Short Answers to Participants' Requests and Questions

ご参加の皆様から事前に頂いたご質問･ご要望のすべてに対して、招待講演者の John Reid 教授と Steve Lionel 氏から回答を頂きました。

凡例

Qn	① 頂いたご質問･ご要望	② J. Reid と S. Lionel への質問･要望（HPFPC による英訳）
An	④ J. Reid と S. Lionel からの回答（HPFPC による和訳）	③ J. Reid と S. Lionel からの回答（英語）

1. 精度

Q1	任意に精度を指定できる数値の型とそれらの間の演算や入出力、基本的な数学関数のサポート。 [HPFPC からのコメント] Fortran 規格では，コンパイラは，任意の精度の種別型パラメタをサポートすることができます。	Support for arbitrary precision numeric types and manipulations on them such as arithmetic operations, I/O, and mathematical functions. [COMMENT by HPFPC] The Fortran standard allows compilers to support kind type parameters for arbitrary precision numeric types.
A1	ご要望は，プログラマが，コンパイラがサポートしている精度に限らず，任意の精度を使用できるようにすることだと思います。これは，規格の大幅な拡張となるでしょう。といいますのは，全ての組込み手続が，任意の精度をサポートしなければならないことになるからです。 Fortran 規格の最も重要な役割は，通常の精度において，効率的な実行ができるようにすることであり，このような拡張は，規格の範囲外だと考えています。そのような機能を提供するパッケージソフトを使用するのが良いでしょう。	We think the question refers to allowing the programmer to ask for any precision, not just those that the compiler chooses to support. It would be a big addition to the Standard because all its intrinsic procedures would need to support arbitrary precision. We see this as outside the main role of Fortran, which is to provide efficient execution at the usual precisions. There are packages that provide such a feature.

Q2	4 倍精度用の組み込み関数の整備。たとえば DPROD で、倍精度と倍精度の乗算で 4 倍精度の積を得ることを可能にするなど。DPROC(DX,DY, KIND=16)	Support for intrinsic functions for quadruple precision. For instance, the function dprod(dx, dy, kind=16) returns a quadruple precision value as the product of the double precision values dx and dy.
A2	規格では，コンパイラは，少なくとも 2 種類の精度の実数型を提供する必要があります。基本実数型は，少なくとも 6 桁の十進精度をもつことが推奨されており，倍精度実数型は，少なくとも 10 桁の十進精度をもつことが要求されています。また，多くのコンパイラは，実質的に 4 倍精度であるような，より精度の高い実数型も提供しています。最近では，半精度実数型への需要もあります。規格では，処理系は，サポートする各実数型に対して，組込み関数を含む全ての機能をサポートする必要があります。ご要望の”dprod”と同等の効果は，(現行の規格の範囲で)次のように書けば得られます。 integer:: quad = selected_real_kind(precision(x)+1) real(x,quad)*real(y,quad)	The standard requires that at least two precisions of reals be provided, default real with recommended decimal precision at least 6 and double precision with required decimal precision at least 10. Many compilers also provide extended precision reals, which is essentially quadruple precision. Recently there has also been a demand for half-precision reals. For each kind of real that an implementation supports, it is required by the standard to support it fully, including the intrinsic functions. The effect of the dprod that you want is available as Integer :: quad = selected_real_kind(precision(x)+1) real(x,quad)* real(y,quad).

2. データ型

Q3	区間演算のサポート。	Support for interval arithmetic (that is, each value is represented as a range of rounding errors and measurement errors in computation to yield reliable results).
A3	区間演算は，Fortran 2003 策定時に議論されましたが，規格に導入するには大きすぎると考えられました。その代わり，区間演算のための派生型が書けるようになる機能が追加されました。これには，算術演算の丸めの制御 及び 入出力における丸めの制御が含まれます。	This was considered for Fortran 2003, but was considered too big to include in the Standard. Instead, features were added that would allow a derived type for interval arithmetic to be written, including control over rounding of arithmetic operations and rounding during input/output.

Q4	Python のような、型が固定されていない変数も認めたらどうか（ただし変数の宣言はする）。そうして Python にあるような、リスト、辞書、タプル、集合の提供（それらについては実行効率が悪くても目をつぶるものとして）。	Support for variables without type declaration and high-level data types such as lists, tuples, and sets like ones in Python. Poor performance would be tolerable for them.
A4	Fortran 規格の最も重要な役割は，効率的な実行を実現することであり，ご要望の機能は，その範囲外だと考えています。Python は，Fortran コードのフロントエンドとして使用できます。	We see this as outside the main role of Fortran, which is to provide efficient execution. Python can be used as a front-end to Fortran code.

