

Fortran向けOSSを活用した デスクトップ開発環境効率化

出川智啓（名古屋大学未来材料・システム研究所）

内容

- ▶ 自己紹介, 背景
- ▶ 迷いを減らす - fpm
- ▶ 再発明を減らす - stdlib, json-fortran, vtkfortran
- ▶ 入力を減らす - VSCode, fypp
- ▶ 迷いを減らす - fprettify

この資料が意図しないこと

- ▶ 紹介するソフトウェアやエディタこそが最高だと主張する
- ▶ 紹介するソフトウェアやエディタを使っていない人を時代遅れだと主張する
- ▶ 紹介するソフトウェアやエディタを使うよう強制する

この資料が意図すること

- ▶ Fortranに関するツールの使い方や情報を共有する
- ▶ Fortranの利便性を改善するOSSを紹介する
- ▶ それらの導入までの障壁を下げる
- ▶ 最近流行っている環境でもFortranが利用できると広報する

FORTTRAN 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。

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。

FORTTRAN 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。

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 点を欠いては数値計算用言語とは言えないと考える。

背景

- ▶ 発表題目

- ▶ Fortran向けOSSを活用したデスクトップ開発環境効率化

- ▶ 発表概要

- ▶ Fortranの開発環境をもっと効率化したいと考えている方に向けて、FortranのOSSの紹介と導入を行います。

- ▶ ねらい

- ▶ Fortranの強み・価値「スパコンで高性能を出すこと」は否定しない
 - ▶ プログラミング言語としての利便性改善にOSSを利用する
 - ▶ 興味を持ったユーザが、スパコンに至る前に逃げ出すことを防ぎたい

内容

- ▶ 自己紹介, 背景
- ▶ 迷いを減らす - fpm
- ▶ 再発明を減らす - stdlib, json-fortran, vtkfortran
- ▶ 入力を減らす - VSCode, fypp
- ▶ 迷いを減らす - fprettify

この発表で用いるプログラム

- ▶ GitHubにアップロード

- ▶ `https://github.com/degawa/hpfpc_fortran_oss`

```
> git clone --recursive https://github.com/degawa/hpfpc_fortran_oss
```

- ▶ 紹介するOSSの使用例

- ▶ `stdlib`, `JSON-Fortran`, `VTKFortran`

- ▶ `stdlib`, `JSON-Fortran`, `VTKFortran`の組合せ

- ▶ `fprettify`, `fypp`

迷いを減らす

fpm - Fortran Package Manager

- ▶ Fortranのパッケージマネージャ兼ビルドシステム
- ▶ 最新バージョンは0.5.0 (11/22リリース)
- ▶ GitHub
 - ▶ <https://github.com/fortran-lang/fpm>
- ▶ ドキュメント
 - ▶ <https://fpm.fortran-lang.org>

fpm - Fortran Package Manager

- ▶ fpmの対象範囲
 - ▶ プログラム・ライブラリの構築
 - ▶ プログラムの実行・テスト
 - ▶ サンプルやデモの実行
 - ▶ プロジェクト構造の規定と配布の簡略化
 - ▶ 他fpmプロジェクトの依存関係の解決

fpmのインストール

- ▶ GitHubからバイナリをダウンロードし，適当な場所に配置

- ▶ `https://github.com/fortran-lang/fpm`

- ▶ Conda

- ▶

```
> conda config --add channels conda-forge  
> conda create -n fpm fpm  
> conda activate fpm
```

- ▶ MSYS2

- ▶

```
> pacman -S mingw-w64-x86_64-fpm
```

- ▶ Spack

- ▶

```
> spack install fpm
```

コマンド - プロジェクトの作成

▶ 新規プロジェクトの作成

▶ `> fpm new プロジェクト名`

▶ プロジェクトの構造

▶ `app` メインルーチン

▶ `src` モジュール群

▶ `test` 単体テスト

▶ `example` 実行例・デモ

▶ プロジェクトの構成ガイド

<https://github.com/fortran-lang/fpm/blob/main/PACKAGING.md>

ディレクトリ構造

```
.
├── .git
├── app
│   └── main.f90
├── src
│   └── sample.f90
├── test
│   └── check.f90
├── fpm.toml
└── README.md
```

コマンド - プロジェクトのビルド

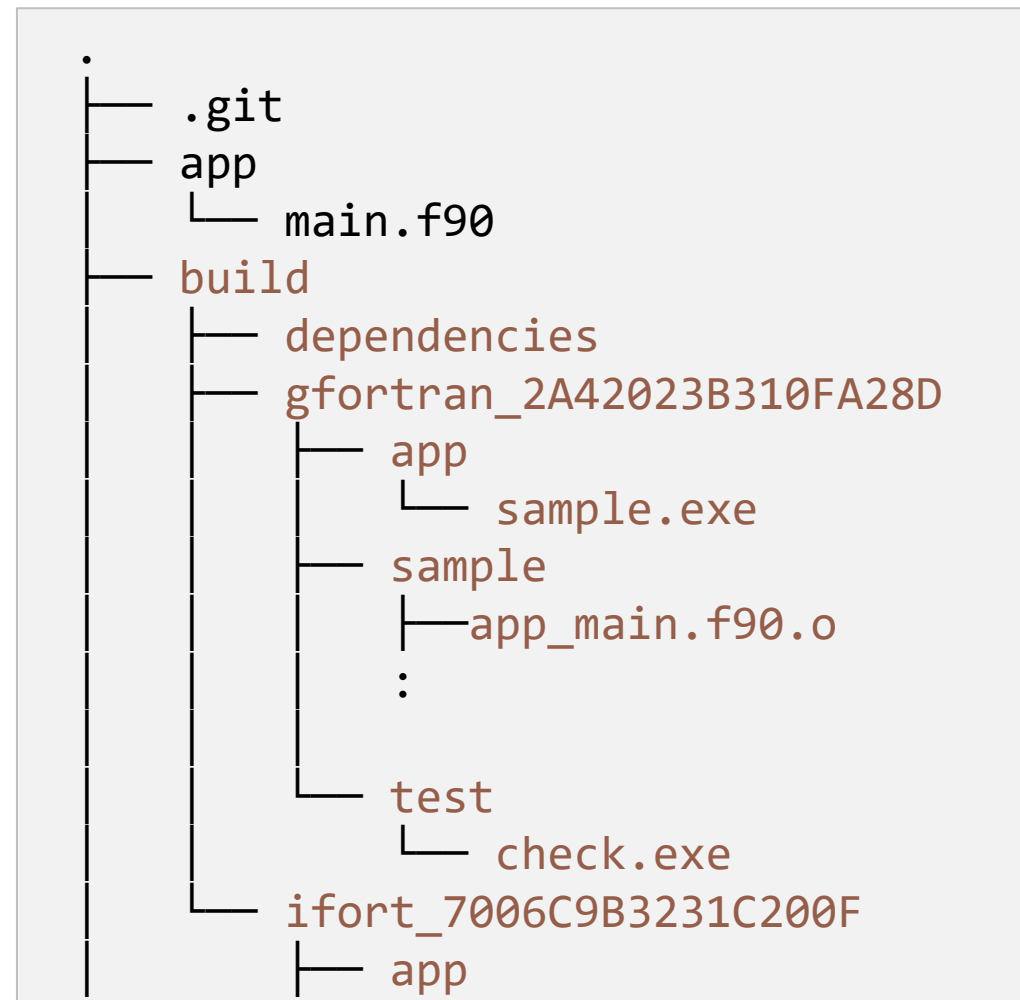
▶ ビルド

▶ `> fpm build`

▶ オプション

- ▶ `--profile {debug|release}`
- ▶ `--flag ""`
- ▶ `--compiler` コンパイラ名
 - `--compile ifort`
- ▶ コンパイルオプションに応じて
出力ディレクトリが変化

ディレクトリ構造



コマンド - プロジェクトの実行・テスト

▶ 実行

▶ `> fpm run`

▶ オプション

- ▶ `--profile {debug|release}`
- ▶ `--flag ""`
- ▶ `--compiler` コンパイラ名
- ▶ `--` 実行ファイルに渡すオプション
- ▶ `--target` アプリケーション名
- ▶ `etc.`

▶ テスト

▶ `> fpm test`

▶ オプション

- ▶ `--profile {debug|release}`
- ▶ `--flag ""`
- ▶ `--compiler` コンパイラ名
- ▶ `--` 実行ファイルに渡すオプション
- ▶ `--target` アプリケーション名
- ▶ `etc.`

実行，テストもビルドと同じオプションが指定できる

コマンド - ターゲットのインストール

▶ インストール

▶ `> fpm install`

▶ オプション

▶ buildのオプション

□ `--profile, --flag, --compiler`

▶ `--no-rebuild`

▶ `--prefix`

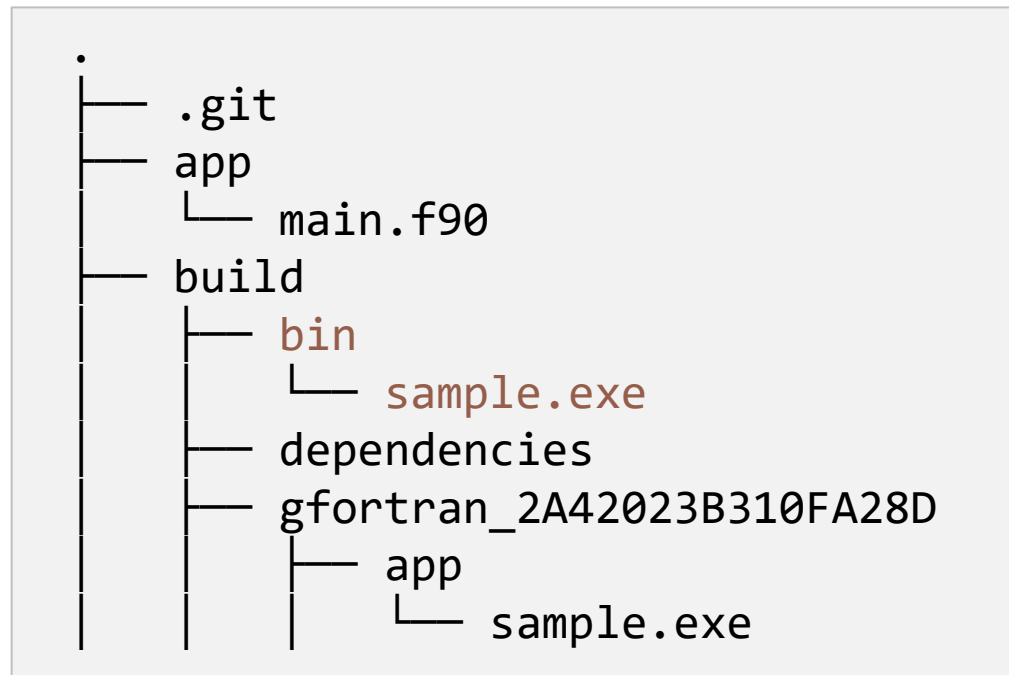
▶ `--bindir`

▶ `--libdir`

▶ `--includedir`

▶ 実行ファイルの取り出しにも利用

▶ `> fpm install --prefix build`



プロジェクトの設定

- ▶ `fpm.toml`に設定を記述
- ▶ `fpm.toml`はマニフェストとよばれている

- ▶ マニフェストの詳しい情報は

Fortran package manager (fpm) manifest reference

<https://github.com/fortran-lang/fpm/blob/main/manifest-reference.md>

- ▶ マニフェストの項目
 - ▶ プロジェクト概要, ビルド, ターゲット (`library`, `executable`, `test`), 依存プロジェクト, インストール

プロジェクトの設定 - プロジェクト概要

▶ name

- ▶ プロジェクト（パッケージ）の名前を設定する
- ▶ ライブラリや実行ファイルの名前としても使われる
- ▶ 他のプロジェクトが利用する場合にも参照される

プロジェクトの設定 - [build]

▶ ビルド, リンクの設定

▶ auto-executables, auto-examples, auto-tests

- ▶ app, example, testディレクトリを自動で探してビルドする/しないを設定

▶ external-modules

- ▶ useするモジュールファイルを指定
- ▶ external-modules = "stdlib_error"
- ▶ external-modules = ["netcdf", "h5lt"]

▶ link

- ▶ リンクするライブラリを指定
- ▶ link = ["blas", "lapack"]

プロジェクトの設定 - [dependencies]

- ▶ 依存するfpmプロジェクトを指定
 - ▶ ローカルにある場合
 - ▶ プロジェクト名 = {path="path/to/fpm/project"}
 - ▶ Gitリポジトリのホスティングサイトから取得する場合
 - ▶ プロジェクト名 = {git="https://github.com/...", branch="..."}
 - ▶ json-fortran = {git="https://github.com/jacobwilliams/json-fortran"}
 - ▶ stdlib = {git="https://github.com/fortran-lang/stdlib",
branch="stdlib-fpm"}

プロジェクトの設定 - [install]


- ▶ インストールするターゲットを設定
 - ▶ 実行ファイルをインストール
 - ▶ 何も書かない
 - ▶ `library = false`
 - ▶ ライブラリをインストール
 - ▶ `library = true`

fpmへの期待

▶ fpmを利用することで

- ▶ プロジェクトの構造に悩まなくてよい
- ▶ 外部依存性の解決に悩まなくてよい
- ▶ ビルド，実行，テスト，インストールを系統的に実行できる

▶ 現状の不満

- ▶ コンパイルオプションをマニフェストに記述できない
- ▶ コンパイルオプションの指定が冗長
- ▶ 標準のディレクトリ名以外を使うと制約が大きい  他ビルドツールからの乗り換えは様子見
 - ▶ 複数のテストをビルドしてくれない等

再発明を減らす

再発明を減らす

- ▶ **stdlib**
 - ▶ 事実上の標準ライブラリを提供する
- ▶ **JSON-Fortran**
 - ▶ JSON形式のファイルを取り扱う
- ▶ **VTKFortran**
 - ▶ VTKファイルを取り扱う

stdlib - Fortran Standard Library

- ▶ Fortranの標準ライブラリを提供する
- ▶ 最新バージョンは0.1.0
- ▶ GitHub
 - ▶ <https://github.com/fortran-lang/stdlib>
- ▶ ドキュメント
 - ▶ <https://stdlib.fortran-lang.org/index.html>

stdlib - Fortran Standard Library

- ▶ Fortran標準で規定された標準ライブラリは存在しない
- ▶ コミュニティが主導し，多くの開発者に合意された，**事実上の**標準ライブラリを提供する
 - ▶ stdlib自身は車輪の再発明を含んでいる
 - ▶ 「多くの人が合意した車輪」を開発することで個別の再発明を回避する
 - ▶ 個人・組織がその用途に合った車輪を開発することは否定しない

stdlibの機能

- ▶ `ascii` ASCII文字を扱う
- ▶ `bitsets` 任意桁のビット集合を扱う
- ▶ `error` エラー処理
- ▶ `IO` I/Oの利便性改善
- ▶ `kinds` 変数`kind`を設定
- ▶ `linalg` 線形代数
- ▶ `logger` ロギング
- ▶ `math` 数学関数
- ▶ `optval` `optional`引数の利便性改善
- ▶ `quadrature` 数値積分
- ▶ `random` 乱数生成
- ▶ `sorting` ソート
- ▶ `specialfunctions` 特殊関数
- ▶ `stats` 統計処理
- ▶ `stats_distributions_uniform`
確率分布
- ▶ `string_type` 文字列を扱う派生型
- ▶ `strings` Fortran標準の文字列操作
- ▶ `stringlist_type` `string_type`のリスト
- ▶ 追加予定機能
 - ▶ 単体テスト, 連結リスト

