。たとえば、次のようになります。

$[2, 3] \cup [5, 6] = [2, 6]$.

区間ボックス (interval box)

区間ボックスは、n 次元のデカルト座標軸に平行な辺を持つ平行六面体です。区間ボックスは n 次元の区間ベクトル $\boldsymbol{X} = (X_1, \ldots, X_n)^T$ を使って簡便に表現することができます。

組み込み区間データ型 (intrinsic INTERVAL data type)

Fortran には、4 つの組み込みの数値データ型として、INTEGER、REAL、DOUBLE PRECISION REAL、COMPLEX があります。コマンド行オプションの -xia または -xinterval を使うと、f95 は区間を組み込みのデータ型として認識します。

組み込みの INTERVAL 固有の関数 (intrinsic INTERVAL-specific function)

f95 には、WID、HULL、MID、INF、SUP を含むいろいろな組み込みの INTERVAL 固有の関数があります。

厳密式処理 (strict expression processing)

厳密式処理のもとでは、コンパイラは自動的な型変換または KTPV 変換を行いません。型の混合と KTPV の混合された区間式は許されません。任意の型と (または) KTPV の変換は明示的にプログラムしなければなりません。

交換可能な式 (exchangeable expression)

2 つの式はそれらが包含集合と同じ (包含集合がすべての箇所で同じ) であれば、交換可能です。

合成式 (composite expression)

g の変域内にある h の変域内のすべての単集合 $\{\underline{x}\} = \{x_1\} \otimes \ldots \otimes \{x_n\}$ について、$f(\{\underline{x}\}) = g(h(\{\underline{x}\}))$ の規定どおりに、所与の式 g と h から新しい式 f (合成式) を形成します。単集合引数は、式が関数または関係のどちらかであるということを暗示しています。

混合型の区間式 (mixed-type INTERVAL expression)

混合型の区間式は異なる型のデータ項目を含みます。たとえば、式 `[0.1] + 0.2D0` は、`[0.1]` が 1 つの区間であり、`0.2D0` が `DOUBLE PRECISION` の定数なので、混合型の区間式です。これらは共に同じ KTPV = 8 を持ちます。

混合モード (型とKTPV) の区間式 (mixed-mode (type and KTPV) INTERVAL expression)

混合モードの区間式は異なる型と KTPV のデータ項目を含みます。たとえば、式 `[0.1] + 0.2` は混合モード式です。`[0.1]` は KTPV = 8 を持つ区間定数ですが、`0.2` は KTPV = 4 を持つ REAL 定数です。

コンパクト集合 (compact set)

コンパクト集合は、集合内にすべての制限ポイントと累積ポイントを含みます。つまり、集合 S とシーケンス $\{s_j\} \in S$ を指定した場合、S の閉包は、$\bar{S} = \{\lim_{j \to \infty} s_j \mid s_j \in S\}$ になります。ここで、$\lim_{j \to \infty}$ は、シーケンス {sj} の累積ポイントまたは制限ポイントを表します。実数 $\{z \mid -\infty < z < +\infty\}$ はコンパクトではありません。拡張された実数の集合は、コンパクトです。

最終位置単位 (ulp) (unit in the last place (ulp))

内部的なマシン数の 1 最終位置単位 (*ulp*) は、マシンの演算機能を使って実行可能な最小のインクリメントまたはデクリメントのことです。このため、計算された区間の幅が 1 ulp であれば、これは所与の KTPV を持つ、可能な最も狭い非縮退区間となります。

最終桁単位 (uid) (unit in the last digit (uld))

単数の入力/出力では、暗黙に区間を構築するために、最終表示桁に対し 1 最終桁単位 (*uld*) の加算または減算が行われます。

最小上限 (supremum(plural, suprema))

数の集合の最小上限は、集合の上限にいちばん近い数です。これは集合内部の最大の数、または、集合内部のすべてのメンバーよりも大きい最小の数のいずれかです。区間定数 [*a*, *b*] の最小上限 sup([*a*, *b*]) は *b* です。

最大下限 (infumum(plural, infima))