Q5	マルチバイト文字のサポート。 [HPFPC からのコメント] Fortran 規格では，コンパイラは，処理系依存の文字集合を用意できます。次のコードは，Fortran 規格書中に記載されている例です： NIHONGO_'彼女なしでは何もできない。' ここで，NIHONGO は，日本語文字を示す処理系依存の種別型パラメタ値をもつ名前付き定数です。	Support for multibyte characters. [COMMENT by HPFPC] The Fortran standard allows compilers to support processor dependent character sets. The following example is from the Fortran standard: NIHONGO_'彼女なしでは何もできない。' where NIHONGO is a named constant whose value is the kind type parameter for processor dependent Japanese characters.
A5	HPFPC のコメントに同意します。 フリーディスカッション時に挙がった別の話題として，識別子にマルチバイト文字を使用できないか，というご質問がありました。これは，言語規格とコンパイラに広範囲に渡る影響を及ぼします。プログラマが，母語で書かれた識別子を使用したい，ということは理解できますが，これを実現するためには，ベンダーによる大きな投資が必要となり，他の要望との間で取捨選択が必要となると思います。	We agree. Separately, during the Free Discussion, it was also asked if the language could allow multibyte characters in identifiers. This would have far-ranging effects in the language and compilers, and while we understand that programmers might prefer to name identifiers in their native language, we believe this would need a big investment by vendors and have to compete with other demands.

3. 非数の扱い

Q6	非数をきちんと扱えるようにする。定数としても書けるし、条件分岐等でも非数の場合も含めて抜けがなく扱えるように。論理値も TRUE と FALSE だけでなくて、非決定（UNDETERMINED）が要るかもしれない。 [HPFPC からのコメント] Fortran 規格の IEEE 機能では，NaN は，型 IEEE_CLASS_TYPE の名前付き定数 IEEE_SIGNALING_NAN 及び IEEE_QUIET_NAN として定義されており，それらに対する演算子 == 及び /=も定義されています。コンパイラは，この機能をサポートすることを選択できます。	Support for Not-a-Number, including constants and use in conditional branches. The logical value .UNDETERMINED. might be needed in addition to .TRUE. and .FALSE. [COMMENT by HPFPC] The Fortran IEEE feature defines Not-a-Number as the named constants IEEE_SIGNALING_NAN and IEEE_QUIET_NAN of type IEEE_CLASS_TYPE and the operators == and /= on them, which compilers can optionally support.
A6	組込みモジュール IEEE_ARITHMETIC 中で，NaN に相当する値が提供されています。	The intrinsic module IEEE_ARITHMETIC provides support for NaN values.

4. 相互利用可能性

Q7	C から fortran で定義した拡張型（class）を使える日は来るのでしょうか？	To treat Fortran "Class" in C++, will WG5 (or such organizations) define C++ bindings as new Fortran Standard?
A7	C との相互利用の機能は使用できますが，現在のところ，それ以上の計画はありません。	The interoperability with C features are available. There are no current plans to go beyond this.