インストール

- ▶ GitHubからソースを取得

- ▶ `> git clone https://github.com/fortran-lang/stdlib`

- ▶ ビルド

- ▶ CMake: `> cmake -B build`

- ▶ make: `> make -f Makefile.manual`

- ▶ fpm: `> git checkout stdlib-fpm & fpm build --profile release`

- ▶ インストール

- ▶ CMake: `> cmake --install build --prefix installdir`

- ▶ fpm: `> fpm install --profile release --prefix installdir`

stdlibの使用例

- ▶ stdlibを利用し，動作確認を行ったプログラムを作成
- ▶ GitHub
 - ▶ https://github.com/degawa/ex_stdlib

```
> git clone https://github.com/degawa/ex_stdlib
> cd ex_stdlib
> fpm build
> fpm run
```

stdlibの使用例

- ▶ stdlibの機能を3種類に分類

- ▶ 公式(Programming/Algorithms/Mathematics)とは別観点から分類

1. System

- ▶ error, kinds, logger, IO, system

2. Type

- ▶ ascii, strings, bitsets, string_type, stringlist_type

3. Function

- ▶ optval, math, linalg, specialfunctions, quadrature, random, stats, stats_distribution_uniform, sorting

stdlibの使用例

- ▶ 各機能の使用例を呼び出す手続を羅列
- ▶ コメントを外すことで実行する機能を選択

```
program main
  use :: ex_stdlib_system
  use :: ex_stdlib_type
  use :: ex_stdlib_function
  implicit none

  ! call ex_stdlib_error_check()
  ! call ex_stdlib_error_error_stop()
  ! call ex_stdlib_kinds()
  :
  :
end program main
```

stdlib_error

- ▶ エラー処理のための手続を定義
 - ▶ check
 - ▶ 条件式の結果を調べる
 - ▶ error_stop
 - ▶ プログラムを停止する

stdlib_errorの使用例

▶ check

```
! 引数conditionが`.false.`のとき, "Check failed."を出力し, 終了コード1でプログラムを終了する.  
call check(condition=.false.)  
! Check failed.  
! ERROR STOP 1
```

```
! `a==0`が`.false.`のとき, 引数msgに渡した文字列を出力し, 終了コード1でプログラムを終了する.  
call check(a == 0, msg="error: actual value of a is not equal to expected value 0")  
! error: acctual value of a is not equal to expected value 0  
! ERROR STOP 1
```

```
! `a==0`が`.false.`のとき, 引数msgに渡した文字列を出力し, 終了コード77でプログラムを終了する.  
call check(a == 0, "error: actual value of a is not equal to expected value 0", code=77)  
! error: acctual value of a is not equal to expected value 0  
! ERROR STOP 77
```

```
! `a==0`が`.false.`のとき, 引数msgに渡した文字列を出力する.  
! 引数warnに`.true.`を渡すと, プログラムは終了しない.  
call check(a == 0, "warning: actual value of a is not equal to expected value 0", warn=.true.)  
! warning: acctual value of a is not equal to expected value 0
```

stdlib_errorの使用例

▶ error_stop

```
! 引数msgに渡した文字列を出力し，非ゼロの終了コードでプログラムを終了する.  
call error_stop(msg="Fatal error")  
! Fatal error  
! ERROR STOP
```

```
! 引数に渡した文字列を出力し，終了コード999でプログラムを終了する.  
call error_stop("Fatal error", code=999)  
! Fatal error  
! ERROR: code          999  was specified.  
! ERROR STOP
```

stdlib_kinds

- ▶ Fortranのデータ型 (integer, logical, real, complex) に対して, 種別 (kind) を表すパラメータを定義
- ▶ 基本的なパラメータはiso_fortran_envで定義されているが, 実質的な標準の導入や利便性の改善が目的
 - ▶ sp - 単精度実数のkind (real32)
 - ▶ dp - 倍精度実数のkind (real64)
 - ▶ xdp - 拡張倍精度実数のkind
 - ▶ qp - 4倍精度実数のkind
 - ▶ lk - 論理型変数のkind

stdlib_kindsで定義されたパラメータの参照

```
real(sp) :: f_sp
real(dp) :: f_dp
logical(lk) :: l
logical(c_bool) :: l_c
integer(int8) :: i1
integer(int16) :: i2
integer(int32) :: i4
integer(int64) :: i8

print '(3g0)', "storage size of real(sp) is ", storage_size(f_sp), " bits"      ! storage size of real(sp) is 32 bits
print '(3g0)', "storage size of real(dp) is ", storage_size(f_dp), " bits"      ! storage size of real(dp) is 64 bits
print '(3g0)', "storage size of logical(lk) is ", storage_size(l), " bits"      ! storage size of logical(lk) is 32 bits
print '(3g0)', "storage size of logical(c_bool) is ", storage_size(l_c), " bits"! storage size of logical(c_bool) is 8 bits
print '(3g0)', "storage size of integer(int8) is ", storage_size(i1), " bits"   ! storage size of integer(int8) is 8 bits
print '(3g0)', "storage size of integer(int16) is ", storage_size(i2), " bits"  ! storage size of integer(int16) is 16 bits
print '(3g0)', "storage size of integer(int32) is ", storage_size(i4), " bits"  ! storage size of integer(int32) is 32 bits
print '(3g0)', "storage size of integer(int64) is ", storage_size(i8), " bits"  ! storage size of integer(int64) is 64 bits
```

stdlib_logger

- ▶ ロギング（エラー報告，メッセージその他の情報の表示）に使用される派生型を定義
 - ▶ 任意のロガーを宣言できる
 - ▶ 規定のロガーとしてglobal_loggerが利用できる

stdlib_loggerの使用例

! debugログの画面出力. 標準設定では出力されない.

```
call global_logger%log_debug(message="log message debug")
```

! informationログの画面出力. "タイムスタンプ: INFO: メッセージ"の形式で出力.

```
call global_logger%log_information(message="log message information")
```

! 2021-11-21 23:29:31.200: INFO: log message information

! warningログの画面出力. "タイムスタンプ: WARN: メッセージ"の形式で出力.

```
call global_logger%log_warning(message="log message warning")
```

! 2021-11-21 23:29:31.201: WARN: log message warning

! errorログの画面出力. "タイムスタンプ: ERROR: メッセージ"の形式で出力.

```
call global_logger%log_error(message="log message error")
```

! 2021-11-21 23:29:31.201: ERROR: log message error

! ロガーを呼び出しているモジュール名と手続名の出力.

! "タイムスタンプ: モジュール名 % 手続名: INFO: メッセージ"の形式で出力.

```
call global_logger%log_information("log message information", &  
                                   module="ex_stdlib_system", &  
                                   procedure="ex_stdlib_logger")
```

stdlib_loggerの使用例

！ ロガーの設定. debugログを出力する

```
call global_logger%configure(level=debug_level)
```

！ ロガーにファイルを追加し, ログメッセージをファイル出力する.

！ 画面には出力されなくなる.

```
call global_logger%add_log_file("log_message.txt", unit=log_file_unit, stat=stat)
```

！ ロガーの出力装置に, 標準出力を追加.

！ これ以降, ログファイルと画面の双方にメッセージが出力される

```
call global_logger%add_log_unit(unit=output_unit)
```

！ ロガーに登録されている装置数を表示.

```
call global_logger%log_debug("number of log units = "//to_string(global_logger%log_units_assigned()))
```

！ 2021-11-21 23:29:31.204: DEBUG: number of log units = 2

！ ロガーにファイルを追加. 既存のログファイルに付きするように設定.

```
call global_logger%add_log_file("log_message.txt", unit=log_file_unit, &  
                                action="write", position="append", status="old")
```

stdlib_loggerの使用例

```
! log_io_errorの動作確認のために存在しないファイルを開く.  
open (newunit=io_unit, file="test.txt", status="old", iostat=iostat, iomsg=iomsg)  
! ioエラーをログメッセージとして画面に表示.  
call global_logger%log_io_error("file //"test.txt//" open failure", iostat=iostat, iomsg=iomsg)  
! 2021-11-21 23:29:31.208: I/O ERROR: file test.txt open failure  
! With iostat = 2  
! With iomsg = "Cannot open file 'test.txt': No such file or directory"
```

```
! テキスト内のエラーをログメッセージとして画面に表示する.  
call global_logger%log_text_error(line="text message", &  
                                   column=7, &  
                                   summary="error summary", &  
                                   filename="test.txt", &  
                                   line_number=0, &  
                                   caret="^")  
  
! 2021-11-21 23:29:31.208  
! test.txt:0:7  
!  
! text message  
!      ^  
! Error: error summary
```

stdlib_loggerの使用例

! 新しいロガーを宣言.

```
type(logger_type) :: new_logger
```

! new_loggerは出力レベルをwarning以上にしたので, informationは表示されない.

```
call new_logger%configure(level=warning_level)
```

```
call global_logger%log_information("global logger")
```

```
call new_logger%log_information("user logger")
```

```
call global_logger%log_warning("global logger")
```

```
call new_logger%log_warning("user logger")
```

```
! 2021-11-23 20:00:02.315: INFO: global logger
```

```
! 2021-11-23 20:00:02.316: WARN: global logger
```

```
! 2021-11-23 20:00:02.317: WARN: user logger
```

stdlib_io

- ▶ ファイル入出力の利便性を改善する処理を定義
 - ▶ open
 - ▶ ファイルを開き，ユニット番号を返す
 - ▶ savetxt
 - ▶ 2次元配列をテキストファイルに書き出す
 - ▶ loadtxt
 - ▶ 2次元配列をテキストファイルから読み込む

stdlib_ioの使用例

▶ open

```
integer(int32) :: unit
character(128) :: str
! open文を簡略化するopen関数.
! 書き込み+テキストモードで開く.
unit = open ("example_io.txt", "wt")
close (unit)

! 読み取り+テキストモードで開く
unit = open ("example_io.txt", "rt")

! 書き込みをしようとするとエラー.
! write (unit, '(A)') "append to example_io.txt"
close (unit)

! 追記モードで開く
unit = open ("example_io.txt", "a")
close (unit)
```


stdlib_ioの使用例

▶ savetxt, loadtxt

```
use :: stdlib_math

real(real32) :: array(16, 2)

! stdlib_mathで定義されているlinspaceを利用してx = [0, 2π]を設定し, sin(x)を計算.
array(:, 1) = linspace(0., 2.*acos(-1.), 16)
array(:, 2) = sin(array(:, 1))
call savetxt("sin.txt", array)
! 2次元配列をテキストモードで出力するsavetxt手続.
!      j      1      2      i
! array = 0.00000000E+00  0.00000000E+00  1
!          4.18879032E-01  4.06736642E-01  2
!          8.37758064E-01  7.43144870E-01  3
!          :
!          5.44542742E+00 -7.43144751E-01
!          5.86430645E+00 -4.06736493E-01
!          6.28318548E+00  1.74845553E-07
call loadtxt("sin.txt", array)
```

stdlib_system

- ▶ OSの機能呼び出す
 - ▶ stdlib本体にはないが, stdlib-fpmブランチに含まれている
- ▶ sleep
 - ▶ ミリ秒単位でプログラムの実行を停止する

```
call system_clock(count_start_c, count_per_sec)

! ミリ秒単位でプログラムの実行を中断
call sleep(millisec=100)

call system_clock(count_end_c, count_per_sec)
print '(3g0)', "sleep ", real(count_end_c - count_start_c)/real(count_per_sec), "sec"
! sleep 0.108999997sec
```

stdlib_ascii

- ▶ 文字の変数や定数を扱う手続を定義

- ▶ 定数

- ▶ ASCIIの各制御文字, `fullhex_digit`, `hex_digits`, `lowerhex_digits`, `digits`, `octal_digits`, `letters`, `uppercase`, `lowercase`, `whitespace`

- ▶ 手続（一文字用）

- ▶ `is_alpha`, `is_alphanum`, `is_digit`, `is_hex_digit`, `is_octal_digit`, `is_control`, `is_white`, `is_blank`, `is_ascii`, `is_punctuation`, `is_graphical`, `is_printable`, `is_lower`, `is_upper`

- ▶ 手続（文字列用）

- ▶ `to_lower`, `to_upper`, `to_title`, `to_sentence`, `reverse`

stdlib_asciiで定義されているASCII文字

```
character(len=1), public, parameter :: NUL = achar(int(z'00')) !! Null
character(len=1), public, parameter :: SOH = achar(int(z'01')) !! Start of heading
character(len=1), public, parameter :: STX = achar(int(z'02')) !! Start of text
character(len=1), public, parameter :: ETX = achar(int(z'03')) !! End of text
character(len=1), public, parameter :: EOT = achar(int(z'04')) !! End of transmission
character(len=1), public, parameter :: ENQ = achar(int(z'05')) !! Enquiry
character(len=1), public, parameter :: ACK = achar(int(z'06')) !! Acknowledge
character(len=1), public, parameter :: BEL = achar(int(z'07')) !! Bell
character(len=1), public, parameter :: BS = achar(int(z'08')) !! Backspace
character(len=1), public, parameter :: TAB = achar(int(z'09')) !! Horizontal tab
character(len=1), public, parameter :: LF = achar(int(z'0A')) !! NL line feed, new line
character(len=1), public, parameter :: VT = achar(int(z'0B')) !! Vertical tab
character(len=1), public, parameter :: FF = achar(int(z'0C')) !! NP form feed, new page
character(len=1), public, parameter :: CR = achar(int(z'0D')) !! Carriage return
character(len=1), public, parameter :: SO = achar(int(z'0E')) !! Shift out
character(len=1), public, parameter :: SI = achar(int(z'0F')) !! Shift in
character(len=1), public, parameter :: DLE = achar(int(z'10')) !! Data link escape
character(len=1), public, parameter :: DC1 = achar(int(z'11')) !! Device control 1
character(len=1), public, parameter :: DC2 = achar(int(z'12')) !! Device control 2
character(len=1), public, parameter :: DC3 = achar(int(z'13')) !! Device control 3
character(len=1), public, parameter :: DC4 = achar(int(z'14')) !! Device control 4
character(len=1), public, parameter :: NAK = achar(int(z'15')) !! Negative acknowledge
character(len=1), public, parameter :: SYN = achar(int(z'16')) !! Synchronous idle
character(len=1), public, parameter :: ETB = achar(int(z'17')) !! End of transmission block
character(len=1), public, parameter :: CAN = achar(int(z'18')) !! Cancel
character(len=1), public, parameter :: EM = achar(int(z'19')) !! End of medium
character(len=1), public, parameter :: SUB = achar(int(z'1A')) !! Substitute
character(len=1), public, parameter :: ESC = achar(int(z'1B')) !! Escape
character(len=1), public, parameter :: FS = achar(int(z'1C')) !! File separator
character(len=1), public, parameter :: GS = achar(int(z'1D')) !! Group separator
character(len=1), public, parameter :: RS = achar(int(z'1E')) !! Record separator
character(len=1), public, parameter :: US = achar(int(z'1F')) !! Unit separator
character(len=1), public, parameter :: DEL = achar(int(z'7F')) !! Delete
```

stdlib_strings

- ▶ 文字列(`character(len=:)`)を扱う手続を定義
 - ▶ `to_string`, `strip`, `chomp`, `starts_with`, `ends_with`, `slice`,
`find`, `replace_all`, `padl`, `padr`, `count`

stdlib_stringsの使用例

! 整数を文字に変換する.

```
print *, to_string(huge(0))! 2147483647
print *, to_string(-huge(0))! -2147483647
```

