テンプレート案

テンプレートの定義（TEMPLATE構文）

- C++のテンプレートに似ているが、接頭辞ではなく構文を構成する。
  - 一つのTEMPLATE構文で、複数の派生型と手続のテンプレートを定義する。
  - TEMPLATE構文は宣言部だけに現れる。

C++の例（比較のため）	テンプレートの定義（TEMPLATE構文）の例

template **<typename T>**
void myswap(**T**& a, **T**& b) {
  T tmp = a;
  a = b;
  b = tmp;
}
	template mytmpl(T)
	  type, deferred :: T ! T は任意の型であると宣言
	  ! ここに派生型を定義（あれば）
	contains ! ここから下に手続を定義
	  subroutine myswap(a, b)
	  type(T) :: a, b, tmp
	  tmp = a
	  a = b
	  b = tmp
	  end subroutine myswap
	end template mytmpl

テンプレート案

テンプレートの具体化(INSTANTIATE文)と使用

- C++では使用時にテンプレートを具体化するのに対し、この案では具体化と使用が別々の文で行われる。
  - INSTANTIATE文は宣言部に現れ、テンプレート名とパラメタを指定して、手続などを具体化する。
  - 同じテンプレートを複数回具体化する場合、具体化された手続などの名前の衝突を避けるため、利用者がリネーミングしなければならない。（総称名を使って回避できるかもしれないが検討中）

C++の例（比較のため）	テンプレートの具体化（INSTANTIATE文）と使用の例

int x = 5, y = 10;
double a = 2.5, b = 3.7;
myswap<int>(x, y);
myswap<double>(a, b);
	instantiate mytmpl(integer), myswap_int => myswap
	instantiate mytmpl(double precision), myswap_dbl => myswap
	
	integer :: x = 5, y = 10
	double precision :: a = 2.5, b = 3.7
	call myswap_int(x, y)
	call myswap_dbl(a, b)

テンプレート案

テンプレートパラメタ

- INSTANTIATE文の引数は、以下の3種類
  - 任意の型名（＋型パラメタ）
    - 例: integer, real(8), mytype
  - 整数型、論理型、または文字型の定数（次元数は任意）
    - 例: 1+1, int(3.14), [.true., .false.], “hello”
  - 手続名
- 仮パラメタ（右例のT,C,F）はすべて構文内で宣言が必要。現仕様では：
  - 型: “deferred” としか宣言できない。CLASSの対応は検討中
  - 定数: 文字型のとき、文字長引継ぎに限る
  - 手続: 明示的引用仕様が必須

深い背景がありそうですが、彼らの意図をまだ完全には理解できていません。

```
template tada(T, C, F)
  type, deferred :: T          ! T は任意の型であることを宣言
  integer, constant :: C(..)   ! C は任意次元数の整定数であると宣言
  interface                    ! 関数 F の引用仕様を宣言
    function F(x, n) result(y)
      import T
      type(T) :: y             ! F は他のパラメタに依存してよい
      type(T), intent(in) :: x
      integer, intent(in) :: n
    end function F
  end interface

contains
  subroutine sub1(x)           ! 手続テンプレートの定義
    type(T), intent(inout) :: x
    x = F(x, SUM(C))
  end subroutine sub1

  subroutine ... end subroutine
  ...
end template tada
```

テンプレート案

その他の特徴的な仕様

- REQUIRES文とREQUIREMENT構文
  - TEMPLATE構文の仮パラメタ（特に手続名）の宣言をTEMPLATEの外に出して、TEMPLATE構文内・間で再利用する。
- TEMPLATE構文はネスト可能
  - 内側テンプレートの具体化は、外側テンプレートの中、または、外側テンプレートの具体化後
  - 内外テンプレート間で名前の親子結合がある。
- TEMPLATE構文は、単独で（具体化される前に）検証（エラーチェックなど）され、内部構造にコンパイルされる。
  - 型不明のまま翻訳し、後で型を確定させる。
  - 特に分割コンパイルの場合に意味がある。

REQUIRES文/REQUIREMENT構文の使用例

```
requirement req1(T, F)
   type, deferred :: T
   function F(x, n) result(y)
      type(T) :: y
      type(T), intent(in) :: x
      integer, intent(in) :: n
   end function F
end requirement

template tada(T, C, F1, F2)
   type, deferred :: T
   requires req1(T, F1)
   requires req1(T, F2)
   integer, constant :: C(..)
contains
   ...
end template tada
```

TEMPLATE構文のネスト モジュールの使い方

```
module mumu
   template outer(T1)
      ...
      template inner(T2)
         ...
      end template inner
      ...
      instantiate inner(T2=real(4))
      ...
   end template outer
end module mumu

module momo
   use mumu
   instantiate outer(T1=mytype)
   instantiate inner(T2=real(8))
   ...
end module momo
```