5. データレイアウト

Q8	配列で行優先や列優先を（デフォルト列優先で）指定可能にできないか。 REAL,ROWMAJOR:: A(100,50) のように。あるいは 3 次元以上でも記憶が連続的になる添字の順序を指定できるように。	Specification of the array element order such as row major or column major (column major by default): real, ROWMAJOR :: a(100,10) More generally, specification of order of dimensions in multidimensional arrays: real, ORDER(3,1,2) :: a(100,1000,10) ! 3rd, 1st, and then 2nd axis
A8	**John Reid の回答：** 私自身も Fortran ユーザーであり，この要望には共感します。解きたい問題を表現するようコードを書きたいですし，性能のために添字の順序を変更したくはありません。さらに，最も良い添字順序は，ハードウェアによっても異なる可能性があります。ご提案になった構文も良いと思います。 **Steve Lionel の回答：** ご要望の機能によって，配列要素順序が，プログラム単位間で異なるとすると，アプリケーションの保守が複雑になってしまうのではないかと思います。入出力処理や形状引継ぎ配列だけでなく，MATMUL のような配列演算も複雑になるでしょう。 **別の人の回答：** 添字の順序を(使用する場所とは)遠い場所で定義するのは良い考えではありません。 **注：** 今回の WG5 会議において，(メンバーの)Tom Clune から，この種の機能への強い要望があり，彼は，ご要望の機能を気に入っていました。テンプレート機能を，次次期の規格改定に組み込むために，検討を今から開始することが決議されました。テンプレート機能は，ご要望に応える代替手段を提供することでしょう。	**Reply from John Reid:** I am a Fortran user and have sympathy with this request. I want the code to represent the problem I am solving and do not want to have to alter the subscript order for the sake of performance. Furthermore, the best ordering may vary according to the underlying hardware. I like your proposed syntax. **Reply from Steve Lionel:** I believe that such a feature would complicate maintenance of an application, if the element order was different among various program units. It would complicate certain array operations, such as MATMUL, as well as certain I/O operations and assumed-shape arrays. **Reply from others:** It is not a good idea for the subscript ordering to be defined remotely. **Note:** At the WG5 meeting, there was a strong request for something of this kind from Tom Clune and he liked this idea. It was decided that work should start soon on templates for inclusion in the standard after the next. Templates should provide an alternative way to meet this request.

Q9	さらに一般的には式の上ではあたかも配列参照のように X(I,J)と書いたときに I と J から一般的な写像を定義して要素への参照法が定義できるような記述を許せば（これはマクロを介すればできるが、関数や組み込み関数では左辺値が作れず不可能）。	Support for user-defined mapping from a subscript-list to an index. For example, when mapping from (i,j) to index(i,j) is defined by a user, the description "x(i,j)" in a code indicates an array element "x(index(i,j))", which can be used as an L-value (definable). (A function "x" that returns the value of an array element cannot be used as an L-value.)
A9	現在の書式はこれに十分であるように思われます。そして、何が起こるかは仕様書を読む人に警告しています。 [HPFPC からの事後のコメント]質問の意図が伝わっていないかもしれません。	The present syntax seems to us to be adequate for this and does alert the reader to what is happening.

6. 構文

Q10	BLOCK 文について詳しく話を伺いたいです。 ブロック構造をちゃんと入れて、ブロックに局所的な変数を使えば、OPENMP などの記述もより美しくできるであろう。 [HPFPC からのコメント]Fortran 2008 以降では，BLOCK 構文が導入されており，BLOCK 構文内で局所的な変数を宣言できるようになりました。	I would like to hear more about the BLOCK statement. Support for block constructs so variables local to the constructs can be used. This feature will make the description of OpenMP directives easer. [COMMENT by HPFPC] The BLOCK construct is defined in Fortran 2008 or later, in which variables local to the construct can be declared.
A10	HPFPC のコメントに同意します。 ご要望の機能は，現在の規格では使用できるようになっています。	We agree. It is available now.

Q11	内部関数はなぜ多重化できないことにしているのだろうか？	Is there any reason why internal procedures cannot be nested?
A11	ご要望の機能は，処理系の実装にとっては負担になり，コードの保守もより難しくなるでしょう。入れ子内部にある手続は，最上位のレベルに移動し，親子結合で参照されていた変数は，引数として追加するのがより良いコーディングだと考えます。	There would be a burden on the implementation and the code would be less easy to maintain. We think it is better coding practice to move nested procedures to the top level and add extra arguments for variables that would have been accessed by host association.

7. 並列化

Q12	OpenMP のような並列記述をディレクティブとせずに、言語そのものの構文として並列化構文を取り入れた方が良くないだろうか？ [HPFPC からのコメント]Fortran 2008 以降では，DO CONCURRENT 構文や coarray 機能のような並列化のための構文が導入されています。	Introduction of parallel constructs as part of the language rather than directives outside the language such as OpenMP. [COMMENT by HPFPC] Some parallel constructs such as DO CONCURRENT and coarray features are defined in Fortran 2008 or later.
A12	HPFPC のコメントに同意します。	We agree.