! 書式付きで整数を文字に変換する

! 4桁の数として文字列に変換する.

```
print *, "|"/to_string(1, '(I4)') ! | 1
```

! 4桁でゼロ埋めして文字列に変換する.

```
print *, "|"/to_string(1, '(I4.4)')! |0001
```

! 2桁の数として文字に変換する. 桁数が足りない場合は*に置き換えられる.

```
print *, to_string(100, '(I2)') ! **
```

block

```
integer(int32) :: i = 16
```

! 2桁の10進数として文字列に変換.

```
print *, to_string(i, '(I2)') ! 16
```

! 8桁の2進数として文字列に変換.

```
print *, to_string(i, '(B8.8)') ! 00010000
```

! 2桁の16進数として文字列に変換.

```
print *, to_string(i, '(Z2)') ! 10
```

end block

stdlib_stringsの使用例

! 実数を文字列に変換する.

```
print *, to_string(huge(0.)) ! 0.340282347E+39  
print *, to_string(-huge(0.))! -0.340282347E+39
```

```
print *, to_string(1.)      ! 1.00000000
```

! 書式付きで実数を文字列に変換する.

! 全体で8文字, 浮動小数点以下を3桁として文字列に変換する.

```
print '(A,F8.3)', to_string(100., '(F8.3)'), 100.      ! 100.000 100.000
```

! 指数表記で文字列に変換する.

```
print '(A,E10.2e2)', to_string(1000., '(E10.2e2)'), 1000.! 0.10E+04 0.10E+04
```

! 科学表記で文字列に変換する.

```
print '(A,ES10.2)', to_string(1000., '(ES10.2)'), 1000. ! 1.00E+03 1.00E+03
```

```
complex(real64) :: c = cmplx(0.75, 0.25)
```

! 複素数を文字列に変換する.

! 書式付きで複素数を文字列に変換する.

```
print *, to_string(c, '(F6.2)') ! ( 0.75, 0.25)
```

stdlib_string_type

- ▶ 任意長さの文字のつながり = 文字列を扱う派生型を定義
- ▶ Fortranの文字列に対する内部手続と互換性のある手続も併せて定義

```
type(string_type) :: str

! コンストラクタを利用した初期化.
str = string_type("string")
! 派生型IOおよびlen関数が利用可能.
print *, str, len(str) ! string          6

! 文字列リテラルに基づいて初期化.
str = "string_type"
print *, str, len(str) ! string_type     11
! 整数リテラルに基づいて初期化.
str = string_type(-huge(0))
print *, str ! -2147483647
```


stdlib_string_typeの使用例

```
type(string_type) :: str1, str2
str1 = "string"

! string_type間の代入
str2 = str1
print *, str2! string

! 二つのstring_typeの比較.
print *, str1 == str2, str1 /= str2, str1 >= str2, str1 <= str2 ! T F T T

! 二つのstring_typeの連結.
print *, str1//str2! stringstring

str2 = ""
! string_type間でデータを移動.
call move(from=str1, to=str2)
print *, str2, len(str2), len(str1) ! string          6          0
```

stdlib_stringlist_type

- ▶ string_typeのリストを扱うための型stringlist_typeを定義
- ▶ stringlist_typeの要素を扱うための型
stringlist_index_typeも定義

```
type(stringlist_type) :: strlist
```

```
! コンストラクタを利用したstringlistの初期化.
```

```
! 要素 (string_type) の長さが異なる.
```

```
strlist = stringlist_type([string_type("one"), string_type("two"), string_type("three")])
```

```
! 要素の指定には, fidx(先頭からの要素番号)もしくはbidx(末尾からの要素番号)の戻り値を利用する.
```

```
print *, strlist%get(fidx(1)), strlist%get(fidx(2)), strlist%get(fidx(3))
```

```
! one two three
```

stdlib_stringlist_typeの使用例

```
type(stringlist_type) :: strlist
type(string_type) :: str
type(stringlist_index_type) :: idx

strlist = stringlist_type([string_type("one"), string_type("two"), string_type("three")])
! stringlist_typeの型束縛手続きlenは、リストの長さ（要素数）を返す.
print *, strlist%len()! 3

! stringlist_typeの要素を取り出してstring_typeに代入.
str = strlist%get(fidx(3))

! 先頭要素はfidx(1)に代わってlist_headを利用可能.
idx = fidx(1)
! stringlist_typeの先頭に要素を追加.
call strlist%insert_at(idx, string_type("four"))
print *, strlist%get(fidx(1)), strlist%get(fidx(2)), strlist%get(fidx(3)), strlist%get(fidx(4))
! four one two three

! stringlist_typeを初期化
call strlist%clear()
print *, strlist%len()! 0
```

stdlib_optval

▶ optval

- ▶ 二つの引数x, defaultを取り, xがあればxを, なければdefaultを返す
- ▶ optional引数を持つ手続を呼び出した際に, optional引数が渡されていないときに標準値を定めるために利用

```
real(real64) :: x = 64.  
print *, root(x), root(x, 3)! 8.0000000000000000 3.9999999999999996  
  
!! xのn乗根を計算する.  
real(real64) function root(x, n)  
    implicit none  
    real(real64), intent(in) :: x!! 被開平数  
    integer(int32), intent(in), optional :: n!! 指数  
  
    root = x**(1d0/optval(n, 2)) ! nが無いときは2が使われる  
end function root
```

stdlib_math

- ▶ 多目的の数学関数を定義
 - ▶ clip, gcd, linspace, logspace, arange
 - ▶ 特殊関数はstdlib_specialfunctionsで定義

! 引数が一つの場合は終端として扱い, 始点を1, 間隔を1として扱う.

```
print *, arange(3.0) != arange(1.0, 3.0, 1.0)
! 1.00000000    2.00000000    3.00000000
```

! 始点と終点を指定. 間隔を1として扱う.

```
print *, arange(0.0, 2.0) != arange(0.0, 2.0, 1.0)
! 0.00000000    1.00000000    2.00000000
```

! 始点, 終点, 間隔を指定.

```
print *, arange(0.0, 0.2, 0.2)
! 0.00000000    0.200000003
```

stdlib_mathの使用例

```
print *, linspace(start=0., end=1., n=6)
! 0.00000000 0.200000003 0.400000006 0.600000024 0.800000012 1.00000000

print *, linspace(0d0, 1d0, 6)
! 0.0000000000000000 0.20000000000000001 0.40000000000000002 0.60000000000000009 0.80000000000000004
1.0000000000000000

! 整数型で範囲を指定すると、戻り値は倍精度実数
print *, linspace(0, 1, 6)
! 0.0000000000000000 0.20000000000000001 0.40000000000000002 0.60000000000000009 0.80000000000000004
1.0000000000000000
```

```
! baseが倍精度実数なので倍精度実数で返す.
print *, logspace(-3, 1, n=5, base=10d0)
! 1.0000000000000000E-003 1.0000000000000000E-
002 0.10000000000000001 1.0000000000000000 10.000000000000000

! start, endが単精度実数なので単精度実数で返す.
print *, logspace(-3., 1., n=5, base=10)
! 1.00000005E-03 9.99999978E-03 0.100000001 1.00000000 10.0000000
```

stdlib_linalg

- ▶ 線形代数に関する手続を定義
 - ▶ `diag`, `eye`, `trace`, `outer_product` (直積)

```
real(real64) :: v(2), A(2, 2)

! ベクトル (1次元配列) を入力として, その値を対角成分にもつ行列を返す.
! [1, 1]を対角成分として行列を作る.
v = [1, 1]
A = diag(v)
print *, A
!  i\j  1  2
!      +-----
!   1 |  1  0
!   2 |  0  1
```

stdlib_linalgの使用例

```
real(real32) :: u(3), A(4, 4)
```

```
u = 1
```

! 対角線を1上にずらし, [1, 1, 1]を成分にもつ行列を返す.

```
A = diag(u, 1)
```

```
print *, A
```

```
! i\j 1 2 3 4
```

```
!      +-----
```

```
!  1| 0  1  0  0
```

```
!  2| 0  0  1  0
```

```
!  3| 0  0  0  1
```

```
!  4| 0  0  0  0
```

! 3行3列の単位行列を返す.

```
print *, eye(3)
```

```
! i\j 1 2 3
```

```
!      +-----
```

```
!  1| 1  0  0
```

```
!  2| 0  1  0
```

```
!  3| 0  0  1
```


stdlib_linalgの使用例

```
real(real64) :: A(3, 3)
A = reshape([1, 2, 3, 4, 5, 6, 7, 8, 9], [3, 3])
! i¥j 1 2 3
!      +-----
!  1| 1 4 7
!  2| 2 5 8
!  3| 3 6 9
print *, trace(A) ! 15
```

```
real(real32), allocatable :: A(:, :), u(:), v(:)
u = [real :: 1, 2, 3]
v = [real :: 4, 5, 6]
! 直積 $u \times v^T$ を計算する.
A = outer_product(u, v)
print *, A
! i¥j 1 2 3
!      +-----
!  1| 4 5 6
!  2| 8 10 12
!  3| 12 15 18
```

stdlib_specialfunctions

▶ 特殊関数を定義

▶ 現状ではLegendre多項式のみ実装されている

```
real(real64) :: x = 1d-1
! P_0(x) = 1
print *, legendre(0, x), 1d0 ! 1.0000000000000000      1.0000000000000000

! P_1(x) = x
print *, legendre(1, x), x ! 0.10000000000000001      0.10000000000000001

! P_2(x) = (3d0*x**2 - 1d0)/2d0
print *, legendre(2, x), (3d0*x**2 - 1d0)/2d0 ! -0.48499999999999999      -0.48499999999999999

! P_3(x) = (5d0*x**3 - 3d0*x)/2d0
print *, legendre(3, x), (5d0*x**3 - 3d0*x)/2d0 ! -0.14749999999999999      -0.14750000000000002

! P_10(x) = (46189d0*x**10 - 109395d0*x**8 + 90090d0*x**6 - 30030d0*x**4 + 3465d0*x**2 - 63d0)/256d0
print *, legendre(10, x), (46189d0*x**10 - 109395d0*x**8 + 90090d0*x**6 - 30030d0*x**4 + 3465d0*x**2 - 63d0)/256d0 ! -0.12212499738710939      -0.12212499738710936
```

stdlib_quadrature

- ▶ 数値積分を計算する手続を定義
 - ▶ 台形則, シンプソン則, Gauss-Legendre求積法, Gauss-Legendre-Lobatto求積法が実装されている

```
real(real32) :: x(6), f(6), w(6)
x = linspace(-1., 1., 6) ! x = [-1.0 -0.6 -0.2 0.2 0.6 1.0 ]
f = 5.*x**4              ! f = [5.0 0.648 8E-03 8E-03 0.648 5.0 ]

! 台形則で関数値fを積分する.
print *, trapz(f, x) ! 2.52479982

! 与えられたx座標値に対して, 重み係数を返す.
w = trapz_weights(x)

! 重み係数wと関数値fを用いて積分する.
print *, sum(f*w) ! 2.52480006
```

stdlib_quadrationの使用例

```
! シンプソン則で関数値fを積分する.  
! 関数値と座標値を指定する.  
! 2.05973339  
print *, simps(f, x)  
  
! 与えられたx座標値に対して, 重み係数を返す.  
w = simps_weights(x)  
  
! 重み係数wと関数値fを用いて積分する.  
print *, sum(f*w) ! 2.05973339  
  
! Gauss-Legendre求積法のノードと重みを計算する.  
call gauss_legendre(x, w)  
! ノードと重みを使って, xが[-1, 1]の範囲で5x^4を積分する.  
print *, sum(5.*x**4*w) ! 1.9999999999999987  
  
! Gauss-Legendre-Lobatto求積法のノードと重みを計算する.  
call gauss_legendre_lobatto(x, w)  
  
! ノードと重みを使って, xが[-1, 1]の範囲で5x^4を積分する.  
print *, sum(5.*x**4*w) ! 1.9999999999999993
```

stdlib_random

- ▶ 擬似乱数生成器を定義
 - ▶ random_seed 擬似乱数の種を設定・取得する
 - ▶ dist_rand 整数の表現範囲内で擬似乱数値を取得する

```
integer(int32) :: seed_put, seed_get
! 乱数の種を設定・取得する.
seed_put = 135792468
call random_seed(seed_put, seed_get)

! 入力された整数のkindに基づいて, その整数の範囲で擬似乱数を生成する.
! 1バイト整数の範囲 [-2^7, 2^7 -1]内で乱数を生成する.
print *, dist_rand(0_int8) ! -90
! 2バイト整数の範囲 [-2^15, 2^15 -1]内で乱数を生成する.
print *, dist_rand(0_int16) ! -32725
! 4バイト整数の範囲 [-2^31, 2^31 -1]内で乱数を生成する.
print *, dist_rand(0_int32) ! -1601563881
```

stdlib_stats

▶ 統計処理の手続を定義

- ▶ mean 平均値を計算する
- ▶ median 中央値を計算する
- ▶ var 分散varianceを計算する
- ▶ cov 共分散covarianceを計算する
- ▶ corr Pearsonの相関係数(correlation)を計算する
- ▶ moment モーメントを計算する

stdlib_stats_distribution_uniform

- ▶ 一様分布に関する手続を定義
 - ▶ shuffle 配列を並び替える
 - ▶ rvs_uniform 一様分布を計算する
 - ▶ pdf_uniform 確率密度関数を計算する
 - ▶ cdf_uniform 累積分布関数を計算する

stdlib_sorting

- ▶ 1次元配列の並び替え処理を定義
 - ▶ `sort`, `ord_sort`, `sort_index`

```
integer(int32), allocatable :: array(:), work(:)
array = [5, 4, 3, 1, 10, 4, 9]
```

