目次

序文

Unidata は National Science Foundataion によって支援されている program で, 米国の大学が革新的なコンピュータおよびネットワークのアプリケーションを通して, 教育および研究のために大気および関連するデータを活用できるようにするものである. これらのデータを解析および表示するために, Unidata Program Center では大学に対して, University of Wisconsin, Purdue University, NASA, および National Weather Service を含む他の組織によって開発されたソフトウェアパッケージを提供している. これらの基礎をなすのが, リアルタイムでデータを取得および扱うために Unidata によって開発されたシステムであり, 各大学が各自の所有データをローカルな必要に応じて取得および扱えるべきであるという Unidata の原則を実地にするものである. Unidata program はデータセンターを持たないとうことが重要な点である – データのマネージメントは分配された任務である.

このガイドで記述されている Network Common Data Form (netCDF) ソフトウェアは, 本来, いろいろな Unidata アプリケーションに対して共通のデータアクセス法を供給することを意図したものである. これらは, 一点観測, 時系列, 規則的に配置された格子点, および衛星またはレーダーイメージを含んだいろいろな種類のデータを扱うものである.

netCDF ソフトウェアは I/O ライブラリとして機能し, C または FORTRAN から呼び出せ, 自己記述的で機種依存の無いファイルにデータを格納および引出すことができる. 各 netCDF ファイルは, 無制限の多次元の名づけられた変数 (整数, 実数, 文字, バイト, 等々のように型が異っていてもよい) を含み, 各変数は測定の単位や説明書きのような補助データを伴っていてもよい. インターフェースは既存の netCDF ファイルに既定の方法でデータを追加する手段を含んでおり, これは (固定長の) レコード構造とは非なる機能性である. しかし, netCDF ライブラリは, 変数の名前およびインデクスによりデータを直接アクセスで格納および引出すことをも許容しており, それゆえにディスク上にある (またはメモリ上にある) ファイルに対してのみ有用なものである.

これまでのところ, Unidata ソフトウェアの約半分が netCDF アクセスを用いるようになっており, このような共通性がすべての Unidata アプリケーションにわたって拡張されるべく計画されている. このことは以下の目的を持つ:

- 共通データを異なるアプリケーションによって使用することを促進する.
- 異なるコンピュータ間でデータファイルが透明, すなわち変換なしで転送または共有されるのを可能にする.
- フォーマットを解釈するために通常費される FORTRAN および C プログラマのプログラミング労力を等しく減らす.
- データおよび補助データの誤解に由来するエラーを減らす.
- あるアプリケーションからの出力を他の入力として使用することを容易にする.
- インターフェースの標準を確立し, Unidata システムに新しいソフトウェアを含めることを簡単にする.

このプロジェクトはある面ではすでに成功を収めている. netCDF は現在, CRAY からパーソナルコンピュータにわたり殆どの UNIX ワークステーションを含むコンピューティングプラットフォームで使用されている. netCDF はあるコンピュータ上で (例えば FORTRAN を用いて) 複雑なデータセットを作成し, 同一の自己記述的なデータセットを他のコンピュータ上で (例えば C を用いて) 中間での変換なしで引出すために使うことができる — netCDF ファイルはネットワークを越えて転送することができ, また, 適切なネットワークファイルシステムを使ってリモートアクセスすることもできる.

Unidata 以外のソフトウェアが netCDF アクセスを使用することは Unidata の主要な顧客の利益になる — このような使用が結果的に Unidata の情報を解析および表示するためのより多くのオプションを与える — と信じているので, netCDF のライブラリはライセンスや他の重大な制約など無しで配布されており, 最新のバージョンをアノニマス FTP で取得することができる. 見たところ, このソフトウェアは大気科学のコミュニティを越えて, 広範囲の学会で大いに受け入れられているようであり, 現在, 多数のパブリックドメインおよび商用のデータ解析システムが netCDF ファイルを入力として受け入れることができるようになっている.

幾つかの組織が netCDF をデータアクセスの標準として採用しており, National Center for Supercomputer Applications (NCSA, Urabana-Champaign の the University of Illinois に関連している) では, netCDF プログラミングインターフェースを “HDF ファイル” にデータを格納および引出すための手段としてサポートしようという努力が進行中である. この “HDF ファイル” は一般的な NCSA のツールによって使われているフォーマットである. 我々はこれらの努力を支援し, 協力してきている.

netCDF ソフトウェアに対して提供されているサポートのレベルについての質問が時々寄せられる. Unidata の公式な立場は, netCDF ライブラリに添付されている著作権注意書に言明されているが, ソフトウェアは “そのまま” 配布される

ということである. 実際, ソフトウェアは時々更新されており, Unidata は近い将来に向けて改善を続けるつもりである. Unidata のミッションは米国の大学の地球科学者に奉仕することであるので, そのコミュニティによる問題の報告は必ず最大限の注意を払われる.

我々は, 読者がこのソフトウェアを有用なものでると感じ, 改良の提案だけでなくアプリケーションにおけるフィードバックをも我々に与えてくれることを期待している.

概要

Network Common Data Form (netCDF) インターフェースの目的は, 配列指向のデータを, 自己記述的でネットワーク透過性のある形式で作成, アクセス, および共有することを可能にすることである. “自己記述的” というのは, ファイルがその内包するデータを定義する情報を含んでいるということを意味している. “ネットワーク透過性” というのは, 整数, 文字および浮動小数点数を異なった方法で格納するコンピュータによってアクセスすることができるような形式でファイルが表現されているということを意味している. 新しいデータセットを作成するために netCDF インターフェースを使うことにより, データに可搬性を持たせることができる. データアクセス, 管理, 解析および表示するためのソフトウェアに netCDF インターフェースを用いることにより, ソフトウェアをより一般的に便利にすることができる.

netCDF ソフトウェアは netCDF データにアクセスするための C および FORTRAN インターフェースを含んでいる. これらのライブラリは普通のコンピューティングプラットフォームの多くで利用できる. netCDF データアクセスのための C++および perl インターフェースもまた Unidata から取得可能である. netCDF のユーザのコミュニティはソフトウェアの付加的なプラットフォームへの移植および他のプログラミング言語のためのインターフェースにも貢献してきた. netCDF ソフトウェアライブラリのソースコードは自由に取得することができ, そのことは配列指向データおよびデータを便利にするソフトウェアの両方を共有するのを支援するものである.

このユーザーズガイドは netCDF データモデルを提出しているが, C および FORTRAN インターフェースのみだけしかドキュメントしていない. UNIX システムのための参考文書は, C および FORTRAN インターフェースのための UNIX‘man’ ページの形式のものとして, netCDF ソフトウェアと一緒に取得できる. netCDF についてのより広範な付加的情報, それは netCDF データを扱う他のソフトウェアへのポインタを含んでいる, は netCDF World Wide Web サイト (‘`http://www.unidata.ucar.edu/packages/netcdf/`’) から取得することができる.

参考文献

1. Brown, S. A, M. Folk, G. Goucher, and R. Rew, “Software for Portable Scientific Data Management,” *Computers in Physics*, American Institute of Physics, Vol. 7, No. 3, May/June 1993.
2. Fahle, J., *TeraScan Applications Programming Interface*, SeaSpace, San Diego, California, 1989.
3. Fulker, D. W., “The netCDF: Self-Describing, Portable Files—a Basis for ‘Plug-Compatible’ Software Modules Connectable by Networks,” *ICSU Workshop on Geophysical Informatics*, Moscow, USSR, August 1988.
4. Fulker, D. W., “Unidata Strawman for Storing Earth-Referencing Data,” *Seventh International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, New Orleans, La., American Meteorology Society, January 1991.
5. Gough, M. L., *NSSDC CDF Implementer's Guide (DEC VAX/VMS) Version 1.1*, National Space Science Data Center, 88-17, NASA/Goddard Space Flight Center, 1988.
6. Jenter, H. L. and R. P. Signell, “NetCDF: A Freely-Available Software-Solution to Data-Access Problems for Numerical Modelers,” *Proceedings of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling*, Tampa, Florida, 1992.
7. Raymond, D. J., “A C Language-Based Modular System for Analyzing and Displaying Gridded Numerical Data,” *Journal of Atmospheric and Oceanic Technology*, **5**, 501-511, 1988.
8. Rew, R. K. and G. P. Davis, “The Unidata netCDF: Software for Scientific Data Access,” *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, California, American Meteorology Society, February 1990.
9. Rew, R. K. and G. P. Davis, “NetCDF: An Interface for Scientific Data Access,” *Computer Graphics and Applications*, IEEE, pp. 76-82, July 1990.
10. Treinish, L. A. and M. L. Gough, “A Software Package for the Data Independent Management of Multi-Dimensional Data,” *EOS Transactions*, American Geophysical Union, **68**, 633-635, 1987.

Chapter 1　イントロダクション

1.1　netCDF データインターフェース

Network Common Data Form, すなわちnetCDF, はデータを配列の形で格納および引出すためのデータアクセス関数のライブラリへのインターフェースである．配列はn次元(nは0,1,2,...)の矩形構造ですべて同じデータタイプ(例えば, 8ビット文字, 32ビット整数)の要素を含んでいるものである．スカラー(単なる一つの値)は0次元の配列である．

netCDFはデータを自己記述的でネットワーク透過性のオブジェクトで単純なインターフェースを通じてアクセスできるものの集合体として見ることをサポートする抽象概念である．配列の値は, データがどのように格納されているかの詳細を知らなくとも直接アクセスできる．どんな単位が使われているかといったようなデータに関する補助情報は, データとともに格納することができる．一般的なユーティリティおよびアプリケーションプログラムはnetCDFファイルにアクセスして, 指定されたデータフィールドを変換, 結合, 解析, および表示することができる．このようなアプリケーションの発展はデータへのアクセシビリティの改善および配列指向のデータを管理, 解析および表示するためのソフトウェアの再利用性の改善につながるものである．

netCDFソフトウェアは抽象的データタイプを実装するものである, これはnetCDFファイルのデータにアクセスし扱うためのすべての命令は, インターフェースによって提供されている関数のみを使わねばならないということを意味している．データの表現はインターフェースを使うアプリケーションからは隠されており, そのため, データの格納法は既存のプログラムに影響を与えることなく変更することができる．netCDFの物理的な表現はデータが書かれたコンピュータに独立であるように設計されている．

UnidataはC, FORTRAN, C++, およびperlのため, およびいろいろなUNIXオペレーティングシステムのためのnetCDFインターフェースをサポートしている．このソフトウェアは, 毎回のメジャーリリースの前に, 他の2, 3のオペレーティングシステムについても, それらのシステムにアクセスするユーザの助力によって移植およびテストされている．UnidataのnetCDFソフトウェアはFTPによって無料で得ることができ, そのことは広範な使用を支援している．

1.2　netCDF はデータベース管理システムではない

配列指向のデータを格納するためになぜ既存のデータベース管理システムを使わないのか? これは, 相関的データベースソフトウェアはnetCDFインターフェースでサポートされているようなデータアクセスには適していないからである．

まず第一に, 相関モデルをサポートする既存のデータベースシステムは, 多次元のオブジェクト(配列)をデータアクセスの基本単位としてはサポートしていない．配列を関係として表現することは便利なデータアクセスの種類のいくつかを不便にしてしまい, 多次元データおよび座標系についての抽象化を殆ど与えない．配列指向のデータを引出し, 修正し, 数学的に扱いそして可視化するためには全く異なったデータモデルが必要である．

このことに関連して, 一般的な目的を持つデータベースシステムには第二の問題点がある: 大きな配列に対してはパフォーマンスが悪いということである．衛星画像の集積, 科学的モデルの出力および長期間の全球気象観測は, 殆どのデータベースシステムが組織化し効率的な引出しのための索引づけする能力を超えている．

最後に, 一般目的のデータベースシステムは, 資源およびアクセス効率を非常に犠牲にして, 配列指向のデータを解析, 管理および表示するためには不必要な機能を提供している．例えば, 手の込んだアップデート機能, 履歴の検査, フォーマットの報告, および業務処理のために設計された機構は, 殆どの科学的アプリケーションには不要である．

1.3　ファイルフォーマット

ネットワーク透過性(マシン独立性)を達成するために, netCDFはXDR (eXternal Data Representation, ‘`ftp://ds.internic.net/rfd/rfc1832.txt`’を参照), すなわちデータの表現およびコード化のために提案されている標準プロトコル, によって実装されている．XDRはデータをマシン独立なビットの列へコード化する方法を提供している．XDRは幅広い種類のコンピューターに実装されており, この実装には, 8ビットのバイトがコンシステントにコード化およびデコード化できるということだけが必要とされる．

netCDFファイルの全体の構造は第9章[netCDFファイル構造および性能] 53ページで記述する．

フォーマットの詳細はAppendix B[ファイルフォーマット仕様] 67ページで記述する．しかし, ユーザがこのフォーマット仕様を用いてnetCDFファイルを読み書きするための独立した低レベルのソフトウェアを開発するというのはお勧めできない, なぜなら, このような行為はフォーマットが変更された際に互換性の問題を引き起こす可能性があるからである．

1.4 性能は?

netCDF の目標の一つは大きなデータセットの小さな部分集合への効率的なアクセスをサポートすることである. この目標をサポートするために, netCDF は順次アクセスではなくむしろ直接アクセスを使用する.

XDR のオーバヘッドの量は, データタイプ, コンピューターのタイプ, データアクセスの不均一性, および実装がコンピューターに合せてどれだけ上手くチューンされているかといったことを含む多くの要素に依存している. このオーバヘッドはアプリケーションに使われる全体の資源に比較すれば概して小さい. どんな場合においても, XDR レイヤーのオーバヘッドは, 移植性のある, ネットワーク透過なデータアクセスのために払う対価として, たいてい理にかなったものである.

データアクセスの効率は netCDF を設計し実装する際の重要な関心事であるが, 非効率的な方法でデータにアクセスするために netCDF インターフェースを用いてしまうようなこともまだありうる: 例えば, 各記録から単一の値だけを必要とするようなデータの断面を要求してしまうようなことによって. インターフェースをいかに効率的に用いるかについての指針は 9 章 [netCDF ファイル構造および性能]53ページに与えている.

1.5 netCDF はよい記録フォーマットか?

netCDF は配列を格納するための一般目的の記録フォーマットとして使うことができる. netCDF でデータを圧縮することは可能である (例えば, 低精度の浮動小数点数をコード化するのに 32 ビットの数の配列の代わりに 8 ビットまたは 16 ビット整数の配列を使うなど) が, netCDF の現行バージョンはデータを最適に圧縮するようには設計されていない. そのため, netCDF は特定のデータセットの特別な性質についての知識を使った特別な目的の記録フォーマットより多くのスペースを必要とするかもしれない.

1.6 慣習に従った自己記述的データの生成

単に netCDF を使うだけではデータを “自己記述的” かつ人間および機械の両方に意味があるようにするためには不十分である. 変数および次元の名前は意味を持ち, あらゆる関連した慣習に従っているべきである. 次元は対応する座標変数の値を (測定できる場合には) 持っているべきである.

属性は補助情報を供給する上で重要な役割を果す. 関連する慣習を用いた関連した標準属性すべてを使うことは重要なことである. 8.1節 [属性の慣習]45ページでは, 予約された属性 (netCDF ライブラリによって使用される) および一般的なアプリケーションソフトウェアのための属性の慣習を記述する.

数多くの団体がそれら自体の付加的な慣習および netCDF データの形式を定義している. これらの慣習の記述は, それらを組み込んだサンプルとともに netCDF Conventions site
(‘`http://www.unidata.ucar.edu/packages/netcdf/conventions.html`’) から参照することができる.

これらの慣習は, ふさわしいところに使われるべきものである. 付加的な慣習はローカルな使用のためにしばしば必要とされている. このような慣習は, もし同じ分野の他のユーザに関心を持たせたいと思うならば, 上記の netCDF Conventions site に寄せられるべきである.

1.7 netCDF インターフェースの背景と発展

netCDF インターフェースの開発は Unidata の必要に関連した控え目な目標: すなわち Unidata アプリケーションと取り込まれたリアルタイムの気象データとの間の共通インターフェースを提供するという目標を持って始まった. Unidata のソフトウェアは, C および FORTRAN の両方からのアクセスを受け付け, 多様なハードウェアプラットフォーム上で実行されるよう意図されていたので, Unidata の目標を達成することは, より広い状況で有用なパッケージを提供しうる可能性を持っていた. このパッケージを幅広く取得可能なものにし, かつ同じような必要を持つ他の組織と共同することによって, 科学的なデータアクセスのためのソフトウェアが同じ学問分野の他の人々にはまれにしか再利用されず, 学問分野間では殆ど全く再利用されなかった (Fulker, 1988) という当時の状況を改善することを我々は希望したのである.

netCDF ソフトウェアに採用された重要なコンセプトは, NASA Goddard National Space Science Data Center (NSSDC) で開発されたデータアクセスソフトウェアを記述した論文 (Treinish and Gough, 1987) に由来する. このソフトウェアによって供給されていたインターフェースは, Common Data Format(CDF) と呼ばれていた. NASA CDF は当初, 配列を格納するための抽象化をサポートするための, プラットフォームに特化した FORTRAN ライブラリとして開発された.

NASA CDF パッケージは, 広範なアプリケーションの山の中で数多くの異る種類のデータを使うために使用されてきていた. これは, (13 サブルーチンのみという) 簡明さ, 格納フォーマットからの独立性, 一般性, ユーザのデータについての論理的な見方をサポートする能力, および一般的なアプリケーションのサポート, のおかげによるものである.

Unidata は CDF に関するワークショップを 1987 年 8 月に Boulder で開いた. 我々は, CDF FORTRAN ライブラリを拡張し, C インターフェースを定義し, かつ既存の NASA インターフェースとの互換性を保ちつつ単一の呼び出しでデータ集合体へのアクセスを許容するために, NASA と共同する可能性を探究することを提案した.

これとは独立に, New Mexico Institute of Minig and Technology の Dave Raymond は, 自己記述的で配列指向のデータへの順次アクセス, およびデータの加工, 解析および表示への "パイプとフィルター"(すなわち "データフロー") 的アプローチをサポートする UNIX 上の C ソフトウェアパッケージを開発していた. このパッケージもまた "Common Data Format" という名前を使っていたが, これは後に C-Based Analysis and Display System (CANDIS) と改められた. Unidata は Raymond の仕事 (Raymond, 1988) を知り, 名前つきの次元および単一のデータオブジェクト内で型の違いをもつ変数を使用するなどといった彼のアイディアのいくつかを Unidata の netCDF インターフェースに組み込んだ.

1988 年初頭, Unidata の Glenn Davis は XDR の上に被せる形で netCDF パッケージのプロトタイプを C で開発した. このプロトタイプは, CDF インターフェースの単一ファイルでネットワーク透過な実装を達成するためのコストが許容範囲内であること, および得られたプログラムは UNIX と VMS システムの両方に移植可能であること, を証明した. しかし, そのプロトタイプはまた, 小型で移植性がありかつ NASA CDF 互換で FORTRAN インターフェースで望まれるような一般性をもつようなものは実際的でないことも示した. 以来, NASA の CDF と Unidata の netCDF は別々に発展してきたが, 最近の CDF バージョンは netCDF と多くの共通点を持っている.

1988 年初頭, 1987 年の Unidata CDF ワークショップの参加者の一人であった, SeaSpace 社 (California 州 SanDiego にある商用ソフトウェア開発会社) の Joe Fahle は, NASA CDF ソフトウェアをいくつか重要な面で拡張した CDF パッケージを C で独立に開発した (Fahle, 1989). Raymond のパッケージと同様に, SeaSpace CDF ソフトウェアは関連しない型を持った変数が同じデータオブジェクトに含まれることを許容し, 多次元配列へのアクセスの一般的な形を可能にしていた. Fahle の実装は, SeaSpace 社において, 画像処理システムのいろいろな段階での中間的な格納形式として使われていた. このインターフェースとフォーマットはその後 Terascan データフォーマットに発展していっている.

我々は, Fahle のインターフェースを研究した後, このインターフェースが NASA インターフェースを我々の目的に拡張しようとして我々が確認した問題の多くを解決しているという結論に達した. 我々は, 1988 年 8 月, 一つの Unidata netCDF インターフェースについての合意を達成し, かつ残った未解決問題を解決するための小規模なワークショップを召集した. 参加者は, SeaSpace の Joe Fahle, Apple の Michael Gough(NASA CDF の著者の一人), University of Miami の Angel Li(我々の netCDF のプロトタイプを VMS 上に実装し, ユーザになる可能性もあった), および Unidata のシステム開発スタッフであった. このワークショップでは, いくつかの簡易化法が見いだされた後に合意が達成された. 提案された Unidata netCDF インターフェース仕様にこのワークショップの結果を組み込んだドキュメントは, Glenn Davis と Russ Rew がこのソフトウェアのファーストバージョンを実装する前に, 意見を求めるために広く配布された. 他のデータアクセスインターフェースとの比較および netCDF を使った経験は, (Rew and Davis, 1990a), (Rew and Davis, 1990b), (Jenter and Singnell, 1992), および (Brown, Folk, Goucher, and Rew, 1993) で議論されている.

1991 年 10 月, 我々はバージョン 2.0 の netCDF ソフトウェア配布を発表した. C インターフェースのちょっとした修正 (次元のサイズを `int` ではなく `long` と宣言するようにした) によって, MS-DOS コンピューターのような安価なプラットフォーム上での netCDF の利便性を向上させた, しかも他のプラットフォーム上での再コンパイルも必要としなかった. インターフェースへのこのこの変更によって, 関連したファイルフォーマットへの変更が必要となることもなかった.

1993 年 6 月にリリースされた netCDF バージョン 2.3.2 は同じファイルフォーマットを維持したが, 記録への単一呼び出しアクセス, 不連続なデータを含む断面へのアクセスの最適化, 特定の次元に沿った部分サンプリング ('stride' を使う), 不連続なデータへのアクセス ('mapped array sections' を使う), ncdump と ncgen ユーティリティの改良, および試験的な C++インターフェース, を追加した.

1.8 過去のリリースから何が新しくなったか?

このガイドは, netCDF2.4 の 1996 年 2 月版を文書化したものであり, この版は以前の版と同じファイルフォーマットを維持しているが, バージョン 2.3.2 からの以下の変更を含んでいる:

- 新しいプラットフォームのサポート
- Cray 上での著しい最適化
- インストールの簡単化

- ドキュメントの改善
- C++インターフェースの追加
- 報告されているバグのフィックス

`long` のサイズが `int` のサイズより大きいような新しいプラットフォーム上で netCDF をサポートするために, 新しいリリースでは, C および C++インターフェースにおいては `nclong` 型宣言を使用するように完全に統合した.

C++インターフェースに追加と変更が加えられ, 記録を渡り歩いたり, 複数の netCDF ファイルへの同一座標アクセス, および単一記録へのアクセスすることをより容易にした.

1.9 netCDF の制限

netCDF データモデルは名前付き属性を持つ名前付き配列変数の集合に系統だてられるデータに広く適用できるが, このモデルおよびソフトウェアの実装にはにはいくつか重要な制限がある.

このデータモデルはデータのネスト構造をサポートしていない. netCDF インターフェースは, ツリー表現, ネストした配列, または他の再帰的なデータ構造, について殆ど役に立つものを提供していないが, これは主に, FORTRAN インターフェースがあらゆる netCDF データセットを読み書きできなければならないという要請によるものである. 間接的な使用および慣習によって, 表現可能なネスト構造もあるが, これは結果として, netCDF の最終目標である "自己記述的データ" の性質を失ってしまう.

現在の実装の著しい制限は, 各 netCDF データセットに対して無制限な次元はたった一つしか許容されていいないということである. 複数の変数が一つの無制限な次元を共有しても良いが, それらはすべて一緒に発展せねばならない. そのため, netCDF モデルは複数の可変な次元サイズを持つ変数を扱えない. また, 同一ファイル内で異なる変数が異なる可変次元を持つことも不可能である. 矩形でない形を持つ変数 (例えば, "不調和な配列") は便利には表現できない.

インターフェースは座標変数に特化した機能, すなわち, 通常の番号付けの代りに次元に沿った位置を指定するために座標変数を使うような, は提供しない. データをビット列にパックする機能もまだない (XDR はこのような能力がない). このため, 9 ビットのデータの配列は, 便利にアクセスされるために 16 ビットの配列として格納されていなければならない. データセットのサイズは現在 2 ギガバイトに制限されている, これは, 32 ビットの符号付きオフセットを使っているからである.

最後に, 現在の実装は netCDF ファイルへの同時アクセスを制限している. 単一のファイルを同時に一人が書き込み, 複数人が読み込むことは許されるが, 複数人が同時に書き込むことはサポートしていない.

1.10 netCDF の将来計画

XDR は開発中の新しいソフトウェアに置き換えられる予定である. これによって, 機能の追加およびよりよい効率がもたらされる予定である.

現在の計画では, 透過なデータパッキング, 改善された同時性のサポート, データへのキーまたは座標値によるアクセス, 効率的な構造変更 (例えば, 新しい変数および属性) のサポート, 新しいデータタイプ, および格納形態に関係なく特定の型としてデータにアクセスするための, 型を気にしない C および FORTRAN インターフェースの追加, を加える予定である. 加えられうる他の望まれる拡張では, もし実現可能ならば, 他のファイルのデータ断面へのポインタ, ネストした配列 (不調和な配列の表現, ツリーおよび他の再帰的データ構造の許容), 2 ギガバイト以上のデータセットへのアクセスできる機能, および複数の無制限な次元, のサポートを加える.

Chapter 2 netCDF ファイルの構成要素

2.1 netCDF データモデル

netCDF ファイルは, 次元, 変数, および属性を含み, それらはすべて, それらを特定するための名前および ID ナンバーの両方を持っている. これらの構成要素は共に, データの意味および配列指向のデータセット中のデータフィールド間の関係を保存するために用いることができる. netCDF ライブラリは, 通常のファイル名に加えて, ファイル ID ナンバーによって指定された複数の netCDF ファイルに同時にアクセスすることを許容している.

netCDF ファイルは, 変数についてその名前, データタイプ, ランク (次元数), 次元および先頭のディスクアドレスを含んだシンボルテーブルを含んでいる. 各要素は, それらが特定される配列番号 (添字) の線形関数のディスクアドレスに格納される. これにより, これらの番号を記録中のフィールドとして, または記録の索引の中 (相関データベースのように) に格納する必要がなくなっている. このことは, 欠損値が多くなければ, 高速でコンパクトな格納法を提供している.

2.1.1 名前付けの慣習

次元, 変数および属性の名前は, 任意の英数字 (と, アンダースコア '_' およびハイフン '-') の列で構成され, 先頭は文字またはアンダースコアでなければならない. (しかし, アンダースコアから始まる名前はシステム用途のために予約されている.) netCDF ファイルにおいては大文字小文字は区別される.