Q13	GPGPU への対応はあるのか？ CPU を大きく増やさずに、GPU を利用して並列計算を行いたい。	Will Fortran support GPGPU? I would like to use GPUs to perform parallel computation, instead of using a lot of CPUs.
A13	ISO Fortran 規格委員会は，特別なハードウェアをサポートするための言語を策定することについては，とても慎重な態度をとっています。なぜなら，言語の寿命は非常に長く，利用者は自分のコードが動作し続けるよう望むので，一旦策定された言語の機能が削除されることは少ないからです。DO PARALLEL (DO CONCURRENT のこと？)で書かれたコードが，可能な場合には GPU 上で実行されると良いと思います。	The committee is very wary of shaping the language towards support of special hardware because the language has a very long life – very little is ever removed because users want their code to continue to execute. It is hoped that code in a do parallel construct will use GPUs when appropriate.

Q14	Cuda のように NVIDIA 社の GPU に特化したような機能は追加することはできるのか？	Is it available to support features for GPU specific to a certain vendor such as CUDA?
A14	規格の目的は，可搬性を促進することにあります。特定のベンダーをサポートすることは，適切とは言えません。OpenMP や OpenACC のような他の規格が，GPGPU･アクセラレータ向けのプログラミングをサポートしており，それらは複数のベンダーがサポートしています。特に，CUDA は，PGI 社が「CUDA Fortran」を提供しています。GPGPU の能力を利用する別の手段として，OpenCL があり，これも広く利用可能になっています。	The intention of the standard is to promote portability. It is not appropriate for it to support a particular vendor. Outside standards such as OpenMP and OpenACC support GPGPU/Accelerator programming and are supported by multiple vendors. For CUDA in particular, PGI offers a “CUDA Fortran” product. OpenCL is another method of accessing GPGPU capabilities and is widely available.

8. I/O 拡張

Q15	I/O の対象としてファイルを拡張して、たとえば URL を指定して Open してネットワーク上の資源に対して読み書きを行えるようにしたらどうなるか？	Support for I/O features on network resources, for example by specifying a URL in an OPEN statement.
A15	大きな I/O の機能が必要のように思われ，一般には受け入れられないかもしれません。なぜなら，利用者は，既存のパッケージソフトを使用するために大きな投資をしているからです。ファイルで指定できるアクセス場所に，ネットワークロケーションを含むオペレーティングシステムもあります。言語規格としての Fortran は，OPEN 時にファイル仕様を指定する構文のような，プラットフォームに依存する機能は取り扱いません。	A large I/O feature would be seen as needed and might not be widely adopted because users have a big investment in use of existing packages. Some operating systems include network locations in their file specification access. Fortran as a language does not get involved in platform-dependent features such as the syntax of a file specification in OPEN.

Q16	MPI の機能のようなものは、Fortran I/O の拡張によっても実現できるのではないか。 たとえばランク（番号）は Fortran I/O の UNIT 番号のようなものとすれば。	Extension to I/O features for communication like MPI, for example by specifying a process number (rank in MPI) as a unit number in I/O statements.
A16	Coarray を使用して，ご要望になっているような通信が記述できます。	Coarrays provide such communication.

9. コンパイラ最適化

Q18	C++11 の alignas 指定子や alignof 演算子のような，データのアラインメントに関するサポート	Support of features for data alignment like the alignas specifier and the alignof operator of C++11.
A18	Fortran では，ご要望の機能が必要だとは考えていません。コンパイラは，派生型変数の格納方法を自由に選択して，性能を向上させることができます。	We do not see the need for this in Fortran. Compilers have the freedom to choose the way they store variables of derived type and use it to enhance performance.

Q19	実行時に割り付けられる変数の（部分的な）形状をコンパイラに教えるため、割付け変数の宣言の書式を拡張する： real, allocatable:: a(20,30,:) ! declare the (partial) shape here ... allocate a(20,30,100) ! or a(*,*,100)	Extension of the allocatable variable declaration to tell compilers a (partial) shape to be allocated: real, allocatable:: a(20,30,:) ! declare the (partial) shape here ... allocate a(20,30,100) ! or a(*,*,100)
A19	変わらない部分を宣言するために定数を使用すれば，ご要望になっている効果を得ることができます。	The effect is already available by using constants to declare the fixed parts.

Q20	ソースコード上のある指定した範囲内での最適化（演算順序の変更など）を明示的に禁止指定できる機能（たとえばブロック構文のオプションとして）。	Specification of code segments where optimizations such as change of operation orders are explicitly prohibited.
A20	規格では，文の並び替えが結果を変えてはならない，と定められています。	The standard already requires that statement reordering does not alter the results.