! イントロソートを利用して並び替える.

```
call sort(array)
print *, array ! 1 3 4 4 5 9 10
```

```
array = [5, 4, 3, 1, 10, 4, 9]
allocate (work, mold=array)
```

! Rustのソートアルゴリズムを利用して並び替える.

```
call ord_sort(array, work)
print *, array ! 1 3 4 4 5 9 10
```


stdlib_sortingの使用例

```
integer(int32), allocatable :: work(:)
integer(int_size), allocatable :: index(:), iwork(:)

array = [5, 4, 3, 1, 10, 4, 9]
allocate (index(1:size(array)))
allocate (work(1:size(array)/2))
allocate (iwork(1:size(array)/2))

! ord_sortを利用して並び替えのインデックスを返す.
call sort_index(array, index, work, iwork)
! 1 3 4 4 5 9 10
! 4 3 2 6 1 7 5
print *, array
print *, index

! indexを用いて並び替える.
array = [5, 4, 3, 1, 10, 4, 9]
array = array(index)
```

JSON-Fortran

- ▶ FortranでJSONファイルを扱うためのAPIを提供する
- ▶ 最新バージョンは8.2.5
- ▶ GitHub
 - ▶ <https://github.com/jacobwilliams/json-fortran>
- ▶ ドキュメント
 - ▶ <https://github.com/jacobwilliams/json-fortran/wiki>
 - ▶ API ドキュメントはFORDを用いてローカルに作成

JSON-Fortran

- ▶ JSON - JavaScript Object Notation
 - ▶ JavaScriptにおけるオブジェクトの記述法のサブセット
 - ▶ 軽量のテキストベースのデータ交換用フォーマット
- ▶ 独自のファイル形式やnamelistと比べて下記の利点がある
(JSON-Fortranの開発者の意見)
 - ▶ 標準的に利用されている
 - ▶ 人間が読めて、編集できる
 - ▶ 他の多くのプログラミング言語にAPIがある

JSON-Fortranのインストール

- ▶ GitHubからソースを取得

- ▶ `> git clone https://github.com/jacobwilliams/json-fortran`

- ▶ ビルド

- ▶ 独自のビルドスクリプト, Visual Studio, CMake, fpm用のビルド設定を整備

- ▶ インストール

- ▶ `make install`, `cmake --install build`, `fpm install`でインストール可能

- ▶ homebrewを利用したインストールも可能

JSON-Fortranの使用例

- ▶ JSON-Fortranを参照し，動作確認を行ったプログラムを作成
- ▶ GitHub
 - ▶ https://github.com/degawa/ex_jsonfortran

```
> git clone https://github.com/degawa/ex_jsonfortran  
> cd ex_jsonfortran  
> fpm build  
> fpm run
```

JSON-Fortranの使用例

- ▶ 3個のサブルーチンを作成
 - 1. 非常に簡単なjsonファイルの読み込み
 - ▶ `read_sample1_json_and_get_value`
 - 2. 数値，配列が書かれたjsonファイルの読み込み
 - ▶ `read_sample2_json_and_get_numerical_values`
 - 3. 文字列をjsonとして解釈し，値を取り出す
 - ▶ `create_json_from_string`

簡単なjsonファイルの読み込み

▶ JSONの書式

- ▶ オブジェクトのキーと値をコロンで対にする.
 - ▶ キーは文字列, 値は数値, 文字列, 論理値, オブジェクト
- ▶ 全体を{}でくくる
- ▶ コメントの記述は許可されていない

▶ サンプルファイル

▶ sample1.json

```
{  
  "key": "value"  
}
```

JSON-Fortranの処理の流れ

```
type(json_file) :: json

! json_fileを初期化する.
call json%initialize()

! sample1.jsonを読み込み.
call json%load(filename="sample1.json")

block
  character(:), allocatable :: value

  ! オブジェクトのキーを指定して値を取得し, valueに代入する.
  call json%get("key", value)
  print *, value ! value
end block

! json_fileを破棄
call json%destroy()
```


get手順のオプション

```
type(json_file) :: json
call json%initialize()
call json%load(filename="sample1.json")

block
  logical :: found
  character(:), allocatable :: value

  ! 追加のオプションfound, defaultの指定.
  ! キー "key"が存在していればfoundに.true.が代入される.
  call json%get("key", value, found=found, default="N/A")
  if (found) then
    print *, "key is found and value is ", value ! key is found and value is value
  end if
  ! defaultが指定されていれば, キーが存在していない場合に値がdefaultで指定された値に設定される.
  call json%get("Key", value, found=found, default="N/A")
  if (.not. found) then
    print *, "Key is not found, value is set to ", value ! Key is not found, value is set to N/A
  end if
end block
call json%destroy()
```

数値，配列が書かれたjsonファイルの読み込み

▶ サンプルファイル

▶ sample2.json

```
{
  "value": {
    "integer": 1,
    "real": {
      "standard notation": 1.0,
      "exponential notation": 1e-4
    },
    "string": "str",
    "logical": "true"
  },
  "array": {
    "integer": [10, 9, 8, 7],
    "real": [6.0, 5.0, 4.0],
    "string": ["3", "2", "1", "-1"]
  }
}
```

数値の読み込み

▶ 整数，文字列，論理値の読み込み

```
integer(int32) :: i
```

！ 整数値を読み込み.

！ オブジェクトを入れ子にした場合は，キーの名前を.で連結する.

```
call json%get("value.integer", i)
```

```
print *, "value.integer", i ! value.integer          1
```

```
character(:), allocatable :: s
```

！ 文字列を読み込み.

！ 変数の型はcharacter(:), allocatableとし，手続get内で自動的に割り付けられる.

```
call json%get("value.string", s)
```

```
print *, "value.string ", s ! value.string str
```

```
logical :: l
```

！ 論理値を読み込み.

```
call json%get("value.logical", l)
```

```
print *, "value.logical", l ! value.logical T
```

実数の読み込み

▶ 通常の表記

```
real(real64) :: f
```

! 実数値を読み込み.

! 実数は全て倍精度として取り扱われる.

! キーの名前は文字列なので, 空白があっても問題ない.

```
call json%get("value.real.standard notation", f)
```

```
print *, "value.real.standard notation", f ! value.real.standard notation 1.0000000000000000
```

▶ 指数表記

```
real(real64) :: f
```

! 指数表記の実数値を読み込み. 通常表記と区別する必要はない.

! 指数の記号はeを用いる.

```
call json%get("value.real.exponential notation", f)
```

```
print *, "value.real.exponential notation", f ! value.real.exponential notation 1.0000000000000000E-004
```

数値，配列が書かれたjsonファイルの読み込み

▶ サンプルファイル

▶ sample2.json

```
{
  "value": {
    "integer": 1,
    "real": {
      "standard notation": 1.0,
      "exponential notation": 1e-4
    },
    "string": "str",
    "logical": "true"
  },
  "array": {
    "integer": [10, 9, 8, 7],
    "real": [6.0, 5.0, 4.0],
    "string": ["3", "2", "1", "-1"]
  }
}
```

数値の配列の読み込み

- ▶ 配列は全てallocatable
- ▶ get内で自動的に割り付け・代入が行われる
- ▶ 整数，実数の読み込み（論理値も同じ）

```
integer(int32), allocatable :: i(:)
```

！ 整数の配列を取得.

！ 要素数はget内で自動的に割り付けられる.

```
call json%get("array.integer", i)
```

```
print *, "array.integer", i ! array.integer          10           9           8           7
```

```
real(real64), allocatable :: f(:)
```

！ 実数の配列を取得.

！ 要素数はget内で自動的に割り付けられる.

```
call json%get("array.real", f)
```

```
print *, "array.real", f  
! array.real    6.0000000000000000    5.0000000000000000    4.0000000000000000
```

文字列の配列の読み込み

- ▶ 長さの異なる文字列をまとめた配列は利用できない
- ▶ 文字列の長さを固定
- ▶ 配列の要素数をget内で自動的に決定

```
"array": {  
  "string": ["3", "2", "1", "-1"]  
}
```

```
character(16), allocatable :: s(:)
```

! 文字列の配列.

! 文字列の配列の場合、配列要素数はget内で自動的に決定されるが、文字列の長さは動的には変更できない.

! そのため、文字列の長さを固定している.

```
call json%get("array.string", s)
```

```
print *, "array.string ", s ! array.string 3          2          1          -1
```

文字列をjsonとして解釈し，値を取り出す

- ▶ JSONファイルを作成しなくても，JSON書式の文字列を解釈
- ▶ 簡易的な連想リストとして利用可能

```
type(json_file) :: json
character(:), allocatable :: json_string
character(1), parameter :: LF = achar(int(z'0A')) !! NL line feed, new line
! jsonの書式に沿って文字列を作る.
json_string = "{"//LF
json_string = json_string//'"value":{'//LF
json_string = json_string//'"integer":2,'//LF
json_string = json_string//'"real":1e+1,'//LF
json_string = json_string//'"string":"str",'//LF
json_string = json_string//'"logical":"false"'//LF
json_string = json_string//"}"//LF
json_string = json_string//"}"

call json%initialize()
! 文字列をjson書式として読み込み，解釈してjson_fileを作る.
call json%deserialize(json_string)
```


VTKFortran

- ▶ FortranでVTKフォーマットを扱うためのAPIを提供する
 - ▶ 多機能だが、ここしばらく開発が止まっている
- ▶ GitHub
 - ▶ <https://github.com/szaghi/VTKFortran>
- ▶ ドキュメント
 - ▶ <https://github.com/szaghi/VTKFortran/wiki>

VTKFortran

- ▶ VTKファイル形式

- ▶ Visualization Toolkitが提供する独自のファイル形式

- ▶ VTKFortranはレガシー，XMLの両方の形式をサポート

- ▶ レガシー

- ▶ Structured Point/Grid, Unstructured Grid, Rectilinear Grid

- ▶ XML

- ▶ Rectilinear Grid, Structured Grid, Unstructured Grid

- ▶ vtkMultiBlockDataSet

VTKFortran

▶ GitHubからソースを取得

```
> git clone --recursive https://github.com/szaghi/VTKFortran.git  
> cd VTKFortran  
> git submodule update --init --recursive
```

▶ ビルド

▶ make:

```
> make
```

▶ CMakeLists.txtは用意されているが，ディレクトリ構成の変更に追従しておらず，ビルドできない

▶ fpmにも未対応

VTKFortran

- ▶ リポジトリをforkし，fpmに対応

- ▶ `> git clone https://github.com/degawa/VTKFortran.git -b vtkfortran-fpm`

- ▶ ビルド

- ▶ fpm: `> fpm build --profile release --flag "-D_R16P"`

- ▶ VTKFortranの依存ライブラリは，ビルドの際に識別子を参照

- 4倍精度実数サポートの有無

- ▶ fpm.tomlにコンパイルオプションを記述する項目がない（今は）

- ▶ fpmはコンパイルオプションを依存ライブラリのビルドにも利用

- 依存ライブラリに必要なオプションを付与してビルド

VTKFortranの使用例

- ▶ VTKFortranを参照し，動作確認を行ったプログラムを作成
- ▶ GitHub
 - ▶ https://github.com/degawa/ex_vtkfortran

```
> git clone https://github.com/degawa/ex_vtkfortran
> cd ex_vtkfortran
> fpm build --profile release --flag "-D_R16P"
> fpm run
```

VTKFortranの使用例

- ▶ 4個のサブルーチンを作成

1. Structured Gridのノード上のスカラー値を出力

- ▶ `write_scalar_node_vts`

2. Rectilinear Gridのノード上のスカラー値を出力

- ▶ `write_scalar_node_vtr`

3. Rectilinear Gridのノード上のスカラー値を出力

- ▶ `write_vector_node_vtr`

4. Rectilinear Gridのセル上のスカラー値を出力

- ▶ `write_scalar_cell_vtr`

Structured Gridのノード上のスカラを出力

▶ 出力されるvtsファイル

```
<?xml version="1.0"?>
<VTKFile type="StructuredGrid" version="1.0" byte_order="LittleEndian">
  <StructuredGrid WholeExtent="+0 +9 +0 +5 +0 +5">
    <Piece Extent="+0 +9 +0 +5 +0 +5">
      <Points>
        <DataArray type="Float64" NumberOfComponents="3" Name="Points" format="appended" offset="0"/>
      </Points>
      <PointData>
        <DataArray type="Float64" NumberOfComponents="1" Name="float64_scalar" format="appended"
offset="8644"/>
      </PointData>
    </Piece>
  </StructuredGrid>
  <AppendedData encoding="raw">
raw data...
  </AppendedData>
</VTKFile>
```

Structured Gridのノード上のスカラを出力

```
use :: vtk_fortran, only: vtk_file
implicit none

type(vtk_file) :: vts !! VTK file

integer(int32), parameter :: nx1 = 0 !! x方向の配列範囲の下限
integer(int32), parameter :: nx2 = 9 !! x方向の配列範囲の上限
integer(int32), parameter :: ny1 = 0 !! y方向の配列範囲の下限
integer(int32), parameter :: ny2 = 5 !! y方向の配列範囲の上限
integer(int32), parameter :: nz1 = 0 !! z方向の配列範囲の下限
integer(int32), parameter :: nz2 = 5 !! z方向の配列範囲の上限
integer(int32), parameter :: nn = (nx2 - nx1 + 1)*(ny2 - ny1 + 1)*(nz2 - nz1 + 1) !! 全要素数

real(real64) :: x(nx1:nx2, ny1:ny2, nz1:nz2) !! x座標値
real(real64) :: y(nx1:nx2, ny1:ny2, nz1:nz2) !! y座標値
real(real64) :: z(nx1:nx2, ny1:ny2, nz1:nz2) !! z座標値
real(real64) :: v(nx1:nx2, ny1:ny2, nz1:nz2) !! スカラ値

integer(int32) :: stat !! 手続の実行結果の状態
```