2.1.2 network Common Data Form Language (CDL)

小さな netCDF の例を使って netCDF データモデルの概念を説明しよう. このファイルは次元, 変数, および属性を含んでいる. この簡単な netCDF オブジェクトを記述するのに使われる表記は, CDL(network Common Data form Language) と呼ばれ, netCDF ファイルを記述するための便利な手段を与える. netCDF ファイルシステムは, バイナリの netCDF ファイルから人間指向の CDL テキストファイルを生成, およびその逆を行うユーティリティを含んでいる.

```
netcdf example_1 {  // netCDF ファイルに対する CDL 表記の例

dimensions:          // 次元名およびサイズがまず宣言される
        lat = 5, lon = 10, level = 4, time = unlimited;

variables:           // 変数の型, 名前, 形, 属性
        float   temp(time,level,lat,lon);
                    temp:long_name     = "temperature";
                    temp:units         = "celsius";
        float   rh(time,lat,lon);
                    rh:long_name = "relative humidity";
                    rh:valid_range = 0.0, 1.0;      // 最小, 最大値
        int     lat(lat), lon(lon), level(level);
                    lat:units       = "degrees_north";
                    lon:units       = "degrees_east";
                    level:units     = "millibars";
        short   time(time);
                    time:units      = "hours since 1996-1-1";
        // グローバル属性
                    :source = "Fictional Model Output";

data:                // オプションのデータ割り当て
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
        rh      =.5,.2,.4,.2,.3,.2,.4,.5,.6,.7,
                 .1,.3,.1,.1,.1,.1,.5,.7,.8,.8,
                 .1,.2,.2,.2,.2,.5,.7,.8,.9,.9,
                 .1,.2,.3,.3,.3,.3,.7,.8,.9,.9,
                  0,.1,.2,.4,.4,.4,.4,.7,.9,.9;
}
```

netCDFファイルのCDL表記は, 後で説明する(10.5節[ncdump]58ページを参照)ユーティリティプログラム**ncdump**を使って自動的に生成される. もう一つのnetCDFユーティリティ**ncgen**はCDLの入力からnetCDFファイルを生成する(または, オプションで, netCDFファイルを生成するのに必要な呼び出しを含んだCまたはFORTRANソースコードを生成する)(10.4節[ncgen]57ページを参照).

CDL表記は簡単でしかも非常に自己説明的である. netCDFファイルの構成要素を説明すれば, より完全な説明になる. さしあたって, CDL文がセミコロンで終っていることに注意されたい. 読み易いように, 空白, タブ, および改行を使うのは自由である. CDL中, どの行でも'//'以下は注釈である. netCDFファイルのCDL記述は以下のような形をとる:

```
netCDF 名前 {
  dimensions: ...
  variables: ...
  data: ...
}
```

ここに, 名前は`ncgen`ユーティリティによってファイル名が生成される際のデフォルトとしてのみ使われる. CDF表記は3つのオプション部分から成り, それらは`dimensions`, `variables`, および`data`のキーワードで導入される. netCDFの次元の宣言は`dimensions`キーワードの後になされ, netCDF変数と属性は`variables`キーワードの後で定義され, 変数のデータ割り付けは`data`キーワードの後になされる.

2.2 次元

次元は実際の物理的な次元を表すために使われうる, 例えば, 時刻, 緯度, 経度, または高度などである. 次元は, 他の量, 例えば, 観測ステーションやモデルの実行番号など, を索引づけするために使うこともできる.

netCDFの次元は名前とサイズの両方を持っている. 次元のサイズは任意の正の整数である, ただし, netCDFファイルの一つの次元は`UNLIMITED`のサイズを持つことができる.

このような次元は無制限次元または記録次元と呼ばれる. 無制限次元を持った変数は, その次元に沿ってどんな長さにすることもできる. 無制限次元の番号は, 伝統的な記録指向のファイル中の記録番号にようなものである. netCDFファイルは無制限次元を一つまでしか持てないが, これは無くてもよい. 変数が無制限次元を持つ場合には, その次元は最も意味のある(最もゆっくり変わる)次元でなければならない. 従って, あらゆる無制限次元はCDL形式における最初の次元でなければならない(Cの宣言では最初のFORTRANでは最後の次元である).

CDLの次元宣言はCDLキーワード`dimensions`の後の一つまたはそれ以上の行ですることができる. 同一行で複数の次元を宣言することもでき, その場合はコンマで区切る. 各宣言は名前=サイズの形式である.

上の例では4つの次元: `lat`, `lon`, `level`, および`time`がある. 最初の3つは割り当てられた固定サイズである; `time`は`UNLIMITED`のサイズを割り当てられているが, これは, 無制限次元であることを意味している.

netCDFファイル中の名づけられた基本単位は変数である. 変数が定義されると, その形は次元のリストとして明示される. これらの次元はすでに存在している.

次元の数はランク(別名ディメンショナリティ)と呼ばれる. スカラー変数はランク0であり, ベクトルはランク1, および行列はランク2である.

変数の形を特定する際に, 同じ次元を一度以上使うことができる. 例えば, `correlation(instrument, instrument)`は異なる測器を用いた測定間の相関を与える相関行列とすることができる. しかし, 物理的な空間/時間の次元に対応する次元を持つデータは, 異なる次元からなる形を持っているべきである, たとえこれらのうちのいくつかが同じサイズであっても.

2.3 変数

変数は, データの大部分をnetCDFファイルに格納するために用いられる. 変数は同じ型の値の配列を表現する. スカラー値は0次元配列として扱われる. 変数は名前, データ型, および変数が生成さえた際に指定された次元のリストで記述された形を持っている. 変数は関連した属性も持つことができ, それは変数が生成さえた後に追加, 削除, または変更することができる.

変数データ型は, netCDFの型の小さな集合の一つであり, その集合は, Cインターフェースでは名前`NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, および`NC_DOUBLE`を持ち, FORTRANインターフェースでは対応する名前`NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および`NCDOUBLE`を持っている. CDL表記では, これらの型はより簡単な`byte`, `char`, `short`, `long`, `float`, および `double` で与えられる．CDL 表記では，`int` は `long` の同意語として使ってもよく，また `real` は `float` の同義語として使ってもよい．各タイプの正確な意味については後に 3.1節 [netCDF データ型]13ページでの議論する．

CDL 変数宣言は CDL ユニット中でキーワード `variables` の後でなされる．それらは以下の型をとる．次元を持つ変数の場合:

型　変数名 (次元名 **1**, 次元名 **2**, ...);

スカラー変数の場合:

型　変数名;

上記の CDL の例において, 6 つの変数がある．以下で議論するように，これらのうちの 4 つは座標変数である．残りの変数 (主変数と呼ばれることもある)`temp` および `rh` は一般的にデータと見なされるものを含んでいる．これらの各変数は無制限次元の `time` を第一次元として持っているので，それらは記録変数と呼ばれる．記録変数ではない変数は，その次元サイズの積で与えられる固定サイズ (データ値の数) を持つ．記録変数のサイズも次元サイズの積であるが，この場合にはこの積は変化する．なぜなら，この積には変化しうる無制限次元のサイズを含んでいるからである．無制限次元のサイズは記録の数である．

2.3.1 座標変数

変数が次元と同じ名前を持つのは正当なことである．このような変数は netCDF ライブラリにおいて特別な意味は持たない．しかし，このような変数はこのライブラリを用いて特別に扱われるという慣習がある．

次元と同じ名前を持つ変数は座標変数と呼ばれる．概して，この変数は，その次元に対応する物理的な座標を定義する．上記の CDL の例は `lat`, `lon`, `level`, および `time` の座標変数を含み，それらは以下のように定義されている:

```
        int     lat(lat), lon(lon), level(level);
        short   time(time);
   ...
 data:
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
```

これらは，これらの座標に沿った位置に対応する緯度，経度，気圧，および時刻を定義している．従って，1000, 850, 700, および 500 ミリバールに対応する高度，および北緯 20, 30, 40, 50, 60 度でのデータがある．各座標変数はベクトルであり，同じ名前の次元のみから構成される形を持つことに注意されたい．

次元に沿った位置は番号によって特定される．これは最小値として C プログラムでは 0, FORTRAN では 1 を持つ整数である．従って，700 ミリバールレベルは番号として C では 2 を FORTRAN では 3 を持つ．

次元が対応する座標変数を持つ場合，この座標変数は，次元に沿った位置を特定するための，代替的でかつより便利なことも多い手段を提供する．座標変数を利用する現在のアプリケーションパッケージは共通して，座標変数が狭義の単調 (すべての値が異なりかつ増加または減少している) な数値ベクトルであることを仮定している．座標変数の値として，文字ラベルのようなものをも許容するようなより一般的な慣習を定義しようという計画がある．

2.4 属性

netCDF の属性はデータについてのデータ (補助データまたはメタデータ) を格納するために使われる．これは伝統的なデータベースにおけるデータ辞書および図表に格納される情報に多くの面で類似したものである．殆どの属性は特定の変数に関する情報を提供する．これらはその変数の名前 (または ID) にその属性の名前を付加したものによって識別される．

いくつかの属性はファイルに関する情報を総括して与え，グローバル属性と呼ばれる．これらは空白の変数名 (CDL では) または特別な空白変数 ID(C または Fortran で) に属性名を付加したもので識別される．

属性は関連する変数 (グローバル属性に対してはない), 名前，データ型，長さ，および値を持っている．現在のバージョンでは，全ての属性をベクトルとして扱う; スカラー値は単一要素ベクトルとして扱われる．

慣習的な属性名は, それが適当な場合には使わるべきである. 新しい名前はできるだけ意味あるものにするべきである.

属性の型は生成の際に指定される. 属性のために許されている型は, 変数のための netCDF データ型と同一である. 同じ名前の属性でも, 対応する変数が異なっていれば, 型が異なっている場合もある. 例えば, `long` 型の変数に対して有効なデータの最大値を指定する属性 `valid_max` は `long` 型であるべきである一方, `double` 型の変数に対する属性 `valid_max` は `double` 型であるべきである.

属性は変数や次元よりも動的である; すなわち, 生成された後でも削除することができ, かつそれらの型, 長さ, および値を変えることができる. 一方, netCDF インターフェースは変数を削除したり, その型や形を変えるための手段を提供していない.

属性を定義する CDL 表記は以下の形をとる. 変数の属性の場合:

変数名:属性名 = 値のリスト;

グローバル属性の場合:

:属性名 = 値のリスト;

各属性の型および長さは CDL において陽には宣言されない; これらは属性に割り当てられた値から得られる. 属性のすべての値は同じ型でなければならない. いろいろな netCDF 型の定数値のために使われる記法は後で議論する (10.3節 [データ定数に対する CDL 表記] 56ページを参照).

netCDF の例 (2 章 [CDL 例]9ページを参照) において, `units` は変数 `lat` に対する属性であり, 13 文字列の値 ‘`degrees_north`’ を持つ. そして, `valid_range` は変数 `rh` に対する属性であり, 長さ 2 で値 ‘`0.0`’ および ‘`1.0`’ を持つ.

例として挙げた netCDF ファイルに対して一つのグローバル属性—`source`—が定義されている. これはデータを説明するための文字列である. 現実の netCDF ファイルでは, その源, 来歴, 慣習, およびファイルの他の性質を総括して記述するためのより多くのグローバル属性を持ちうる.

netCDF ファイルを加工する一般的なアプリケーションの多くは, 標準的な属性の慣習を前提としており, 特に理由がなければその慣習に従うことを強く勧める. `units`, `long_name`, `valid_min`, `valid_max`, `valid_range`, `scale_factor`, `add_offset`, `_FillValue`, および他の慣習的な属性の情報については 8.1節 [属性の慣習]45ページを参照.

netCDF ファイルが最初に定義されてからずっと後になっても, 属性は追加することができるので, 後で便利になるかもしれない属性すべてを見越して使う必要はない. しかし, 既存のファイルに新しい属性を追加することは, そのファイルをコピーするのと同じコストがかかってしまう. より広範な議論については, 9 章 [netCDF ファイル構造および性能] 53ページを参照のこと.

2.5 属性と変数の違い

データ大部分のために使われる変数とは対照的に, 属性は補助的データまたはデータに関する情報のために使われる. netCDF オブジェクトに関連, 属性に格納される補助的データの総量は, 概してメモリ上に常駐できるほど小さい. しかし, 変数はメモリ上に完全にはめこむには大き過ぎることが多く, 加工のためには部分に分ける必要がある.

属性と変数との間のもう一つの違いは, 変数は複数次元を持ちうるということである. 属性はすべてスカラー (単一の値) またはベクトル (一つの固定された次元) のどちらかである.

変数は, データ値を割り当てられる前に, 名前, 型, および形を決められて生成されるので, 変数は値をもたずに存在しうる. 属性の値は属性が生成される際に指定されるので, 属性は値を持たずには存在できない.

変数は属性を持つことができるが, 属性は属性を持つことができない. 変数に割り当てられた属性はその変数と同じ単位を持ってもよいし (例: `valid_range`), または単位を持たなくてもよい (例: `scale_factor`). 関連する変数の単位と異なる単位を必要とするデータを格納したい場合は, 属性よりむしろ変数を使うべきである. より一般には, もしデータがそれらを説明するための補助データを必要とするか, 複数次元であるか, それらの値を索引づけするための定義された netCDF 次元のいずれかを必要とするか, または重大な格納量を必要とする場合には, そのようなデータは属性としてではなくむしろ変数として表現されるべきである.

Chapter 3 データ

この章では 6 つの基本 netCDF データ型, netCDF でサポートされているデータアクセスの種類, および配列以外のデータ構造を netCDF ファイルにいかに実装できるかということについて議論する.

3.1 netCDF データ型

netCDF でサポートされている基本の型は現在以下のようになっている:

`character`	文を表現するための 8 ビット文字
`byte`	符号つきまたは符号なしの 8 ビット整数 (以下の議論を参照)
`short`	符号つき 16 ビット整数
`long`	符号つき 32 ビット整数
`float`	32 ビット IEEE 浮動小数点数
`double`	64 ビット IEEE 浮動小数点数

`byte` が追加されていることおよび符号なしの型が足りないことを除けば, netCDF は C と同じ基本データ型をサポートしている. 基本データ型の名前は CDL の予約語になっているので, 変数, 次元, および属性の名前として型の名前を使ってはならない.

今のところ, `byte` は符号つき (-128 から 127) または符号なし (0 から 255) のどちらとも解釈できる. netCDF の現在のバージョンは 8 ビットのバイトをそれが符号つきかどうかを知る必要なしに単純に読み書きする. しかし, netCDF の将来のバージョンでデータの詰め込み機能が追加されれば, 値の算術演算が必要となるので, このような目的のために `byte` データは符号つきと解釈されるようになるだろう.

これらの型は, C および FORTRAN プログラマにとってなじみがあり, 特定のコンピューターに独立した (XDR を使えば) 明確な外部表現を持ち, かつデータ精度と各データを表現するのに必要なビット数との間のトレードオフ関係を適度に広い範囲で与える, といった理由で選ばれた. netCDF データ型と言語のデータ型との間の対応については, 7節 [変数]31ページを参照のこと.

64 ビット整数および n ビットへの詰め込みを含んだ新しいデータ型も計画されている.

3.2 データアクセス

netCDF データにアクセス (読みまたは書き) するために, オープンされた netCDF ファイル, netCDF 変数, および変数の要素を特定する情報 (例えば番号) を指定する. さらに, netCDF インターフェースは記録指向のデータアクセス形式もサポートしている.

データへのアクセスは直接である. これはすなわち大きなデータセット中の小さな部分集合に, 最初にその前にあるすべてのデータにアクセスするようなことなく効率的にアクセスできるということを意味している. 変数を指定してデータの読み書きをすると, ファイル中での位置を指定していなければ, ファイル中に他の変数がどのくらいあるかということと独立してデータアクセスができ, データにより多くの変数を追加するようなデータフォーマットの変更によってプログラムが影響されなくなる.

C および FORTRAN インターフェースにおいて, データにアクセスしようとするたびにファイルを名前で指定するのではなく, 代わりにファイル ID と呼ばれる小さな整数 (これはファイルが最初に生成またはオープンされた際に得られる) によって指定される. 同様に, 変数もデータアクセスのたびに名前で指定されるわけではなく, 変数 ID(netCDF ファイル中の変数を特定するために使われる小さな整数) で指定される. (C++インターフェースにおいては, オープンされた netCDF ファイルおよび変数はオブジェクトであるので, ID は必要ない.)

3.2.1 データアクセスの形式

netCDF インターフェースは, オープンされた netCDF ファイル中のデータ値への直接アクセスについていくつかの形式をサポートしている. これらのアクセスの形式一つ一つを, 一般性を増していく順番で説明する:

- インデックスベクトルを指定した個々の要素へのアクセス;
- インデックスベクトルおよびカウントベクトルを指定した配列部分へのアクセス;
- インデックスベクトル, カウントベクトルおよびストライドベクトルを指定した部分的にサンプルされた配列部分へのアクセス; および

- インデックスベクトル, カウントベクトル, ストライドベクトルおよびインデックスマッピングベクトルを指定したマップされた配列部分へのアクセス

これらの4つの型のベクトル (インデックスベクトル, カウントベクトル, ストライドベクトルおよびインデックスマッピングベクトル) はすべて各次元の要素を持ったベクトルである. n 次元 (ランク = n) の変数について, n 要素のベクトルが必要である. 変数がスカラー (次元なし) なら, これらのベクトルは無視される.

配列部分は “板きれ” すなわち2つのベクトルで指定された連続した矩形のブロックである. インデックスベクトルは原点に最も近い隅の要素の番号を与える. カウントベクトルは, 変数の各次元に沿った縁の長さを順番に与える.

部分的にサンプルされた配列部分は, ストライドベクトルが追加されてサンプリングを指定するということを除いて配列部分に類似している. このベクトルは, 各次元について, その次元に沿ってとられるべきストライドの長さを与える要素を持つ. 例えば, ストライドが4であるというのは, 対応する次元に沿って4つおきの値という意味である. アクセスされる値の総数は, 各縁の長さ (カウントベクトルで定義される) を対応するストライドで割ったものの上限 (すなわち, 整数へ丸め) の積である.

マップされた配列部分は, インデックスマッピングベクトルが追加され, メモリ上に並べられた netCDF 変数に関連したデータ値にどのようにアクセスするかを指定できるようになっているという点を除けば部分的にサンプルされた配列部分に類似している. 参照位置からの各値の (バイトでの) オフセットは, 各インデックスに対応するインデックスマッピングベクトルの要素を掛けたものの和で与えられる. アクセスされる値の個数は部分的にサンプルされた配列部分の場合と同じである.

マップされた配列部分の使用については以下でより完全に議論するが, より普通に使われる配列部分アクセスの例を示すことにする.

3.2.2 配列部分アクセスの例

先の netCDF ファイルの例 (第2章 [CDL 例]9ページ参照) において, `temp` すべての値のあるレベル (例えば, 2番目) における断面を読み込みたいとし, その netCDF ファイルには現在3つの記録 (`time` の値) があるとする. 次元が

```
lat = 5, lon = 10, level = 4, time = unlimited;
```

と定義されており, 変数 `temp` が CDL 表記において

```
float   temp(time, level, lat, lon);
```

と宣言されていることを思い出そう.

対応する C 変数で, ある唯一のレベルのデータを保持するものは以下のように宣言される:

```
#define LATS  5
#define LONS 10
#define LEVELS 1
#define TIMES 3                    /* 今のところでは */
    ...
float   temp[TIMES*LEVELS*LATS*LONS];
```

これはデータを1次元配列に保持するためのもので,

```
    ...
float   temp[TIMES][LEVELS][LATS][LONS];
```

とすれば多次元配列のための宣言になる.

FORTRAN では, 次元は CDL 宣言とは逆で, 第一次元が先に変化し, 記録変数の最後の次元が記録次元となる. 従って, 唯一のレベルのデータを保持する対応変数の FORTRAN 宣言は以下のようになる:

```
INTEGER LATS, LONS, LEVELS, TIMES
PARAMETER (LATS=5, LONS=10, LEVELS=1, TIMES=3)
   ...
REAL TEMP(LONS, LATS, LEVELS, TIMES)
```

第2レベルのみで, すべての時刻, すべての緯度, そしてすべての経度を表現するデータブロックを指定するには, 隅と縁の長さを与えなければならない. この隅はCでは(0,1,0,0)で, — または, FORTRANでは(1,1,2,1) — である. なぜなら, `time`, `lon`, および`lat`の各次元については最初から, `level`次元については2番目の値から始めたいからである. 縁の長さはCでは(3,1,5,10), — FORTRANでは(10,5,1,3) — になる. これは, `time`についてはすべての3つの値, `level`の唯一の値, `lat`のすべての5つの値, および`lon`のすべての10個の値を取得したいからである. 全部で150(3*1*5*10)の浮動小数点数の値が返されるので, この数に十分な配列のスペースを用意しておかなければならない. データが返される順番は, Cでは最後の次元`lon`が最初に変化し, FORTRANでは最後の次元`LON`が最初に変化する:

```
               C                      FORTRAN

        temp[0][1][0][0]       TEMP(1, 1, 2, 1)
        temp[0][1][0][1]       TEMP(2, 1, 2, 1)
        temp[0][1][0][2]       TEMP(3, 1, 2, 1)
        temp[0][1][0][3]       TEMP(4, 1, 2, 1)
             ...                     ...

        temp[2][1][4][7]       TEMP( 8, 5, 2, 3)
        temp[2][1][4][8]       TEMP( 9, 5, 2, 3)
        temp[2][1][4][9]       TEMP(10, 5, 2, 3)
```

CおよびFORTRANインターフェースにおける次元の順番の相違は, ディスク上に値が格納されている順番が異なっているということではなく, 単にこの2つの言語の手続きインターフェースによってサポートされている順番が異なっているということの反映である. 一般に, netCDFファイルがCかFORTRANのどちらで書かれたかということは意味がない; どちらの言語一方で書かれたnetCDFファイルは, 他の言語で書かれたプログラムによっても読むことができる.

3.2.3 一般的な配列部分アクセスについてもうすこし

マップされた配列部分を使うことによって, 変数要素のディスクアドレスとそれらがメモリ上に収納されているアドレスとの間に意味のある関係づけをすることができる. 例えば, メモリ上の行列は, ディスク上の行列を並べ変えて, 要素の順番を全く異なるようにすることができる. 正常な配列部分において, ディスクとメモリアドレスとの間のマッピングは当たり前なものである: メモリ内の値の構造(すなわち, 次元サイズとその順番)は配列部分のそれに一致している. しかし, マップされた配列部分では, インデックスマッピングベクトルが使われ, netCDF変数の要素の順番とそれらのメモリアドレスとの間のマッピングを定義する. メモリに常駐している配列の原点からある特別の点へのオフセット(バイト単位)は, インデックスマッピングベクトルとその点のインデックスベクトル[1]との内積[2]で与えられる. 正常な配列部分のインデックスマッピングベクトルは, — 最も速く変化する次元から最も遅い次元への順番で — メモリに常駐するデータのバイトサイズ(例えば, 浮動小数点数の値なら4), そして, その値に配列部分の最も速く変化する次元の縁の長さを掛けたもの, そして, その値に次に速く変化する次元の縁の長さを掛けたもの, 以下同様, といった要素を持つ. しかし, マップされた配列においては, netCDF変数のディスク位置とメモリ位置との間の関係は根本的に異なりうる. 例えば, 以下のC定義

```
struct vel {
    int flags;
    float u;
    float v;
} vel[NX][NY];
long imap[2] = {
    sizeof(struct vel),
    sizeof(struct vel)*NY};
```

(ここに`imap`はインデックスマッピングベクトルである)は, メモリ上にあるnetCDF変数(`vel(NY,NX)`)の値にアクセスするために使うことができる. たとえ次元が並べ変えられたり, かつデータが浮動小数点数の値の2次元配列ではなく構造体の2次元配列に含まれているような場合でも.

---

[1] ある点の座標オフセットベクトルは, 各次元に対して, その点を含む配列の原点からその点へのオフセットを与える. Cでは, ある点の座標オフセットベクトルはその座標ベクトルと同じである. しかし, FORTRANではある点の座標オフセットベクトルは, その点の座標ベクトルの対応する値より一つ少ない値である.

[2] 二つのベクトル[x0, x1, ..., xn]と[y0, y1, ..., yn]との内積は単にx0*y0 + x1*y1 + ... + xn*ynである.

マップされた配列アクセスのより詳細な例は, マップされた配列アクセスのためのCおよびFORTRANインターフェースの記述の中で与えられる. 7.7節 [部分的にサンプルされたまたはマップされた値の配列を書き出す: NCVPTGおよびNCVPGC]36ページを参照.

netCDF抽象化は, もし状況によって正当化されていれば, 部分的にサンプルされたまたはマップされた配列部分へのアクセスを使うことを許容しているが, それらは要求されているものではない. もし, これらの, より一般的なアクセス形態を必要としないならば, これらの機能を無視して, 代わりに単一の値へのアクセスまたは正常な配列部分へのアクセスを使ってもよい.

3.3 データ構造

netCDF抽象化によって直接サポートされている唯一の種類のデータ構造は, ベクトル属性を付加された名前つき配列の集合である. netCDFは, 結合されたリスト, ツリー, 粗な行列, 不均一な配列, またはポインタを必要とするような他種のデータ構造, を格納するのに特に適合しているわけではない. 一つの配列中のデータを他の配列へのポインタとして使うことに関する種々の慣例を採用して, 配列の集合から多種のデータ構造を構築することも可能である. netCDFライブラリはこのようなデータ構造について手助けも妨げもしないが, netCDFはこのような慣習が設計できるような機構は提供している.

以下の例では, 各列の始点のインデックスを与える関連インデックス変数を名付ける属性`row_index`使って, 不均一な配列`ragged_mat`を格納している. 第一の列は12個(12-0)の要素を含んでおり, 第二の列は7個(19-12)を含んでいる, 等々である.

```
        float   ragged_mat(max_elements);
                ragged_mat:row_index = "row_start";
        int     row_start(max_rows);
    data:
        row_start   = 0, 12, 19, ...
```

もう一つの例を挙げると, 各グループの変数の名前を空白やコンマのような伝統的な区切り文字で区切ってリスト化する属性を定義することによってnetCDF変数を一つのnetCDFファイル中にグループ化できる. このようなグループ化のために特別な種類の属性名を使うために慣習を使うことができ, それによって, 変数の名前つきグループを任意の個数サポートすることができる. もし必要ならば, 各変数に対する特別な慣習的属性により, その変数がメンバーになっているグループの名前をリスト化することができる. 属性, または他の属性か変数を参照する変数を使うことにより, netCDFファイルにおけるいくつかの種類の複雑な構造を表現するための柔軟な機構が得られる.

Chapter 4　netCDFライブラリの使用

別にnetCDFライブラリすべてについて知っていなくとも, netCDFライブラリを使うことができる. netCDFファイルを作るだけなら, 必要なのは, 必要な次元, 変数および属性を定義し, netCDFファイルにデータを書き出すのに必要な一握りのルーチンだけである. (ncgenユーティリティを使ってファイルを作っておいてから, プログラムを走らせ, netCDFライブラリを使ってデータを書けば, 必要なライブラリはずっと少なくて済む.) 同様に, 特定のnetCDFオブジェクトに格納されたデータにアクセスするためのソフトウェアを書くには, netCDFファイルをオープンし, データにアクセスするためのほんの一部のnetCDFライブラリが必要なだけである. 任意のnetCDFファイルにアクセスする包括的なアプリケーションを書くには, より多くのnetCDFライブラリに精通している必要がある.

本章では, 通常の使用に必要な一般的netCDFサブルーチン呼び出しの一覧表を与える. 明解にするために, ルーチンの名前だけを提示し; 宣言およびエラーチェックを省略し; 一般に複数回呼び出される文を字下げし; さらに記号... を使って他の文の任意の列を表している. 手続きおよびサブルーチンの引数の完全なリストは後の章で説明する.

4.1　netCDFファイルの生成

新規にnetCDFファイルを生成するのに使われる一般的な一連のnetCDF呼び出しは以下のようである.

```
NCCRE                ! netCDFファイルを生成: 定義モードに入る
      ...
   NCDDEF            ! 次元の定義: 名前とサイズから
      ...
   NCVDEF            ! 変数の定義: 名前, 型, および次元から
      ...
   NCAPT or NCAPTC   ! 属性を置く: 属性値を割り当てる
      ...
NCENDF               ! 定義の終了: 定義モードから抜ける
      ...
   NCVPT or NCVPTC   ! 変数を置く: 変数に値を与える
      ...
NCCLOS               ! 閉じる: 新しいnetCDFファイルを保存する
```

netCDFファイルを生成するためにはたった一つの呼び出ししか必要なく, その際に2つのnetCDFモードのうちの最初のモードに入る. オープンされたnetCDFファイルにアクセスする際には, そのアクセスは定義モードまたはデータモードのいずれかで行われる. 定義モードでは, 次元, 変数および新規の属性を生成できるが, 変数データの読み書きはできない. データモードでは, データへのアクセスおよび既存の属性を変えるとができるが, 新規の次元, 変数または属性を生成することはできない.