Q21	あるいは組み込み関数 NOOPT でもって、NOOPT(式）とすると、”式”の部分はコードの最適化をしないとするなど。	Or an intrinsic function NOOPT(expr), where optimizations in computation of the expression "expr" are prohibited.
A21	規格では，式の演算は，括弧に従った順序で行われなければならない，と定められています。	The standard already requires that parentheses within expressions are respected.

10. 組込み手続とライブラリ

Q22	FORTRAN は数値計算のための言語、とされている。しかし驚くべきことに、組込数学関数としては未だに初等関数しかサポートしていない。「各種超越関数は過去の資産のソースコードを用いればよい」などという考えは数値計算のためのコンピュータ言語という設計基本と矛盾している。最低限としてもベッセル関数群と楕円関数群は含まれなければなるまい。精度の問題は、別途明示すればよいはずである。さらに行列解法の標準コールがない。	FORTRAN is said to be a language for numerical computations, but only elementary mathematical functions are supported as the intrinsic function. More mathematical functions, such as Jacobi elliptic functions should be supported.
A22	一般的には必要とされない関数については，専用のライブラリを使用したほうがいいと考えています。処理系の実装にとって負担となるからです。	We feel that it is better to use a specialized library for functions that are not widely required. Adding them to the standard is a burden on all implementations.

Q23	未だに高々3 次元のソルバーを吐き出し法などで書かねばならない状況にユーザーはウンザリさせられていると思う。ただ大次元のソルバーは精度保証が出来ないから言語仕様には適さないかもしれない。そこで例えば 10 次元までは公式にサポートするとし、さらに 10 次元から 10 万次元さらに数億次元へスケーラブルに対応できるサブルーチン CALL の具体的な手続きを言語仕様内に定めることが重要である。以上の 2 点を欠いていては数値計算用言語とは言えないと考える。	Furthermore there is no standard library for the matrix solution. Even for 3 order matrix, users must write the code using Gaussian elimination method. Of course, large-order solvers may not be suitable for language specifications because accuracy cannot be guaranteed. But it is important to define the concrete procedure of the library call for matrix solutions in the language specification.
A23	ご要望の機能については，利用者はライブラリの使用で満足していると思われます。	Users seem to be content to use libraries for this.

Q24	データの可視化は別ライブラリが必要で、だから異なったシステム間での可視化のポータビリティがない。加えて可視化ライブラリー群が開発者の廃れてゆくことでコードが使えなくなってゆく。むしろこのような現状から可視化は備えないのかもしれないが、数多くの可視化ライブラリーが FORTRAN BINDING をもっていて、 それは需要の高さに他ならない。継続的なコードの活用のため、何らかの可視化標準手続きを備えてはどうだろうか。	Data visualization requires a separate library, so there is little portability of visualization between different systems. In addition, legacy visualization libraries are not maintained, and the code that uses them will become unusable. Visualization is high demand, and some kind of visualization standard procedure as the language specification should be useful for continuous use and development of the code.
A24	大きな機能が必要となり，一般には受け入れられないかもしれません。というのは，利用者は，既存のパッケージソフトを使用するために大きな投資を行っているからです。	A large feature would be needed and might not be widely adopted because users have a big investment in use of existing packages.

11. 概念的な問題

Q25	FORTRAN コンパイラが広くまた長く使われ続けるためには、大規模計算だけでなく小規模計算にも簡単に使える仕様であることが、非常に大切だと思う。例えば、実験データ解析を 30 行程度の短いプログラムで　データ読み込み、最小 2 乗法でのデータ近似、可視化まで行える、言わば「やりたいことがすぐにできて、かつ計算速度も一番だ」という言語仕様であることだが、この簡単さの意識に欠けていて、かつ上記のように「実は」数値計算言語としてまだまだ不親切で不十分なのが現在の FORTRAN 言語仕様だ。この簡単さに優れた MATLAB は FORTRAN をその利用者の幅広さという点で遙かに凌いでしまった。「簡単にできる」意識に沿った FORTRAN の改訂が望まれていると思います。	"Easy-to-use" functionality is important for not only large-scale computations but also small-scale calculations. For example, it is possible to write a short program of about 30 lines, which reads data, performs data analysis using the least squares method, and so on, with MATLAB, and this simplicity makes MATLAB very, very popular and familiar tool than Fortran. The language specification of Fortran should be revised to adopt this "easy-to-use" functionality.
A25	この要望には，共感する部分もありますが，Fortran の強みは，特にスーパーコンピューター上での高性能にあります。MATLAB と競争しようとするのは，間違っているでしょう。	While we are sympathetic to this wish, Fortran’s strength lies in high performance, especially on super-computers. It would be a mistake to try to compete with MATLAB.