Structured Gridのノード上のスカラを出力

```
! ファイルを開き, VTKFileタグとStructuredGridタグを開く.
! <VTKFile type="StructuredGrid" version="1.0" byte_order="LittleEndian">
!   <StructuredGrid WholeExtent="+0 +9 +0 +5 +0 +5">
stat = vts%initialize(format='raw', &
                      filename='xml_structured_grid_raw_scalar.vts', &
                      mesh_topology='StructuredGrid', &
                      nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)

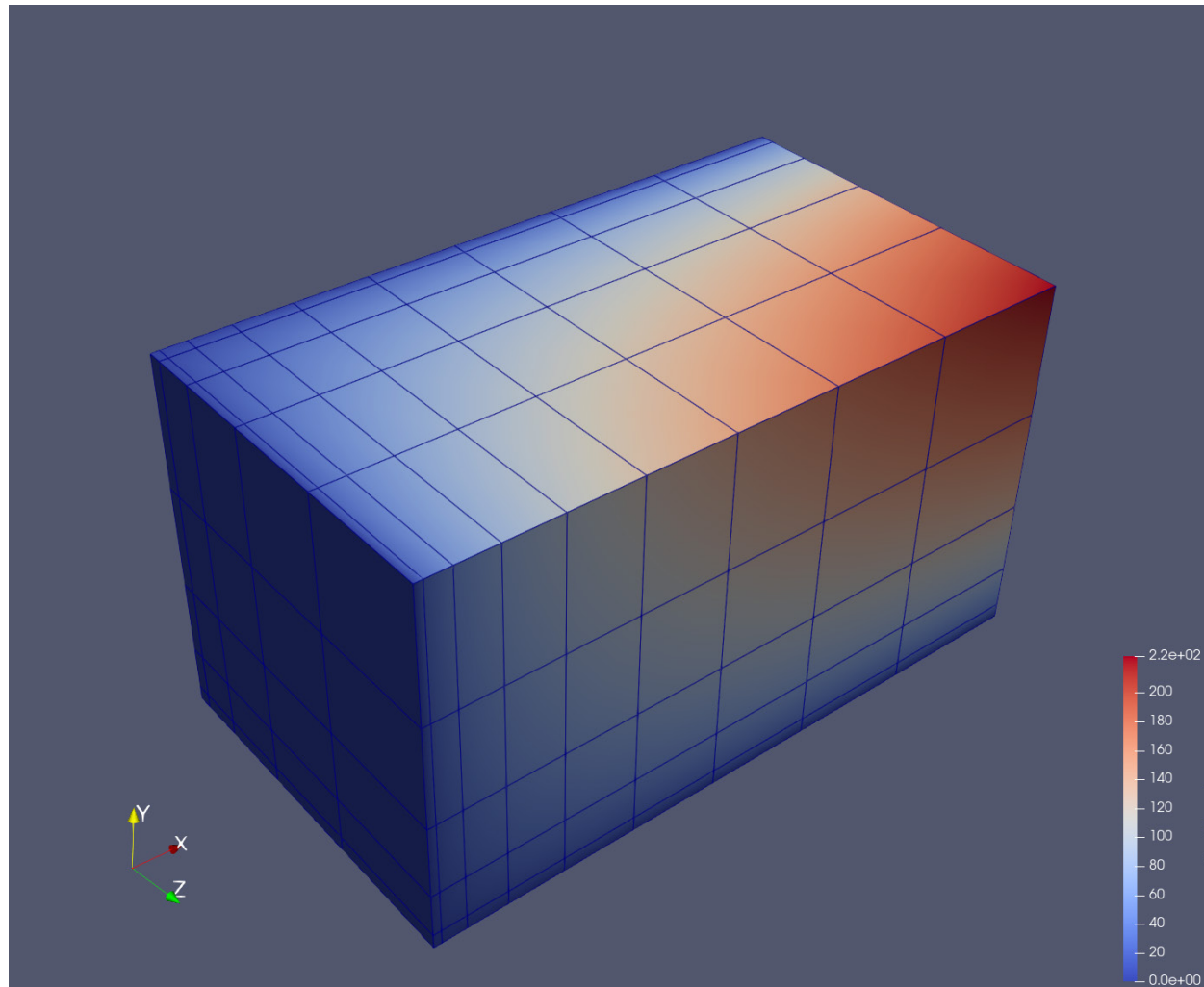
! Pieceタグ open.
! <Piece Extent="+0 +9 +0 +5 +0 +5">
stat = vts%xml_writer%write_piece(nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)

! 各方向座標値の出力情報の書き出し.
! 実際のデータは </StructuredGrid>と</VTKFile>の間に,
! <AppendedData encoding="raw"></AppendedData>が設けられ,
! そこに書き出される.
! <Points>
!   <DataArray type="Float64" NumberOfComponents="3" Name="Points" format="appended" offset="0"/>
! </Points>
stat = vts%xml_writer%write_geo(n=nn, x=x, y=y, z=z)
```

Structured Gridのノード上のスカラを出力

```
! 配列の書き出し.  
! 実際のデータはAppendedDataに追記される.  
! <PointData>  
stat = vts%xml_writer%write_dataarray(location='node', action='open')  
! <DataArray type="Float64" NumberOfComponents="1" Name="float64_scalar" format="appended" offset="8644"/>  
stat = vts%xml_writer%write_dataarray(data_name='float64_scalar', x=v, one_component=.true.)  
! </PointData>  
stat = vts%xml_writer%write_dataarray(location='node', action='close')  
  
! Pieceタグ close  
! </Piece>  
stat = vts%xml_writer%write_piece()  
  
! VTKFileタグとStructuredGridタグ close.  
! vtsファイルを閉じる.  
! </StructuredGrid>  
! </VTKFile>  
stat = vts%finalize()
```

Structured Gridのノード上のスカラを出力



Rectilinear Gridのノード上のスカラを出力

▶ 出力されるvtrファイル

```
<?xml version="1.0"?>
<VTKFile type="RectilinearGrid" version="1.0" byte_order="LittleEndian">
  <RectilinearGrid WholeExtent="+1 +10 +1 +6 +1 +6">
    <Piece Extent="+1 +10 +1 +6 +1 +6">
      <Coordinates>
        <DataArray type="Float64" NumberOfComponents="1" Name="X" format="appended" offset="0"/>
        <DataArray type="Float64" NumberOfComponents="1" Name="Y" format="appended" offset="84"/>
        <DataArray type="Float64" NumberOfComponents="1" Name="Z" format="appended" offset="136"/>
      </Coordinates>
      <PointData>
        <DataArray type="Float64" NumberOfComponents="1" Name="float64_scalar" format="appended" offset="188"/>
      </PointData>
    </Piece>
  </RectilinearGrid>
  <AppendedData encoding="raw">
raw data...
  </AppendedData>
</VTKFile>
```

Rectilinear Gridのノード上のスカラを出力

```
use :: vtk_fortran, only: vtk_file
implicit none

type(vtk_file) :: vtr !! VTK file

integer(int32), parameter :: nx1 = 1 !! x方向の配列範囲の下限
integer(int32), parameter :: nx2 = 10 !! x方向の配列範囲の上限
integer(int32), parameter :: ny1 = 1 !! y方向の配列範囲の下限
integer(int32), parameter :: ny2 = 6 !! y方向の配列範囲の上限
integer(int32), parameter :: nz1 = 1 !! z方向の配列範囲の下限
integer(int32), parameter :: nz2 = 6 !! z方向の配列範囲の上限

real(real64) :: x(nx1:nx2) !! x座標値
real(real64) :: y(ny1:ny2) !! y座標値
real(real64) :: z(nz1:nz2) !! z座標値
real(real64) :: v(nx1:nx2, ny1:ny2, nz1:nz2) !! スカラ値

integer(int32) :: stat !! 手続の実行結果の状態
```

Rectilinear Gridのノード上のスカラを出力

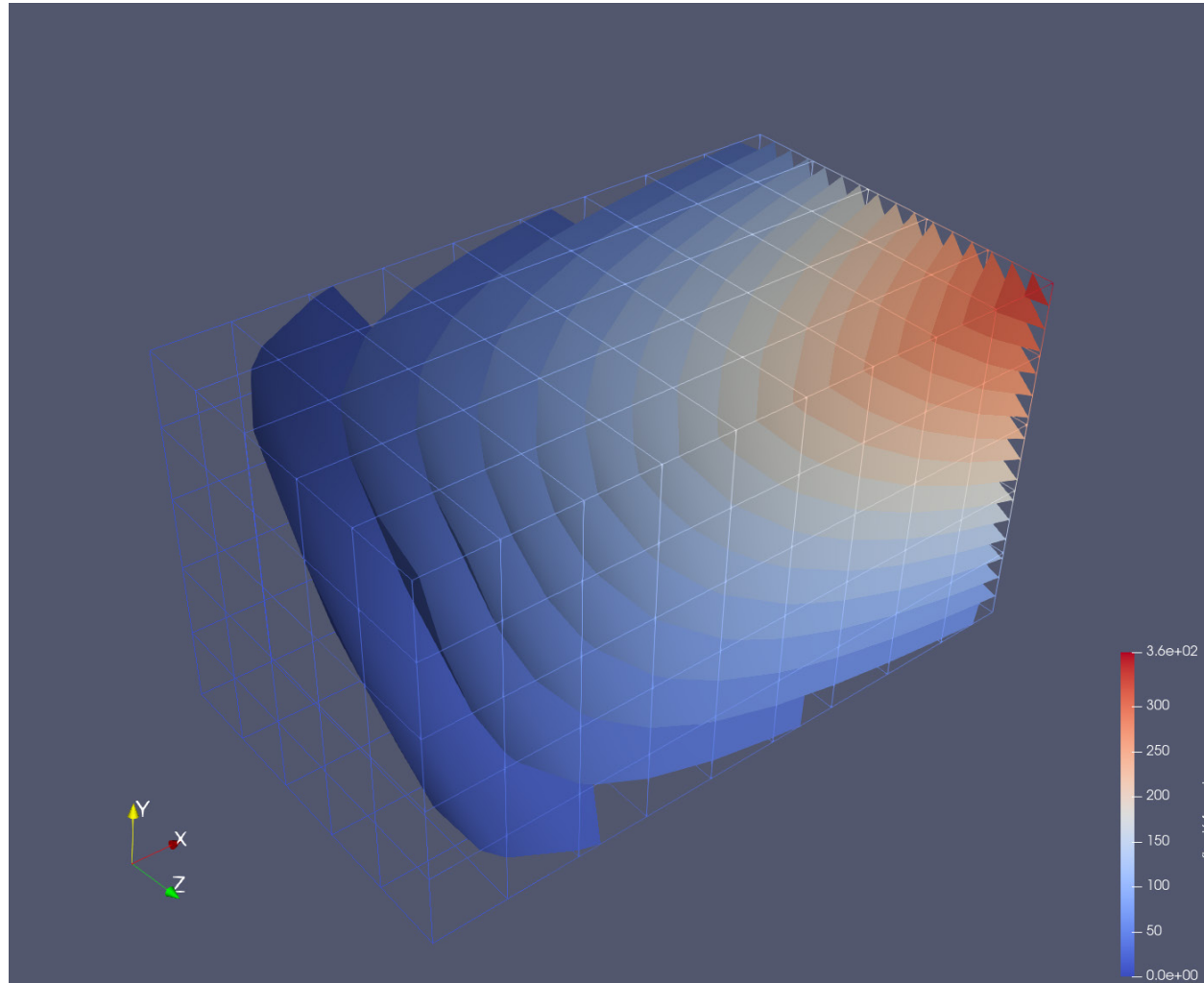
```
! ファイルを開き, VTKFileタグとStructuredGridタグを開く.
! <VTKFile type="RectilinearGrid" version="1.0" byte_order="LittleEndian">
!   <RectilinearGrid WholeExtent="+1 +10 +1 +6 +1 +6">
stat = vtr%initialize(format="raw", &
                      filename="xml_rectilinear_grid_raw_scalar.vtr", &
                      mesh_topology="RectilinearGrid", &
                      nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)

! Pieceタグ open.
! <Piece Extent="+1 +10 +1 +6 +1 +6">
stat = vtr%xml_writer%write_piece(nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)
! 各方向座標値の出力情報の書き出し.
! 実際のデータは </StructuredGrid>と</VTKFile>の間に,
! <AppendedData encoding="raw"></AppendedData>が設けられ,
! そこに書き出される.
!
! <Coordinates>
!   <DataArray type="Float64" NumberOfComponents="1" Name="X" format="appended" offset="0"/>
!   <DataArray type="Float64" NumberOfComponents="1" Name="Y" format="appended" offset="84"/>
!   <DataArray type="Float64" NumberOfComponents="1" Name="Z" format="appended" offset="136"/>
! </Coordinates>
stat = vtr%xml_writer%write_geo(x=x, y=y, z=z)
```

Rectilinear Gridのノード上のスカラを出力

```
! 配列の書き出し.  
! 実際のデータはAppendedDataに追記される.  
!  
! <PointData>  
stat = vtr$xml_writer%write_dataarray(location="node", action="open")  
!   <DataArray type="Float64" NumberOfComponents="1" Name="float64_scalar" format="appended"  
offset="188"/>  
stat = vtr$xml_writer%write_dataarray(x=v, data_name="float64_scalar", one_component=.true.)  
! </PointData>  
stat = vtr$xml_writer%write_dataarray(location="node", action="close")  
! Pieceタグ close  
! </Piece>  
stat = vtr$xml_writer%write_piece()  
! VTKFileタグとStructuredGridタグ close.  
! vtrファイルを閉じる.  
!   </StructuredGrid>  
! </VTKFile>  
stat = vtr%finalize()
```

Rectilinear Gridのノード上のスカラを出力



Rectilinear Gridのノード上のベクトルを出力

▶ 出力されるvtrファイル

```
<?xml version="1.0"?>
<VTKFile type="RectilinearGrid" version="1.0" byte_order="LittleEndian">
  <RectilinearGrid WholeExtent="+1 +10 +1 +6 +1 +6">
    <Piece Extent="+1 +10 +1 +6 +1 +6">
      <Coordinates>
        <DataArray type="Float64" NumberOfComponents="1" Name="X" format="appended" offset="0"/>
        <DataArray type="Float64" NumberOfComponents="1" Name="Y" format="appended" offset="84"/>
        <DataArray type="Float64" NumberOfComponents="1" Name="Z" format="appended" offset="136"/>
      </Coordinates>
      <PointData>
        <DataArray type="Float64" NumberOfComponents="3" Name="float64_vector" format="appended" offset="188"/>
      </PointData>
    </Piece>
  </RectilinearGrid>
  <AppendedData encoding="raw">
raw data...
  </AppendedData>
</VTKFile>
```