それぞれの次元を生成するためには, `NCDDEF`を1回ずつ呼び出す必要がある. 同様に, それぞれの変数を生成するためには, `NCVDEF`を1回ずつ呼び出す必要があり, それぞれの属性を定義しかつ値を割り当てるためには`NCAPT`または`NCAPTC`を1回ずつ呼び出す必要がある. 定義モードを抜けてデータモードに入るには, `NCENDF`を呼び出せばよい.

いったんデータモードに入れば, 変数に新しいデータを追加したり, 古い値を変えたり, 既存の属性の値を変えたり (その属性の変更がより多くの格納スペースを必要としない限り) することができる. FORTRANインターフェースは, 属性を定義し, 変数に値を与えるサブルーチンを2つ用意している. これは, 数か文字列かのどちらの値が使われるかということに依存している. 単一の値を変数に書き出すには`NCVPT1`または`NCVPT1C`を使う; 一方, 任意のデータ配列を書くには`NCVPT`, `NCVPTC`, `NCVPTG`, または`NCVPTGC`を代りに使う. 複数変数のデータ記録を書くには, `NCVPT`を複数回呼び出せばよい.

最後に, 書き込みのためにオープンされているnetCDFファイルすべてを`NCCLOS`を呼び出して陽にクローズしなければならない. 書き込みのためにnetCDFファイルをオープンしたままプログラムが異常終了すると, `NCSNC`を最後に呼び出してから最も新しく書かれた記録変数データおよび任意の属性の変更の1つまたはそれ以上のデータが失われてしまう可能性がある. 異常終了によるデータ損失の可能性を減らすには, netCDF変数に書き出したり, 属性値を変更したりするたびごとに`NCSNC`を陽に呼び出せば可能である. このようなことはコンピューター資源において高くつくので, このような呼び出しは通常, 必要なとき以外には省略するべきである.

4.2　名前を知って, netCDFファイルを読む

ここでは, netCDFファイルの名前だけではなく, それらの次元, 変数および属性の名前を知っている場合を考える. (そうでない場合は, "問い合わせ" 呼び出しをしなければならない.) netCDFファイル中のそれらの変数からデータを読む

ための一般的な FORTRAN 呼び出しの順番は以下のようである:

```
NCOPN              ! 既存の netCDF ファイルをオープンする
     ...
   NCDID           ! 次元の ID を取得する
     ...
   NCVID           ! 変数の ID を取得する
     ...
   NCAGT or NCAGTC ! 属性の値を取得する
     ...
   NCVGT or NCVGTC ! 変数の値を取得する
     ...
NCCLOS             ! netCDF ファイルをクローズする
```

最初に, 単一の呼び出しにより, 与えられたファイル名で netCDF ファイルがオープンされ, オープンされた netCDF ファイルを参照するために以降のすべての呼び出しで使われる netCDF ID が返される.

次に, 興味のある各次元について `NCDID` を呼び出して次元の名前から次元の ID を取得する. 同様にして, 必要な変数 ID はそれぞれ, `NCVID` を呼び出すことによって名前から決定される. 一旦, 変数 ID が分かれば, 変数の属性値は, 要求される属性それぞれについて, netCDF ID, 変数 ID, および要求される属性名を `NCAGT` または `NCAGT` の入力として与えれば引き出せる. 変数データ値は netCDF ファイルから直接アクセスできる; 単一の値の場合には `NCVGT1` または `NCVG1C` を使って, 値の断面については `NCVGT`, `NCVGTC`, `NCVGTG`, または `NCVGGC` を使って.

最後に, netCDF ファイルは `NCCLOS` を使ってクローズされる. 単に読むだけなら, ファイルをクローズする必要はない.

4.3 名前を知らずに, netCDF ファイルを読む

多くのプログラム (例えば, 総括的なソフトウェア) は (訳注: この部分が原文で落丁している) を取得する必要がある. 各変数を (多分, 座標変数は除いて), それらの変数の名前を前もって知ることなしに, 何らかの方法で加工するようなプログラム (例えば, 総括的なソフトウェア) を書くことは可能である. たとえ, 次元および属性の名前が分かっていなくとも.

名前および他の情報は問合せ関数を呼び出すことによって netCDF ファイルから得ることができる. これらの呼び出しは, netCDF ファイル全体, 次元, 変数, または属性についての情報を返す. 以下の表は, それらの呼び出しの使われ方を説明するものである:

```
NCOPN                ! 既存の netCDF ファイルをオープンする
   ...
NCINQ                ! 中身を調べる
     ...
   NCDINQ            ! 次元の名前, サイズを取得する
     ...
   NCVINQ            ! 変数の名前, 型, 形を取得する
       ...
      NCANAM         ! 属性の名前を取得する
       ...
      NCAINQ         ! 属性の型と長さを取得する
       ...
      NCAGT or NCAGTC ! 属性の値を取得する
       ...
   NCVGT or NCVGTC   ! 変数の値を取得する
     ...
NCCLOS               ! netCDF ファイルをクローズする
```

前の例と同様に, 単一の呼び出しによって既存の netCDF ファイルがオープンされ, netCDF ID が返される. この netCDF は `NCINQ` ルーチンに与えられ, 次元の数, 変数の数, グローバル属性の数, および, もしあれば無制限次元の ID, を返す.

問合せ関数のすべては, 使用にあたってのコストは非常に少なくかつ I/O を必要としない, なぜなら, それらが与える情報はメモリ上に格納されているからである. C インターフェースにおいては, 必要ない情報については有効なアドレスの代わりにヌルポインタを与えるようにして, 情報の一部を得ることもサポートしている.

次元の ID は連続した整数 (C では 0 から, FORTRAN では 1 から始まる) を使って割り当てられる. 次元もまた, 一旦生成されてしまうと, 削除できない. 従って, ある netCDF ファイル中の次元 ID の数を知るということは, すべての次元 ID を知ることを意味している: それらは, 0, 1, 2, ...(FORTRAN では 1, 2, 3, ...) という整数である. 各次元 ID について, 呼び出し `NCDINQ` は次元の名前とサイズを返す.

変数 ID もまた連続した整数 0, 1, 2, ...(FORTRAN では 1, 2, 3, ...) が割り当てられる. これらの ID は, 各変数に割り当てられた名前, 型, 形, および属性の数を調べるために, `NCVINQ` 呼び出しで使われる.

変数の属性の数が一旦分かれば, 以下の `NCANAM` の呼び出しにより, 与えられた netCDF ID, 変数 ID, および属性番号から, 各属性の名前が返される. 属性名が分かれば, `NCAINQ` の呼び出しにより, その型および長さが返される. 型と長さが与えられれば, 属性の値を保持するのに十分なスペースを割り付けることができる. それから, `NCAGT` または `NCAG1TC` を呼び出すと, 属性の値が返される.

すべての netCDF 構成要素の名前, 型, 形, および長さが一旦分かれば, データ値はアクセス可能である; 単一の値については `NCVGT1` または `NCVGT1C` を, 配列アクセスを使う値の集合体については `NCVGT`, `NCVGTC`, `NCVGTG`, または `NCVGGC` を, 呼び出すことによって.

4.4 新しい次元, 変数, 属性を追加する

既存の netCDF ファイルは幅広く変更可能である. 新しい次元, 変数, 属性を追加したり, 既存のものの名前を変更したり, 既存の属性を削除したりできる. 既存の次元, 変数, および属性は名前を変更できる. 以下のコード表は, 既存のファイルに新しい netCDF 構成要素を加えるための一般的な一連の呼び出しをリスト化したものである:

```
NCOPN           ! 既存の netCDF ファイルをオープンする
  ...
NCREDF          ! 定義モードに入る
    ...
  NCDDEF        ! 追加する次元を定義する（もしあれば）
    ...
  NCVDEF        ! 追加する変数を定義する（もしあれば）
    ...
  NCAPT or NCAPTC ! 追加する属性を定義する（もしあれば）
    ...
NCENDF          ! 定義をチェックし，定義モードから抜ける
    ...
  NCVPT or NCVPTC ! 新しい変数に値を与える
    ...
NCCLOS          ! netCDF をクローズする
```

netCDF ファイルはまず `NCOPN` 呼び出しによってオープンされる. この呼び出しにより, データモードに入り, それによって既存のデータ値へのアクセスおよび変更が可能であり, 既存の属性が変更できる (それらが大きくならない限り) が, 何も追加することができない. 新しい netCDF 次元, 変数, または属性を追加するためには, データモードを抜けて定義モードに入らねばならず, このために `NCREDF` を呼び出せばよい. 定義モードにおいては, `NCDDEF` を呼び出して新しい次元を定義し, `NCVDEF`) により新しい変数を定義し, `NCAPT` または `NCAPTC` により変数に新しい属性を割り当てたり古い属性を大きくしたりする.

`NCENDF` を呼び出すことによって, 定義モードを抜けて再びデータモードに入る; この際, 新しい定義すべてに矛盾がないかチェックされ, 変更がディスクへ書き出される. データモードに再び入る必要がないなら, 単に `NCCLOS` を呼び出せばよい; これは先に `NCENDF` を呼び出したのと同じ効果がある.

`NCENDF` を呼び出すまでは, `NCABOR` を呼び出すことによって, 定義モードでなされた再定義すべてを外して netCDF データセットの前の状態を復旧できる. また, `NCENDF` 呼び出しが失敗した場合に, `NCABOR` を呼び出すことによって netCDF データセットを矛盾のない状態に復旧できる. 定義モードから `NCCLOS` を呼び出した際に暗に呼び出される `NCENDF` が失敗すると, `NCABOR` が自動的に呼び出され, netCDF ファイルがクローズされ, ファイルは前の矛盾のない状態 (定義モードに入る前の状態) のままになる.

4.5 エラー処理

netCDF ライブラリは柔軟にエラーを処理するために必要な機能を提供している. しかし, エラー処理について何か指定する必要はたいていない, なぜなら, デフォルトではすべての netCDF ライブラリルーチンはエラーが生じると単にエラーメッセージを出力して終了してしまうだけだからである. もし, このような (確かに徹底的な) エラー時の挙動が容認できるものであれば, 返り値をチェックする必要はない; なぜなら, エラーに至るようなどのような状態でも説明メッセージを出力して終了してしまうからである. 簡単のために, このガイドにおける例では, このデフォルトのエラー処理の挙動を仮定するので, 返り値のチェックはしない.

FORTRAN インターフェースにおいては, エラーオプションには `NCPOPT` および `NCGOPT` ルーチンを使ってアクセスできる. エラー処理の 2 つの面: エラーメッセージの抑制およびエラーの致死性が独立に修正できる. エラーが生じるとメッセージを出力して終了するというデフォルトの挙動は, 以下の文に等しい:

```
CALL NCPOPT(NCVERBOS+NCFATAL)
```

ここに, `NCVERBOS` および `NCFATAL` の値は FORTRAN インクルードファイル '`netcdf.inc`' で事前に定義されている定数である. もし, エラーメッセージは残して, エラー時に終了したくない場合には, 致死的エラーフラグを以下のように切ればよい:

```
CALL NCPOPT(NCVERBOS)
```

もし, エラーメッセージもエラー時の終了もさせたくないならば, 両方のフラグを以下のように切ればよい:

```
CALL NCPOPT(0)
```

エラーオプションの現在の値を取得するには, 以下のように使う:

```
CALL NCGOPT(NCOPTS)
```

どちらの場合においても, 整数の返り値 (すべての FORTRAN サブルーチンおよび関数の最後のパラメター) は, エラーの性質を決定するために使うことができる 0 以外の netCDF 特定のエラーナンバーを含んでいる. netCDF エラーコードの名前および記述は '`netcdf.inc`' ファイルに含まれている.

時折, netCDF ライブラリの下の XDR ライブラリ層で下位レベルの書き込みエラーが起きる可能性がある. 例えば, 書き込みのオペレーションにより, ディスクの割り当てを超えたり, もはや使えないデバイスに書き込もうとしたりすると, netCDF ライブラリからではなく, XDR 関数の一つからエラーメッセージを受け取る可能性がある.

4.6 netCDF ライブラリのコンパイルとリンク

netCDF の C または FORTRAN インターフェースを使うプログラムのコンパイルおよびリンクの仕方の詳細は, オペレーティングシステム, 使えるコンパイラ, および netCDF ライブラリおよびインクルードファイルがどこにインストールされているか, に依存して異なる. ではあるが, ここでは Unix プラットフォーム上で netCDF ライブラリを使うプログラムのコンパイルおよびリンク方法の例を与えるので, これらの例をインストールの状況に応じて直せばよい.

netCDF 関数または定数を参照する FORTRAN ファイルはそれぞれ, そのような最初の参照の前に適切な `INCLUDE` 文を含んでいなければならない:

```
INCLUDE 'netcdf.inc'
```

'`netcdf.inc`' ファイルが FORTRAN コンパイラがいつも参照する標準的ディレクトリにインストールされていない場合には, コンパイラを呼び出す際に`-I` オプションを使って '`netcdf.inc`' がインストールされているディレクトリを指定しなければならない, 例えば:

```
f77 -c -I/usr/local/netcdf/include myprogram.f
```

代わりに, `INCLUDE` 文に絶対パスを指定することもできるが, そうすると netCDF が異なった位置にインストールされている他のプラットフォームでそのプログラムがコンパイルできなくなってしまう.

netCDF ライブラリがリンカがいつも参照する標準的ディレクトリにインストールされていない場合には, ライブラリを使うオブジェクトファイルをリンクするために`-L` および`-l` オプションを使わねばならない, 例えば:

```
f77 -o myprogram myprogram.o -L/usr/local/netcdf/lib -lnetcdf
```

代わりに, ライブラリの絶対パスを指定することもできる:

```
f77 -o myprogram myprogram.o -l/usr/local/netcdf/lib/libnetcdf.a
```

netCDF ライブラリに加えて, システムの XDR ライブラリのあるライブラリを追加して指定しなければならないシステムもある.

Chapter 5 ファイル

この章では, netCDF ファイル全体を扱う netCDF ルーチンのインターフェースについて述べる.

まだオープンされていない netCDF ファイルはそのファイル名によってのみ参照できる. いったん netCDF ファイルがオープンされた後は, そのファイルは **netCDF ID** によって参照することができ, その ID はファイルを生成またはオープンした際に返される小さな非負の整数である. netCDF ID は C におけるファイル記述子または FORTRAN における論理装置番号によく似たものである. 単一のプログラムにおいてはいつも, オープンされた異なる netCDF ファイルの netCDF ID は別々である. 一つの etCDF ファイルは複数回オープンすることができ, 従って複数の異なる netCDF ID を持つことになる; しかし, 一つの netCDF ファイルに対してオープンされている装置のうち一つまでしか書き込みは許されない. オープンされていた netCDF ファイルがクローズされると, その ID はもはや netCDF ファイルとは関連しなくなる.

netCDF ファイル上で一つのオブジェクトとしてサポートされている操作は:

- ファイル名および上書きするかどうかを決めて, ファイルを生成する.
- ファイル名および読み/書きどちらを行うかを決めて, アクセスのためにファイルをオープンする.
- 次元, 変数, 属性を追加するために定義モードに入る.
- 追加した内容の一貫性をチェックして定義モードから抜ける.
- 必要ならばディスクに書き込んでからファイルをクローズする.
- 次元の数, 変数の数, グローバル属性の数, およびもしあれば無制限次元の ID, を取得する.
- 現在の状態を確保するためにディスクとの同期をとる.
- 最適な連続的書き込みのための **nofill** モードをセットまたは解除する.

この章では, netCDF の C および FORTRAN インターフェースを記述するのに使われる慣習をまとめたあと, 残りの部分でこれらの操作のためのインターフェースについて述べる.

5.1 netCDF ライブラリインターフェースの記述

本章および後の章における, ある特定の netCDF 関数に対する各インターフェースの記述は以下の事項を含む:

- 関数の目的の記述
- 起りうるエラー状態のリスト
- FOTRAN 関数のプロトタイプ: その FORTRAN 関数への正式なパラメターの型および順序を示す.
- その FORTRAN インターフェースにおける正式パラメターそれぞれについての説明
- その netCDF 関数 (および他の netCDF 関数) を呼び出す FORTRAN プログラム断片の例

FORTRAN 関数のプロトタイプは正式なパラメターの型および順序を簡潔に示す.

5.2 netCDF ファイルの生成: NCCRE

関数 `NCCRE` は新たな netCDF ファイルを生成し, 以降その netCDF を参照するために使われる netCDF ID を返す. その netCDF ファイルは定義モードになっている.

エラーが起きた場合には, `NCCRE` は `RCODE` に零以外の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 存在しないディレクトリを含んだファイル名を渡している場合.
- すでに存在しているファイルを指定し, かつ `NCNOCLOB` を指定した場合.
- ファイルを作成するためのパーミッションが与えられていないディレクトリにファイルを生成しようとしている場合.

```
INTEGER FUNCTION NCCRE (CHARACTER*(*) FILENAME, INTEGER CMODE, INTEGER RCODE)
```

`FILENAME` 新規の netCDF ファイルのファイル名.

`CMODE` `NCCLOB` または `NCNOCLOB` のどちらかを指定する. これらの定数は, インクルードファイル '`netcdf.inc`' で定義される. `NCCLOB` は, たとえファイルがすでに存在しても, 古いファイルの内容を消去して同じ名前の新しいファイルを作成したい場合に指定する. `NCCLOB` は, 与えられたファイル名が既存のファイルを指していない場合のみに新規の netCDF ファイルを作成したい場合に指定する.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, ‘`foo.nc`’ という名前の netCDF ファイルを作成している. ここでは, その名前のファイルがまだ存在していない場合のみ, カレントディレクトリにそのファイルを作成したい場合であるとしている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
```

5.3 アクセスのために netCDF ファイルをオープンする: NCOPN

関数 `NCOPN` はアクセスのために既存の netCDF ファイルをオープンする.

エラーが生じた場合には, `NCOPN` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された netCDF ファイルが存在していない.
- 指定されたモードが `NCWRITE` または `NCNOWRITE` 以外である.

```
INTEGER FUNCTION NCOPN (CHARACTER*(*) FILENAME, INTEGER RWMODE, INTEGER RCODE)
```

`FILENAME` オープンする netCDF ファイルの名前.

`RWMODE` ファイルを書き込みのためにオープンする `NCWRITE` または, ファイルを読み込みのみでオープンする `NCNOWRIT` のどちらかを指定する. “書き込み” は, データの変更, 次元, 変数および属性の追加または名前のつけかえ, または属性の削除, を含んだあらゆる種類のファイルの変更を意味している.

`RCODE` 返されたエラーコードを. エラーが生じていなければ, 0 が返される.

以下の例では, `NCOPN` を使って ‘`foo.nc`’ という名前の既存の netCDF ファイルを読み込みのためにオープンしている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
```

5.4 netCDF ファイルを定義モードにする: NCREDF

関数 `NCREDF` はオープンされた netCDF ファイルを定義モードにし, 次元, 変数および属性を追加またはその名前を変更できるように, また属性を削除できるようにする.

エラーが生じた場合には, `NCREDF` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された netCDF ファイルがすでに定義モードになっている.
- 指定された netCDF ファイルが読み込みのみでオープンされている.
- 指定された netCDF ID が netCDF ファイルを参照していない.

```
SUBROUTINE NCREDF (INTEGER NCID, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCREDF` を使って ‘`foo.nc`’ という名前の既存の netCDF ファイルをオープンし, それを定義モードにしている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCOPN('foo.nc', NCWRITE, RCODE)
   ...
CALL  NCREDF(NCID, RCODE)
```

5.5 定義モードから抜ける: NCENDF

関数 `NCENDF` はオープンされた netCDF ファイルを定義モードから出す. 定義モードになっているうちに加えられた netCDF ファイルへの変更はチェックされ, 問題が生じていなければディスクに書き込まれる. このとき, netCDF ファイルはデータモードになっており, 変数データが読み書きできる.

この呼び出しは, ある環境においては記録変数ではない変数の初期化およびデータのコピーを含むので, コストがかかる. より広範な議論については, 9 章 [netCDF ファイル構造および性能] 53ページを参照.

エラーが生じた場合には, `NCENDF` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された netCDF ファイルが定義モードになっていない.
- 指定された netCDF ID が netCDF ファイルを参照していない.

```
SUBROUTINE NCENDF (INTEGER NCID, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCENDF` を使って ‘`foo.nc`’ という名前の新しい netCDF ファイルの定義モードを終了し, それをデータモードにしている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
   ... ! 次元, 変数, 属性を作成する.
CALL  NCENDF(NCID, RCODE)
```

5.6 netCDF ファイルをクローズする: NCCLOS

関数 `NCCLOS` はオープンされた netCDF ファイルをクローズする. もしそのファイルが定義モードであれば, クローズする前に `NCENDF` が呼び出される. (この場合, もし `NCENDF` がエラーを返せば, `NCABOR` が自動的に呼び出され, 定義モードに最後に入る前の矛盾のない状態に復旧する.) オープンされた netCDF ファイルがクローズされた後では, その netCDF ID は次にオープンまたは作成される netCDF ファイルに割り当てられる.

エラーが生じた場合には, `NCCLOS` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 定義モードになっていて, `NCENDF` の自動的な呼び出しが失敗した.
- 指定された netCDF ID が netCDF ファイルを参照していない.

```
SUBROUTINE NCENDF (INTEGER NCID, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCCLOS` を使って ‘`foo.nc`’ という名前の新しい netCDF ファイルの定義モードを終了し, その netCDF ID を解放している:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
   ... ! 次元, 変数, 属性を作成する.
CALL  NCCLOS(NCID, RCODE)
```

5.7 オープンされた netCDF ファイルについて問合せをする: NCINQ

関数 `NCINQ` は, netCDF ID を与えて, オープンされた netCDF ファイルについての情報を返す. それは定義モードまたはデータモードから呼び出すことができる. それは, 次元の数, 変数の数, グローバル属性の数, および, 無制限サイズを持つ次元の次元 ID(もしあれば) を返す. これまたは netCDF インターフェース中の他のあらゆる '問合せ' 関数が呼び出されても I/O は要求されない. なぜなら, それらの関数はオープンされた各 netCDF ファイルについての表に格納された情報を返すだけだからである.

エラーが生じた場合には, `NCINQ` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された netCDF ID が netCDF ファイルを参照していない.

```
      SUBROUTINE NCINQ (INTEGER NCID, INTEGER NDIMS, INTEGER NVARS,
     *                  INTEGER NGATTS, INTEGER RECDIM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`NCDIMS` この netCDF ファイルに定義されている次元の数を返す.

`NCVARS` この netCDF ファイルに定義されている変数の数を返す.

`NCGATTS` この netCDF ファイルに定義されているグローバル属性の数を返す.

`RECDIM` この netCDF ファイルに定義されていれば, 無制限次元の ID を返す. 無制限サイズの次元が定義されていなければ, `RECDIM` の値として -1 が返される.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCINQ` を使って '`foo.nc`' という名前の netCDF ファイルについて調べている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE
   ...
NCID = NCCRE('foo.nc', NCNOWRIT, RCODE)
   ...
CALL  NCINQ(NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE)
```

5.8 オープンされた netCDF ファイルをディスクに同期する: NCSNC

関数 `NCSNC` は, 書き込みのためにオープンされている netCDF ファイルのディスク上のコピーが現在の状態であるようにする. 定義モードにある netCDF ファイルは `NCENDF` が呼び出されたときにのみディスクに同期される. 他のプロセスが書き込んでいる netCDF ファイルを読み込んでいるプロセスが, そのファイルをクローズして再オープンしたりせずに, `NCSNC` を呼び出して書き込みプロセスによってなされた変更 (例えば, 書かれた記録の数など) に対応した更新をすることもできる.

変数データの書き込みまたは属性値の変更のたびにいつもディスクと同期をとると, コンピューター資源におけるコストが大きくなってしまいうる. 書き込みの後で同期をとりたい理由として, 以下の 2 つが挙げられる:

- 異常終了の場合のデータ損失を最小にする.
- データが書かれた直後にそれを読む他のプロセスがそのデータにアクセスできるようにする. しかし, 読み込みのためにそのファイルをすでにオープンしているプロセスは, 書き込みプロセスが `NCSNC` を呼び出した際の記録数の増大を感知できないということに注意が必要である; これを感知するためには, 読み込みプロセスが `NCSNC` を呼び出す必要がある.

netCDF ファイルがクローズされるか, または定義モードを抜ける際いつでもデータは自動的にディスクに同期される.

エラーが生じた場合には, `NCSNC` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- その netCDF ファイルが定義モードにある.
- 指定された netCDF ID が netCDF ファイルを参照していない.

```
      SUBROUTINE NCSNC (INTEGER NCID, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCSNC` を使って ‘`foo.nc`’ という名前の netCDF ファイルのディスク書き込みの同期をとっている:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
*     データを書く, または属性を変更する.
         ...
      CALL  NCSNC(NCID, RCODE)
```

5.9 最新の定義をやめる: NCABOR

関数 `NCABOR` は, もし定義モードになっていなければ, netCDF ファイルをクローズする. もしそのファイルが作成されたばかりでまだ定義モードにあれば, そのファイルは削除される. `NCREDF` の呼び出しによって定義モードに入っているなら, netCDF ファイルはその定義モードに入る前の状態に復旧され, そのファイルはクローズされる. `NCABOR` を呼び出す主な理由は, 新しい次元, 変数, または属性を定義している間に何か障害が起った場合に, 既知の矛盾のない状態にファイルを復旧するということである.

この関数は, 定義モードから `NCCLOS` が呼び出されたり, クローズの前に定義モードから抜けるための呼び出しが失敗したりした場合に自動的に呼び出される.

エラーが生じた場合には, `NCABOR` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- netCDF ファイルを生成している際の定義モードから呼び出された場合に, ファイルの削除が失敗した.
- 指定された netCDF ID が netCDF ファイルを参照していない.

```
SUBROUTINE NCABOR (INTEGER NCID, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCABOR` を使って ‘`foo.nc`’ という名前のファイルの再定義をやめている:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
      CALL  NCREDF(NCID, RCODE)
         ...
      LATID = NCDDEF(NCID, 'LAT', 18, RCODE)
      IF (RCODE .EQ. -1) THEN   ! 次元の定義が失敗
         CALL NCABOR(NCID, RCODE)   ! 再定義をやめる
      END IF
         ...
```

5.10 書き込みのフィルモードの設定: NCSFIL

この呼び出しは, 以下に述べるようないくつかの環境における書き込みを最適化するための, 高度な使用のために意図されたものである. 関数 `NCSFIL` は, 書き込みのためにオープンされている netCDF ファイルにフィルモードを設定し, 現在のモードを返す. フィルモードは `NCFILL` または `NCNOFILL` のどちらかが指定できる. `NCFILL` に対応するデフォルトの動作では, データはフィル値で前もって満される. すなわち, 記録のない変数を生成したり, まだ書かれていないところのデータを越えて値を書いたりする際にフィル値が書かれる. これによって, データを, それが書かれる前に読もうとしているような試みを検出することができる. フィル値の使用についてのより多くの情報については 7.12節 [フィル値]43 ページを参照. 独自のフィル値の定義法についての情報は 8.1節 [属性の慣習] 45ページを参照.

`NCNOFILL` に対応した動作では, データを前もってフィル値で満すというデフォルトの動作を無効にする. これは効率を上げるために使うことができる. なぜなら, netCDF ライブラリが直後にデータで上書きされてしまうようなフィル値を書き込むときに生じる重複書き込みを回避できるからである.