12. モダン Fortran 勉強会からの要望・質問

Q26	implicit none の標準化 暗黙の型宣言は標準で無効とし，有効にする場合はコンパイルオプションを付けるようにしてほしい． 現代では，変数にはわかりやすい適切な名前を付けることが推奨されており，そのような状況において，変数名で型が決まる暗黙の型宣言は利用するべきではないと考えます．そのため，標準で無効とし，後方互換のためにコンパイルオプションで有効化できるようにすることが望ましいと考えます． あるいは，型のない言語のように型推論を導入し，型を右辺から決定できるようにするという道もあり得るかと思います．とにかく，名前から型が決まるという機能はなくすべきです．	The standard should make IMPLICIT NONE default. The implicit type declaration should be ineffective in default.
A26	現在動作している古いプログラムが，動作し続けるということには，大きな需要があります。ご要望の機能は，この目標と両立できないでしょう。多くのコンパイラが，ご要望の効果をもつオプションを提供しています。	There is a huge demand for old working programs to continue to work and this would not be compatible with this aim. Many compilers provide an option to have this effect.
Q27	言語として物理定数を内包してほしい 数値計算向けの言語のはずなのに，π を毎回定義する，あるいは定義してあるモジュールを読み込むのは手間です．いくつかの定数を言語として内包し，可能であれば，内部関数のように何も use しなくても参照できると助かります．そのような機能が Fortran にないのであれば，次の規格で導入していただきたい．	The standard should include some physical constants such as pi (π). It is more helpful if they can be used without using any modules, similarly to intrinsic procedures.
A27	大きな拡張ではありませんが，そのような要望はこれまでありませんでした。	This would not be a big addition to the language, but it has not been requested.

Q28	iso_fortran_env は標準で use しておいてほしい Fortran は大量の方言を有し，その中でも変数の型宣言はひどい有様です．2008 で int32 や real32 等のパラメータが導入された事は（幾分遅いように感じますが）喜ばしい限りです．しかし，Fortran を初心者に教える際，導入の段階で use,intrinsic :: iso_fortran_env が登場することは，悪名高い C 言語の #include<stdio.h> //おまじない と同じ状況を生みます．上の要望とも関係しますが，iso_fortran_env は標準で use された状態になっていることが望ましいと考えます．	The intrinsic module ISO_FORTRAN_ENV should be used by default. This will promote portability in kind type parameters such as int64 and real128, which have a lot of dialects.
A28	組込みモジュール ISO_FORTRAN_ENV 中では，極めて多くの機能が定義されており，利用者は，USE 文の ONLY 句を使用して，必要な機能を選択できます。ご要望は，現在動作している古いプログラムが，動作し続ける，という目標と相容れないでしょう。さらに，REAL128 のような種別型パラメタ定数は，多くの人が要望している可搬性という利点を提供できません。なぜなら，マシン上の表現ではなく，ビット幅を指定しているに過ぎないからです。	There are a significant number of features defined in ISO_FORTRAN_ENV and user can be selective with USE ONLY. This suggestion would not be compatible with the aim that old working programs should continue to work. In addition, the kind type constants such as real128 do not provide the portability benefits many claim, as they denote only storage size, not machine properties.

Q29	unsigned な変数を取り扱いたい iso_c_binding で C 言語との相互利用は高いレベルで可能になりましたが，unsigned を Fortran で取り扱えないため，相互利用性を損なっています．また，計算の大規模化に伴って配列も大きくなる現代では，Fortran 単体でもループカウンタなど unsigned な値を利用する機会はあると考えます．	The standard should include unsigned types for interoperability with C.
A29	規格委員会は，符号なしの型を以前議論したことがありますが，その利点は，誤解されていると思います。符号なし整数型は，ほとんどの場合，算術演算のためではなく，ビット列を格納するために使用されていると思います。ビット列の使用に役立つ機能を言語に追加し続けていきます。	The committee has considered unsigned types before but feels that the benefits are misunderstood. We believe that most use of unsigned integers is to treat integers as containers of bits, not to do arithmetic, and we continue to add features to the language to assist with this.