Rectilinear Gridのノード上のベクトルを出力

```
use :: vtk_fortran, only: vtk_file
implicit none

type(vtk_file) :: vtr !! VTK file

integer(int32), parameter :: nx1 = 1 !! x方向の配列範囲の下限
integer(int32), parameter :: nx2 = 10 !! x方向の配列範囲の上限
integer(int32), parameter :: ny1 = 1 !! y方向の配列範囲の下限
integer(int32), parameter :: ny2 = 6 !! y方向の配列範囲の上限
integer(int32), parameter :: nz1 = 1 !! z方向の配列範囲の下限
integer(int32), parameter :: nz2 = 6 !! z方向の配列範囲の上限

real(real64) :: x(nx1:nx2) !! x座標値
real(real64) :: y(ny1:ny2) !! y座標値
real(real64) :: z(nz1:nz2) !! z座標値
real(real64) :: u(nx1:nx2, ny1:ny2, nz1:nz2) !! ベクトル値のx成分
real(real64) :: v(nx1:nx2, ny1:ny2, nz1:nz2) !! ベクトル値のy成分
real(real64) :: w(nx1:nx2, ny1:ny2, nz1:nz2) !! ベクトル値のz成分

integer(int32) :: stat !! 手続の実行結果の状態
```

Rectilinear Gridのノード上のベクトルを出力

```
! ファイルを開き, VTKFileタグとStructuredGridタグを開く.
! <VTKFile type="RectilinearGrid" version="1.0" byte_order="LittleEndian">
!   <RectilinearGrid WholeExtent="+1 +10 +1 +6 +1 +6">
stat = vtr%initialize(format="raw", &
                      filename="xml_rectilinear_grid_raw_vector.vtr", &
                      mesh_topology="RectilinearGrid", &
                      nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)

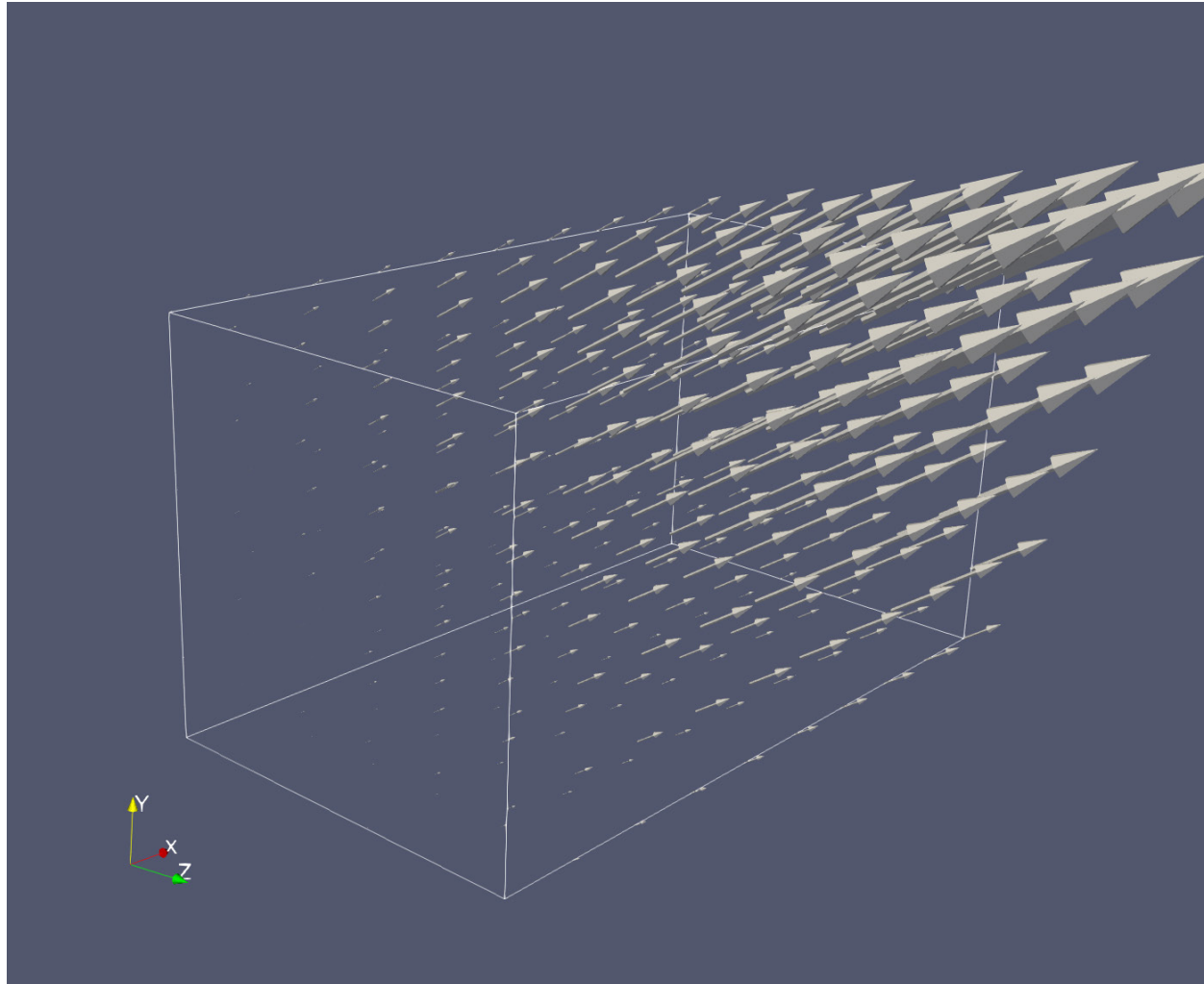
! Pieceタグ open.
! <Piece Extent="+1 +10 +1 +6 +1 +6">
stat = vtr%xml_writer%write_piece(nx1=nx1, nx2=nx2, ny1=ny1, ny2=ny2, nz1=nz1, nz2=nz2)

! 各方向座標値の出力情報の書き出し.
! 実際のデータは </StructuredGrid>と</VTKFile>の間に,
! <AppendedData encoding="raw"></AppendedData>が設けられ,
! そこに書き出される.
! <Coordinates>
!   <DataArray type="Float64" NumberOfComponents="1" Name="X" format="appended" offset="0"/>
!   <DataArray type="Float64" NumberOfComponents="1" Name="Y" format="appended" offset="84"/>
!   <DataArray type="Float64" NumberOfComponents="1" Name="Z" format="appended" offset="136"/>
! </Coordinates>
stat = vtr%xml_writer%write_geo(x=x, y=y, z=z)
```

Rectilinear Gridのノード上のベクトルを出力

```
! 配列の書き出し.  
! 実際のデータはAppendedDataに追記される.  
!  
! <PointData>  
stat = vtr$xml_writer%write_dataarray(location="node", action="open")  
!   <DataArray type="Float64" NumberOfComponents="3" Name="float64_vector" format="appended"  
offset="188"/>  
stat = vtr$xml_writer%write_dataarray(x=u, y=v, z=w, data_name="float64_vector", is_tuples=.false.)  
! </PointData>  
stat = vtr$xml_writer%write_dataarray(location="node", action="close")  
  
! Pieceタグ close  
! </Piece>  
stat = vtr$xml_writer%write_piece()  
  
! VTKFileタグとStructuredGridタグ close.  
! vtrファイルを閉じる.  
!   </StructuredGrid>  
! </VTKFile>  
stat = vtr%finalize()
```

Rectilinear Gridのノード上のベクトルを出力



組み合わせた例

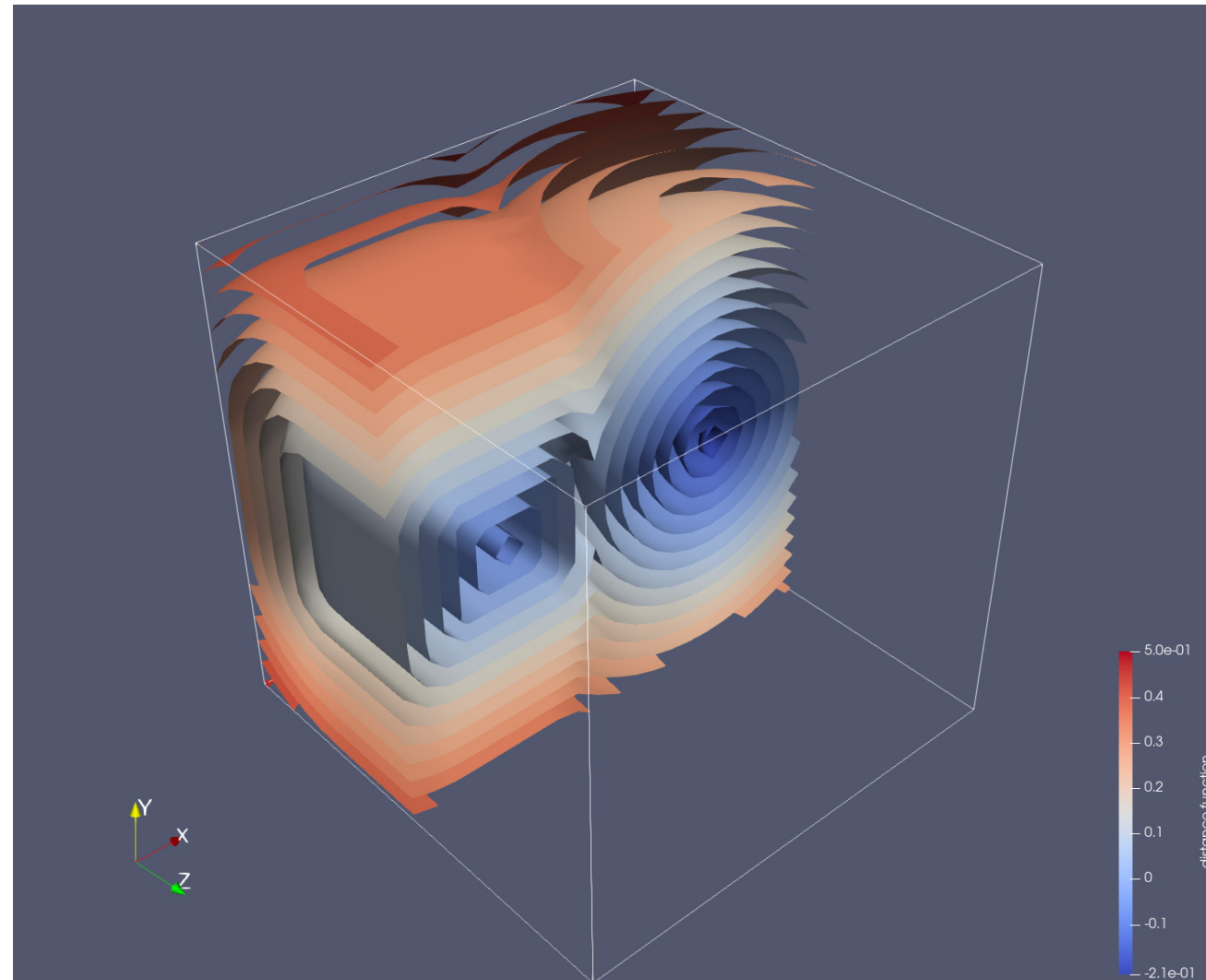
- ▶ json形式で書かれた設定ファイルから設定を読み込み，距離関数を計算してvtrファイルに出力する
- ▶ GitHub
 - ▶ `https://github.com/degawa/ex_io`

```
> git clone https://github.com/degawa/ex_io
> cd ex_io
> fpm build --profile release --flag "-D_R16P"
> fpm run
```

設定ファイル

```
{
  "physical conditions": {
    "space": {
      "x": {
        "min": 0.0,
        "max": 1.0
      },
      "y": {
        "min": 0.0,
        "max": 1.0
      },
      "z": {
        "min": 0.0,
        "max": 1.0
      }
    },
    "discrete conditions": {
      "number of grid points": {
        "x": 21,
        "y": 21,
        "z": 21
      }
    }
  },
  "objects": {
    "objects name": [
      "rect",
      "sphere"
    ],
    "rect": {
      "center": {
        "x": 0.25,
        "y": 0.5,
        "z": 0.5
      },
      "length": {
        "x": 0.25,
        "y": 0.25,
        "z": 0.5
      }
    },
    "sphere": {
      "center": {
        "x": 0.75,
        "y": 0.5,
        "z": 0.5
      },
      "radius": 0.25
    }
  }
}
```

出力された距離関数



入力を減らす

入力を減らす

- ▶ VSCode
 - ▶ VSCodeの拡張機能
 - ▶ コードスニペット
 - ▶ タスクの設定
- ▶ fypp – python powered Fortran metaprogramming
 - ▶ Pythonを利用したFortranのメタプログラミングツール

VSCode - Visual Studio Code

- ▶ Microsoftが開発しているソースコードエディタ
 - ▶ シンタックスハイライト, コードスニペット, IntelliSense, リファクタリング, デバッグ, Git/GitHubとの連携
- ▶ 拡張機能を利用して様々な機能を追加
 - ▶ VSCode本体のFortranサポートは皆無
 - ▶ 拡張機能を追加することでシンタックスハイライト, コードスニペット, IntelliSense, リファクタリングが利用可能

Modern Fortran

- ▶ Fortran言語サポートを提供
 - ▶ シンタックスハイライト
 - ▶ コードスニペット
 - ▶ マウスホバーで組込関数のドキュメントを表示(`ctrl+k` `ctrl+i`)
 - ▶ gfortranを用いたコードの静的解析
 - ▶ シンボルプロバイダ (`ctrl+shift+o`)
 - ▶ 変数, 関数, サブルーチン定義への移動
 - ▶ デバッグとの連携
 - ▶ findentもしくはfprettifyを用いたコード整形

Modern Fortran

- ▶ gfortranを用いたコードの静的解析
 - ▶ ソースファイルを開くとlint用の.modを作成
 - ▶ 異なるディレクトリのmodファイルは参照できない
 - ▶ srcディレクトリに余分なものが作られる
 - ▶ linterModOutputを設定することで、modファイルを一箇所に集約
 - ▶ fpmのプロジェクトルート (=VSCodeのワークスペースのルート) に.vscodeディレクトリとmodの出力ディレクトリを作成
 - ▶ .vscode内にsettings.jsonファイルを作成し、設定を記述

```
{  
  "fortran.linterModOutput": "D:¥¥hpfpfc_fortran_oss¥¥ex_stdlib¥¥.mod"  
}
```

FORTRAN IntelliSense

- ▶ Fortranに対するIntelliSenseサポートを提供
 - ▶ ユーザ定義の手続のインタフェースの表示
 - ▶ Doxygen, FORDスタイルのドキュメントを表示
 - ▶ 入力補完
 - ▶ 変数の定義・参照位置への移動
 - ▶ 手続の定義・実装・参照位置への移動
 - ▶ シンボル（変数・関数・サブルーチン）の検索, 名前の変更
 - ▶ コードの診断

FORTRAN IntelliSense

- ▶ Fortran-Language-Server
 - ▶ FORTRAN IntelliSenseが要求
 - ▶ VSCodeだけでなくAtom, Visual Studio, vim, Emacsでも利用可能
- ▶ GitHubリポジトリ
 - ▶ <https://github.com/hansec/fortran-language-server>
- ▶ インストール
 - ▶ `> pip install fortran-language-server`
 - ▶ WindowsでAnacondaを使っている場合は、仮想環境を作ること！