返り値によって, netCDF ファイルがどのモードにあったかということが分かる. この値を使って, オープンされた netCDF ファイルのフィルモードを一時的に変えて, それから元のモードに復旧することができる.

オープンされた netCDF ファイルを `NCNOFILL` モードにした後は, 後で読まれるすべての位置に必ず妥当なデータを書き込まなければならない. `NCNOFILL` モードは書き込みのためにオープンされた netCDF ファイルの一時的な性質でしかないことに注意が必要である: もしそのファイルをクローズして再オープンすると, デフォルトの動作に戻る. `NCFILL` に陽に設定するために, `NCSFIL` を再び呼び出して, デフォルトの動作に戻ることもできる.

ノーフィルモードに設定するのが有効な状況としては, 以下の 3 つが考えられる:

1. netCDF ファイルを生成または初期化しているとき. この場合, `NCENDF` を呼び出す前にノーフィルモードを設定し, それからすべての記録なし変数および初期化したいすべての記録変数の初期記録を完璧に書くべきである.
2. 記録指向の既存の netCDF ファイルを拡張しているとき. 書き込みのためにファイルをオープンした後でノーフィルモードに設定し, その後, 書かれていない記録が入り込まないように, 追加の記録をファイルに完璧に追加すること.
3. 初期化する予定の新しい変数を既存の netCDF ファイルに追加するとき. `NCENDF` を呼び出す前にノーフィルモードを設定し, それから新しい変数を完璧に書くこと.

netCDF ファイルが無制限次元を持ち, その最後の記録が `NCNOFILL` モードで書かれた場合には, ファイルは `NCNOFILL` モードが設定されていない場合よりも 4 バイト長くなるが, netCDF インターフェースだけを使ってデータにアクセスしていればこの変化は全く透過性を損わない.

エラーが生じた場合には, `NCSFIL` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された netCDF ID が netCDF ファイルを参照していない.
- 指定された netCDF ID が読み込みのみでオープンされているファイルを参照している.
- フィルモードの引数が `NCNOFILL` および `NCFILL` のどちらでもない.

```
INTEGER FUNCTION NCSFIL (INTEGER NCID, INTEGER FILLMODE, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`FILLMODE` そのファイルに対して希望するフィルモード. `NCNOFILL` または `NCFILL` のどちらか.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

`NCSFIL` この呼び出しの前のファイルのフィルモードの現状を返す. `NCNOFILL` または `NCFILL` のどちらか.

以下の例では, `NCSFIL` を使って ‘`foo.nc`’ という名前の netCDF ファイルに対してノーフィルモードを設定している:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, OMODE
         ...
* 前もってフィル値を満すデフォルトの動作でデータを書く
         ...
      OMODE = NCSFIL(NCID, NCNOFILL, RCODE)
         ...
* 前もってフィル値を満さずにデータを書く
         ...
```

Chapter 6 次元

netCDF ファイルにおける次元は, 生成される際に定義される. これは netCDF ファイルが定義モードにある間に行われる. 再び定義モードに入ることによって, 後で追加の次元を追加することもできる. netCDF の次元は名前とサイズを持っている. 一つの netCDF ファイルにおいては, 一つの次元までが `NC_UNLIMITED` サイズを持つことができる. このサイズは, この次元を使う変数がこの次元に沿って成長できるということを意味している.

一つの netCDF ファイルで定義できる次元の数には上限 (100) が提唱されている. この上限は前もって定義されているマクロ `MAXNCDIM` の値である. この上限の目的は, 一般的なアプリケーションをより簡明に書くことである. それによって, これらのアプリケーションは, `MAXNCDIM` 次元の配列さえ用意しておけば, 任意の netCDF ファイルを扱うことができる. netCDF ライブラリの実装においてはこの勧告された上限を強要してはいないので, もし必要ならばより多くの次元を使うことができる. しかし, できた netCDF ファイルを一般的なアプリケーションや netCDF ユーティリティが扱えると思ってはならない.

通常, 次元の名前とサイズは, その次元が最初に定義されたときに固定される. その名前は後で変えることができるが, 次元のサイズを変えるためには, 次元サイズを再定義した新しい netCDF ファイルにすべてのデータをコピーしなければならない.

C インターフェースにおける次元サイズは `int` 型ではなく `long` 型である. これは MSDOS のように, 16 ビット `int` データ型しかサポートしていないプラットフォーム上でも netCDF ファイル内のすべてのデータにアクセスすることを可能にするためである. そうでなく, もし次元サイズが `int` だと, 16 ビット `int` が収容できるのより大きな次元サイズをもつ変数からのデータにアクセスすることができなくなってしまう.

オープンされた netCDF ファイル中の netCDF 次元は, C および FORTRAN インターフェースにおいて次元 **ID** と呼ばれる小さな整数によて参照される. C インターフェースにおいては, 次元 ID は 0, 1, 2, ..., であるが, FORTRAN インターフェースにおいては, 対応する ID は次元が定義された順番に 1, 2, 3, ..., となる.

次元に対してサポートされている操作は以下のようである:

- 名前とサイズを与えて次元を生成する.
- 名前から次元 ID を取得する.
- その ID から次元の名前とサイズを取得する.
- 次元の名前を変える.

6.1 次元を生成する: NCDDEF

関数 `NCDDEF` は定義モードにあるオープンされた netCDF ファイルに新しい次元を追加する. これは, netCDF ID, 次元名, および次元サイズを与えると次元 ID を返す. それぞれの netCDF ファイルに対して, 一つまでの無制限サイズ次元 (これは記録次元と呼ばれる) がが定義できる.

エラーが生じた場合には, `NCDDEF` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- netCDF ファイルが定義モードになっていない.
- 指定された次元名が他の既存の次元の名前である.
- 指定されたサイズが 0 以下である.
- 指定されたサイズが無制限だが, この netCDF ファイルに対してすでに無制限サイズの次元が定義されている.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      INTEGER FUNCTION NCDDEF (INTEGER NCID, CHARACTER*(*) DIMNAM,
     +                         INTEGER DIMSIZ, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`DIMNAM` 次元名. アルファベット文字で始まり, アンダースコア ('_') を含む零またはそれ以上の英数字が続かねばならない. 大文字小文字は区別される.

`DIMSIZ` 次元のサイズ; すなわち, この次元を使う変数へのインデックスとしてのこの次元に対する値の数である. これは正の整数か, 前もって定義されている定数 `NCUNLIM` のどちらかでなければならない.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCDDEF` を使って `lat` という名前でサイズ 18 の次元および `rec` という名前の記録次元を ‘`foo.nc`’ という名前の新しい netCDF ファイルに生成している:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE, LATID, RECID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
   ...
LATID = NCDDEF(NCID, 'lat', 18, RCODE)
RECID = NCDDEF(NCID, 'rec', NCUNLIM, RCODE)
```

6.2 名前から次元 ID を取得する: NCDID

関数 `NCDID` は, 名前を与えて, netCDF 次元の ID を返す. `ndims` は netCDF ファイルで定義されている次元の数で, それぞれの次元は `1` から `ndims` の間の ID を持つ.

エラーが生じた場合には, `NCDDEF` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された名前がその neCDF ファイル中で現在定義されているどの次元の名前でもない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
INTEGER FUNCTION NCDID (INTEGER NCID, CHARACTER*(*) DIMNAME, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`DIMNAME` 次元名. アルファベット文字で始まり, アンダースコア (‘_’) を含む零またはそれ以上の英数字が続かねばならない. 大文字小文字は区別される.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCDID` を使って, `lat` という名前の次元の次元 ID を決定している (既存の ‘`foo.nc`’ という名前の netCDF ファイルにおいて過去に定義されていると仮定している):

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE, LATID
   ...
NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
   ...
LATID = NCDDEF(NCID, 'lat', RCODE)
```

6.3 次元について問合せる: NCDINQ

関数 `NCDINQ` は, 次元 ID を与えて, 次元の名前とサイズを返す. 無制限次元 (もしあれば) のサイズはそれまで書かれている記録の数である.

エラーが生じた場合には, `NCDINQ` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された neCDF ファイルにおいて, その次元 ID が不適当である.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
 SUBROUTINE NCDINQ (INTEGER NCID, INTEGER DIMID, CHARACTER*(*) DIMNAM,
+                   INTEGER DIMSIZ, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`DIMID` 以前の `NC1DID` または `NCDDEF` 呼び出しで返された次元 ID.

`DIMNAM` 返された次元名. 返される名前のためのスペースを割り当てておかねばならない. 次元名の可能な最大文字長は, 前もって定義されている定数 `MAXNCNAM` で与えられている.

`DIMSIZ` 返された次元サイズ. 無制限次元については, これは, この次元をもつ任意の変数に書き込むのに使われた現在の最大値である. すなわち, 記録数の最大値である.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCDINQ` を使って, 既存の ‘`foo.nc`’ という名前の netCDF ファイルについて, `lat` という名前の次元のサイズ, および無制限 (記録) 次元の名前と現在の最大サイズを決めている:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID, LATSIZ
      INTEGER NDIMS, NVARS, NGATTS, RECID, NRECS
* 以下の宣言における 31 はパラメター MAXNCNAM である.
      CHARACTER*31 LATNAM, RECNAM
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      LATID = NCDDEF(NCID, 'lat', RCODE)
* lat の名前とサイズを取得する (すでに名前は知っているが)
      CALL NCDINQ(NCID, LATID, LATNAM, LATSIZ, RCODE)
* 記録次元の ID を取得する (まず第 1 に)
      CALL NCINQ(NCID, NDIMS, NVARS, NGATTS, RECID, RCODE)
* 記録次元の名前と現在のサイズを取得する
      CALL NCDINQ(NCID, RECID, RECNAME, NRECS, RCODE)
```

6.4 次元の名前を変える: NCDREN

関数 `NCDREN` は, 書き込みのためにオープンされている netCDF ファイルにおける既存の次元の名前を変える. もし新しい名前が古い名前より長ければ, その netCDF データセットは定義モードでなければならない. ある次元の名前を他の次元と同じ名前にるすることはできない.

エラーが生じた場合には, `NCDREN` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 新しい名前が他の次元の名前である.
- 指定された neCDF ファイルにおいて, その次元 ID が不適当である.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.
- 新しい名前が古い名前より長く, かつ netCDF ファイルが定義モードにない.

```
SUBROUTINE NCDREN (INTEGER NCID, INTEGER DIMID, CHARACTER*(*) DIMNAME, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`DIMID` 以前の `NC1DID` または `NCDDEF` 呼び出しで返された次元 ID.

`DIMNAM` 次元の新しい名前.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCDREN` を使って, 既存の ‘`foo.nc`’ という名前の netCDF ファイルについて, 次元の名前を ‘‘`lat`’’ から ‘‘`latitude`’’ に変えている:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
* 次元名を変えるために定義モードに入る
      CALL NCREDF(NCID, RCODE)
      LATID = NCDID(NCID, 'lat', RCODE)
      CALL NCDREN(NCID, LATID, 'latitude', RCODE)
* 定義モードを抜ける
      CALL NCENDF(NCID, RCODE)
```

Chapter 7　変数

netCDF ファイルにおける変数はファイルが生成される際に定義される．これは netCDF ファイルが定義モードにある間に行われる．再び定義モードに入ることによって，後で他の変数を追加することもできる．netCDF の変数は名前，型，および形を持ち，これらは変数が定義される際に定義される．変数は値も持つことができ，これは後にデータモードで確定される．

通常，名前，型，および形は変数が最初に定義される際に固定される．名前は変えることができるが，変数の型および形は変えることができない．しかし，無制限次元を使って定義されている変数はその次元における上限なしで成長できる．

オープンされた netCDF ファイル中の netCDF 変数は，C および FORTRAN インターフェースにおいて変数 **ID** と呼ばれる小さな整数で参照される．変数 ID は netCDF ファイル内で変数が定義された順番を反映している．FORTRAN インターフェースにおいては，変数 ID は変数が定義された順番に 1, 2, 3, ..., である．どちらのインターフェースにおいても，変数名から変数 ID を取得およびその逆を行うための関数が用意されている．

単位のような性質を指定するために，変数に属性 (8 章 [属性] 45ページを参照) を伴っていてもよい．

変数に対してサポートされている操作は以下のようである:

- 名前，データ型，および形を与えて変数を生成する．
- 名前から変数 ID を取得する．
- その ID から変数の名前，データ型，形および属性の数を取得する．
- 変数 ID，番号，および値を与えて変数にデータ値を入れる．
- 変数 ID，隅の番号，縁の長さ，および一まとまりのデータを与えて変数に値の配列を入れる．
- 変数 ID，隅の番号，縁の長さ，ストライドベクトル，インデックスマッピングベクトルおよび一まとまりのデータを与えて部分的にサンプルされたまたはマップされた値の配列断面を変数に入れる．
- 記録番号，および値のまとまりへのポインタを与えて値を記録変数に入れる．
- 変数 ID および番号を与えて変数からデータを取得する．
- 変数 ID，隅の番号および縁の長さを与えて変数から値の配列を取得する．
- 変数 ID，隅の番号，縁の長さ，ストライドベクトルおよびインデックスマッピングベクトルを与えて部分的にサンプルされたまたはマップされた値の配列断面を変数から取得する．
- 記録番号，および各記録変数のデータがどこに格納されているかを示すポインタを与えて，記録変数から値を取得する．
- 変数の名前を変える．
- 与えられたデータ型のバイト数を取得する．
- 記録変数の数，それらの ID，およびそれらの記録サイズを取得する．

7.1　netCDF データ型に対応した言語の型

以下の表に netCDF データ型と FOTRAN データ型との対応を示す:

netCDF/CDL	FORTRAN		
Data Type	Data Type	API Mnemonic	Bits
`byte`	`BYTE, LOGICAL*1 (INTEGER)`	`NCBYTE`	8
`char`	`CHARACTER`	`NCCHAR`	8
`short`	`INTEGER*2 (INTEGER)`	`NCSHORT`	16
`long`	`INTEGER*4 (INTEGER)`	`NCLONG`	32
`float`	`REAL*4 (REAL)`	`NCFLOAT`	32
`double`	`REAL*8 (DOUBLEPRECISION)`	`NCDOUBLE`	64

最初のコラムは netCDF データを与えているが，これは CDL データ型と同じである．次の一組のコラムは，それぞれ，最初のコラムに対応する FORTRAN データ型および，netCDF ルーチンを呼び出す際に使われる対応する FORTRAN パラメターである (このパラメターは netCDF FORTRAN インクルードファイル `netcdf.inc` で定義されている)．できれば，括弧で括っていない FORTRAN 型を使った方がよい．手持ちの FORTRAN コンパイラがサポートしていない型については，対応する括弧で括られた型を使うこと．最後のコラムは対応する型の値を外部表現するのに使われるビット数である．

netCDF ライブラリの現在のバージョンでは, 64 ビット整数または複数バイトの文字に対応する netCDF 型が無いことに注意されたい.

7.2 変数を生成する: NCVDEF

関数 `NCVDEF` は, 定義モードにあるオープンされている netCDF ファイルに新しい変数を追加する. netCDF ID, 変数名, 変数型, 次元の数, および次元 ID のリストを与えると, 変数 ID が返される.

エラーが生じた場合には, `NCVDEF` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- netCDF ファイルが定義モードにない.
- 指定された変数名が他の既存の変数名である.
- 指定された型が, 適当な netCDF 型でない.
- 指定された次元の数が負であるか, または定数 `MAX_VAR_DIMS` より大きい; この定数は, netCDF 変数に対して許されている次元の最大数である.
- 次元のリストにおける一つまたはそれ以上の次元 ID がその netCDF ファイルに対して適当でない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      INTEGER FUNCTION NCVDEF (INTEGER NCID, CHARACTER*(*) VARNAM, INTEGER VARTYP,
     +                         INTEGER NVDIMS, INTEGER VDIMS(*), INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARNAM` 変数名. アルファベット文字で始まり, アンダースコア ('_') を含む零またはそれ以上の英数字が続かねばならない. 大文字小文字は区別される.

`VARTYP` 前もって定義されている netCDF データ型集合の一つ. 適当な netCDF データ型は `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE` である.

`NVDIMS` その変数に対する次元の数. 例えば, 2 は行列, 1 はベクトル, および 0 はその変数が次元を持たないスカラーであることを意味する. これは負であったり, 前もって定義されている定数 `MAXVDIMS` より大きくてはならない.

`VDIMS` 変数次元に対応する次元 ID の `NVDIMS` ベクトル. 無制限次元の ID が含まれている場合には, それば最後でなければならない. この引数は, `NVDIMS` が 0 の場合には無視される.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVDEF` を使って, 既存の 'foo.nc' という名前の新しい netCDF ファイルについて, 3 つの次元 `time`, `lat`, および `lon` を持つ名前 `rh` の `long` 型変数を生成している:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  LATDIM, LONDIM, TIMDIM  ! 次元 ID
INTEGER  RHID                    ! 変数 ID
INTEGER  RHDIMS(3)               ! 変数形
   ...
NCID = NCCRE ('foo.nc', NC_CLOBBER, RCODE)
   ...
                                 ! 次元の定義
LATDIM = NCDDEF(NCID, 'lat', 5, RCODE)
LONDIM = NCDDEF(NCID, 'lon', 10, RCODE)
TIMDIM = NCDDEF(NCID, 'time', NCUNLIM, RCODE)
   ...
                                 ! 変数の定義
RHDIMS(1) = LONDIM
RHDIMS(2) = LATDIM
RHDIMS(3) = TIMDIM
RHID = NCVDEF (NCID, 'rh', NCDOUBLE, 3, RHDIMS, RCODE)
```

7.3 名前から変数 ID を取得する: NCVID

関数 `NCVID` は, 名前を与えて, netCDF 変数の ID を返す.

エラーが生じた場合には, `NCVID` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数名が, 指定された netCDF ファイル中の変数名として適当でない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
        INTEGER FUNCTION NCVID (INTEGER NCID, CHARACTER*(*) VARNAM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARNAM` ID が欲しい変数名.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVID` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数の ID を調べている:

```
   INCLUDE 'netcdf.inc'
      ...
   INTEGER  NCID, RCODE
   INTEGER  RHID                        ! 変数 ID
      ...
   NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
      ...
   RHID = NCVID (NCID, 'rh', RCODE)
```

7.4 ID から変数についての情報を取得する: NCVINQ

関数 `NCVINQ` は, ID を与えて, netCDF 変数についての情報を返す. 返される情報は, 名前, 型, 次元の数, 変数の形を記述する次元 ID のリスト, および変数に割り当てられている変数属性の数, である.

エラーが生じた場合には, `NCVINQ` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
        SUBROUTINE NCVINQ (INTEGER NCID, INTEGER VARID, CHARACTER*(*) VARNAM,
       +                   INTEGER VARTYP, INTEGER NVDIMS, INTEGER VDIMS(*),
       +                   INTEGER NVATTS, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`VARNAM` 返された変数名. 返される名前のためのスペースを割り当てておかねばならない. 変数名の可能な最大文字長は, 前もって定義されている定数 `MAXNCNAM` で与えられている.

`VARTYP` 返されるデータ型であり, 前もって定義されている netCDF データ型集合の一つ. 適当な netCDF データ型は `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE` である.

`NVDIMS` その変数に対する次元の数が返される. 例えば, 2 は行列, `1` はベクトル, および `0` はその変数が次元を持たないスカラーであることを意味する.

`VDIMS` 変数次元に対応する次元 ID の `NVDIMS` ベクトルが返される. 少なくとも, `NVDIMS` の整数のベクトルが返されるのに十分なスペースを割り当てておかねばならない. 変数に対する次元数の可能な最大値は前もって定義されている定数 `MAXVDIMS` で与えられている.

`NVATS` この変数に割り当てられている変数属性の数を返す.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVINQ` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数について調べている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID               ! 変数 ID
CHARACTER*31 RHNAME         ! 変数名
INTEGER  RHTYPE             ! 変数型
INTEGER  RHN                ! 次元の数
INTEGER  RHDIMS(MAXVDIMS)   ! 変数の形
INTEGER  RHNATT             ! 属性の数
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)  ! ID を取得
CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT, RCODE)
```

7.5 単一のデータ値を書き込む: NCVPT1 および NCVP1C

関数 `NCVPT1` または `NCVP1C` は, データモードにあるオープンされた netCDF ファイルの変数に単一のデータ値を書く. 入力は, netCDF ID, 変数 ID, どの値が追加または変更されるかを指定する複数次元のインデックス, およびデータ値, である.

エラーが生じた場合には, `NCVPT1` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定されたインデックスが指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された netCDF ファイルがデータモードでなはく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCVPT1 (INTEGER NCID, INTEGER VARID, INTEGER MINDEX(*),
     +                   型 VALUE, INTEGER RCODE)

      SUBROUTINE NCVP1C (INTEGER NCID, INTEGER VARID, INTEGER MINDEX(*),
     +                   CHARACTER CHVAL, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVPT1` および `NCVP1C` は変数に単一の値を書く. 最初のものは数値型の変数に数値を書き, 2 番目のものは文字型変数に文字値を書く.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`MINDEX` 書くデータ値の複数次元のインデックス. このインデックスは 1 に相対的なものであり, それゆえ, 例えば 2 次元変数の最初のデータ値はインデックス `(1,1)` を持つ. `MINDEX` の要素はその変数の次元に対応していなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスは記録数に対応していなければならない.

`VALUE` `NCVPT1` における, 書くデータ値である. このデータは netCDF 型 `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE`, のどれに対応していてもよいが, netCDF 変数の型として適当でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`CHVAL` `NCVP1C` における, 書くデータ値である. このデータは netCDF 型 `NCCHAR` または `NCBYTE` に対応した文字型でなければならない.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVPT1` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数の `(4,3,2)` 要素に `0.5` をセットしている. この例においては簡単のため, `rh` が次元 `lon`, `lat`, および `time` を持っていることを知っているとする. そこで, `lon` の 4 番目の値, `lat` の 3 番目の値, および `time` の 2 番目の値に対応する `rh` の値をセットしたいわけである:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID                ! 変数 ID
INTEGER  RHINDX(3)           ! 値を書く場所
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)  ! ID を取得
CALL NCVPT1 (NCID, RHID, RHINDX, 0.5, RCODE)
```

7.6 値の配列を書き込む: NCVPT(C)

関数 `NCVPT` または `NCVPTC` は, オープンされた netCDF ファイルの変数に値を書く. 書きこむ netCDF 変数の部分は, netCDF 変数の配列断面を示す隅および縁の長さのベクトルを与えることによって指定される. 書かれる値は, netCDF 変数の第 1 次元が最も先に変わるような順番で, netCDF 変数と関係づけられる. その netCDF ファイルはデータモードになければならない. [1]

エラーが生じた場合には, `NCVPT` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定された隅のインデックスが, 指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された隅のインデックスに指定された縁の長さを加えたものが, 指定された変数のランクの範囲を外れたデータを参照している. 例えば, 縁の長さが (対応する次元サイズ)−(隅のインデックス) より大きい場合はエラーを引き起こす.
- 指定された netCDF ファイルがデータモードでなはく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
 SUBROUTINE NCVPT (INTEGER NCID, INTEGER VARID, INTEGER START(*),
+                  INTEGER COUNT(*), 型 VALUES, INTEGER RCODE)

 SUBROUTINE NCVPTC (INTEGER NCID, INTEGER VARID, INTEGER START(*),
+                   INTEGER COUNTS(*), CHARACTER*(*) STRING,
+                   INTEGER LENSTER, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVPT` および `NCVPTC` は netCDF 変数に値の配列を書く. 最初のものは数値型の変数に数値を書き, 2 番目のものは文字型変数に文字値を書く.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`START` データ値の先頭が書かれる変数のインデックスを指定する整数のベクトルである. このインデックスは 1 に相対的なものであり, それゆえ, 例えば変数の最初のデータ値はインデックス `(1, 1, ..., 1)` を持つ. `START` のサイズは, 指定された変数の次元数と同じでなければならない. `START` の要素は, 変数の次元に順番に対応しなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスはデータ値を書くための開始記録番号に対応していなければならない.

`COUNT` 書かれるデータ値のブロックの, 各次元に沿った縁の長さを指定する整数のベクトル. 例えば, 単一の値を書くには, `COUNT` を `(1, 1, ..., 1)` と指定する. `COUNT` のサイズは, 指定された変数の次元数である. `COUNT` の要素は, 変数の次元に対応している. 従って, その変数が記録変数ならば, `COUNT` の最後の要素は, 書く記録数の総計に対応している.

---

[1] MSDOS システム上における XDR の現在の実装では, `NCVPT` および `NCVPTC` の各呼び出しにおいてアクセスされるデータの量は, 64K バイト以下に制限されている.

`VALUES` `NCVPT`において, 書かれるデータ値のブロックである. データが指定された変数に書かれる順番は, 1番目の次元から先に変わるような順番である (通常のFORTRANの慣習どおり). データはnetCDF型`NCSHORT`, `NCLONG`, `NCFLOAT`, および`NCDOUBLE`, のどれに対応していてもよいが, そのnetCDF変数の型として適当でなければならない. **警告: コンパイラもnetCDFソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`STRING` `NCVPTC`において, 書かれる文字である. netCDF変数に文字が書かれる順番は, 1番目の次元から先に変わるような順番である (通常のFORTRANの慣習どおり). データはnetCDF型`NCCHAR`または`NCBYTE`に対応する型が許される.

`LENSTR` `NCVPTC`において, `STRING`引数の宣言されている文字長の総計である. これは, `COUNT`ベクトルの要素の積以上の値であるはずである. 引数として配列が与えられることもあるので, この値は, FORTRANの`LEN`関数が返す値と必ずしも同じであるとは限らないことに注意されたい.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0が返される.

以下の例では, `NCVPT`を使って, 既存の‘`foo.nc`’という名前の既存のnetCDFファイルにおいて, 名前`rh`の変数のすべての値を`0.5`にしている. この例においては簡単のため, `rh`が次元`lon`, `lat`, および`time`を持っていること, および`lon`の値が10個, `lat`の値が5個, および`time`の値が3個, であることを知っているとする:

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (NDIMS=3)          ! 次元の数
      PARAMETER (TIMES=3, LATS=5, LONS=10) ! 次元のサイズ
      INTEGER  NCID, RCODE, TIMES
      INTEGER  RHID                ! 変数ID
      INTEGER  START(NDIMS), COUNT(NDIMS)
      DOUBLE RHVALS(LONS, LATS, TIMES)
      DATA START /1, 1, 1/         ! 最初の値からスタート
      DATA COUNT /LONS, LATS, TIMES/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)    ! IDを取得
      DO 10 ILON = 1, LONS
         DO 10 ILAT = 1, LATS
            DO 10 ITIME = 1, TIMES
               RHVALS(ILON, ILAT, ITIME) = 0.5
   10 CONTINUE
      CALL NCVPT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.7 部分的にサンプルされた, またはマップされた値の配列を書く: NCVPTG および NCVPGC

関数`NCVPTG`または`NCVPGC`は, 部分的にサンプルされた, またはマップされた値の配列断面をオープンされたnetCDFファイルの変数に書き込む. 部分的にサンプルされた, またはマップされた値の配列断面は, 隅, 縁の長さのベクトル, ストライドベクトル, およびインデックスマッピングベクトル, を与えて指定される. データ配列の次元の順番またはサイズに関しては特に制限はない. そのnetCDFはデータモードになければならない.

エラーが生じた場合には, `NCVPTG`および`NCVPGC`は`RCODE`として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数IDが, 指定されたnetCDFファイルに対して適当でない.
- 指定された隅のインデックスが, 指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された隅のインデックスに指定された縁の長さおよびストライドを加えたものが, 指定された変数のランクの範囲を外れたデータを参照している. 例えば, 縁の長さが (対応する次元サイズ)−(隅のインデックス) より大きい場合や, そのnetCDF変数の対応する次元のサイズより大きなストライドを使って2つまたはそれ以上の値にアクセスしようとした場合, にはエラーを引き起こす.
- ストライドが正でない.