数の集合の最大下限は、集合の下限にいちばん近い数です。これは、集合内部の最小の数、または、集合内部のすべてのメンバーよりも小さい最大の数のいずれかです。区間定数 [*a*, *b*] の最大下限 inf([*a*, *b*]) は a です。

最大幅要求式処理 (widest-need expression processing)

最大幅要求式処理のもとでは、コンパイラにより、自動的な型変換と KTPV 変換が行われます。任意の非区間の添字式は区間に昇格され、KTPV は $KTPV_{max}$ へと設定されます。

最大幅要求式処理：スコープ (widest-need expression processing: scope)

Fortran では、スコープは、データと (または) 演算が定義されていてあいまいさのない、実行プログラムの一部分を意味します。最大幅要求式処理のスコープは、関数とサブルーチンの呼び出しにより制限を受けます。

最大幅要求式処理のスコープ (scope of widest-need expression processing)

最大幅要求式処理： スコープ (widest-need expression processing: scope)を参照してください。

式処理：strict (expression processing: strict)

厳密式処理 (strict expression processing) を参照してください。

式処理：widest-need (expression processing: widest-need)

最大幅要求式処理(widest-need expression processing) を参照してください。

式の文脈 (expression context)

最大幅要求式処理では、式の文脈を定義する 2 つの属性は、式の型と最大 KTPV ($KTPV_{max}$) です。

実装特性 (quality of implementation)

実装特性はコンパイラの区間サポートを特徴付けます。狭幅 (narrow width) は区間データ型用に組み込みのコンパイラサポートにより提供される新しい実装品質です。

集合理論 (set theoretic)

集合理論は、集合の手段、または、集合の代数に属するものです。

縮退した区間
(degenerate interval)

縮退した区間はゼロ幅の区間です。縮退した区間は、その唯一の要素が点である単集合です。ほとんどの場合、縮退区間は 1 つの点と考えることができます。たとえば、区間 [2, 2] は縮退しており、区間 [2, 3] は縮退していません。

種別型パラメータ値
(KTPV)

Fortran では、各組み込みデータ型は、データ型の種別 (精度) を選択する種別型パラメータ値 (KTPV) を使ってパラメータ化されます。`f95` には、4、8、16 の 3 つの区間 KTPV が存在します。デフォルトの KTPV は、8 です。

上限
(upper bound)

最小上限 (supremum(plural, suprema)) を参照してください。

鋭い区間結果
(sharp interval result)

鋭い区間結果は可能な限り狭い幅を持ちます。鋭い区間結果は式の包含の包と同じになります。有限精度の算術による制限が課せられると、鋭い区間結果は、式の包含集合を含む、可能なかぎり最も狭い有限の区間となります。

代入文
(assignment statement)

代入文は、次の形式を持つ Fortran 文です。

`V = expression`

代入文の左側は、変数、配列要素、または配列 V です。

互いに素の区間
(disjoint interval)

2 つの互いに素の区間は共通の要素を持ちません。区間 [2, 3] と [4, 5] は、互いに素の区間です。2 つの互いに素の区間の積集合は空の区間となります。

多用途式 (MUE = multiple-use expression)

多用途式 (*MUE*) は、少なくとも 1 つの独立変数が複数回出現する式のことです。

単一用途式 (SUE) (single-use expression (SUE))

単一用途式 (*SUE*) は、式内部で各変数が 1 度だけしか発生しない式のことです。たとえば、次の式は単一用途式ですが、

$$\frac{1}{1+\frac{Y}{X}}$$

次の式は、そうではありません。

$$\frac{X}{X+Y}$$

単数区間データ変換 (single-number INTERVAL data conversion)

単数区間データ変換は、`Y` 編集記述子による読み取りと単数表現を使った外部区間の表示に使用します。**単数の入力/出力 (single-number input/output)**を参照してください。

単数の入力/出力 (single-number input/output)

単数の入力/出力は区間用に、区間 $[-1, +1]_{uld}$ が暗黙のうちに最終表示桁に追加される、単数外部表現を使用します。下付き文字 uld は、最終桁での単位を表す頭字語 (Unit in the Last Digit) です。たとえば、0.12300 は区間 $0.12300 + [-1, +1]_{uld} = [0.12299, 0.12301]$ を表します。