他のFortran向け拡張機能

- ▶ Fortran Break Point
 - ▶ ブレークポイント機能を提供する
- ▶ vscode-modern-fortran-formatter
 - ▶ fprettifyを用いたコード整形
 - ▶ VSCodeでコード整形を利用する場合は、下記2項目を設定

Editor: Format On Save

☒ ファイルを保存するときにフォーマットします。フォーマッタが有効でなければなりません。ファイルの遅延保存やエディターを閉じることは許可されていません。

Editor: Default Formatter

他のすべてのフォーマッタ設定よりも優先される、既定のフォーマッタを定義します。フォーマッタを提供している拡張機能の識別子にする必要があります。

vscode-modern-fortran-formatter

←他形式の整形が効かなくなるので、解決策を模索中

ユーザズニペット

- ▶ 入力されたキーワード (prefix) をまとめたコード片に変換
 - ▶ r8+Tabでreal(real64)が入力される

```
"type declaration of binary64": {  
  "prefix": "r8",  
  "body": "real(real64)",  
  "description": "type declaration of binary32"  
},
```

- ▶ \$+数字を利用すると, 入力すべき箇所にカーソルを移動

```
"allocation inspection": {  
  "prefix": "allocated?",  
  "body": "if(.not.allocated($1)) allocate($1($0))",  
  "description": "allocated function"  
},
```

複数行のコードへの変換

- ▶ bodyに複数行のコード片を書く

```
"module template": {  
  "prefix": "mod",  
  "body": [  
    "module $1",  
    "  use, intrinsic :: iso_fortran_env",  
    "  implicit none",  
    "  private",  
    "  $0",  
    "end module $1"  
  ],  
  "description": "module routine template"  
},
```

タスクの設定

- ▶ fpmではビルド，テスト，インストールの度に同じコンパイルオプションが必要
- ▶ VSCodeと外部ツールの連携
 - ▶ fpmのプロジェクトルートに.vscodeディレクトリを作成
 - ▶ .vscode内にtasks.jsonファイルを作成し，外部ツールの呼出しを記述
 - ▶ コンパイルオプションをtasks.jsonに記述
- ▶ タスクを使う利点
 - ▶ ショートカットキーで実行できる
 - ▶ 未保存のファイルを自動で保存してくれる

ビルドタスクの登録

▶ ビルドタスクの設定

- ▶ groupでkind: build, isDefault: trueに設定すると, ctrl+shift+bでタスクが実行される

```
"tasks": [  
  {  
    "label": "fpm build",  
    "type": "shell",  
    "options": {  
      "cwd": "${workspaceFolder}",  
    },  
    "command": "fpm",  
    "args": ["build", "--profile", "release", "--flag", "¥"-D_R16P¥""],  
    "group": {  
      "kind": "build",  
      "isDefault": true  
    },  
  },  
],
```

他のタスクの登録

- ▶ テストタスク
 - ▶ ビルドタスクとほぼ同じ
 - ▶ argsの"build"を"test"に変更
 - ▶ groupのkindをtestに変更

```
"command": "fpm",  
"args": ["test", "--profile", "release", "--flag", "¥"-D_R16P¥"],  
"group": {  
  "kind": "test",  
  "isDefault": true  
},
```

- ▶ 同じ要領でfpm run, fpm installも登録できる
 - ▶ groupはnone

fypp

- ▶ Python powered Fortran metaprogramming
- ▶ Fortran用のメタプログラミングツール
- ▶ GitHub
 - ▶ <https://github.com/aradi/fypp>
- ▶ ドキュメント
 - ▶ <https://fypp.readthedocs.io/en/stable/>

fypの用途

▶ 手続のテンプレートの作成

- ▶ Fortranでは，総称名を使えば同じ名前で異なる型の引数を取る手続を定義できる

```
interface is_positive
  procedure :: is_positive_int8
  procedure :: is_positive_int16
  procedure :: is_positive_int32
  procedure :: is_positive_int64
  procedure :: is_positive_real32
  procedure :: is_positive_real64
end interface

is_positive(-1_int8), is_positive(-1._real64)
```

- ▶ 異なる型に対して同じ処理を何回も書く必要がある

```
logical function is_positive_int8(val)
  integer(int8), intent(in) :: val
  is_positive_int8 = .false.
  if (val >= 0_int8) then
    is_positive_int8 = .true.
  end if
end function is_positive_int8

logical function is_positive_real64(val)
  real(real64), intent(in) :: val
  is_positive_real64 = .false.
  if (val >= 0.0_real64) then
    is_positive_real64 = .true.
  end if
end function is_positive_real64
```

fypの用途

- ▶ 手続のテンプレートの作成
 - ▶ テンプレートからFortranコードを作成
 - ▶ Pythonを使っている人には比較的簡単

```
#:for k, t in IR_KINDS_TYPES

logical function is_positive_${k}$ (val)
  implicit none
  ${t}$, intent(in) :: val

  is_positive_${k}$ = .false.
  if (val >= 0${decimal_suffix(t)}_${k}$) then
    is_positive_${k}$ = .true.
  end if
end function is_positive_${k}$
#:endfor
```



```
logical function is_positive_int8(val)
  integer(int8), intent(in) :: val
  is_positive_int8 = .false.
  if (val >= 0_int8) then
    is_positive_int8 = .true.
  end if
end function is_positive_int8

logical function is_positive_int16(val)
  integer(int16), intent(in) :: val
  is_positive_int16 = .false.
  if (val >= 0_int16) then
    is_positive_int16 = .true.
  end if
end function is_positive_int16

logical function is_positive_int32(val)
  integer(int32), intent(in) :: val
  is_positive_int32 = .false.
  if (val >= 0_int32) then
    is_positive_int32 = .true.
  end if
end function is_positive_int32

logical function is_positive_int64(val)
  integer(int64), intent(in) :: val
  is_positive_int64 = .false.
  if (val >= 0_int64) then
    is_positive_int64 = .true.
  end if
end function is_positive_int64

logical function is_positive_real32(val)
  real(real32), intent(in) :: val
  is_positive_real32 = .false.
  if (val >= 0.0_real32) then
    is_positive_real32 = .true.
  end if
end function is_positive_real32

logical function is_positive_real64(val)
  real(real64), intent(in) :: val
  is_positive_real64 = .false.
  if (val >= 0.0_real64) then
    is_positive_real64 = .true.
  end if
end function is_positive_real64
```


インストール

▶ Conda

- ▶ `> conda install -c conda-forge fypp`

▶ pip

- ▶ `> pip install --user fypp`

- ▶ WindowsでAnacondaを使っている場合は，仮想環境を作ること！

▶ MSYS2

- ▶ `> pacman -S mingw-w64-x86_64-python-fypp`

fyppの例

- ▶ fpyyの各機能を検証するためのテンプレート
- ▶ GitHub
 - ▶ https://github.com/degawa/ex_fypp

```
> git clone https://github.com/degawa/ex_fypp
> cd ex_fypp
> cd src
> fypp is_positive.fy90 is_positive.f90 --include=inc
> fypp mean.fy90 mean.f90 --include=inc
> fypp ..
> fpm build --flag "-cpp"
> fpm run --flag "-cpp" --example example
```

fyppの基本

▶ fyppのディレクティブ

- ▶ `#:`, `#!` `{}`, `$:`, `${}`, `@:`
- ▶ `#:if` ~ `#:elif` ~ `#:else` ~ `#:endif`
- ▶ `#:def` ~ `#:enddef`
- ▶ `#:for`, `#:set`, `#:del`

▶ fyppの実行

- ▶ `> fypp 入力 出力`
- ▶ 入出力はファイルでも標準入出力でもよい
- ▶ ファイル拡張子に制限はないが、fyppがよく使われる

識別子の定義

▶ #:if defined('識別子')

▶ ifdef.fy90

```
program ifdef
  implicit none

  #! 定数DEBUGは文字列としてdefined()に渡す.
  #:if defined('DEBUG')
    print *, "debug print"
  #:endif
end program ifdef
```

> fypp ifdef.fy90 ifdef.f90

```
program ifdef
  implicit none

end program ifdef
```

> fypp ifdef.fy90 ifdef.f90 --define=DEBUG

```
program ifdef
  implicit none

    print *, "debug print"
end program ifdef
```

インラインディレクティブ

▶ #{}#

▶ {}の中にディレクティブを記述

```
program ifdef
  implicit none

  #{if defined('DEBUG')}# print *, "debug"
  #{else}# print *, "release" #{endif}#
end program ifdef
```

> fypp ifdef.fy90 ifdef.f90

```
program ifdef
  implicit none

  print *, "release"
end program ifdef
```

> fypp ifdef.fy90 ifdef.f90 --define=DEBUG

```
program ifdef
  implicit none

  print *, "debug"
end program ifdef
```

set, forディレクティブによるコード生成

- ▶ setでリストを定義
- ▶ forでリストの要素を一つずつ取り出して反映

> fypp

```
program for
  use, intrinsic :: iso_fortran_env
  implicit none
  #! integer kinds
  #:set INTEGER_KINDS = ['int8', 'int16', 'int32', 'int64']

  #:for int_kind in INTEGER_KINDS
    integer(${int_kind}$), parameter :: zero_${int_kind}$ = 0_${int_kind}$
  #:endfor
end program for
```

```
program for
  use, intrinsic :: iso_fortran_env
  implicit none

  integer(int8), parameter :: zero_int8 = 0_int8
  integer(int16), parameter :: zero_int16 = 0_int16
  integer(int32), parameter :: zero_int32 = 0_int32
  integer(int64), parameter :: zero_int64 = 0_int64
end program for
```

defディレクティブによるマクロの定義

- ▶ `def` マクロ名(引数) ~ `#:enddef` でマクロを定義
 - ▶ `@:マクロ名(引数), $:マクロ名("引数")`
 - ▶ `${マクロ名(引数)}$`

```
#:def ASSERT(cond)
  if (.not. ${cond}$) then
    write (error_unit, '(3A,I0,A)') "assertion failed: ", __FILE__, ", line ", __LINE__, "."
    error stop
  end if
#:enddef ASSERT

@:ASSERT(result == 3)
```



```
if (.not. result == 3) then
  write (error_unit, '(3A,I0,A)') "assertion failed: ", __FILE__, ", line ", __LINE__, "."
  error stop
end if
```

defディレクティブによる関数の定義

- ▶ defは単純な置換
- ▶ 関数のような動作を定義することも可能

```
#:set IR_TYPES = ['integer(int32)', 'real(real32)']

#! typeとして渡される型の名前にrealが含まれていたら.0を
返し, それ以外は何も返さない.
#:def decimal_suffix(type)
#{if 'real' in type}#.0#{endif}#
#:enddef

#:for t in IR_TYPES
! 型名tに"real"が含まれていたら0.0になる
0${decimal_suffix(t)}$
#:endfor
```

```
#! 数字を文字に変換する
#:def toString(i)
${"{}".format(i)}
#:enddef

#! rankとして渡される配列のランクが1以上なら, rankに応
じた数のコロンを返す.
#:def rank_suffix(rank)
#{if rank > 0}#{ "${":" + ","*(rank - 1)}$"}#{endif}#
#:enddef

#:for rank in range(1, 15+1)
#:set str_rank = toString(rank)
! rankの値に応じてx(:), x(:, :), x(:, :, :)となる
x${rank_suffix(rank)}$
#:endfor
```


異なる型を引数に取る手続

```
#:set I_KINDS = ['int8', 'int16', 'int32', 'int64']
#:set I_TYPES = ['integer(int8)', ...]
#:set R_KINDS = ['real32', 'real64']
#:set R_TYPES = ['real(real32)', 'real(real64)']
```

```
#:set IR_KINDS = I_KINDS + R_KINDS
#:set IR_TYPES = I_TYPES + R_TYPES
#:set IR_KINDS_TYPES = list(zip(IR_KINDS, IR_TYPES))
```

```
interface is_positive
  #:for k in IR_KINDS
    procedure :: is_positive_${k}$
  #:endfor
end interface
contains
#:for k, t in IR_KINDS_TYPES

  logical function is_positive_${k}$(val)
    implicit none
    ${t}$, intent(in) :: val

    is_positive_${k}$ = .false.
    if (val >= 0${decimal_suffix(t)}_${k}$) then
      is_positive_${k}$ = .true.
    end if
  end function is_positive_${k}$
#:endfor
```



```
interface is_positive
  procedure :: is_positive_int8
  procedure :: is_positive_int16
  procedure :: is_positive_int32
  procedure :: is_positive_int64
  procedure :: is_positive_real32
  procedure :: is_positive_real64
end interface
```

contains

```
logical function is_positive_int8(val)
  integer(int8), intent(in) :: val
  is_positive_int8 = .false.
  if (val >= 0_int8) then
    is_positive_int8 = .true.
  end if
end function is_positive_int8
```

```
logical function is_positive_int16(val)
  integer(int16), intent(in) :: val
  is_positive_int16 = .false.
  if (val >= 0_int16) then
    is_positive_int16 = .true.
  end if
end function is_positive_int16
```

```
logical function is_positive_int32(val)
  integer(int32), intent(in) :: val
  is_positive_int32 = .false.
  if (val >= 0_int32) then
    is_positive_int32 = .true.
  end if
end function is_positive_int32
```

```
logical function is_positive_int64(val)
  integer(int64), intent(in) :: val
  is_positive_int64 = .false.
  if (val >= 0_int64) then
    is_positive_int64 = .true.
  end if
end function is_positive_int64
```

```
logical function is_positive_real32(val)
  real(real32), intent(in) :: val
  is_positive_real32 = .false.
  if (val >= 0.0_real32) then
    is_positive_real32 = .true.
  end if
end function is_positive_real32
```

```
logical function is_positive_real64(val)
  real(real64), intent(in) :: val
  is_positive_real64 = .false.
  if (val >= 0.0_real64) then
    is_positive_real64 = .true.
  end if
end function is_positive_real64
```

異なるランクの配列を引数に取る手続

```
#!/usr/bin/perl
# rankとして渡される配列のランクが1以上なら、rankに応じた
# 数のコロンを返す。
# def rank_suffix(rank)
#   if rank > 0
#     printf "%d", rank
#   else
#     printf ""
#   end
# end