- 指定された netCDF ファイルがデータモードでなはく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCVPTG (INTEGER NCID, INTEGER VARID, INTEGER START(*),
     +                   INTEGER COUNT(*), INTEGER STRIDE(*), INTEGER IMAP(*),
     +                   型 VALUES, INTEGER RCODE)

      SUBROUTINE NCVPGC (INTEGER NCID, INTEGER VARID, INTEGER START(*),
     +                   INTEGER COUNT(*), INTEGER STRIDE(*), INTEGER IMAP(*),
     +                   CHARACTER*(*) STRING, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVPTG` および `NCVPGC` は netCDF 変数に部分的にサンプルされた, またはマップされた値の配列断面を書き込む. 最初のものは数値型の変数に数値を書き, 2 番目のものは文字型変数に文字値を書く.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`START` データ値の先頭が書かれる変数のインデックスを指定する整数のベクトルである. このインデックスは 1 に相対的なものであり, それゆえ, 例えば変数の最初のデータ値はインデックス `(1, 1, ..., 1)` を持つ. `START` のサイズは, 指定された変数の次元数と同じでなければならない. `START` の要素は, 変数の次元に順番に対応しなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスはデータ値を書くための開始記録番号に対応していなければならない.

`COUNT` 書かれるデータ値のブロックの, 各次元に沿った縁の長さを指定する整数のベクトル. 例えば, 単一の値を書くには, `COUNT` を `(1, 1, ..., 1)` と指定する. `COUNT` のサイズは, 指定された変数の次元数である. `COUNT` の要素は, 変数の次元に対応している. 従って, その変数が記録変数ならば, `COUNT` の最後の要素は, 書く記録数の総計に対応している.

`STRIDE` 各次元に対して, アクセスされる netCDF 変数の値の間隔または `0` を指定する整数のベクトルまたは, 値 `0`, である. ベクトルの要素は, 順番に, 変数の次元に対応している. この値が 1 なら, netCDF 変数の対応する次元の隣接した値にアクセスしていく; 値が 2 なら, netCDF 変数の対応する次元の 1 つおきの値にアクセスしていく; 等々である. 引数が `0` なら, は各次元に沿って, 隣接した値にアクセスしていくというデフォルトの動作になる.

`IMAP` 各次元に対して, netCDF 変数に関連したデータ値がメモリ上にどのように並べられているかを指定する整数のベクトル, または値 `0` である. `VALUE` 引数で指されるメモリ上の位置から特定のデータへのオフセット (バイト単位) は, インデックスマッピングベクトルとデータの (原点を 0 とした) 座標との内積で与えられる. (二つのベクトル [x0, x1, ..., xn] と [y0, y1, ..., yn] と内積は, 単純に x0*y0 + x1*y1 + ... + xn*yn である.) `VALUE` 引数が適切に指定されていれば, このベクトルは負の値を含んでいてもよい. 引数が `0` なら, メモリ上の値が, 関連する netCDF 変数と同じ構造をしていうことが仮定されるデフォルトの動作になる.

`VALUES` `NCVPTG` において, 書かれるデータ値のブロックである. データが指定された変数に書かれる順番は, 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE`, のどれに対応していてもよいが, その netCDF 変数の型として適当でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`STRING` `NCVPGC` において, 書かれる文字である. netCDF 変数に文字が書かれる順番は, 部分的にサンプルされたまたはマップされた配列断面の 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 `NCCHAR` または `NCBYTE` に対応する型が許される.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVPTG` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数の各次元の 1 つおきの値を `0.5` にしている. 同じ次元のストライドを用いて, netCDF 変数と次元が逆である 2 をパラメターとする配列から値を取り出している. この例においては簡単のため, `rh` が次元 `lon`, `lat`, および `time` を持っていること, および `lon` の値が 10 個, `lat` の値が 5 個, および `time` の値が 3 個, であることを知っているとする:

```
      INCLUDE 'netcdf.inc'
```

```
      ...
   PARAMETER (NDIMS=3)          ! 次元数
   PARAMETER (TIMES=3, LATS=5, LONS=10) ! 次元サイズ
   INTEGER  NCID, RCODE
   INTEGER  RHID                ! variable ID
   INTEGER  START(NDIMS), COUNT(NDIMS), STRIDE(NDIMS), IMAP(NDIMS)
                                ! 部分的にサンプルされたまたはマップされた配列
   DOUBLE DATA(2, TIMES, LATS, LONS)    ! rh は 2 番目のパラメターである
   DATA START /1, 1, 1/         ! 最初の値からスタートする
   DATA COUNT /LONS, LATS, TIMES/
   DATA STRIDE /2, 2, 2/
      ...
   NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
      ...
   RHID = NCVID (NCID, 'rh', RCODE)    ! ID を取得
   DO 10 ILON = 1, LONGS, STRIDE(1)
      DO 10 ILAT = 1, LATS, STRIDE(2)
         DO 10 ITIME = 1, TIMES, STRIDE(3)
            DATA(2, ITIME, ILAT, ILON) = 0.5
10 CONTINUE
   IMAP(3) = 8*2*2    ! 倍精度変数 2 個組のベクトルの一つおきの値
   IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
   IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
   CALL NCVPTG (NCID, RHID, START, COUNT, STRIDE, IMAP, DATA(2,1,1,1), RCODE)
```

7.8 単一のデータ値を読む: NCVGT1 および NCVG1C

関数 `NCVGT1` または `NCVG1C` は, データモードにあるオープンされた netCDF ファイルの変数から単一のデータ値を読む. 入力は, netCDF ID, 変数 ID, どの値が追加または変更されるかを指定する複数次元のインデックス, およびデータ値が読みこまれる位置アドレス, である.

エラーが生じた場合には, `NCVGT1` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定されたインデックスが指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された netCDF ファイルがデータモードでなはく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCVGT1 (INTEGER NCID, INTEGER VARID, INTEGER MINDEX(*),
     +                   型 VALUE, INTEGER RCODE)

      SUBROUTINE NCVG1C (INTEGER NCID, INTEGER VARID, INTEGER MINDEX(*),
     +                   CHARACTER CHVAL, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVGT1` および `NCVG1C` は変数から単一の値を読む. 最初のものは数値型の変数中の数値を読み, 2 番目のものは文字型変数の文字値を読む.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`MINDEX` 読むデータ値の複数次元のインデックス. このインデックスは 1 に相対的なものであり, それゆえ, 例えば 2 次元変数の最初のデータ値はインデックス `(1,1)` を持つ. `MINDEX` の要素はその変数の次元に対応していなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスは記録数に対応していなければならない.

`VALUE` `NCVPT1` における, データ値を読みこむ位置である. このデータは netCDF 型 `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE`, のどれに対応していてもよいが, netCDF 変数の型として適当でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`CHVAL` `NCVP1C` における, データ値を読みこむ位置である. このデータは netCDF 型 `NCCHAR` または `NCBYTE` に対応した文字型でなければならない.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVGT1` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数の `(4,3,2)` 要素を取得している. この例においては簡単のため, `rh` が次元 `lon`, `lat`, および `time` を持っていることを知っているとする. そこで, `lon` の 4 番目の値, `lat` の 3 番目の値, および `time` の 2 番目の値に対応する `rh` の値を取得したいわけである:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE
INTEGER RHID ! 変数 ID
INTEGER RHINDX(3) ! 値を取得する位置
DOUBLE PRECISION RHVAL ! ここに入れる
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)    ! ID を取得
CALL NCVGT1 (NCID, RHID, RHINDX, RHVAL, RCODE)
```

7.9 値の配列を読み込む: NCVGT(C)

関数 `NCVGT` または `NCVGTC` は, オープンされた netCDF ファイルの変数から値の配列を読む. 配列は, 隅および縁の長さのベクトルを与えることによって指定される. 値は, netCDF 変数の第 1 次元が最も先に変わるような順番で, 連続的な位置に読み込まれる. その netCDF ファイルはデータモードになければならない. [2]

エラーが生じた場合には, `NCVGT` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定された隅のインデックスが, 指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された隅のインデックスに指定された縁の長さを加えたものが, 指定された変数のランクの範囲を外れたデータを参照している. 例えば, 縁の長さが (対応する次元サイズ)−(隅のインデックス) より大きい場合はエラーを引き起こす.
- 指定された netCDF ファイルがデータモードでなはく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
 SUBROUTINE NCVGT (INTEGER NCID, INTEGER VARID, INTEGER START(*),
+                  INTEGER COUNT(*), 型 VALUES, INTEGER RCODE)

 SUBROUTINE NCVGTC (INTEGER NCID, INTEGER VARID, INTEGER START(*),
+                   INTEGER COUNTS(*), CHARACTER*(*) STRING,
+                   INTEGER LENSTER, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVGT` および `NCVGTC` は netCDF 変数から値の配列を読む. 最初のものは数値型の変数からの数値を読み, 2 番目のものは文字型変数から文字値を読む.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`START` データ値の先頭が読まれる変数のインデックスを指定する整数のベクトルである. このインデックスは 1 に相対的なものであり, それゆえ, 例えば変数の最初のデータ値はインデックス `(1, 1, ..., 1)` を持つ. `START` のサイズは, 指定された変数の次元数と同じでなければならない. `START` の要素は, 変数の次元に順番に対応しなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスはデータ値を読むための開始記録番号に対応していなければならない.

---

[2] MSDOS システム上における XDR の現在の実装では, `NCVGT` および `NCVGTC` の各呼び出しにおいてアクセスされるデータの量は, 64K バイト以下に制限されている.

COUNT 読まれるデータ値のブロックの, 各次元に沿った縁の長さを指定する整数のベクトル. 例えば, 単一の値を読むには, COUNT を (1, 1, ..., 1) と指定する. COUNT のサイズは, 指定された変数の次元数である. COUNT の要素は, 変数の次元に対応している. 従って, その変数が記録変数ならば, COUNT の最後の要素は, 読む記録数の総計に対応している.

VALUES NCVGT において, データ値が読み込まれる位置である. データが指定された変数から読まれる順番は, 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 NCSHORT, NCLONG, NCFLOAT, および NCDOUBLE, のどれに対応していてもよいが, その netCDF 変数の型として適当でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

STRING NCVGTC において, 文字データが読み込まれる文字である. netCDF 変数に文字が書かれる順番は, 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 NCCHAR または NCBYTE に対応する型が許される.

LENSTR NCVGTC において, STRING 引数の宣言されている文字長の総計である. これは, COUNT ベクトルの要素の積以上の値であるはずである. 引数として配列が与えられることもあるので, この値は, FORTRAN の LEN 関数が返す値と必ずしも同じであるとは限らないことに注意されたい. NCVGTC は, 要求されるデータが LENSTR 個の文字に合っていることをチェックする.

RCODE 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, NCVGT を使って, 既存の 'foo.nc' という名前の既存の netCDF ファイルにおいて, 名前 rh の変数のすべての値を読み込んでいる. この例においては簡単のため, rh が次元 lon, lat, および time を持っていること, および lon の値が 10 個, lat の値が 5 個, および time の値が 3 個, であることを知っているとする:

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3) ! 次元の数
PARAMETER (TIMES=3, LATS=5, LONS=10) ! 次元のサイズ
INTEGER NCID, RCODE
INTEGER RHID ! 次元 ID
INTEGER START(NDIMS), COUNT(NDIMS)
DOUBLE RHVALS(LONS, LATS, TIMES)
DATA START /1, 1, 1/ ! 最初の値からスタート
DATA COUNT /LONS, LATS, TIMES/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)! ID を取得
CALL NCVGT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.10 部分的にサンプルされた, またはマップされた値の配列を読む: NCVGTG および NCVGGC

関数 NCVGTG または NCVGGC は, 部分的にサンプルされた, またはマップされた値の配列断面をオープンされた netCDF ファイルの変数から読む. 部分的にサンプルされた, またはマップされた値の配列断面は, 隅, 縁の長さのベクトル, ストライドベクトル, およびインデックスマッピングベクトル, を与えて指定される. 値は第 1 次元が先に変わるような順番で読まれる. その netCDF はデータモードになければならない.

エラーが生じた場合には, NCVGTG および NCVGGC は RCODE として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID が, 指定された netCDF ファイルに対して適当でない.
- 指定された隅のインデックスが, 指定された変数のランクの範囲から外れている. 例えば, 負のインデックスまたは, 対応する次元サイズより大きなインデックス, はエラーを起す.
- 指定された隅のインデックスに指定された縁の長さおよびストライドを加えたものが, 指定された変数のランクの範囲を外れたデータを参照している. 例えば, 縁の長さが (対応する次元サイズ)−(隅のインデックス) より大きい場合や, その netCDF 変数の対応する次元のサイズより大きなストライドを使って 2 つまたはそれ以上の値にアクセスしようとした場合, にはエラーを引き起こす.

- ストライドが正でない.
- 指定された netCDF ファイルがデータモードではなく, 定義モードにある.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCVGTG (INTEGER NCID, INTEGER VARID, INTEGER START(*),
     +                   INTEGER COUNT(*), INTEGER STRIDE(*),
     +                   INTEGER IMAP(*), 型 VALUES, INTEGER RCODE)

      SUBROUTINE NCVGGC (INTEGER NCID, INTEGER VARID, INTEGER START(*),
     +                   INTEGER COUNT(*), INTEGER STRIDE(*),
     +                   INTEGER IMAP(*), CHARACTER*(*) STRING, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCVGTG` および `NCVGGC` は netCDF 変数から部分的にサンプルされた, またはマップされた値の配列断面を読み込む. 最初のものは数値型の変数から数値を読み, 2 番目のものは文字型変数から文字値を読む.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`START` データ値の先頭が読まれる変数のインデックスを指定する整数のベクトルである. このインデックスは 1 に相対的なものであり, それゆえ, 例えば変数の最初のデータ値はインデックス `(1, 1, ..., 1)` を持つ. `START` のサイズは, 指定された変数の次元数と同じでなければならない. `START` の要素は, 変数の次元に順番に対応しなければならない. 従って, もしその変数が記録変数ならば, 最後のインデックスはデータ値を読むための開始記録番号に対応していなければならない.

`COUNT` 読まれるデータ値のブロックの, 各次元に沿った縁の長さを指定する整数のベクトル. 例えば, 単一の値を読むには, `COUNT` を `(1, 1, ..., 1)` と指定する. `COUNT` のサイズは, 指定された変数の次元数である. `COUNT` の要素は, 変数の次元に対応している. 従って, その変数が記録変数ならば, `COUNT` の最後の要素は, 読む記録数の総計に対応している.

`STRIDE` 各次元に対して, アクセスされる netCDF 変数の値の間隔または `0` を指定する整数のベクトルまたは, 値 `0`, である. ベクトルの要素は, 順番に, 変数の次元に対応している. この値が 1 なら, netCDF 変数の対応する次元の隣接した値にアクセスしていく; 値が 2 なら, netCDF 変数の対応する次元の 1 つおきの値にアクセスしていく; 等々である. 引数が `0` なら, は各次元に沿って, 隣接した値にアクセスしていくというデフォルトの動作になる.

`IMAP` 各次元に対して, netCDF 変数に関連したデータ値がメモリ上にどのように並べられているかを指定する整数のベクトル, または値 `0` である. `VALUE` 引数で指されるメモリ上の位置から特定のデータへのオフセット (バイト単位) は, インデックスマッピングベクトルとデータの (原点を 0 とした) 座標との内積で与えられる. (二つのベクトル [x0, x1, ..., xn] と [y0, y1, ..., yn] と内積は, 単純に x0*y0 + x1*y1 + ... + xn*yn である.) `VALUE` 引数が適切に指定されていれば, このベクトルは負の値を含んでいてもよい. 引数が `0` なら, メモリ上の値が, 関連する netCDF 変数と同じ構造をしていうことが仮定されるデフォルトの動作になる.

`VALUES` `NCVGTG` において, データ値が読み込まれる位置である. データが netCDF 変数から読まれる順番は, 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE`, のどれに対応していてもよいが, その netCDF 変数の型として適当でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`STRING` `NCVGGC` において, 文字データが読み込まれる文字列である. netCDF 変数から文字が読まれる順番は, 部分的にサンプルされたまたはマップされた配列断面の 1 番目の次元から先に変わるような順番である (通常の FORTRAN の慣習どおり). データは netCDF 型 `NCCHAR` または `NCBYTE` に対応する型が許される.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVGTG` を使って, 既存の '`foo.nc`' という名前の既存の netCDF ファイルにおいて, 名前 `rh` の変数の各次元の 1 つおきの値を読んでいる. 同じ次元のストライドを用いて, netCDF 変数と次元が逆である 2 をパラメターとする配列に値を割り当てている. この例においては簡単のため, `rh` が次元 `lon`, `lat`, および `time` を持っていること, および `lon` の値が 10 個, `lat` の値が 5 個, および `time` の値が 3 個, であることを知っているとする:

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3) ! 次元の数
PARAMETER (TIMES=3, LATS=5, LONS=10) ! 次元サイズ
INTEGER NCID, RCODE
INTEGER RHID ! 変数 ID
INTEGER START(NDIMS), COUNT(NDIMS), STRIDE(NDIMS), IMAP(NDIMS)
DOUBLE DATA(2, TIMES, LATS, LONS) ! rh は 2 番目のパラメターであ
DATA START /1, 1, 1/ ! 最初の値からスタートする
DATA COUNT /LONS, LATS, TIMES/
DATA STRIDE /2, 2, 2/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)  ! ID を取得
IMAP(3) = 8*2*2 ! 倍精度変数 2 個組のベクトルの一つおきの値
IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
CALL NCVGTG (NCID, RHID, START, COUNT, STRIDE, IMAP, DATA(2,1,1,1), RCODE)
```

7.11 文字列値を読み書きする

文字列は, 基本的な netCDF データ型ではない. これは一つには, FORTRAN が可変長の文字列の抽象化をサポートしていないからである (FORTRANLEN 関数は, 文字列の動的長さではなく, 静的長さを返す). 結果として, netCDF インターフェース中において, 文字列は単一のオブジェクトとして読み書きできない. その代わりに, 文字列は文字の配列として扱われねばならず, 文字列を netCDF ファイル中の変数データとして読み書きするには, 配列アクセスを使う必要がある. さらに, 可変長の文字列は, 慣習による場合を除いて, netCDF インターフェースではサポートされていない; このような慣習の例として, 零のバイトを文字列を終了するものとして扱うことができるが, netCDF 変数から読み込まれるまたはそれに書き込まれる文字列の長さは陽に指定しなければならない.

属性値として文字列は, もっと使いやすい; なぜなら, 文字列がアクセスの単一単位として扱われるからである. しかし, それでも, 文字列属性の値は, その属性が定義された際に指定されているはずの長さを陽に与えた文字の配列である.

文字列値を持つ変数を定義する際には, その変数の最も先に変わる次元 (FORTRAN では第 1 次元) を文字位置次元として使わねばならない. このとき, 文字位置次元のサイズは, 文字列変数に格納されるあらゆる文字列値の最大の長さになる. 最大サイズの列のためのスペースは, そのスペースを使う使わないにかかわらず, 文字列変数のディスク表現の中に割り当てられる. 2 つまたはそれ以上の変数が同じ最大長を持つ場合は, 変数の形を定義するのに同じ文字位置次元を使うことができる.

文字列値を文字列変数に書き込むには, 配列アクセスを使わねばならない. このためには, 隅および縁の長さのベクトルの両方を指定しなければならない. 隅の文字位置次元は 1 になる. もし, 書かれる文字列の長さが `n` であるなら, 縁の長さのベクトルの文字位置次元には `n` が, 他のすべての次元には 1 が, 指定される; すなわち, `(n, 1, 1, ..., 1)` である.

FORTRAN においては, 固定サイズの文字列は, 終了文字なしで netCDF ファイルに書くことができ, これによりスペースを節約することができる. 可変長の文字列は C の慣習に従って, 終了文字である 0 のバイトを付けて書かれる; これは, 後で C または FORTRAN インターフェースで読まれる際に文字列の長さが決められるようにするためである.

文字列を読み書きするための FORTRAN インターフェースでは, 文字列値と数値にアクセスするためには, 異るサブルーチンを用いなければならない; なぜなら, 標準 FORTRAN では, 同じ正式のパラメターが文字値と数値の両方に使われることが許されていないからである. `NCVPTC` および `NCVGTC` では, 値として渡される宣言された文字列長を指定する余分の引数が必要である. 文字列の実際の長さは, 縁の長さベクトルの文字位置次元に対応する値として指定される.

以下の例では, 文字列のために `tx` なる記録変数を定義し, `NCVPTC` を使って 3 番目の記録に文字列値を格納している. この例では, この文字列変数およびデータを, 無制限記録次元 `time` を既に持つ '`foo.nc`' という名前の既存の netCDF ファイルに追加するという状況を考えている.

```
INCLUDE 'netcdf.inc'
   ...
INTEGER   TDIMS, TXLEN
PARAMETER (TDIMS=2)    ! TX 次元の数
PARAMETER (TXLEN = 15) ! 文字列例の長さ
INTEGER  NCID, RCODE
INTEGER  CHID          ! 文字位置次元の id
```

```
      INTEGER  TIMEID        ! 記録次元の id
      INTEGER  TXID          ! 変数 ID
      INTEGER  TXDIMS(TDIMS) ! 変数の形
      INTEGER  TSTART(TDIMS), TCOUNT(TDIMS)
      CHARACTER*40 TXVAL     ! 最大長 40
      DATA TXVAL /'example string'/
         ...
      TXVAL(TXLEN:TXLEN) = CHAR(0)   ! ヌル文字で終了
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
      CALL NCREDF(NCID, RCODE) ! 定義モードに入る
         ...
* 最大長 40 の文字列のための文字位置次元を定義する
      CHID = NCDDEF(NCID, "chid", 40, RCODE)
         ...
* 文字列変数を定義する
      TXDIMS(1) = CHID   ! 文字位置次元が先
      TXDIMS(2) = TIMEID
      TXID = NCVDEF(NCID, "tx", NCCHAR, TDIMS, TXDIMS, RCODE)
         ...
      CALL NCENDF(NCID, RCODE) ! 定義モードから抜ける
         ...
* 3 番目の記録中の netCDF 変数 tx に txval を書き込む
      TSTART(1) = 0      ! 変数の始めからスタート
      TSTART(2) = 3      ! 書き込む記録番号
      TCOUNT(1) = TXLEN  ! 書き込む文字の数
      TCOUNT(2) = 1      ! 1 つの記録だけを書く
      CALL NCVPTC (NCID, TXID, TSTART, TCOUNT, TXVAL, 40, RCODE)
```

7.12 フィル値

オープンされた netCDF ファイルに一度も書かれていない変数を読もうとするとどうなるか? これはいつもエラーとなって, エラーメッセージが得られるかエラー状態が返されると考えるかもしれない. 確かに, 読み込みのためにオープンされていない netCDF ファイルからデータを読もうとしたり, 指定された netCDF において変数 ID が不適当であったり, 指定されたインデックスが指定された次元サイズで定義される範囲に適切に収まっていなかったり, する場合には, エラーとなる. 一方, 書かれていない値を読むと, netCDF ファイルが最初に書かれた際に定義されなかったすべての値を埋めるために使われる特別のフィル値が返される.

フィル値を無視し, netCDF データ型の全範囲を使うことができるが, この場合, 読む前に, すべてのデータ値をちゃんと書いておかねばならない. 読む前にすべてのデータを書くことが確かな場合には, 書く前に `ncsetfill` を呼び出すことによって, 前もってフィル値で変数を埋めるのを止めるように指定することができる. これによって, netCDF の書き込みの効率が著しく向上することもある.

フィル値を指定するためには変数属性 `_FillValue` が使われる. デフォルトのフィル値はインクルードファイル '`netcdf.inc`' で定義される.

netCDF バイト型と文字型の一つの違いは, この二つの型は異なるデフォルトのフィル値を持つということである. 文字のデフォルトのフィル値は 0 のバイトであり, これは可変長の C 文字列の末尾を検出するのに便利な値である. もし, バイト型にフィル値が必要なら, 適切な`_FillValue` 属性を陽に定義することを勧める; なぜなら, ncdump のような一般的なユーティリティはバイト型のデフォルトのフィル値を仮定しないからである.

7.13 変数の名前を変える: NCVREN

関数 `NCVREN` は, オープンされた netCDF ファイル中の変数の名前を変える. 新しい名前が古い名前より長い場合には, その netCDF ファイルは定義モードになければならない. 変数の名前を既存の変数と同じ名前に変えることはできない.

エラーが生じた場合には, `NCVREN` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 新しい名前が他の変数の名前として使われている.
- 変数 ID が指定された netCDF ファイルにおいて適当でない.

- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCVREN (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`NEWNAM` 指定された変数の新しい名前.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCVREN` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 変数 `rh` の名前を `rel_hum` に変えている.

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! 変数 ID
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      CALL NCREDF (CDFFID, RCODE)  ! enter definition mode
      RHID = NCVID (NCID, 'rh', RCODE)  ! ID を取得
      CALL NCVREN (NCID, RHID, 'rel_hum', RCODE)
      CALL NCENDF (CDFFID, RCODE)  ! 定義モードを抜ける
```

7.14 あるデータ型を表現するためのバイト数を得る: NCTLEN

関数 `NCTLEN` は, netCDF 型あたりのバイト数を返す.

エラーが生じた場合には, `NCTLEN` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定されたデータ型が netCDF データ型として適当でない.

```
      INTEGER FUNCTION NCTLEN (INTEGER TYPE ,INTEGER RCODE)
```

`TYPE` 前もって定義されている netCDF データ型集合のうちの一つ. 適当な netCDF データ型は `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE` である.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCTLEN` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, 変数 `rh` の単一の値を格納するのに必要なバイト数を決めている:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID               ! netCDF ID
      INTEGER  RHID               ! 変数 ID
      CHARACTER*31 RHNAME         ! 変数名
      INTEGER  RHTYPE             ! 変数型
      INTEGER  RHN                ! 次元の数
      INTEGER  RHDIMS(MAXVDIMS)   ! 変数形
      INTEGER  RHNATT             ! 属性の数
      INTEGER  RHBYTS             ! 値あたりのバイト
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
* "rh"の型を取得
      CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT, RCODE)
      RHBYTS = NCTLEN (RHTYPE)
```

Chapter 8 属性

各 netCDF 変数は, 単位, 特別な値, 有効な値の最大および最小値, スケーリングファクター, およびオフセットのような性質を指定するために属性を伴っていてもよい. netCDF ファイルに対する属性は, そのファイルが最初に生成される際に, それが定義モードにある間に定義される. 後で再び定義モードに入ることによって, 属性を追加することができる. netCDF 属性は, その属性が割り当てられている変数, 名前, 型, 長さ, および一つまたはそれ以上の値の並び, を持っている. 属性は, その変数 ID および名前で示される. ただし, 一つの場合 (`NCANAM`) を除いて; そこでは, 属性の名前が不明なので, 属性は変数 ID および番号で示される.

変数に関連している属性は, 普通, その変数が生成された直後で, まだ定義モードにあるうちに定義される. 属性のデータ型, 長さ, および値はデータモードにおいても変更できる; その変更によって, 属性が初めに定義されたときよりも多くのスペースが必要とならない限りは.

どの変数とも関連しない属性を持つことも可能である. これらは, グローバル属性と呼ばれ, `NC_GLOBAL` を仮の変数 ID として使って見分けられる. グローバル属性は, 大抵, netCDF ファイル全体と関連づけられており, netCDF データセットに対してタイトルまたは作業記録を付けるために使うことができる.