断定的な関係 (affirmative relation)

certainly、possibly または同じでない集合を除く順位関係。断定的な関係は、たとえば、$a < b$ のように、何かを断言する関係です。

断定的な関係演算子 (affirmative relational operators)

断定的な関係演算子は集合の要素です。：{<, ≤, =, ≥, >}

定数式 (constant expression)

Fortran での定数式は変数または配列を含みません。定数とオペランドは含めることができます。式 `[2, 3] + [4, 5]` は定数式です。`X` が変数であれば、式 `X + [2, 3]` は定数式ではありません。`Y` が名前付き定数であれば、`Y + [2, 3]` は定数式です。

点 (point)

(区間の対立概念としての) 点は、1 つの数です。N 次元空間での 1 つの点は、n 次元のベクトル $\boldsymbol{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ を使って表されます。点と縮退した区間、あるいは、区間ベクトルは同じものと考えることができます。厳密には、任意の区間はその要素が複数の点である 1 つの集合です。

透明なデータ型 (opaque data type)

透明なデータ型は内部的な近似値の構造を未指定のままにします。区間データ項目は透明です。このため、プログラマは区間データ項目が内部的に何か特別な方法で表現されることを期待することができません。組み込みの関数 `INF` と `SUP` は区間の構成要素へのアクセスを提供します。手作業で任意の有効な区間を構築するには、`INTERVAL` 構成子を使うことができます。

名前付き定数 (named constant)

名前付き定数は `PARAMETER` 文の中で宣言、初期化されます。名前付き定数の値は文脈に依存しないので、`PARAMETER` 宣言でのデータ項目のより適切な名前は、「読み取り専用変数」です。

反断定的な関係 (anti-affirmative relation)

反断定的な関係は true となりえないことがらについての陳述です。順位関係 ≠ は、Fortran での唯一の反断定的な関係です。

反断定的な関係演算子 (anti-affirmative relation operator)

Fortran の .NE. と /= 演算子は反断定的な関係を実装しています。certainly、possible、区間オペランド用の集合バージョンは、それぞれ、.CNE.、.PNE.、.SNE.で表されます。

左側の終了点 (left endpoint)

1 つの区間の左側の終了点は、その最大下限または下限と同じです。

文脈依存の区間定数 (context-dependent INTERVAL constant)

最大幅要求式処理のもとでの区間定数の内部的な近似値は、$KTPV_{max}$ と同じ KTPV を持つ鋭い区間なので文脈依存となります。次の前提で、区間定数 [*a*, *b*] についての任意の近似値を使うことができます。

$[a,b] \supseteq ev([a,b])$,

ここでは、ev([*a*,*b*]) は区間定数 [*a*,*b*] の外部値を表します。包含の違反がなければ、任意の内部的な近似値を選択することが許されます。たとえば、内部的な近似値 [0.1_4]、[0.1_8] と [0.1_16] は、すべてが外部値 ev(0.1) = 1/10 を持ち、このため、包含の制約に違反しません。最大幅要求式処理のもとでは、$KTPV_{max}$ と同じ KTPV を持つ内部的な近似値が使用します。

閉鎖区間 (closed interval)

閉鎖区間はその終了点を含みます。閉鎖区間は閉集合 (closed set) です。[2, 3] = $\{z \mid 2 \le z \le 3\}$である区間はその終了点が含まれるので閉じられています。区間 (2, 3) = $\{z \mid 2 < z < 3\}$ である区間はその終了点が含まれないので、開いています。f95 で実装された区間演算は、閉鎖区間だけを取り扱います。

閉鎖数学システム (closed mathematical system)

閉鎖数学システム内部では、未定義の演算子オペランドの組み合わせが存在してはいけません。閉鎖システムの要素に関するすべての定義済み演算はそのシステムの要素を生成しなければなりません。実数システムは、ゼロによる除算がそのシステムで未定義なので、閉じられていません。

閉包 (hull)

区間包 (interval hull) を参照してください。

包含集合 (containment set)