```

```
interface mean
  #:for rank in range(1, 15+1)
    procedure :: mean_r64_rank${k}$
  #:endfor
end interface
contains
#:for rank in range(1, 15+1)
function mean_r64_rank${rank}$(x) result(res)
  real(real64), intent(in) :: x${rank_suffix(rank)}$
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank${rank}$
#:endfor
```



```

interface mean
  procedure :: mean_r64_rank1
  procedure :: mean_r64_rank2
  procedure :: mean_r64_rank3
  procedure :: mean_r64_rank4
  procedure :: mean_r64_rank5
  procedure :: mean_r64_rank6
  procedure :: mean_r64_rank7
  procedure :: mean_r64_rank8
  procedure :: mean_r64_rank9
  procedure :: mean_r64_rank10
  procedure :: mean_r64_rank11
  procedure :: mean_r64_rank12
  procedure :: mean_r64_rank13
  procedure :: mean_r64_rank14
  procedure :: mean_r64_rank15

end interface

contains

function mean_r64_rank1(x) result(res)
  real(real64), intent(in) :: x(:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank1

function mean_r64_rank2(x) result(res)
  real(real64), intent(in) :: x(:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank2

function mean_r64_rank3(x) result(res)
  real(real64), intent(in) :: x(:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank3

function mean_r64_rank4(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank4

function mean_r64_rank5(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank5

function mean_r64_rank6(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank6

function mean_r64_rank7(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank7

function mean_r64_rank8(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank8

function mean_r64_rank9(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank9

function mean_r64_rank10(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank10

function mean_r64_rank11(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank11

function mean_r64_rank12(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank12

function mean_r64_rank13(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank13

function mean_r64_rank14(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank14

function mean_r64_rank15(x) result(res)
  real(real64), intent(in) :: x(:,:,:,:,:,:,:,:,:,:,:,:,:,:,:)
  real(real64) :: res

  res = sum(x) / real(size(x, kind = int64), real64)
end function mean_r64_rank15

```

迷いを減らす

fprettify

- ▶ Fortran用のソースファイル自動成形ツール
 - ▶ 自由形式のみをサポート
- ▶ GitHub
 - ▶ <https://github.com/pseewald/fprettify>

コードスタイルの多様性

- ▶ インデント
 - ▶ 2文字, 3文字, 4文字
- ▶ if文やwrite文の後ろの空白
 - ▶ 空けない (`if(),read*,write()`) , 空ける (`if (),read *,write ()`)
- ▶ 文や組込手続
 - ▶ 大文字, 小文字
- ▶ 演算子間のスペース
 - ▶ `=*/`の間は空けるが`+-`は空けない, 全て空ける

コードスタイルの多様性

▶ module文に対するインデント

▶ する,しない

```
module mod_hoge
contains
  subroutine hoge()
    real(real32)...
  end subroutine hoge
end module mod_hoge
```

```
module mod_hoge
contains
subroutine hoge()
    real(real32)...
end subroutine hoge
end module mod_hoge
```

▶ 多重ループ内のインデント

▶ 強制する,しない

```
do k =...
  do j =...
    do i =...
      end do
    end do
  end do
```

```
do k =...
do j =...
do i =...
end do
end do
end do
```

fprettifyの機能

- ▶ インデント幅の指定
- ▶ 行継続時のぶら下げインデント
- ▶ 演算子や区切り記号(, : %)前後の空白の指定
- ▶ 2行以上の空行の削除
- ▶ 組込手続の大文字/小文字の変換
- ▶ 論理演算子のスタイル(.eq., ==)の変換
- ▶ cpp, fyppディレクティブとの協調
- ▶ エディタとの統合

インストールと実行

▶ pip

- ▶ `> pip install fprettify`

- ▶ WindowsでAnacondaを使っている場合は，仮想環境を作ること！

▶ Setuptools

- ▶ `> https://github.com/pseewald/fprettify.git`
`> cd fprettify`
`> ./setup.py install`

▶ 実行

- ▶ `> fprettify ファイル1 ファイル2 ...`

fprettifyの例

- ▶ fprettifyの各機能を検証するためのソース

- ▶ GitHub

- ▶ https://github.com/degawa/ex_fprettify

```
> git clone https://github.com/degawa/ex_fprettify
> cd ex_fprettify
> fprettify --recursive . --fortran f90 --fortran f08
```

fprettify使用時の注意

- ▶ fprettifyは整形した内容で入力ファイルを上書きする
- ▶ 整形したくない場合
 - ▶ 文末に!&を付ける
 - ▶ !&< ~ !&>で囲む
 - ▶ fyppを使っていると混同するが!&< ~ !>&ではない
- ▶ VSCodeとの連携時，ファイルの内容が消えることがある
 - ▶ 拡張機能に起因
 - ▶ 特に&が関係する場合に発生
 - ▶ 慌てずにctrl+zで戻す

インデント制御

▶ インデント幅の制御

▶ > fprettify --indent インデント数 ファイル名

▶ program, module内のインデントの無効化

▶ > fprettify --disable-indent-mod ファイル名

```
module type_vector2
contains
  subroutine clear_vector2(vec)
    use, intrinsic :: iso_fortran_env
    implicit none
    type(vector2), intent(inout) :: vec
    vec%x = 0d0
    vec%y = 0d0
  end subroutine clear_vector2
end module type_vector2
```



```
module type_vector2
contains
  subroutine clear_vector2(vec)
    use, intrinsic :: iso_fortran_env
    implicit none
    type(vector2), intent(inout) :: vec
    vec%x = 0d0
    vec%y = 0d0
  end subroutine clear_vector2
end module type_vector2
```

インデント制御

- ▶ 多重ループのインデント

- ▶ > fprettify --strict-indent ファイル名

```
block
  integer(int32) :: i, j, k, Nx, Ny, Nz
  do k = 1, Nz
  do j = 1, Ny
  do i = 1, Nx
    print *
  end do
  end do
  end do
end block
```



```
block
  integer(int32) :: i, j, k, Nx, Ny, Nz
  do k = 1, Nz
    do j = 1, Ny
      do i = 1, Nx
        print *
      end do
    end do
  end do
end block
```

- ▶ インデント制御の無効化

- ▶ > fprettify --disable-indent ファイル名

- ▶ インデント幅を変えずに、空白などを整形したい場合に利用

空白の制御

▶ --whitespace プリセット

- ▶ プリセットは0, 1, 2(標準), 3, 4から選択

| プリセット番号 | 空白が入る場所 |
|---------|--|
| 0 | カンマの後, セミコロンの後, 行継続記号&の前, thenの前 |
| 1 | プリセット0で空白が入る場所, 代入・関係演算子・論理演算子の前後 print, read文と書式の間 print* -> print * endと文の名前の間 enddo -> end do 文の名前とその後の括弧の間 if() -> if () カンマの後 |
| 2 | プリセット1で空白が入る場所, +, -の前後 |
| 3 | プリセット2で空白が入る場所, *, /の前後 |
| 4 | プリセット3で空白が入る場所, 派生型の成分を指定する%の前後 |

空白の制御

- ▶ `--enable-decl`

- ▶ `use`や変数宣言のセミコロン`::`を整形の対象にする

- ▶ `--whitespace-decl`

- ▶ `--enable-decl`と併せて利用
 - ▶ パラメータ`true/false`で`::`前後に空白を入れる/削除を切り替え

- ▶ `> fprettify --enable-decl --whitespace-decl true ファイル名`

```
use,intrinsic::iso_fortran_env
use::type_vector2
real(real64),parameter::pi=acos(-1d0)
```



```
use, intrinsic :: iso_fortran_env
use :: type_vector2
real(real64), parameter :: pi = acos(-1d0)
```

空白の制御

▶ 個別の空白の制御

- ▶ `--whitespace-comma`, `--whitespace-assignment`, `--whitespace-relational`, `--whitespace-logical`, `--whitespace-plusminus`, `--whitespace-multdiv`, `--whitespace-print`, `--whitespace-type`, `--whitespace-intrinsics`, `--strip-comments`
- ▶ `--whitespace-`から始まるオプションは, `true/false`で有効化/無効化

▶ 空白制御の無効化

- ▶ `> fprettify --disable-whitespace ファイル名`

組込手続のキャピタリゼーション

- ▶ `--case` パラメータ1 パラメータ2 パラメータ3 パラメータ4
- ▶ パラメータの値は0, 1, 2から選択
- ▶ 0:入力ファイルの状態を維持, 1:小文字に変換, 2:大文字に変換

| パラメータ | 大文字になる項目 | | |
|-------|---|-----------------|--------------------|
| 1 | 文 program, print, write, read, if, do 等 | | |
| 2 | 組込のモジュール名 | iso_fortran_env | -> ISO_FORTRAN_ENV |
| | 組込の手続名 | acos | -> ACOS |
| 3 | 関係演算子 | .eq., .gt. | -> .EQ., .GT. |
| | 論理演算子 | .and., .not. | -> .AND., .NOT. |
| | 論理値 | .true., .false. | -> .TRUE., .FALSE. |
| 4 | 組込の定数 | real64 | -> REAL64 |
| | 指数記号 | 1d0 | -> 1D0 |

論理演算子のスタイル

▶ --enable-replacements

- ▶ 90スタイルの関係演算子を77スタイルの関係演算子に置き換える

▶ `> fprettify --enable-replacements ファイル名`

```
if (num2 == 0 .and. num2 .lt. 0)
```



```
if (num2 .eq. 0 .and. num2 .lt. 0)
```

▶ --c-relations

- ▶ --enable-replacementsと併せて利用
- ▶ 77スタイルを90スタイルに置き換える

▶ `> fprettify --enable-replacements --c-relations ファイル名`

```
if (num2 == 0 .and. num2 .lt. 0)
```



```
if (num2 == 0 .and. num2 < 0)
```

VSCoDeとの統合

▶ vscode-modern-fortran-formatter拡張

Modern Fortran Formatter: Fprettify Args

fprettify arguments (DO NOT SET: -h, -r, -S, --version)

```
--indent 4 --line-length 256 --enable-decl --strip-comments --case 1 1 1 1
```

▶ Modern Fortran拡張

Fortran > Formatting: Args

Additional arguments for the formatter

[settings.json](#) で編集

Fortran > Formatting: Formatter

Fortran formatter, currently supports findent and fprettify

fprettify

ここ1週間ほどでModern Fortran拡張に変更が入ったため挙動が変化。
最適な運用は模索中。

その他紹介できなかったFortran向けOSS

- ▶ FORD
- ▶ TOML-Fortran
- ▶ M_CLI2
- ▶ vegetables
- ▶ Quickstart Fortran
- ▶ GTK-Fortran

FORD - Fortran Documenter

- ▶ <https://github.com/Fortran-FOSS-Programmers/ford>
- ▶ <https://github.com/Fortran-FOSS-Programmers/ford/wiki>
- ▶ ドキュメント生成ツール
 - ▶ ソースからAPIドキュメントを生成
 - ▶ Pagesとよばれるドキュメントを追加で生成可能
- ▶ Fortran-Language-Serverと連携でき，ホバーする手順の説明にドキュメントを表示可能
- ▶ Graphvizをpipとcondaでインストールする必要がある

TOML-Fortran

- ▶ <https://github.com/toml-f/toml-f>
- ▶ <https://toml-f.github.io/toml-f/>
- ▶ FortranでTOMLファイルを取り扱う
 - ▶ TOML - Tom's Obvious Minimal Language
 - ▶ 設定ファイル向けファイルフォーマット
 - ▶ 可読性が高く，小規模で明確な構文を採用
 - ▶ テーブルがある場合のデータ取得手順が直感的でなく，慣れるのに少々時間がかかる

```
[value]
integer = 1
string = "str"
logical = "true"

[value.real]
"standard notation" = 1.0
"exponential notation" = 1e-4

[array]
integer = [10, 9, 8, 7]
real = [6.0, 5.0, 4.0]
string = ["3", "2", "1", "-1"]
```

M_CLI2

- ▶ https://github.com/urbanjost/M_CLI2
- ▶ コマンドラインオプションの取り扱いを改善
 - ▶ 使いやすいインタフェース
 - ▶ `set_args(文字列)`で設定
 - ▶ `?get("オプション")`で値を取得
 - ▶ ソースファイルが一つなので取り回しがしやすい

```
call set_args('-x 1 -y 2.0 -z 3.5e0 -p 11,-22,33 --title "my title" -l F -L F')  
  
sum = rget('x') + rget('y') + rget('z')  
title = sget('title')  
p = igets('p')  
l = lget('l')  
lbig = lget('L')
```

vegetables

- ▶ <https://gitlab.com/everythingfunctional/vegetables>
- ▶ Fortranのテストツール
 - ▶ given-when-thenのBehavior-Driven Developmentスタイルのテストフレームワークを提供

```
tests = describe("is_leap_year", &
  [it("returns false for years that are not divisible by 4", &
    [example_t(integer_input_t(2002)), example_t(integer_input_t(2003))], &
    check_not_leap_year), &
  it("returns true for years that are divisible by 4 but not by 100", &
    [example_t(integer_input_t(2004)), example_t(integer_input_t(2008))], &
    check_leap_year), &
  it("returns false for years that are divisible by 100 but not by 400", &
    [example_t(integer_input_t(1900)), example_t(integer_input_t(2100))], &
    check_not_leap_year), &
  it("returns true for years that are divisible by 400", &
    [example_t(integer_input_t(2000)), example_t(integer_input_t(2400))], &
    check_leap_year) &
  ])
```

Quickstart Fortran on Windows

- ▶ <https://github.com/LKedward/quickstart-fortran>
- ▶ Windowsにgfortran, fpmをまとめてインストール
 - ▶ gfortran, fpm, OpenBLAS, make
 - ▶ intelコンパイラの環境変数を設定するバッチファイル
 - ▶ stdlibをビルド, インストールするためのバッチファイル
- ▶ インストールが不完全?
 - ▶ ビルドして実行した場合に, 必要なDLLが見つからないとのエラーが出る場合がある

GTK-Fortran

- ▶ <https://github.com/vmagnin/gtk-fortran>
- ▶ <https://github.com/vmagnin/gtk-fortran/wiki>
- ▶ GUIを構築するためのGIMP ToolkitのFortranバインディング
 - ▶ iso_c_bindingを利用し, 100%Fortranで記述
 - ▶ Windows上でのgtk-fortranの動作
 - ▶ <https://www.youtube.com/watch?v=WWY3SiVq3V8>

まとめ

- ▶ FortranのOSSの紹介と簡単な解説を行った
- ▶ 3通りの方法で開発環境を効率化した
 - ▶ 迷いを減らす - fpm, fprettify
 - ▶ 再発明を減らす - stdlib, json-fortran, vtkfortran
 - ▶ 入力を減らす - VSCode, fypp, fortran-language-server

まとめ

- ▶ 「Fortranのここが不便」は世界中でみんなが感じている
- ▶ それを自分たちで改善しようという動きが活発になっている
- ▶ あなたの感じている不便の解消にOSSが役立つかもしれない
- ▶ FortranのOSSは開発者・メンテナ不足
- ▶ 共感・協力できる方はぜひ一緒に盛り上げていきましょう