属性に対してサポートされている操作は以下のようである:

- 変数 ID, 名前, データ型, 長さ, および値を与えて属性を生成する.
- 変数 ID および名前から, 属性のデータ型と長さを取得する.
- 変数 ID および名前から属性の値を取得する.
- ある netCDF 変数から他の変数に属性をコピーする.
- 番号から属性名を取得する.
- 属性の名前を変える.
- 属性を削除する.

8.1 属性の慣習

アンダースコア ('_') から始まる名前は netCDF ライブラリの使用のために予約されている. netCDF ファイルを加工する一般の多くのアプリケーションは, 標準的な属性の慣習を仮定しており, そうしないだけの正当な理由が無い限りは, これらのとりめに従うことを強く勧める. 以下に, 推奨される標準的な属性 (便利であることがすでに示されている) の名前および意味のリストを示す. これらのうち, 数値データが仮定され, 文字データとしては使われないものもある (例えば, `units`, `valid_range`, `scale_factor`) ことに注意が必要である.

`units` 変数データに対して使われている単位を指定する文字列. Unidata は, 自由に取得可能なルーチンのライブラリを提供しており, それによって文字列と単位記述のバイナリ形式との間の変換を行うことができ, さらにそのバイナリ形式においては, いろいろと便利な操作を実行できる. このライブラリはいくつかの netCDF アプリケーションにおいて使われている. 推奨される単位構文を使えば, 整合単位で表現されたデータを, 算術演算できる一般的な単位に自動的に変換できる. より多くの情報については, Appendix A [単位]65ページを参照.

`long_name` 長い記述的な名前. これは, 例えば, プロットのラベルづけに使うことができる. もし, 変数に `long_name` 属性が割り当てられていなければ, 変数名がデフォルトとして使われる.

`valid_min` この変数に対する有効な値の最小値を示すスカラー.

`valid_max` この変数に対する有効な値の最大値を示すスカラー.

`valid_range` この変数に対する有効な値の最小値および最大値を示す 2 つの数値のベクトルであり, `valid_min` および `valid_max` 属性の両方の値を指定するのに等しい. これらのどれもが有効範囲を定義する. もし, `valid_min` または `valid_max` のどちらかが定義されている場合には `valid_range` を定義してはならない.

一般的なアプリケーションは, 有効範囲を外れた値を欠損として扱う. `valid_range`, `valid_min`, および `valid_max` のそれぞれの型は, その変数の型に合っていなければならない (ただし, `byte` 型データを除く. この場合は, これらの属性の型は, 意図する範囲を指定するために, 符号つきの整数型であってもよい).

`valid_min`, `valid_max`, および `valid_range` のどれも定義されていない場合には, 一般的なアプリケーションは, 有効範囲を以下のように定義する. データ型がバイト型で, `_FillValue` が陽に定義されていない場合は, 有効範囲はすべての可能な値を含む. 一方, 有効範囲は, `_FillValue` を以下のように除外する. もし, `_FillValue` が正ならば, それが有効な最大値を定義し, そうでなければ, それは有効な最小値を定義する. 整数型については, `_FillValue` と

この有効な最大または最小値との間には 1 だけ違いがある. 浮動小数点型については, この違いは丸め誤差を許すために, 表現可能な最小値 (最も下位のビットが 1) の 2 倍の値である.

`scale_factor` ある変数についてこの属性が与えられている場合, データは, それにアクセスするアプリケーションによって読み込まれる際にこのファクター倍される.

`add_offset` ある変数についてこの属性が与えられている場合, データは, それにアクセスするアプリケーションによって読み込まれた後にこの数が加えられる. `scale_factor` および `add_offset` の両方の属性が与えられている場合には, データはまずスケールされ, その後にオフセットが加えられる. `scale_factor` および `add_offset` 属性を同時に使うことによって, 簡単なデータ圧縮を行うことができる; 有効桁数の小さい浮動小数点データを小さな整数として netCDF ファイルに格納することによって. スケールされたデータを書く際には, アプリケーションは, まずオフセットを引いて, その後でスケールファクターで割ればよい.

`scale_factor` および `add_offset` を圧縮のために使う際には, 関係する変数 (圧縮されたデータを格納している) は, 普通, byte 型または short 型である; 一方, 解凍されたデータは float または double 型になるようにしたい. というわけで, `scale_factor` および `add_offset` 属性は両方とも, 解凍されたデータの持つべき型 (例えば, float または double 型) でなければならない.

`_FillValue` `_FillValue` 属性は, 変数に割り当てられているディスクスペースを前もって埋めるために使われるフィル値を指定する. このような前もって埋める動作は, `NCSFIL` を使ってノーフィルモードが設定されていない場合になされる. 詳細については, 5.10節 [書き込みのためのフィルモードの設定: NCSFIL]25ページを参照. フィル値は, まだ書かれていない値を読むと返される. `_FillValue` が定義される場合, それはスカラーで, 変数と同じ型でなければならない. `_FillValue` は通常, 有効範囲の外側にあり, それゆえ, 一般のアプリケーションではでは欠損として扱われる. しかし, `_FillValue` を有効範囲内にして, 他の有効な値のように扱えるようにするのは正当なことである. 変数の型に対するデフォルトのフィル値が適当であれば, 独自の`_FillValue` を定義する必要はない. しかし, byte データ型に対しては, デフォルトのフィル値を使うことは勧められない. この属性の値を変えた場合, その変更された値は引き続く書き込みにのみ適用されることに注意されたい; すなわち, それまでに書かれたデータは変更されない. より多くの情報については, 7.12節 [フィル値] 43ページを参照.

`missing_value` この属性は, ライブラリまたは慣習に従った一般的なアプリケーションで特別に扱われるわけではないが, 記録として便利なこともあり, また特定のアプリケーションで使われることもありうる. `missing_value` 属性は, 欠損データを示す値を含むスカラーまたはベクトルでありうる. これらの値はすべて, 一般的なアプリケーションが欠損として扱えるように, 有効範囲の外側にあるべきである. より多くの情報については, 7.12節 [フィル値] 43ページを参照.

`signedness` 価値のなくなった属性である; もともとは, バイト値が符号つきまたは符号なしのどちらで扱われるべきかを指定するように作られたものである. `valid_min` および `valid_max` 属性をこの目的のために使うことができる. 例えば, バイト変数を非負の値として格納したい場合には, `valid_min = 0` および `valid_max = 255` を使うことができる. この属性は, netCDF ライブラリでは無視される.

`C_format` この変数の値をプリントする C アプリケーションにおいて使われるフォーマットを与える文字列である. 例えば, 変数が有効数字 3 桁の精度しかないことが分かっているなら, `C_format` 属性を`"%.3g"`と定義するのが適当である. `ncdump` ユーティリティプログラムは, この属性を, それが定義されている変数に対して使う. このフォーマットは, スケーリングを行う `scale_factor` および `add_offset` 属性のあるなしにかかわらず, スケールされた (内部の) 型および値に対して適用される.

`FORTRAN_format` この変数の値をプリントする FORTRAN アプリケーションにおいて使われるフォーマットを与える文字列である. 例えば, 変数が有効数字 3 桁の精度しかないことが分かっているなら, `FORTRAN_format` 属性を`"(G10.3)"`と定義するのが適当である.

`title` グローバル属性で, そのデータセットの中身を簡単に記述する文字列.

`history` 履歴の検査のためのグローバル属性. これはそのファイルを修正したプログラムの各呼び出しに対する行からなる文字配列である. たちのよい一般的な netCDF アプリケーションは, 日付, 時刻, ユーザ名, プログラム名, およびコマンド引数, を含む行を追加する.

`Conventions` ある変数についてこの属性が与えられている場合, ‘`Conventions`’ は, グローバル属性で, そのファイルが従っている慣習の名前を表す文字配列である; それは, 分野に特化した慣習の集合を記述した文書を貯蔵しているディレクトリからの, 相対ディレクトリ名として解釈できる文字列の形式になっている. これによって, 慣習の階層構造が可能となり, 場所を与えて, 慣習の記述および例を, それを定義している機関およびグループに維持してもらうこ

とができる. 慣習のディレクトリ名は, 現在, ホストマシン `ftp.unidata.ucar.edu` の `put/netcdf/Conventions/` ディレクトリから相対的に解釈される. 代わりに, 慣習を記述する文章が維持されている WWW サイトを指定するために, 完全な URL 指定子を使ってもよい.

例えば, NUWG という名前のグループにおいて, 次元名, 変数名, 必要な属性, およびある分野に特化したデータ構造のための netCDF 表現, に対する一まとまりの慣習についての合意ができているのなら, その合意されたとりきめを記述する文章を, 慣習のディレクトリのサブディレクトリ `NUWG/` 以下のファイルに保管しておくことができる. これらの慣習に従ったファイルは, `"NUWG"`という値を持つグローバル `Conventions` 属性を含むことになる.

後になって, もしそのグループが, NUWG データの特定の部分集合, 例えば, 時系列データ, に対する追加の慣習について合意したなら, その追加の慣習の記述はサブディレクトリ `NUWG/TIME_series/`に保管されることになる; また, これらの追加の慣習に従ったファイルは, `"NUWG/Time_series"`という値を持つグローバル `Conventions` 属性を使って, このファイルが NUWG の慣習および追加的な NUWG の時系列の慣習, に従っていることを示すことになる.

8.2 属性を生成する: NCAPT および NCAPTC

関数 `NCAPT` または `NCAPTC` は, オープンされた netCDF ファイルにおいて, 変数属性またはグローバル属性を追加または変更する. この属性が新しいものであるか, またはその属性を格納するのに必要なスペースが以前より大きな場合には, netCDF ファイルは定義モードになければならない.

エラーが生じた場合には, `NCAPT` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 変数 ID が指定された netCDF ファイルにおいて適当でない.
- 指定された netCDF 型が適当でない.
- 指定された長さが負である.
- 指定されたオープンされている netCDF ファイルがデータモードであり, かつ指定された属性の大きさが大きくなっている.
- 指定されたオープンされている netCDF ファイルがデータモードであり, かつ指定された属性がまだ存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCAPT (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM,
     +                  INTEGER ATTYPE, INTEGER ATTLEN, 型 VALUE, INTEGER RCODE)

      SUBROUTINE NCAPTC (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                   INTEGER LENSTR, CHARACTER*(*) STRING, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCAPT` および `NCAPTC` は属性を生成する. 最初のものは数値型の属性に対するもので, 2 番目のものは文字列型の属性に対するものである.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 変数 ID.

`ATTNAM` 属性名. アルファベット文字で始まり, アンダースコア ('_') を含む零またはそれ以上の英数字が続かねばならない. 大文字小文字は区別される. いくつかの netCDF 一般アプリケーションでは, 属性名の慣習が仮定されている, 例えば, `units` は netCDF 変数に対する単位を与える文字列属性の名前である. 慣習的な属性名の表は netCDF インターフェースに関する前の章で提示している.

`ATTTYP` 前もって定義されている netCDF データ型集合の一つ. 適当な netCDF データ型は `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE` である. `NCAPTC` においては, これは常に `NCCHAR` でなければならない (このインターフェースの欠点であるが, 以前とのコンパチビリティのために必要である).

`ATTLEN` `NCAPT` において, その属性に与えられる数値の数である.

`VALUE` `NCAPT` において, `ATTLEN` のデータ値の配列である. このデータは, その netCDF 属性として適当な型でなければならない. **警告: コンパイラも netCDF ソフトウェアも間違ったデータ型が使われているかどうかを検出できない.**

`STRING` NCAPTC において, 属性の文字列値である.

`LENSTR` NCAPTC において, `STRING` パラメターの宣言されている文字長の総計である. 引数として配列が与えられることもあるので, この値は, FORTRAN の `LEN` 関数が返す値と必ずしも同じであるとは限らないことに注意されたい.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCAPT` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する `valid_range` という名前の属性および, `title` という名前のグローバル属性, を追加している:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID               ! 変数 ID
DOUBLE RHRNGE(2)
DATA RHRNGE /0.0D0, 100.0D0/
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF (NCID, RCODE)     ! 定義モードに入る
RHID = NCVID (NCID, 'rh', RCODE)! ID を取得
   ...
CALL NCAPT (NCID, RHID, 'valid_range', NCDOUBLE, 2, RHRNGE, RCODE)
CALL NCAPTC (NCID, NCGLOBAL, 'title', NCCHAR, 19, 'example netCDF file', RCODE)
   ...
CALL NCENDF (NCID, RCODE)     ! 定義モードを抜ける
```

8.3 属性に関する情報を取得する: NCAINQ

関数 `NCAINQ` は, 変数 ID および名前を与えて, netCDF 属性に関する情報を返す. 返される情報は, 属性の型および長さである.

エラーが生じた場合には, `NCAINQ` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 変数 ID が指定された netCDF ファイルにおいて適当でない.
- 指定された属性が存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCAINQ (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM,
     +                   INTEGER ATTYPE, INTEGER ATTLEN,INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 属性変数の変数 ID, または, グローバル属性については `NCGLOBAL`.

`ATTNAM` 属性名.

`ATTTYP` 属性の型 (前もって定義されている netCDF データ型集合の一つ) が返される. 適当な netCDF データ型は `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, および `NCDOUBLE` である.

`ATTLEN` その属性に現在格納されている値の数が返される. 文字列値を持つ属性については, これは文字列中の文字数となる.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCAINQ` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する `valid_range` という名前の属性および, `title` という名前のグローバル属性, を調べている:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID               ! 変数 ID
INTEGER  VRTYPE, TTYPE      ! 属性の型
INTEGER  VRLEN, TLEN        ! 属性の長さ
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
```

```
   ...
RHID = NCVID (NCID, 'rh', RCODE)! IDを取得
   ...
CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN, RCODE)
CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN, RCODE)
```

8.4 属性値を取得する: NCAGT(C)

関数 `NCAGT` または `NCAGTC` は, 変数 ID および名前を与えて, netCDF 属性の値を返す.

エラーが生じた場合には, `NCAGT` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 変数 ID が指定された netCDF ファイルにおいて適当でない.
- 指定された属性が存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCAGT (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM,
     +                  型 VALUES, INTEGER RCODE)

      SUBROUTINE NCAGTC (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM,
     +                   CHARACTER*(*) STRING, INTEGER LENSTR, INTEGER RCODE)
```

2 つの FORTRAN サブルーチン `NCAGT` および `NCAGTC` は属性値を取り出す. 最初のものは数値型の属性に対するもので, 2 番目のものは文字列型の属性に対するものである.

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 属性変数の変数 ID, または, グローバル属性については `NCGLOBAL`.

`ATTNAM` 属性名.

`VALUES` 属性値が返される. 属性値ベクトルのすべての要素が返されるので, それらを保持するのに十分なスペースを与えておかねばならない. どれだけのスペースを用意しておかねばならないか不明な場合には, まず `NCAINQ` を呼び出して属性の長さを調べればよい.

`STRING` `NCAGTC` において, 属性の文字列値である.

`LENSTR` `NCAGTC` において, 呼び出し時の `STRING` パラメターの宣言されている文字長の総計である. 引数として配列が与えられることもあるので, この値は, FORTRAN の `LEN` 関数が返す値と必ずしも同じであるとは限らないことに注意されたい. `NCAGTC` は, 要求されるデータが `LENSTR` 個の文字に合っていることをチェックする.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCAGT` を使って, 既存の '`foo.nc`' という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する `valid_range` という名前の属性および, `title` という名前のグローバル属性, を決定している. この例において, どれだけ多くの値が返されるかを知らないとしている. そのため, まず属性の長さを問い合わせ, 値を格納するのに十分なスペースを確保している:

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (MVRLEN=3) ! "valid_range"の値の最大数
PARAMETER (MTLEN=80) ! "title"属性の最大長さ
INTEGER  NCID, RCODE
INTEGER  RHID               ! 変数 ID
INTEGER  VRTYPE, TTYPE      ! 属性の型
INTEGER  VRLEN, TLEN        ! 属性の長さ
DOUBLE PRECISION VRVAL(MVRLEN) ! vr 属性の値
CHARACTER*80 TITLE          ! title 属性の値
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE) ! ID を取得
   ...
```

```
* 属性の長さを調べ，十分なスペースを確保している
      CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN, RCODE)
      CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN, RCODE)
* もし大きすぎないならば，属性の値を取得する
      IF (VRLEN .GT. MVRLEN) THEN
          WRITE (*,*) 'valid_range attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGT (NCID, RHID, 'valid_range', VRVAL, RCODE)
      ENDIF
      IF (TLEN .GT. MTLEN) THEN
          WRITE (*,*) 'title attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGTC (NCID, NCGLOBAL, 'title', TITLE, MTLEN, RCODE)
      ENDIF
```

8.5 ある netCDF から他へ属性をコピーする: NCACPY

関数 `NCACPY` または `NCAGTC` は, あるオープンされた netCDF ファイルから他へ属性をコピーする. これは, 同一の netCDF ファイル内で, 一つの変数から他の変数へ属性をコピーするのにも使うことができる.

エラーが生じた場合には, `NCACPY` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 入力または出力の変数 ID が指定された netCDF ファイルにおいて適当でない.
- 指定された属性が存在していない.
- 出力の netCDF が定義モードになく, かつその出力ファイルにおいてその属性が新しいまたは既存の属性より大きい.
- 入力または出力の指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
       SUBROUTINE NCACPY (INTEGER INCDF, INTEGER INVAR, CHARACTER*(*) ATTNAM,
      +                   INTEGER OUTCDF, INTEGER OUTVAR, INTEGER RCODE)
```

`INCDF` 属性がそこからコピーされるべき入力 netCDF ファイルの netCDF ID であり, 以前の `NCOPN` または `NCCRE` 呼び出しで返される.

`INVAR` 属性がそこからコピーされるべき入力 netCDF ファイル中の変数 ID または, グローバル属性については `NCGLOBAL`.

`ATTNAM` コピーされる入力 netCDF ファイル中の属性名.

`OUTCDF` 属性がコピーされるべき宛先の出力 netCDF ファイルの netCDF ID であり, 以前の `NCOPN` または `NCCRE` 呼び出しで返される. 入力と出力の netCDF ID が同じでもよい. コピーされるべき属性が宛先の変数にまだ存在しない場合, またはコピーによって既存の宛先となる属性が大きくなる場合, には出力 netCDF ファイルは定義モードになければならない.

`OUTVAR` 属性がコピーされるべき宛先の出力 netCDF ファイルの変数 ID, または, グローバル属性へコピーする際には, `NCGLOBAL`.

以下の例では, `NCACPY` を使って, 既存の '`foo.nc`' という名前の既存の netCDF ファイル中の `rh` という名前の netCDF 変数から変数属性 `units` を, 他の既存の '`bar.nc`' という名前の netCDF ファイル中の `avgrh` という名前の netCDF 変数へコピーしている; ここで, 変数 `avgrh` が既に存在しているが, まだ `units` 属性を持っていないということを仮定している:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID1, NCID2      ! netCDF ID
      INTEGER  RHID, AVRHID        ! 変数 ID
         ...
      NCID1 = NCOPN ('foo.nc', NCNOWRIT, RCODE)
      NCID2 = NCOPN ('bar.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID1, 'rh', RCODE)
```

```
      AVRHID = NCVID (NCID2, 'avgrh', RCODE)
         ...
      CALL NCREDF (NCID2, RCODE)  ! 定義モードに入る
* 変数属性を"rh"から"avgrh"へコピーする
      CALL NCACPY (NCID1, RHID, 'units', NCID2, AVRHID, RCODE)
         ...
      CALL NCENDF (NCID2, RCODE)  ! 定義モードから抜ける
```

8.6 番号から属性名を取得する: NCANAM

関数 `NCANAM` は, 変数 ID および番号を与えて, 属性の名前を取得する. この関数は, ある変数に関連したすべての変数の名前を取得する必要がある一般アプリケーションにおいて便利である, なぜなら, 他のすべての属性関数は番号ではなく名前で属性にアクセスするからである. 属性の番号は, 名前よりも変わりやすい, なぜなら, 同じ変数の他の属性が削除された際に, 番号は変わりうるからである. 属性番号が属性 ID とは呼ばれないのはこのためである.

エラーが生じた場合には, `NCANAM` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 変数 ID が指定された netCDF ファイルにおいて適当でない.
- 指定された属性番号が負であるか, 指定された変数に対して定義されている属性の数より大きい.
- 指定された属性が存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCANAM (INTEGER NCID, INTEGER VARID, INTEGER ATTNUM,
     +                   CHARACTER*(*) ATTNAM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 属性変数の変数 ID, または, グローバル属性については `NCGLOBAL`.

`ATTNUM` 属性の数. 各変数に対する属性は 1(第 1 属性) から `NVATTS` まで番号づけられている. ここに, `NVATTS` は変数に対する属性の数で, `NCVINQ` 呼び出しで返される.

`ATTNAM` 属性名が返される. 返される名前のためのスペースを割り当てておかねばならない. 属性名の可能な最大文字長は, 前もって定義されている定数 `MAXNCNAM` で与えられている.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCANAM` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する第 1 属性の名前を決定している:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID      ! netCDF ID
      INTEGER  RHID       ! 変数 ID
* 以下の 31 は, MAXNCNAM である
      CHARACTER*31 ATTNAM
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
         ...
* 第 1 属性 (番号 1) の名前を取得する
      CALL NCANAM (NCID, RHID, 1, ATTNAM, RCODE)
```

8.7 属性名を変更する: NCAREN

関数 `NCAREN` は, 属性名を変更する. 新しい名前がもとの名前より長い場合には, netCDF ファイルは定義モードになければならない. 同じ変数の他の属性と同じ名前となってしまうような属性名の変更はできない.

エラーが生じた場合には, `NCAREN` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 変数 ID が指定された netCDF ファイルにおいて適当でない.
- 新しい属性名が, 指定された変数の他の属性にすでに使われている.

- 指定された netCDF ファイルがデータモードにあり, かつ新しい名前が古い名前よりも長い.
- 指定された属性が存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCAREN (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM,
     +                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 属性変数の変数 ID, または, グローバル属性については `NCGLOBAL`.

`ATTNAM` もとの属性名.

`NEWNAM` 指定された属性に割り当てられる新しい名前. 新しい名前が古い名前よりも長い場合には, netCDF ファイルは定義モードになければならない.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCAREN` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する変数属性の名前を `units` から `Units` に変更している:

```
      INCLUDE "netcdf.inc"
         ...
      INTEGER  NCID      ! netCDF ID
      INTEGER  RHID       ! 変数 ID
         ...
      NCID = NCOPN ("foo.nc", NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, "rh", RCODE)
         ...
* 属性名を変更
      CALL NCAREN (NCID, RHID, "units", "Units", RCODE)
```

8.8 属性を削除する: NCADEL

関数 `NCADEL` は, オープンされている netCDF ファイルから netCDF 属性を削除する. netCDF ファイルは定義モードになければならない.

エラーが生じた場合には, `NCADEL` は `RCODE` として非零の値を返す. エラーの原因として考えられるものとしては以下のようなものが挙げられる:

- 指定された変数 ID がて適当でない.
- 指定された netCDF ファイルがデータモードにない.
- 指定された属性が存在していない.
- 指定された netCDF ID がオープンされた netCDF ファイルを参照していない.

```
      SUBROUTINE NCADEL (INTEGER NCID, INTEGER VARID, CHARACTER*(*) ATTNAM, INTEGER RCODE)
```

`NCID` 以前の `NCOPN` または `NCCRE` 呼び出しで返された netCDF ID.

`VARID` 属性変数の変数 ID, または, グローバル属性については `NCGLOBAL`.

`ATTNAM` もとの属性名.

`RCODE` 返されたエラーコード. エラーが生じていなければ, 0 が返される.

以下の例では, `NCADEL` を使って, 既存の ‘`foo.nc`’ という名前の既存の netCDF ファイルにおいて, `rh` という名前の netCDF 変数に対する変数属性 `Units` を削除している:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID      ! netCDF ID
      INTEGER  RHID       ! 変数 ID
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
         ...
* 属性を削除
      CALL NCREDF (NCID, RCODE)  ! 定義モードに入る
      CALL NCADEL (NCID, RHID, 'Units', RCODE)
      CALL NCENDF (NCID, RCODE)  ! 定義モードから抜ける
```

Chapter 9　netCDF ファイル構造および性能

netCDF は, 配列指向のデータアクセスのためのデータ抽象化, およびその抽象化をサポートするインターフェースの具体的な実装を与えるソフトウェアライブラリである. その実装によって, 配列を表現するための機種独立なフォーマットが与えられる. netCDF ファイルフォーマットはインターフェースから隠されているが, 実装および関連したファイル構造をいくらかでも理解することによって, どの netCDF 操作になぜコストがかかるかを明白にする手助けになる.

netCDF フォーマットのより詳細な記述については, Appendix B[ファイルフォーマット仕様]67ページを参照. これは, netCDF ファイルを読み書きしたり, 効率に関することを理解したりするためには必要ない. 文書化されたインターフェースだけを使い, かつフォーマットについて他の仮定をしていないプログラムは, 将来, netCDF フォーマットがたとえ変更されても機能し続けるであろう, なぜなら, このようなあらゆる変更は, 文書化されたインターフェースの下層でなされ, かつ以前のバージョンの netCDF データもサポートされるからである.

この章では, netCDF ファイルの構造, およびネットワーク透過性を与える XDR 層のいくつかの性質を, netCDF の性能の点について理解するのに十分なくらい詳細に記述する.

9.1　netCDF ファイルの部分部分

netCDF データセットは, 2 つの部分より成る 1 つのファイルに格納される.

- ヘッダ: 次元, 属性, および変数データを除く変数, に関するすべての情報を含む.
- データ部分: 固定サイズデータ (無制限次元を持たない変数に対するデータを含む) および記録データ (無制限次元を持つ変数に対するデータ記録を含む)

すべてのデータは, 機種独立にするために XDR 形式で表現される.

netCDF ファイルの初めにある記述的なヘッダは, ファイル中の次元, 変数, および属性に関する情報を表す高度なデータ構造を XDR でコード化したものである. このヘッダにおける変数記述には, 各変数データの先頭へのオフセット, または, ある記録中のある変数の相対オフセットが含まれている. この記述には, 次元サイズ, および各変数に対する複数次元のインデックスを適切なオフセットにどのようにマップするかを決定するために必要な情報も含まれている.

このヘッダには使用可能な余分スペースはない; 各 netCDF ファイル内の次元, 変数, および属性のために必要な大きさしかない. これにより, netCDF ファイルがコンパクトになり, ファイルを自己記述的にするための補助データを格納するために殆どオーバヘッドを必要としない, という利点がある. この機構の潜在的な欠点は, ヘッダを拡張する必要のある netCDF ファイルへの操作, 例えば, 既存の netCDF ファイルに新しい次元, および新しい変数を追加するなど, はそのファイルをコピーするのと同じコストがかかる. このコストは, `NCREDF` が呼ばれた後で `NCENDF` が呼ばれる際にかかる. 必要なすべての次元, 変数, および属性を変数データを書く前に生成し, 後になってファイルのヘッダ部分により多くのスペースを必要とするような netCDF 要素の追加および名前の変更をしないようにすれば, ヘッダの拡張に伴うコストを回避することができる.

ヘッダに続く固定サイズデータ部分は, 無次元 (記録) 次元を使っていない変数に対するすべての変数データを含む. 各変数のデータはファイルのこの部分に連続的に格納される. もし無制限次元がなければ, これは netCDF ファイルの最後の部分である.

固定サイズデータに続く記録データ部分は, 可変数の記録を含み, 各記録がすべての記録変数のためのデータを含んでいる. 各変数に対する記録データは, 各記録内に連続的に格納される.

固定サイズデータ部分内および各記録内にデータが現れる順番は, 変数が定義されたのと同じ順番であり, これは netCDF 変数 ID の昇順である. この知識は, データアクセス性能を向上させるために使われることもある; なぜなら, 最もよいデータアクセスは今のところ, データを順次的に読み書きすることによって達成されるからである.

