Improving Strings Support in Fortran

Presenter: Aman Godara
Thanks to: Sebastian Ehlert and Milan Curcic

Terminologies (for the talk)

- Character Sequence: a sequence of characters; intrinsic

```fortran
character(len=28) :: char_seq_variable
char_seq_variable = "this is a character sequence"
```

- String: an instance of type *string_type*; extrinsic (provided by **stdlib**)

```fortran
type(string_type) :: string_variable
string_variable = string_type( "this is a character sequence" )
```

- Stringlist: an instance of type *stringlist_type*; extrinsic (provided in **stdlib**)

```fortran
type(stringlist_type) :: stringlist_variable
stringlist_variable = stringlist_type( ["char_seq #1", "char_seq #2"] )
```

Immutability of Strings

- Immutable Data Types: once created CAN'T be modified

- Character Sequences are mutable
  - but length (*len*) is pre-specified

```
character(len=28) :: char_seq_variable
char_seq_variable = "this is a character sequence"
char_seq_variable(1:4) = "THIS"
```

- Strings are immutable; reassign to change value
  - same variable can be assigned strings of different lengths

```
type(string_type) :: string_variable
string_variable = string_type( "character sequence of length 31" )
string_variable = string_type( "This character sequence is of length 39" )
```

Advantages of Immutability

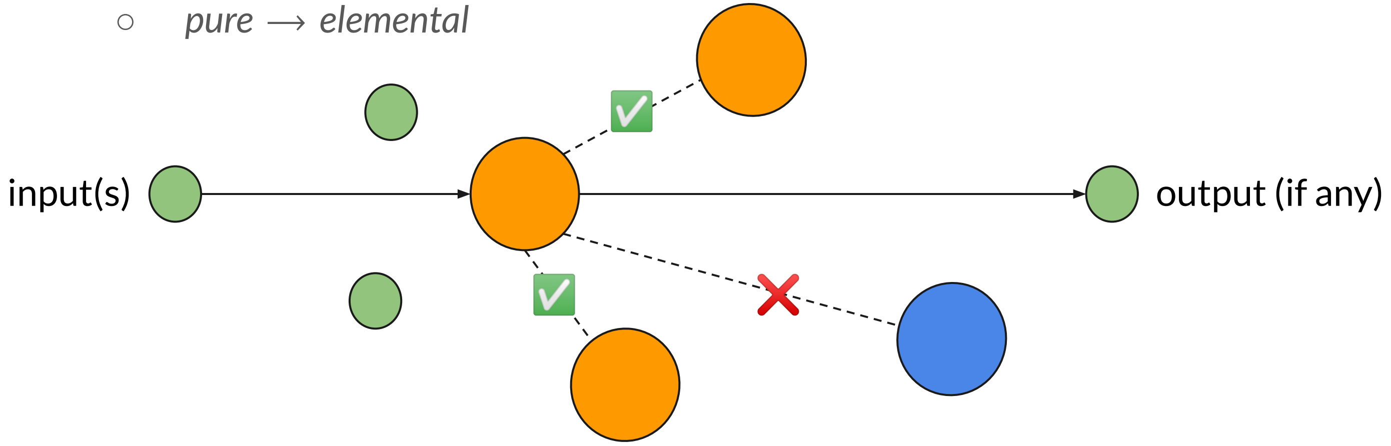
```
type(string_type) :: variable_1, variable_2
! Defined two variables

variable_1 = string_type( "This is a char-seq of length 31" )
! assigned a value (the initialised string) to variable_1

variable_2 = variable_1
! Assigned the value of variable_1 to variable_2


...
...
...
! Hidden behind ... are some operations involving variable_2 and NO variable_1

! variable_1 being unaffected by these ... operations smiles
```

∗ this behaviour can be expressed by Mutable Data Types as well

🔵 impure procedure 🟠 pure procedure 🟢 data

Features of APIs Provided

- *pure* procedures (functions & subroutines)
  - NO effects on any outside data except for input
  - *pure* ⟶ *elemental*



- functions
  - *intent(in)* arguments: NO side-effects on input(s)

Continued ...

- integration with Character Sequences, *allocatable/pointer* Character Sequences/Strings

```
type(string_type) :: string_variable
string_variable = "character sequence of length 31"
string_variable = "This character sequence is of length 39"
```

- high level APIs

```
string_variable = "demo for slice"
sliced_string = slice(string_variable, last=8)                  ! "demo for"

sliced_char_seq = slice("demo for slice", first=8, stride=-1)   ! "rof omed"

sliced_char_seq = slice("demo for slice", first=8, last=6)      ! "rof"
```

- low level APIs

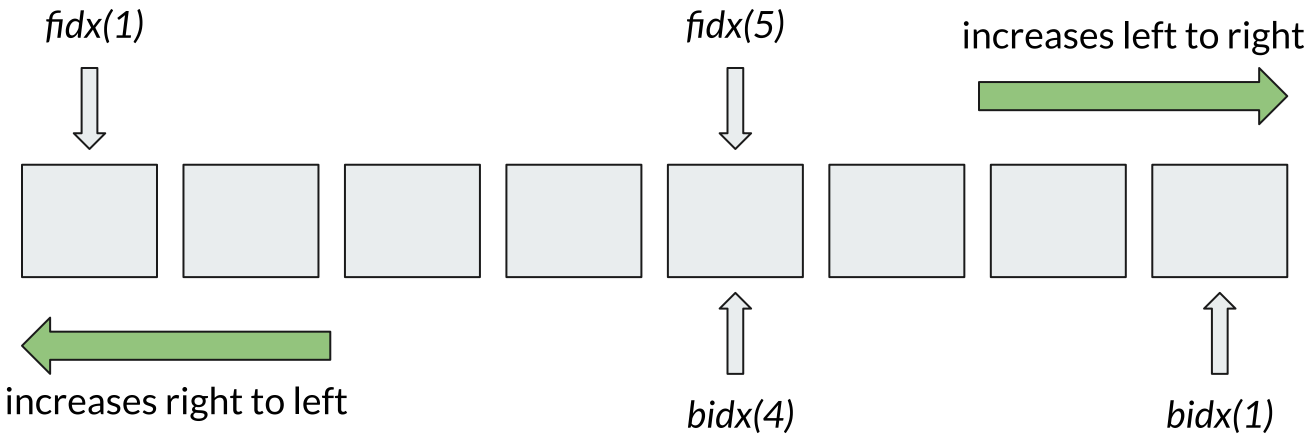
```
print *, find("qwqwqwq", pattern="qwq", occurrence=3, consider_overlapping=.true.)      ! 5
! find 3rd occurrence of "qwq" in "qwqwqwq" considering overlapping substrings
```

Underlying Implementation: *allocatable*

- more secure
- NO memory leaks

Stringlist (list of Strings)

- adopted philosophy of Strings
- forward and backward indexes through *fidx* and *bidx* functions



To Know More!

- *stdlib_string_type*: https://stdlib.fortran-lang.org/page/specs/stdlib_string_type
- *stdlib_strings*: https://stdlib.fortran-lang.org/page/specs/stdlib_strings
- *stdlib_stringlist_type*: https://stdlib.fortran-lang.org/page/specs/stdlib_stringlist_type
- *stdlib_stringlists*: under development
- *pointer* vs *allocatable* in Fortran: https://www.youtube.com/watch?v=hPBGpyX--W8 (Everything Functional)