Q30	大文字小文字の区別 Fortran は大文字小文字を区別しません．他のプログラミング言語で蓄積された適切な命名規則を流用する場合，大文字小文字を混ぜて名前を付けたとしても，それを守らなくてもコンパイルが通ってしまい，コーディングルールが容易に破られます．	Lower and upper case letters should be differentiated.
A30	ご要望は，現在動作している古いプログラムが動作し続ける，という目標と相容れないでしょう。	This suggestion would not be compatible with the aim that old working programs should continue to work.

Q31	変数名等の 31 文字制限の撤廃 変数名等には 31 文字制限があります．適切な名前を付けるにはスネークケースを用いなければならず，名前が長くなりがちです． これは多人数での開発や，プログラムを資産として正しく伝えていくためにも，この矛盾した環境を是正し，適切に名前付けできる環境が必要と考えます．	The standard should extend the maximum length of a name from 31.
A31	現在，規格では，63 文字まで許されています。	The current standard allows 63 characters.

Q32	random_number に用いるアルゴリズムの統一 Fortran の random_number は，擬似乱数生成アルゴリズムが規格で定められていないと記憶しています．それを明記してほしいと考えます．実装がコンパイラ依存だと，あるコンパイラでは非常に質のよい擬似乱数が発生できるのに，別のコンパイラに移植した途端質が悪くなるという状況が発生する可能性があります．たとえ質の悪いアルゴリズムでも，どのアルゴリズムを使っているかが統一されていれば，外部のライブラリを利用するなどの回避策が取れます．	The standard should specify the algorithm for the intrinsic function random_number() to avoid variations in quality among venders.
A32	乱数生成は，いまだ研究対象であり，ベストだと合意が得られているアルゴリズムは存在しません。組込み関数は，乱数の品質が重要ではない場面での使用が意図されています。品質が重要な場合，利用者は，その必要性に応じたパッケージソフトを使用すべきです。	This remains a subject of research and there is no algorithm that everyone agrees is best. This function is intended for use where the quality is not critical. If quality is important, the user should use a package that meets his/her requirements.

Q33	コンパイラオプションのいくつかを，プログラム内で指定できるようにしてほしい コンパイラが異なる場合，オプションを探すのがかなりの労力になるので，プログラム中で指定できるようにしてもらいたい．gfortran では，独自拡張によって open 文でエンディアンや record-marker を指定できるようになっている．	The standard should include a way of specifying compiler options in source programs. For example, gfortran supports the specification of endian and record-marker in OPEN statements.
A33	IMPLICIT NONE は，同等のコンパイラオプションが広く利用されるようになってから，規格に追加された文の一例です。他にも適切な文があるかもしれません。貴国の規格委員会で，議論されると良いと思います。	IMPLICIT NONE is an example of a statement that was added following wide use of an equivalent compiler option. Others may be appropriate. We suggest that you discuss this within your national body.

Q34	機能追加の方法の抜本的見直し Fortran の規格策定において，機能の追加＝文の追加となっているように見受けられます．これは，他言語の機能追加のスタイルとは大きく違います．このような機能追加のやり方では，新たな機能やそれに付随する機能を実現するための文が大量に追加されることになりますし，機能追加の度に新しい書き方を覚える必要があります．気軽に試して，分からなければ元のコードに戻すということもやりにくくなります． そうではなく，既存機能を整理し，既存の文に新たな機能を加えるようなやり方もあるかと思います．例えば，do に do while や do concurrent があるように，where 文を追加するのではなく do where という形式を設けるのもあり得た形だと思います．その他にも，クラス（派生型）を整理して新たな機能を提供するという方法や，他言語でいう filter や map のようにシンプルだが柔軟な機能を導入して，それらの組合せで新たな処理を実現するという方法もあり得るかと思います．template はこのような機能の代表であり，導入の要望が高い（おそらく次で導入される）ということは，ユーザの望んでいる機能のあり方と，Fortran の機能の追加のされ方が上手くかみ合っていないのではないかと感じています．	New features should be added as extensions to existing statements or as classes, not as new statements. Too many statements are added to add new features.
A34	ご要望の問題点を理解できていません。新しい文が規格に追加されるのは，それが，プログラムを理解しやすくするためにベストだと考えられる場合です。既存のプログラムが不正とならないような仕様が常に選択されています。	We do not understand the problem. New statements are added when this seems the best way to make programs easy to understand. They are always chosen to avoid making an old program invalid.