式 h の包含集合 $h(x)$ は、h が任意の合成 $f(\{x\}) = g(h(x), x)$ の構成要素として使われると、包含の制約に違反しない最小の集合となります。

For $h(x, y) = x \div y$ については、

$h(+\infty, +\infty) = [0, +\infty]$ となります。

f(set) も参照してください。

包含集合の同値 (containment set equivalent)

包含集合のどの部分も同じである場合、2 つの式は包含集合の同値です。

包含集合の閉包恒等式 (containment set closure identity)

n 個の変数の任意の式 $f(\{\boldsymbol{x}\}) = f(\{x_1\} \ldots, \{x_n\})$ と点 $\boldsymbol{x}_0$ が与えられると、点 $\boldsymbol{x}_0$ での f の閉包は、 $\mathrm{f}(\mathrm{x})) = \bar{f}(\{\boldsymbol{x}_0\})$ となります。

包含の失敗
(containment failure)

包含の失敗は、包含の制約を満たせない失敗です。たとえば、[1]/[0] が [*empty*] であると定義されると、包含の失敗となります。これは、$0 \notin Y$ でない Y である X=[0]と Y について、次のような区間式を想定すれば確認することができます。

$$\frac{X}{X+Y} = \frac{1}{1+\frac{Y}{X}}$$

最初の式の包含集合は [0] です。しかし、[1]/[0] が [*empty*] であると定義されると、2 番目の式もまた [*empty*] となります。これは、包含の失敗です。

包含の制約
(containment
constraint)

縮退した区間 [x] での式 f の区間の評価 $f(x)$ に関する包含の制約は、$f([x]) \supseteq f(x)$ のようになります。ここで、$f(x)$ は、$f(x)$ が含まなければならないすべての可能な値の包含集合を表します。包含集合は、1 / 0 = {-∞, +∞} ですから、[1] / [0] = hull({-∞, +∞}) = [-∞, +∞] となります。包含集合 (**包含集合** (containment set)) も参照してください。

右側の終了点
(right endpoint)

最小上限 (supremum(plural, suprema)) を参照してください。

有効な区間結果
(valid interval result)

1 つの有効な区間結果 [a, b] は、次の 2 つの要件を満たさなければなりません。

- $a \leq b$
- [a, b] は包含の制約に違反してはいけません。

有向の丸め
(directed rounding)

有向の丸めは特別な向きを持つ丸めです。区間演算の文脈では、丸め切り上げは +∞ 方向の、丸め切り下げは -∞ 方向の向きを持ちます。有向の丸めは矢印↓と↑により記号化されます。このため、小数 5 桁の算術を使う場合、↑ 2.00001 = 2.0001 のようになります。有向の丸めは、コンピュータ上で包含の制約に違反することがないよう、区間算術の実装で使用します。

読み取り専用変数
(read-only variable)

読み取り専用変数は標準 Fortran で定義済みの構成概念ではありません。しかし、それでも読み取り専用変数は一度初期化された後は変更できなくなります。区間サポートのない標準 Fortran では、名前付き定数と読み取り専用変数を区別する必要がありません。しかし、最大幅要求式処理は定数の外部値を使うので、読み取り専用変数と名前付き定数を区別しなければなりません。f95 の実装では、PARAMETER 宣言の中で初期化される記号名は読み取り専用変数となります。

リテラル定数
(literal constant)

f95 では、区間リテラル定数は、定数の外部値を定義するために使用される文字列です。

リテラル定数の外部値
(literal constant's
external value)

f95 では、リテラル定数の外部値は定数の文字列により定義された数学上の値です。**外部値** (external value)も参照してください。

リテラル定数の内部的
な近似値
(literal constant's
internal approximation)

f95では、内部的なリテラル定数の内部的近似値は、定数の外部値を含む鋭いマシン表現可能な区間です。

例外 (exception)

IEEE754 浮動小数点標準では、ゼロによる除算のような未定義の演算を実行しようとすると、例外が発生します。

索引

D

E

F

G

I

K

L

き

く

け

こ

さ

し

す

せ

た