9.2　XDR 層

XDR は, データの記述およびコード化の標準であり, 外部データ表現のためのライブラリである. これを使って, 機種独立な形でデータ構造をコード化することができる. netCDF はヘッダ部分およびデータ部分の両方において, すべてのデータの表現のために XDR を採用している. XDR は, XDR ライブラリが実装されている他のマシン上で読めるような可搬性のあるデータを書くために使われる.

9.3 XDR についての一般的考察

多くのベンダが, C ランタイムライブラリと一緒に XDR を提供している. 配布される netCDF ソフトウェアもまた, ベンダーがまだ XDR ライブラリを供給していないプラットフォームのために, Sun の可搬的な XDR の実装を含んでいる.

XDR 層が XDR でコード化されたデータを netCDF ファイルに読み書きするには, C 標準 I/O(stdio) ライブラリとそっくりに実装されている I/O 層を使う. 従って, データに同時にアクセスする多重プロセス, I/O バッファの使用, および netCDF ファイルをオープン/クローズするコスト, についてのほとんどの質問には, 標準 I/O ライブラリを理解すれば答えられる. 特に, あるプロセスが netCDF ファイルを書いて, その間に他のプロセスがそれを読むということが可能である. データの読み書きには, stdio の `fread()` および `fread()` より下位の呼び出しは使っていない. `NCSNC()` は標準 I/O ライブラリの `fflush()` に類似したものであり, まだ書かれていないバッファされてるデータを書き出して, 他のプロセスがそれを読めるようにする; `NCSYNC()` はヘッダの変更を最新のものにする (例えば, 属性値への変更など).

stdio ライブラリのときと同様に, ファイルの異なる領域への “探索” が生じた際に flush が実行される. 従って, 読み書きの操作の順番は, I/O の性能に著しく影響を与えうる. 各記録中にデータが書かれたのと同じ順番でデータを読めば, バッファの flush は最小限になる.

状況がより複雑になる普通でない場合が一つある: 他のプロセスも読み込みのためにオープンしている既存の netCDF ファイルに, いくつかの追加次元, 変数, または属性, を追加するために, 書き込み側が定義モードに入るような場合である. この場合, 書き込み側が定義モードから抜けると, 新しい次元, 属性, または変数, と古いデータを持つファイルのコピーが新しく生成されるが, そのファイルをまだオープンしている読み込み側は, そのファイルをクローズして, 再オープンしない限り, その変更を感知できない.

同じファイルを同時に書き込みオープンしている書き込みが多重にあるような場合には, netCDF データアクセスはまともに機能しないかもしれない.

XDR 下の I/O 層を, プラットフォームに特化した他の I/O 層に置き換えることによって, netCDF の実装をチューンできるプラットフォームもある. これは, 例えば, Cray についてなされている. これによって, netCDF と標準 I/O の類似性が変わってしまい, それゆえにデータの共有化, バッファリング, および I/O 操作のコストに関連した性質も変わってしまう可能性はある.

XDR のような, データに対するまっとうな表現を使うことによるコストは, データの型および, そのデータに対するそのマシン本来形式が XDR 形式と同じであるかどうかに依存して変わりうる. XDR は, byte, character, および short 整数データについては特に効率が良い.

とあるマシン上におけるとあるデータ型については, データを XDR(に/から) 変換するのに必要な時間が重大になることもある. 最もよいのは, byte の配列で, そのためには, ほとんど変換コストが生じない; なぜなら, XDR ライブラリは, それらに対するサポートを内蔵しているからである. netCDF の実装には, short(16 ビット) 整数の配列に対する同様なサポートを XDR に追加したものが含まれている. 最悪の場合は, IEEE 浮動小数点をそれ本来の表現としていないマシン上で浮動小数点データの大きな配列を読み書きする場合である. XDR ライブラリは, 浮動小数点数がアクセスされるたびごとに, 関数を呼出し, コストがかかる. 各浮動小数点数のための関数呼び出しのコストが, 浮動小数点部分への netCDF アクセスのコストの大半を占めてしまうようなマシンもある.

配布されている netCDF の実装は, 可搬性を旨としている. I/O の性能をよりよくするための実装のさらなる最適化を行ったり, long 整数および浮動小数点の配列の XDR 変換を最適化するために XDR ライブラリ中のループのアンローリングを行ったりするような, プラットフォームに特化した移植は, データアクセスにより高い性能が求められるような場合には実際的でありかつ望まれるものである.

Chapter 10　netCDFユーティリティ

配列を扱うアプリケーションがnetCDFインターフェースを使う主な理由の一つは，高位レベルのnetCDFユーティリティおよびnetCDFデータに対する一般的なアプリケーションを利用するためである．現在，netCDFソフトウェア配布の一部として，2つのnetCDFユーティリティ：`ncgen`および`ncdump`が利用できる．多くの利用者が他のnetCDFユーティリティを提供しており，netCDFデータにアクセスする様々な可視化および解析パッケージが利用可能である．netCDFデータにアクセスまたはそれを扱う無料の，および商用のソフトウェアの最新のリストについては，netCDFソフトウェアリスト（‘`http://www.unidata.ucar.edu/packages/netcdf/software.html`’）を参照されたい．

この章では，`ncgen`および`ncdump`ユーティリティについて説明する．これら2つのツールは，バイナリのnetCDFファイルと，netCDFファイルのテキスト表現との間の変換をするものである．`ncdump`の出力および`ncgen`の入力は，netCDFファイルをCDL(network Common data form Description Language)として知られるささやかな言語によってテキスト表現したものである．

10.1　CDL構文

以下のCDLの例では，数個の名前つき次元（`lat`，`lon`，`time`），変数（`z`，`t`，`p`，`rh`，`lat`，`lon`，`time`），変数属性（`units`，`_FillValue`，`valid_range`），およびいくらかのデータを持ったnetCDFファイルを記述している．

```
netcdf foo {     // CDLによるnetCDF指定の例

dimensions:
lat = 10, lon = 5, time = unlimited ;

variables:
  long    lat(lat), lon(lon), time(time);
  float   z(time,lat,lon), t(time,lat,lon);
  double  p(time,lat,lon);
  long    rh(time,lat,lon);

  lat:units = "degrees_north";
  lon:units = "degrees_east";
  time:units = "seconds";
  z:units = "meters";
  z:valid_range = 0., 5000.;
  p:_FillValue = -9999.;
  rh:_FillValue = -1;

data:
  lat   = 0, 10, 20, 30, 40, 50, 60, 70, 80, 90;
  lon   = -140, -118, -96, -84, -52;
}
```

すべてのCDF文の終りはセミコロンである．可読性のために，空白，タブ，および改行を使うのは自由である．すべての行において，ダブルスラッシュ//以下に続くものはコメントである．

CDL記述は，3つのオプション部分：次元，変数，およびデータ，から成る．変数部は，変数宣言および属性の割り当てを含んでいてもよい．

次元は，このCDL記述で記述される1つまたはそれ以上の複数次元の変数の形を定義するために使われる．次元は名前およびサイズを持つ．CDL記述の中の1つの次元だけが無制限サイズを持つことができる．すなわち，この次元を持つデータは任意の長さに成長しうる（ファイル中の記録数と同様に）．

変数は，同じ型の値の複数次元の配列を表現する．変数は，名前，データ型，および，その次元のリストで記述される形持つ．各変数はデータ値だけではなく，関連した属性（以下参照）も持てる．名前，データ型，および変数の形は，CDL記述の変数部分における宣言によって指定される．変数は次元と同じ名前を持ってもよい；慣習では，このような変数はその名前の次元の座標値を含んでいる．

属性は，変数またはnetCDFデータセット全体についての情報を含んでいる．属性は，単位，特別な値，有効な値の最大および最小，および圧縮パラメターのような特性を指定するために使うことができる．属性情報は，単一の値または値の配列によって表現される．例えば，`units`は`celsius`のような文字配列で表現される属性である．属性は，関連する変数，名前，データ型，長さ，および値を持っている．データを格納するための変数とは対照的に，属性は，補助データ（データについてのデータ）を格納するためのものである．

CDL においては, 属性は変数および属性の名前をコロン (‘:’) で区切ったもので指定される. 変数名を省略し, 属性名をコロン (‘:’) で始めることによって, netCDF ファイル全体にグローバル属性を割り当てることができる. CDL 中の属性のデータ型は, それに割り当てられている値の型で決まる. 属性の長さは, データ値の数またはその属性に割り当てられている文字列中の文字数である. 値をコンマ (‘,’) で区切ることによって, 文字でない属性に複数の値を割り当てることができる. 属性に割り当てられているすべての値は, 同じ型でなければならない.

変数, 属性, および次元に対する CDL 名には, 英文字または数字と (‘_’) および (‘-’) 文字の任意の組合せが許されるが, (‘_’) から始まる名前はライブラリのために予約されている. CDL 名では大文字小文字は区別される. netCDF ライブラリは netCDF 名に何ら制限を設けていないので, CDL 名として適当でない名前をもつ変数を定義することも (愚かなことではあるが) 可能である. 基本的なデータ型の名前は CDL 中の予約語なので, 変数, 次元および属性の名前は型の名前であってはならない.

CDL 記述のオプショナルデータ部分では, netCDF 変数が初期化される. 初期化の構文は簡単である:

変数 = 値 **1**, 値 **2**,...;

コンマで区切られた定数のリストは, 空白, タブ, および改行で分けることができる. 複数次元の配列については, 最後の次元が先に変わる. というわけで, 行列については, 行順ではなく列順が使われる. 変数を満すのに必要なのよりも少ない値しか与えられない場合には, フィル値が展開される. その定数の型は変数に宣言された型と合致している必要はない; 例えば, 整数を強制的に浮動小数点に変換するといった操作がなされる. 意味のあるすべての型変換がサポートされている.

フィル値のための特別の記法がサポートされている: ‘_’ 文字は, 変数のためのフィル値を示す.

10.2 CDL データ型

CDL データ型は以下のようである:

`char` 文字
`byte` 8 ビット整数
`short` 16 ビット符号つき整数
`long` 32 ビット符号つき整数
`int` long と同義
`float` IEEE 単精度浮動小数点 (32 ビット)
`real` float と同義
`double` IEEE 倍精度浮動小数点 (64 ビット)

`byte` データ型が追加されていること, および `unsigned` 修飾子が無いことを除けば, CDL は C と同じ基本データ型をサポートしている. 宣言文において, データ型は, 大文字または小文字のどちらかで指定される.

`byte` 型は, 8 ビットデータを格納するためにあるという点において, `char` 型と異なっており, 0 のバイトは, 文字データにおけるような特別な意味を持たない. `ncgen` ユーティリティは `byte` 宣言を, 出力 FORTRAN コードでは, `BYTE`, `INTEGER*1`, または同様のプラットフォーム依存の宣言に変換する.

`short` 型は -32768 から 32767 までの値を保持する. `ncgen` ユーティリティは `byte` 宣言を, 出力 FORTRAN コードでは, `INTEGER*2` 宣言に変換する.

`long` 型は -2147483648 から 2147483647 までの値を保持する. `ncgen` ユーティリティは `byte` 宣言を, 出力 FORTRAN コードでは, `INTEGER` 宣言に変換する. CDL 宣言においては, `int` と `integer` は `long` の同義語として解釈される.

`float` 型は -3.4+38 から 3.4+38 までの値を, 32 ビット IEEE 規格の単精度浮動小数点数の外部表現を用いて保持する. `ncgen` ユーティリティは `float` 宣言を, 出力 FORTRAN コードでは, `REAL` 宣言に変換する. CDL 宣言においては, `real` は `float` の同義語として解釈される.

`double` 型は -1.7+308 から 1.7+308 までの値を, 64 ビット IEEE 規格の単精度浮動小数点数の外部表現を用いて保持する. `ncgen` ユーティリティは `double` 宣言を, 出力 FORTRAN コードでは, `DOUBLE PRECISION` 宣言に変換する.

10.3 データ定数に対する CDL 表記

この節では, 定数に対する CDL 表記について説明する.

属性は, CDL記述の`variables`節において, 属性の型および長さを決める定数のリストを与えることによって初期化される. (netCDFライブラリへのCおよびFORTRAN手続きインターフェースにおいて, 属性の型および長さは, それが定義される再に陽に与えられなければならない.) CDLは異なるnetCDF型の中から型を区別できるような, 定数値のための構文を定義している. CDL定数のための構文は, Cの構文に類似しているが, `shorts`および`floats`を`long`および`double`から区別するために, 型を示す添字が追加されている.

`byte`定数は単一の文字または, シングルクオートで囲まれた複数文字のエスケープ列, で表現される. 例えば:

```
'a'      // ASCII a
'\0'     // 0のバイト
'\n'     // ASCII改行文字
'\33'    // ASCIIエスケープ文字 (8進数で33)
'\x2b'   // ASCIIプラス (16進数で2b)
'\376'   // 8進数で377 = 10進数で-127 (または254)
```

文字定数はダブルクオートで囲まれる. 文字配列はダブルクオートで囲まれた文字列で表現できる. 複数の文字列は, 1つの文字配列に連結される; これによって, 長い文字配列を複数行に書くことができる. 複数の可変長の文字列値をサポートするために, ‘,’ のような慣習的な区切り文字を使うことはできるが, 文字列区切りのためのこのような慣習は, netCDFライブラリ層の上のソフトウェアで実装されなければならない. C文字列に対する通常のエスケープ文字の慣習はそのまま使える. 例えば:

```
"a"              // ASCII ‘a’
"Two\nlines\n"   // 2つの改行文字を埋めこまれた10文字の文字列
"a bell:\007"    // ASCII ベルを含んだ文字列
"ab","cde"       // "abcde"に同じ
```

`short`定数の形式は, ‘`s`’ または ‘`S`’ を付加した整定数である. `short`定数が ‘`0`’ で始まっている場合は, 8進数であると解釈される. ‘`0x`’ で始まっている場合は, 16進数であると解釈される. 例えば:

```
2s       // short型の2
0123s    // 8進数
0x7ffs   // 16進数
```

`long`定数の形式は, 通常の整定数である; ただし, オプションとして ‘`l`’ または ‘`L`’ を付加してもよい. `long`定数が ‘`0`’ で始まっている場合は, 8進数であると解釈される. ‘`0x`’ で始まっている場合は, 16進数であると解釈される. 妥当な`long`定数の例は以下のようである:

```
-2
1234567890L
0123              // 8進数
0x7ff             // 16進数
```

`float`型は, 有効数字約7桁の精度を持つデータを表現するのに適当なものである. `float`型の形式は, Cの浮動小数点定数に ‘`f`’ または ‘`F`’ を付加したものと同じである. CDL`float`においては, 整数と区別するために, 小数点が必須である. 妥当な`float`定数の例は以下のようになる:

```
-2.0f
3.14159265358979f        // 低精度に丸められる
1.f
.1f
```

`double`型は, 有効数字約16桁の精度を持つデータを表現するのに適当なものである. `double`型の形式は, Cの浮動小数点定数と同じである. オプションとして ‘`d`’ または ‘`D`’ を付加してもよい. CDL`double`においては, 整数と区別するために, 小数点が必須である. 妥当な`double`定数の例は以下のようになる:

```
-2.0
3.141592653589793
1.0e-20
1.d
```

10.4 ncgen

`ncgen`ツールは, netCDFファイルまたは, netCDFファイルを生成するCまたはFORTRANプログラム, を作成する. `ncgen`を呼び出す際にオプションを指定しなければ, このプログラムはCDL入力の構文をチェックするのみで, CDL構文違反についてのエラーメッセージを出すだけである.

`ncgen`をUNIXで呼び出すための書式は:

```
ncgen [-b] [-o netCDFファイル] [-c] [-f] [-n] [入力ファイル]
```

ここに,

‘-b’ (バイナリの)netCDFファイルを生成する. ‘-o’ が指定されていない場合は, デフォルトのファイル名がnetCDF名(入力中のnetcdfキーワードの後に指定される)から, 拡張子‘.nc’に付加して作成される. **警告: もし指定された名前のファイルが既に存在していると, それは上書きされる.**

‘-o netcdfファイル’ 生成されるnetCDFファイルの名前を指定する. このオプションが指定されると, ‘-b’オプションが暗黙に仮定される. (netCDFファイルは探索呼び出しによって生成される直接参照ファイルであり, そのため標準出力に書くことができないので, このオプションが必要である.)

‘-c’ netCDF指定に合致したnetCDFファイルを生成するCのソースコードを作成する. このCソースコードは, 標準出力に出力される. これは, 比較的小さなCDLファイルについてのみ有用なものでである; なぜなら, 作成されたプログラム中ではすべてのデータは変数初期化の中に含まれてしまうからである.

‘-f’ netCDF指定に合致したnetCDFファイルを生成するFORTRANのソースコードを作成する. このFORTRANソースコードは, 標準出力に出力される. これは, 比較的小さなCDLファイルについてのみ有用なものでである; なぜなら, 作成されたプログラム中ではすべてのデータは変数初期化の中に含まれてしまうからである.

‘-n’ ‘-b’オプションと同様であるが, ‘-o’オプションで出力ファイル名が指定されていない場合には‘.nc’拡張子の代わりに旧式の‘.cdf’拡張子を持ったnetCDFファイルを生成する. このオプションは以前とのコンパチビリティのためにサポートされているだけである.

例

CDLファイル‘foo.cdl’の構文をチェックする:

```
ncgen foo.cdl
```

CDLファイル‘foo.cdl’から,‘bar.nc’という名前の等価なバイナリファイルを作成する.

```
ncgen -o bar.nc foo.cdl
```

CDLファイル‘foo.cdl’から, 等価なバイナリファイルを生成するために必要なnetCDF関数呼出しを含むCプログラムを作成する:

```
ncgen -c foo.cdl > foo.c
```

10.5 ncdump

`ncdump`ツールは, netCDFファイルのCDLテキスト表現を標準出力に出力する; オプションで, 入力中のデータの変数の一部または全てを除外することもできる. `ncdump`の出力は, `ncgen`への入力として使えるようになっている. 従って, `ncdump`および`ncgen`は, バイナリおよびテキスト表現の間でデータ表現を変換するための正逆変換として使うことができる.

`ncdump`は, netCDFデータファイルの簡単な閲覧手段として使うことができ, netCDFファイルの中身について以下のような表示ができる: 次元名およびサイズ; 変数名, 型および形; 属性名および値; およびオプションですべての変数または選択された変数の値.

`ncdump`は, netCDF変数データの各型に使われるデフォルトのフォーマットを定義しているが, これは`C_format`属性がnetCDF変数に対して定義されていれば上書きされる. この場合, `ncdump`は`C_format`属性を使って, その変数の値のフォーマットを行う. 例えば, netCDF変数`Z`に対する浮動小数データが有効数字3桁の精度しかないことが分かっているならば, この変数属性を使うのが適当である.

`ncdump`は, ‘_’を使って, 変数に対する`_FillValue`属性(これはまだ書かれていないデータを表現するためのものである)と等しいデータ値を表す. もし, 変数が`_FillValue`属性を持っていなければ, その変数型に対するデフォルトのフィル値が使われる; その変数がバイト型でなければ.

```
Z:C_format = "%.3g"
```

`ncdump`をUNIXで呼び出すための書式は:

```
ncdump  [ -c | -h]  [-v 変数1,...]  [-b 言語]  [-f 言語]
[-l 長さ]  [ -d floatの桁[,doubleの桁]] [ -n 名前]  [入力ファイル]
```

ここに,

‘`-c`’ 全ての次元, 変数, および属性値の宣言と, 座標変数 (次元にもなっている変数) の値を表示する. 座標変数でない変数のデータ値は出力には含まれない. これは, netCDF ファイルの構造および中身を簡潔に見るのにたいてい最も適したオプションである.

‘`-h`’ ヘッダの情報のみを出力する, すなわち, 入力ファイルの netCDF 次元, 変数, および属性の宣言のみを出力し, 変数のデータ値は出力しない. 出力は, 座標変数の値が含まれないことを除けば, ‘`-c`’ オプションを使うのと同じである. (‘`-c`’ または ‘`-h`’ オプションのどちらか一方しか指定してはならない.)

‘`-v 変数1,...`’ すべての次元, 変数, および属性の宣言に加えて, 指定された変数のデータ値が出力に含まれる. このオプションに続けてコンマで区切った名前のリストを与えることによって, 1 つまたはそれ以上の変数を指定しなければならない. このリストはコマンドへの単一の引数でなければならないので, 空白または他の空白文字を含めることはできない. 名前つき変数は, 入力ファイル中の妥当な netCDF 変数でなければならない. このオプションおよび ‘`-c`’ または ‘`-h`’ オプションを指定しないデフォルトでは, すべての変数に対するデータ値が出力に含まれる.

‘`-b 言語`’ CDL コメント (‘`//`’ で始まる文字) の形式の簡潔な注釈が, 出力のデータ節中におけるデータの各 ‘列’ に対して含まれるようになり, 複数次元の変数に対するデータ値を確認する助けになる. 言語が ‘`C`’ または ‘`c`’ で始まっている場合には, C 言語の慣例が使われる (0 を基準としたインデックス, 最後の次元が真っ先に変わる). 言語が ‘`F`’ または ‘`f`’ で始まっている場合には, FORTRAN 言語の慣習が使われる (1 を基準としたインデックス, 最初の次元が真っ先に変わる). どちらの場合においても, データは同じ順番で表示される; 注釈のみが異なる. このオプションは, 大量の複数次元のデータを一覧するのに便利なオプションである.

‘`-f 言語`’ CDL コメント (‘`//`’ で始まる文字) の形式の, 各データ値 (文字配列の個々の文字は除いて) に対する完全な注釈がデータ節に含まれるようになる. 言語が ‘`C`’ または ‘`c`’ で始まっている場合には, C 言語の慣例が使われる (0 を基準としたインデックス, 最後の次元が真っ先に変わる). 言語が ‘`F`’ または ‘`f`’ で始まっている場合には, FORTRAN 言語の慣習が使われる (1 を基準としたインデックス, 最初の次元が真っ先に変わる). どちらの場合においても, データは同じ順番で表示される; 注釈のみが異なる. このオプションは, データを他のフィルターにパイプするのに便利なオプションである; なぜなら, 各データ値は別々の行に表示され, 完全に識別されるからである. (‘`-b`’ または ‘`-f`’ オプションのどちらか一方しか指定してはならない.)

‘`-l 長さ`’ 文字でないデータ値のリストをフォーマットする際に使われる行の最大長のデフォルト (80) を変更する.

‘`-d floatの桁[,doubleの桁]`’ `C_format` 属性を持たない変数に対する浮動小数点または倍精度のデータ値を表示するのに使われる有効数字の数のデフォルトを指定する. 浮動小数点データは **float** の桁の有効桁で表示される. **double** の桁も指定された場合には, 倍精度値はその分の有効桁で表示される. 変数が `C_format` 属性を持っている場合には, 指定されたすべての浮動小数点のデフォルトが上書きされる. ‘`-d`’ オプションが指定されなければ, 浮動小数点および倍精度データは, それぞれ有効桁 7 および 15 で表示される. もし, より低い精度しか必要ないなら, CDL ファイルはより小さくできる. 浮動小数点および倍精度の精度の両方が指定される場合は, この 2 つの値はコマンドに対する単一の引数となるようにコンマ (空白はなし) で区切って与えねばならない.

‘`-n 名前`’ CDL では, ‘`ncgen -b`’ でデフォルトの netCDF ファイル名を生成する際に使うために, netCDF データセットの名前が必要である. デフォルトでは, `ncdump` は入力 netCDF ファイルの名前の最後の要素より任意の拡張子を取り除いたものからこの名前を構成する. 異なる名前を指定するためには ‘`-n`’ オプションを使う. ‘`ncgen -b`’ で使われる出力ファイル名を指定することはできるが, `ncdump` を使って生成される CDL ファイルを編集し, さらに ‘`ncgen -b`’ を使ってその編集した CDL ファイルから新しい netCDF ファイルを生成したりする際に, うっかりと貴重な netCDF ファイルを上書きしてしまうのを避けるために, `ncdump` を使ってデフォルトの名前を変更しておくのが賢明である.

例

netCDL ファイル ‘`foo.nc`’ 内のデータ構造を見る:

```
ncdump -c foo.nc
```

注釈のために C スタイルのインデックスづけを使って, netCDL ファイル ‘`foo.nc`’ 内の構造およびデータの CDL 版を注釈つきで生成する.

```
ncdump -b c foo.nc > foo.cdl
```

netCDL ファイル‘`foo.nc`’から変数 `uwind` および `vwind` のみに対するデータを出力し, 浮動小数点データを有効数字 3 桁だけの精度で表示する:

```
ncdump -v uwind,vwind -d 3 foo.nc
```

インデックスづけに FORTRAN の慣習を使って, 変数 `omega` に対するデータの完全な注釈つき (1 行につき 1 つのデータ) のリストを生成し, 生成される CDL ファイル中の netCDF データセット名を `omega` に変更する.

```
ncdump -v omega -f fortran -n omega foo.nc > Z.cdl
```

Chapter 11　よくある質問への答

この章では, netCDF について最もよくある質問のうちのいくつかへの答を書いている. netCDF についてのより包括的で最新の FAQ 文書は,

‘`http://www.unidata.ucar.edu/packages/netcdf/faq.html`’

に維持されている.

netCDF とは?

netCDF(network Common Data Form) は配列指向のデータアクセスのためのインターフェース, およびそのインターフェースの実装を与える C, FORTRAN, C++ ソフトウェアライブラリの収集物で, これらは無償で配布される. netCDF ソフトウェアは Colorado 州 Boulder にある Unidata Program Center の Glenn Davis, Russ Rew, および Steve Emmerson によって開発され, 他の netCDF ユーザからの貢献によって増強された. netCDF ライブラリは配列を表現するための機種独立のフォーマットを定義している. 同時に, このインターフェース, ライブラリおよびフォーマットは, 配列の生成, アクセス, および共有, をサポートしている.

netCDF データは:

- 自己記述的である. netCDF ファイルはその中身のデータについての情報を含んでいる.
- ネットワーク透過である. netCDF ファイルは, 整数, 文字, および浮動小数点数を異なる形式で格納するコンピューターでもアクセスできる形式で表現されている.
- 直接アクセスである. 大きなデータセットの小さな部分集合に効率的にアクセスできる; 最初にその前のすべてのデータを読み込んだりすることなしに.
- 追加ができる. 1 つの次元に沿って netCDF データセットにデータを追加することができる; そのデータをコピーしたり, 構造を再定義したりすることなしに. netCDF データセットの構造は変更できる; この場合, データセットのコピーを引き起こしてしまうこともあるが.
- 共有できる. 1 つの書き込みおよび複数の読み込みが同一の netCDF ファイルに同時にアクセスできる.

netCDF ソフトウェアパッケージはどのように取得できるか?

配布されているソースは, 以下のディレクトリからアノニマス FTP で取得できる:

‘`ftp://ftp.unidata.ucar.edu/pub/netcdf/`’

このディレクトリには, 以下のファイルが含まれている:

`netcdf.tar.Z` 一般的な最新版のソースコードの tar ファイルを圧縮したもの.

`netcdf-beta.tar.Z` 現在のベータテスト版.

いくつかのプラットフォームのためのバイナリ配布は以下のディレクトリから取得できる:

‘`ftp://ftp.unidata.ucar.edu/pub/binary/`’

perl インターフェースのソースは, 別のパッケージとして以下のディレクトリからアノニマス FTP で取得できる:

‘`ftp://ftp.unidata.ucar.edu/pub/netcdf-perl/`’

World Wide Web 上で netCDF の情報にアクセスできるか?

はい. netCDF ユーザーズガイドのハイパーテキスト版および netCDF についての他の情報, さらにこの FAQ 文章の最新版は, 以下から取得できる:

‘`http://www.unidata.ucar.edu/packages/netcdf`’

以前の版から何が変わったか?

2.4 版は, 新しいプラットフォームのサポートおよび以前からサポートされていたプラットフォームの最新版を組込み, Cray/UNICOS のための新しい最適化を与え, 報告されていたバグを修正し, 文章を改良し, さらにインストールをより簡単にした. より詳細については, 以下を参照:

`‘http://www.unidata.ucar.edu/packages/netcdf/release-notes.html’`

netCDF についての議論および質問のためのメイリングリストはありますか?

はい. このメイリングリスト, およびどうしたら参加/脱退ができるかについての情報については,

`majordomo@unidata.ucar.edu`

にサブジェクトなしで本文に以下の行を書いたメッセージを送られたい:

```
info netcdfgroup
```

他の誰が netCDF を使っていますか?

netCDF メイリングリストは 15 の国にわたる 500 程のアドレスを持っている (このうちのいくつかはより多くのアドレスへのエイリアスである). 多くのグループが netCDF をいくつかの形式の配列指向データを表現する標準手法として採用している; この中には, 大気科学, 水理学, 海洋学, 環境モデリング, 地球物理学, クロマトグラフィ, 質量分光学, およびニューロイメージング, が含まれている.

netCDF を使っているプロジェクトおよびグループのいくつかについての記述は, 以下から取得できる:

`‘http://www.unidata.ucar.edu/packages/netcdf/usage.html’`

netCDF ファイルの物理的なフォーマットはどのようなものですか?

9 章 [netCDF ファイル構造および性能]53ページを参照されたい. ここでは, netCDF データの物理的構造を説明しており, その説明は, 異なったデータ構成の性能の含みを明解にするのに十分なレベルである. ファイルフォーマットの詳細な指定については, Appendix B[ファイルフォーマット仕様]67ページを参照されたい.

netCDF データにアクセスするプログラムは, すべてのアクセスを文書化されたインターフェースを通して実行するべきであり, netCDF データの物理フォーマットに依存すべきではない. そうすることにより, たとえ将来, このフォーマットが変更されても, プログラムの変更の必要はなくなる; なぜなら, このような変更の際にはライブラリも変更され, 旧版および新版の両方のフォーマットがサポートされるからである.

netCDF はどこで動作しているか?

netCDF の現在の版は, 以下のプラットフォーム上でテストされ, うまくいっている:

- AIX-4.1
- HPUX-9.05
- IRIX-5.3
- IRIX64-6.1
- MSDOS (gcc, f2c, および GNU make を使う)
- OSF1-3.2
- OpenVMS-6.2
- OS/2 2.1
- SUNOS-4.1.4
- SUNOS-5.5
- ULTRIX-4.5
- UNICOS-8
- Windows NT-3.51

netCDF データに対して, 他にどんなソフトウェアが使えるか?

Unidata から配布されている現在の netCDF で取得可能なユーティリティは, netCDF ファイルを可読な ASCII 形式に変換する ncdump, および可読な ASCII 形式からバイナリの netCDF ファイル, またはその netCDF ファイルを生成する C または FORTRAN プログラム, に戻す ncgen である.

多くの商用または無償の解析およびデータ可視化パッケージが netCDF データアクセスに適応している. これらのパッケージおよび, netCDF データを処理または表示するために使える他のソフトウェア, についてのより多くの情報は, 以下から取得できる:

‘`http://www.unidata.ucar.edu/packages/netcdf/software.html`’

配列指向のデータには, 他にどんなフォーマットが使えるか?

科学データフォーマット情報 **FAQ** が以下から取得できる:

‘`http://www.cis.ohio-state.edu/hypertext/faq/usenet/sci-data-formats/faq.html`’

ここでは, 他のアクセスインターフェースおよび, CDF と HDF を含む配列指向データのためのフォーマット, について詳述されている.

なぜ netCDF 呼出しは, エラー標識を返す代わりに単に終了してしまうのか?

すべての netCDF 関数のデフォルトのエラー処理の動作は, エラーがあると終了してしまうというものであるが, この動作はプログラマがコントロールできる. エラーの致命度, および netCDF ライブラリ呼出しで検出されるエラーからのメッセージの表示, は独立にコントロールできる. より多くの情報については, 4.5節 [エラー処理]19ページを参照.

バグ報告はどうしたらよいですか?

もしバグを見つけたら, `support@unidata.ucar.edu` に報告を送って下さい. これは, netCDF についての質問または議論のために使われているアドレスでもある. 質問または議論を `netcdfgroup` メイリングリスト全体に送ることは適当ではない.

過去の問題の報告はどうしたら検索できますか?

netCDF ホームページの一番下に検索フォームがあり, Unidata サポートスタッフによって提供されているサポートの質問および回答について, 完全なテキスト検索ができるようになっている.

C++インターフェースは C インターフェースとどう違いますか?

C++のインターフェースは C インターフェースのすべての機能を提供しており (`ncvarputg` および `ncvargetg` における部分的にサンプルされた, またはマップされた配列アクセスを除く), `void*`ポインタの使用をやめて型の安全性を改善し, C インターフェースよりもいくらか簡単に使えるようになっている. C++インターフェースを使うと, netCDF 要素に ID は不要になり, 属性を生成する際に型の指定が不要になり, さらに次元をより直接的に扱うことができる. しかし, C++インターフェースは C インターフェースよりも成熟しておらず, かつそれほど広く使われていない. さらに, C++インターフェースの文章は, netCDF データモデルおよび C インターフェースに慣れていることを前提としており, あまり広範なものではない.

FORTRAN インターフェースは C インターフェースとどう違いますか?

FORTRAN インターフェースは C インターフェースのすべての機能を提供している. FORTRAN インターフェースは, 配列のインデックス, 添字の順番, および文字列に対する FORTRAN の慣習を使っている. 異なる言語インターフェースから書かれたデータについて, ディスク上のフォーマットの違いはない. C 言語で書かれたデータは FORTRAN プログラムから読むことができ, 逆も可能である.

Perl インターフェースは C インターフェースとどう違いますか?

Perl インターフェースは C インターフェースのすべての機能を提供している. Perl インターフェースは, 配列および文字列に対する Perl の慣習を使っている. 異なる言語インターフェースから書かれたデータについて, ディスク上のフォーマットの違いはない. C 言語で書かれたデータは Perl プログラムから読むことができ, 逆も可能である.

Appendix A 単位

Unidata Program Center は, フォーマットされたおよびバイナリ形の単位間の変換を行い, かつバイナリ形式における単位の代数演算を行う単位ライブラリを開発している. 単位ライブラリは自己完結的であり, それと netCDF ライブラリとの間には依存関係はないが, それでも, このライブラリは一般的な netCDF プログラムを書く際に有用であるので取得することを勧める. このライブラリおよび関連文書は, ‘`http://www.unidata.ucar.edu/packages/udunits/`’ から取得できる.

以下は, Unidata 単位ライブラリの関数 `utScan()` で解釈できる単位の文字列の例である:

```
10 kilogram.meters/seconds2
10 kg-m/sec2
10 kg m/s^2
10 kilogram meter second-2
(PI radian)2
degF
100rpm
geopotential meters
33 feet water
milliseconds since 1992-12-31 12:34:0.1 -7:00
```

単位は, 単位名の任意の整数冪に任意の定数を掛けたものとして指定される. 割り算はスラッシュ‘/’ で表される. 乗算は, 空白, ピリオド ‘`.`’, またはハイフン ‘`-`’ で表される. 冪乗は, 整数の添字, または冪乗演算子 ‘`^`’ および ‘`**`’ によって表される. グループ分けおよび明瞭化のために括弧を用いてもよい. 最後の例のタイムスタンプは特別な場合として扱われる.

任意のガリレオ変換 (すなわち, $y = ax + b$) が許されている. 特に, 温度の変換は正しく扱える. 以下の指定:

```
degF @ 32
```

は, 原点を華氏 32 度分シフト (すなわち, 摂氏 0 度) した華氏目盛を指す. 従って, 摂氏目盛は, 以下の単位に等しい:

```
1.8 degF @ 32
```

原点移動演算は乗算より優先されることに注意. 演算の優先順位は (下位から上位へ), 除算, 乗算, 原点移動, 冪乗, の順である.

`utScan()` はすべての SI 接頭語 (例えば, “mega” および “milli”) およびそのそれらの短縮形 (例えば, “M” および “m”)

関数 `utPrint()` は常に, 単位指定を一意にコード化する. 間違いを減らすために,, このコード化スタイルをデフォルトとして使うことを勧める. 基本単位は空白で区切られ, 任意の指数は対応する単位に直接付加される. 上の例は以下のようにコード化される:

```
10 kilogram meter second-2
9.8696044 radian2
0.555556 kelvin @ 255.372
10.471976 radian second-1
9.80665 meter2 second-2
98636.5 kilogram meter-1 second-2
0.001 seconds since 1992-12-31 19:34:0.1000 UTC
```

(華氏単位は原点 255.372kelvin からの分数の偏差としてコード化されており, 最後の例における時刻は, UTC に変換されていることに注意.)

単位ライブラリのためのデータベースは, 単位定義を含むフォーマットされたファイルであり, このパッケージを初期化するために使われる. 妥当な名前と記号を見付けるために, まずここが参照される.

この単位ファイルのフォーマットは内部に説明があり, 必要ならば, このファイルはユーザが修正できる. 特に, 付加的な単位および定数は簡単に追加できる (既存の単位または定数の異なる綴りを含む).

`utScan()` は大文字小文字を区別する. もしこれで支障があるなら, 単位ファイルに適当な項目を追加してもよい.

デフォルトの単位ファイルにおける単位短縮形は, 直感に反することもある. 特に, 以下に注意されたい:

For	Use	Not	Which Instead Means
Celsius	‘Celsius’	‘C’	coulomb
gram	‘gram’	‘g’	<standard free fall>

`gallon`	`‘gallon’`	`‘gal’`	`<acceleration>`
`radian`	`‘radian’`	`‘rad’`	`<absorbed dose>`
`Newton`	`‘newton’ or ‘N’`	`‘nt’`	`nit (unit of photometry)`

この単位ライブラリについてのさらなる情報については, 配布版に付属のマニュアルページを参照されたい.

Appendix B ファイルフォーマット仕様

このappendixではnetCDFファイルフォーマットバージョン1の仕様を述べる．このフォーマットは，少なくともnetCDFライブラリバージョン3.0までは使われる予定である．

このフォーマットは，BNF文法表記で正式に表現される．この文法では，オプションの要素は括弧（‘[’と‘]’）で囲まれている．‘//’以下は注釈である．端末語でないものは小文字で，端末語は大文字で表記されている．0またはそれ以上の項目を並べる場合は，‘`[entity ...]`’と表記される．

ファイルフォーマット仕様

```
netcdf_file := header  data

header   := magic  numrecs  dim_array  gatt_array  var_array

magic    := 'C'  'D'  'F'  VERSION_BYTE

VERSION_BYTE := '\001'    // ファイルフォーマットのバージョン数

numrecs     := NON_NEG

dim_array  :=  ABSENT | NC_DIMENSION  nelems  [dim ...]

gatt_array :=  att_array  // グローバル属性

att_array  :=  ABSENT | NC_ATTRIBUTE  nelems  [attr ...]

var_array  :=  ABSENT | NC_VARIABLE   nelems  [var ...]

ABSENT   := ZERO  ZERO      // 配列が無いことを意味する (nelems == 0 に同じ).

nelems   := NON_NEG         // 以下の並ぶ要素の数

dim      := name  dim_size

name     := string

dim_size := NON_NEG         // もし 0 なら，これは記録次元である.
                            // 記録次元は 1 つまでである.

attr     := name  nc_type  nelems  [values]

nc_type := NC_BYTE | NC_CHAR | NC_SHORT | NC_LONG | NC_FLOAT | NC_DOUBLE

var      := name  nelems  [dimid ...]  vatt_array  nc_type  vsize  begin
                            // nelems は変数のランク (次元数) である.
                            // スカラーなら 0，ベクトルなら 1，行列なら 2，...

vatt_array :=  att_array  // 変数を特定した属性

dimid    := NON_NEG         // 変数形状のための次元 ID(dim_array へのインデックス).
                            // もし最初の次元 (のみ) が記録次元ならば,
                            // これは記録変数と呼ばれる.

vsize     := NON_NEG        // 変数サイズ. 記録変数でない場合は，その変数データ
                            // に割り当てられるバイト単位のスペースの量である.
                            // この数は，次元サイズの積に型のサイズを掛けたものを
                            // 4 バイトの境界に合せて詰め込んだものである.
                            // もし記録変数なら，これは各記録あたりのスペースの量
                            // である. netCDF"記録サイズ"は記録変数の vsize の和と
                            // して計算される.

begin    := NON_NEG         // 変数のスタート位置. この変数データの先頭の
```

```
                                    // ファイル中におけるバイト単位のオフセット.

data     := non_recs  recs

non_recs := [values ...]  // 記録変数でない最初の変数, 2 番目の変数, ... のデータ

recs     := [rec ...]     // 最初の記録, 2 番目の記録, ...

rec      := [values ...]  // 記録 n に対する最初の記録変数, 記録 n に対する 2 番目の
                          // 記録変数, ... のデータ.
                          // 特別の場合については以下の注意を参照.

values   := [bytes] | [chars] | [shorts] | [ints] | [floats] | [doubles]

string   := nelems  [chars]

bytes    := [BYTE ...]  padding

chars    := [CHAR ...]  padding

shorts   := [SHORT ...]  padding

ints     := [INT ...]

floats   := [FLOAT ...]

doubles  := [DOUBLE ...]

padding  := <次の 4 バイト境界までの 0, 1, 2, または 3 バイト>
                         // ヘッダでは, 詰め込むのは 0 のバイトである.
                         // データでは, 詰め込むのは変数のフィル値である.

NON_NEG  := <非負の値を持つ整数>

ZERO     := <0 の値を持つ整数>

BYTE     := <8 ビット byte>

CHAR     := <8 ビット ACSII/ISO でコード化された character>

SHORT    := <16 ビット符号つき, ビッグエンディアン, 2 の補数表現>

INT      := <32 ビット符号つき, ビッグエンディアン, 2 の補数表現>

FLOAT    := <32 ビット IEEE 単精度浮動小数点, ビッグエンディアン>

DOUBLE   := <64 ビット IEEE 単精度浮動小数点, ビッグエンディアン>

// タグは 32 ビット整数である.
NC_BYTE      := 1         // データは 8 ビットの符号つき整数の配列
NC_CHAR      := 2         // データは文字の配列, すなわちテキスト
NC_SHORT     := 3         // データは 16 ビットの符号つき整数の配列
NC_LONG      := 4         // データは 32 ビットの符号つき整数の配列
NC_FLOAT     := 5         // データは IEEE 符号つきの単精度浮動小数点の配列
NC_DOUBLE    := 6         // データは IEEE 符号つきの倍精度浮動小数点の配列
NC_DIMENSION := 10
NC_VARIABLE  := 11
NC_ATTRIBUTE := 12
```

ファイルオフセットの計算

指定されたデータ値のオフセット (ファイル中の位置) を計算するために, 指定された変数に対する適当な型のデータ値 1 つの外部サイズ (バイト単位) を *external_sizeof* とする. 各 *nc_type* に対する *external_sizeof* は以下のようである:

`NC_BYTE`	1
`NC_CHAR`	1
`NC_SHORT`	2
`NC_LONG`	4
`NC_FLOAT`	4
`NC_DOUBLE`	8

open() (または endef()) においては, まえもって *var_array* で示された変数の並びが調べられ, *recsize* を計算するために, “記録” 変数の *vsize* を合計する.

変数の次元サイズの積を右から左にとっていき, 記録変数に対しては最も左の (記録) 次元はとばして, その結果を各変数について *product* 配列に格納する. 例えば:

非記録変数:
```
        dimension sizes:        [  5  3  2 7]
        product:                [210 42 14 7]
```

記録変数:
```
        dimension sizes:        [0  2  9 4]
        product:                [0 72 36 4]
```

この段階において, 最も左の積を次の 4 の倍数に丸めたものが変数サイズ, すなわち, 上の文法における *vsize* である. 例えば, 上記の非記録変数に対しては, *vsize* フィールドの値は 212 (210 を 4 の倍数に丸めたもの) である. 記録変数に対しては, *vsize* の値はちょうど 72 である. なぜなら, これはすでに 4 の倍数であるからである.

求めるデータ値の座標の配列を *coord* とし, 求める結果を *offset* とする. このとき, *offset* は単に, 求める変数の最初のデータ値のオフセット (その変数の *begin* フィールド) に, *coord* および *product* ベクトルの内積を変数の各データのサイズ (バイト単位) 倍したものを加えたものになる. 最後に, 変数が記録変数である場合には, 記録数 (‘`coord[0]`’) と記録サイズ (*recsize*) の積が加えられ, それが最終的な *offset* 値になる.

offset を計算する C 的なコードは以下のようになる:

```
for (innerProduct = i = 0; i < var.rank; i++)
        innerProduct += product[i] * coord[i]
offset = var.begin;
offset += external_sizeof * innerProduct
if(IS_RECVAR(var))
        offset += coord[0] * recsize;
```

そこで, (外部表現の) データ値を取得するには,

```
lseek(fd, offset, SEEK_SET);
read(fd, buf, external_sizeof);
```

特別な場合: 記録変数が 1 つしかない場合には, 各記録が 4 バイト境界に合っていなければならないという制約が無いので, この場合は記録の詰め込みは行われない.

例

上の文法を使うと, 次元, 変数, 属性を持たず, 従ってデータも持たない妥当な netCDF ファイルで最も小さいものが導かれる. この空の netCDF ファイルの CDL 表現は以下のようになる.

```
netcdf empty { }
```

この空 netCDF ファイルは 32 バイトの大きさがあり, これは ‘`ncgen -b empty.cdl`’ を使って, CDL 表現から空 netCDF ファイルを生成して確認することができる. この空 netCDF ファイルは, 4 バイトの “マジックナンバー” から始まり, これはそれが netCDF バージョン 1 ファイルと認識するためのものである. このマジックナンバーは: ‘`C`’,‘`D`’,’`F`’,‘`\001`’ である. その次には, 32 ビットの 0 が 7 個の続く. これは以下のことを表している: 記録数が 0, 次元の並びが空, グローバル属性が空, および変数の並びが空.

以下は, Unix コマンド `od -xcs empty.nc` を使って, ビッグエンディアンのマシン上において生成されたファイルの (編集された) ダンプを表示したものである.

ファイルは, 16 バイトの各部分が 4 行で表示されている. 第 1 行はバイトを 16 進数表示している. 第 2 行はバイトを文字表示している. 第 3 行は 2 バイトごとをグループ化して, それを符号つき 16 ビット整数として表示している. 第 4 行 (手作業で追加されたものである) は, netCDF 要素および値としてバイトを解釈して表示したものである.

```
   4344    4601    0000    0000    0000    0000    0000    0000
  C   D   F 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
  17220   17921   00000   00000   00000   00000   00000   00000
[magic number ] [  0 records  ] [  0 dimensions   (ABSENT)    ]

   0000    0000    0000    0000    0000    0000    0000    0000
 \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
  00000   00000   00000   00000   00000   00000   00000   00000
[  0 global atts  (ABSENT)    ] [  0 variables    (ABSENT)    ]
```

もう少し意味のある例として, 以下の CDL を考える:

```
netcdf tiny {
dimensions:
        dim = 5;
variables:
        short vx(dim);
data:
        vx = 3, 1, 4, 1, 5 ;
}
```

これは 92 バイトの netCDF ファイルに対応する. 以下はこのファイルの編集されたダンプである:

```
   4344    4601    0000    0000    0000    000a    0000    0001
  C   D   F 001  \0  \0  \0  \0  \0  \0  \0  \n  \0  \0  \0 001
  17220   17921   00000   00000   00000   00010   00000   00001
[magic number ] [  0 records  ] [NC_DIMENSION ] [ 1 dimension ]

   0000    0003    6469    6d00    0000    0005    0000    0000
 \0  \0  \0 003   d   i   m  \0  \0  \0  \0 005  \0  \0  \0  \0
  00000   00003   25705   27904   00000   00005   00000   00000
[  3 char name = "dim"        ] [ size = 5    ] [ 0 global atts

   0000    0000    0000    000b    0000    0001    0000    0002
 \0  \0  \0  \0  \0  \0  \0 013  \0  \0  \0 001  \0  \0  \0 002
  00000   00000   00000   00011   00000   00001   00000   00002
 (ABSENT)     ] [NC_VARIABLE  ] [ 1 variable  ] [ 2 char name =

   7678    0000    0000    0001    0000    0000    0000    0000
  v   x  \0  \0  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0  \0
  30328   00000   00000   00001   00000   00000   00000   00000
 "vx"         ] [1 dimension  ] [ with ID 0   ] [ 0 attributes

   0000    0000    0000    0003    0000    000c    0000    0050
 \0  \0  \0  \0  \0  \0  \0 003  \0  \0  \0  \f  \0  \0  \0   P
  00000   00000   00000   00003   00000   00012   00000   00080
 (ABSENT)     ] [type NC_SHORT] [size 12 bytes] [offset:    80]

   0003    0001    0004    0001    0005    8001
 \0 003  \0 001  \0 004  \0 001  \0 005 200 001
  00003   00001   00004   00001   00005  -32767
[    3] [    1] [    4] [    1] [    5] [fill ]
```

Appendix C FORTRAN インターフェースのまとめ

入力引数を大文字で, 出力引数を小文字で表記している. すべての引数の FORTRAN 型のリスト (引数のファベット順) は, サブルーチンおよび関数の宣言の下に付けてある.

```
INTEGER FUNCTION NCCRE (FILENAME,CLOBMODE, rcode)
INTEGER FUNCTION NCOPN (FILENAME,RWMODE, rcode)
SUBROUTINE NCREDF (NCID, rcode)
SUBROUTINE NCENDF (NCID, rcode)
SUBROUTINE NCCLOS (NCID, rcode)
SUBROUTINE NCINQ (NCID, ndims,nvars,natts,recdim,rcode)
SUBROUTINE NCSNC (NCID, rcode)
SUBROUTINE NCABOR (NCID, rcode)
INTEGER FUNCTION NCDDEF (NCID,DIMNAME,SIZE, rcode)
INTEGER FUNCTION NCDID (NCID,DIMNAME, rcode)
SUBROUTINE NCDINQ (NCID,DIMID, dimname,size,rcode)
SUBROUTINE NCDREN (NCID,DIMID,DIMNAME, rcode)
INTEGER FUNCTION NCVDEF (NCID,VARNAME,DATATYPE,NVDIMS,VDIMS, rcode)
INTEGER FUNCTION NCVID (NCID,VARNAME, rcode)
SUBROUTINE NCVINQ (NCID,VARID, varname,datatype,nvdims,vdims,nvatts,rcode)
SUBROUTINE NCVPT1 (NCID,VARID,INDICES,VALUE, rcode)
SUBROUTINE NCVP1C (NCID,VARID,INDICES, CHVAL, rcode)
SUBROUTINE NCVGT1 (NCID,VARID,INDICES, value, rcode)
SUBROUTINE NCVG1C (NCID,VARID,INDICES, chval, rcode)
SUBROUTINE NCVPT (NCID,VARID,START,COUNTS,VALUE, rcode)
SUBROUTINE NCVPTC (NCID,VARID,START,COUNTS,STRING,LENSTR, rcode)
SUBROUTINE NCVPTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,VALUE, rcode)
SUBROUTINE NCVPGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,STRING,rcode)
SUBROUTINE NCVGT (NCID,VARID,START,COUNTS, value,rcode)
SUBROUTINE NCVGTC (NCID,VARID,START,COUNTS, string,LENSTR,rcode)
SUBROUTINE NCVGTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,value,rcode)
SUBROUTINE NCVGGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,string,rcode)
SUBROUTINE NCVREN (NCID,VARID,VARNAME, rcode)
SUBROUTINE NCAPT (NCID,VARID,ATTNAME,DATATYPE,ATTLEN,VALUE, rcode)
SUBROUTINE NCAPTC (NCID,VARID,ATTNAME,DATATYPE,LENSTR,STRING, rcode)
SUBROUTINE NCAINQ (NCID,VARID,ATTNAME, datatype,attlen,rcode)
SUBROUTINE NCAGT (NCID,VARID,ATTNAME, value,rcode)
SUBROUTINE NCAGTC (NCID,VARID,ATTNAME, string,LENSTR,rcode)
SUBROUTINE NCACPY (INNCID,INVARID,ATTNAME,OUTNCID,OUTVARID, rcode)
SUBROUTINE NCANAM (NCID,VARID,ATTNUM, attname,rcode)
SUBROUTINE NCAREN (NCID,VARID,ATTNAME,NEWNAME, rcode)
SUBROUTINE NCADEL (NCID,VARID,ATTNAME, rcode)
INTEGER FUNCTION NCTLEN (DATATYPE, rcode)
SUBROUTINE NCPOPT (NCOPTS)
SUBROUTINE NCGOPT (ncopts)
INTEGER FUNCTION NCSFIL (NCID,FILLMODE, rcode)

INTEGER ATTLEN          ! 属性ベクトル中の要素数
CHARACTER*(*) ATTNAME   ! 属性名
INTEGER ATTNUM          ! 属性数
CHARACTER CHVAL         ! 変数または属性の文字値
INTEGER CLOBMODE        ! NCCLOB または NCNOCLOB
INTEGER COUNTS(NVDIMS)  ! 値のブロックの縁の長さ
INTEGER DATATYPE        ! 型: NCBYTE, ..., または NCDOUBLE
INTEGER DIMID           ! 次元 ID
CHARACTER*(*) DIMNAME   ! 次元名
CHARACTER*(*) FILENAME  ! netCDF ファイル名
INTEGER FILLMODE        ! フィルモードの設定: NCNOFILL または NCFILL
INTEGER IMAP(NVDIMS)    ! インデックスマッピングベクトル
INTEGER INDICES(NDIMS)  ! 変数の単一要素の座標
INTEGER INNCID          ! 入力 netCDF ID
INTEGER INVARID         ! 入力変数 ID
INTEGER LENSTR          ! 文字配列の長さ
```

```
INTEGER NATTS           ! グローバル属性の数
INTEGER NCID            ! NCCREまたはNCOPNで返されるnetCDF ID
INTEGER NCOPTS          ! エラー処理オプションのフラグ
INTEGER NDIMS           ! 次元数
CHARACTER*(*) NEWNAME   ! 新規の属性名
INTEGER NVARS           ! 変数の数
INTEGER NVATTS          ! 変数に割り当てられている属性の数
INTEGER NVDIMS          ! 変数中の次元数
INTEGER OUTNCID         ! 出力netCDF ID
INTEGER OUTVARID        ! 出力変数ID
INTEGER RCODE           ! 返されるエラーコード．エラーがなければ0.
INTEGER RECDIM          ! 無制限次元のID
INTEGER RWMODE          ! NCWRITEまたはNCNOWRIT
INTEGER SIZE            ! 次元サイズ
INTEGER START(NVDIMS)   ! アクセスされる最初の値のインデックス
INTEGER STRIDE(NVDIMS)  ! netCDF変数次元のストライド
CHARACTER*(*) STRING    ! 変数または属性の文字配列値
DOUBLE VALUE            ! 変数または属性の倍精度値
REAL VALUE              ! 変数または属性の実数値
INTEGER VALUE           ! 変数または属性の整数値
INTEGER VARID           ! NCVDEFまたはNCVIDから得られる変数ID，またはNCGLOBAL
CHARACTER*(*) VARNAME   ! 変数名
INTEGER VDIMS(NDIMS)    ! 変数の次元ID．変数の形を与える．
```