

Tecniche di programmazione in chimica computazionale

Derived types & pointers

Emanuele Coccia

Dipartimento di Scienze Chimiche e Farmaceutiche

Derived type

- User-defined combination of existing types

type name

...

components

...

end type name

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component selector %

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component selector %
- Read and write statements for the full derived-type variable

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component **selector %**
- Read and write statements for the full derived-type variable
- Easy to treat with **single** components of the derived-type variable (as ordinary variables)

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component **selector %**
- Read and write statements for the full derived-type variable
- Easy to treat with **single** components of the derived-type variable (as ordinary variables)
- Used as dummy arguments in procedures

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component **selector %**
- Read and write statements for the full derived-type variable
- Easy to treat with **single** components of the derived-type variable (as ordinary variables)
- Used as dummy arguments in procedures
- Example **example1.f90**

Derived type

- User-defined combination of existing types
 - type name*
 - ...
 - components*
 - ...
 - end type name*
- Component **selector %**
- Read and write statements for the full derived-type variable
- Easy to treat with **single** components of the derived-type variable (as ordinary variables)
- Used as dummy arguments in procedures
- Example **example1.f90**
- Example **example2.f90**

- Ordinary variable: stores a data value

- Ordinary variable: stores a data value
- Pointer: stores the address of memory location of an ordinary variable

- Ordinary variable: stores a data value
- Pointer: stores the address of memory location of an ordinary variable
- Advantages:
 - Dynamic management of variables
 - No *a priori* knowledge of the size problem
 - Flexibility in producing a more efficient program (with caveats)

- A pointer can point to **any** variable

- A pointer can point to **any** variable
- Must specify the **target** attribute

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays
- Pointer assignment: *pointer => target*

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays
- Pointer assignment: *pointer => target*
- After the assignment, any reference to the pointer will be a reference to the **data stored** in the target

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays
- Pointer assignment: *pointer => target*
- After the assignment, any reference to the pointer will be a reference to the **data stored** in the target
- Example **p1.f90**

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays
- Pointer assignment: *pointer => target*
- After the assignment, any reference to the pointer will be a reference to the **data stored** in the target
- Example **p1.f90**
- *pointer2 => pointer1*

- A pointer can point to **any** variable
- Must specify the **target** attribute
- **Deferred-shape array specification** for arrays
- Pointer assignment: *pointer => target*
- After the assignment, any reference to the pointer will be a reference to the **data stored** in the target
- Example **p1.f90**
- **pointer2 => pointer1**
- Example **p2.f90**

Pointer association status

- Pointer status:
 - **Undefined**: just declared pointer

Pointer association status

- Pointer status:
 - **Undefined**: just declared pointer
 - **Associated**: pointer pointing to a target

Pointer association status

- Pointer status:
 - **Undefined**: just declared pointer
 - **Associated**: pointer pointing to a target
 - **Disassociated**: pointer not associated anymore

Pointer association status

- Pointer status:
 - **Undefined**: just declared pointer
 - **Associated**: pointer pointing to a target
 - **Disassociated**: pointer not associated anymore
- `nullify(p1)` to disassociate the pointer p1

Pointer association status

- Pointer status:
 - **Undefined**: just declared pointer
 - **Associated**: pointer pointing to a target
 - **Disassociated**: pointer not associated anymore
- *nullify(p1)* to disassociate the pointer p1
- Example **p3.f90**

Pointer dereferencing

- Doing operations and assignments on the **target data**

Pointer dereferencing

- Doing operations and assignments on the target data
- If $p1 \Rightarrow n1$ and $p2 \Rightarrow n2$, then $p1 = p2$ means $n1 = n2$

Pointer dereferencing

- Doing operations and assignments on the **target data**
- If $p1 \Rightarrow n1$ and $p2 \Rightarrow n2$, then $p1 = p2$ means $n1 = n2$
- $p1 = p2 + p3$ means $n1 = n2 + n3$, if $p3 \Rightarrow n3$

Pointer dereferencing

- Doing operations and assignments on the **target data**
- If $p1 \Rightarrow n1$ and $p2 \Rightarrow n2$, then $p1 = p2$ means $n1 = n2$
- $p1 = p2 + p3$ means $n1 = n2 + n3$, if $p3 \Rightarrow n3$
- Example **p4.f90**

Pointer dereferencing

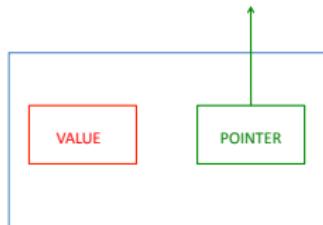
- Doing operations and assignments on the **target data**
- If $p1 \Rightarrow n1$ and $p2 \Rightarrow n2$, then $p1 = p2$ means $n1 = n2$
- $p1 = p2 + p3$ means $n1 = n2 + n3$, if $p3 \Rightarrow n3$
- Example **p4.f90**
- Make things faster: examples **swap.f90** and **p_swap.f90**

Pointer dereferencing

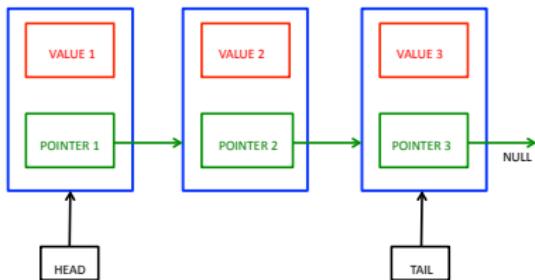
- Doing operations and assignments on the **target data**
- If $p1 \Rightarrow n1$ and $p2 \Rightarrow n2$, then $p1 = p2$ means $n1 = n2$
- $p1 = p2 + p3$ means $n1 = n2 + n3$, if $p3 \Rightarrow n3$
- Example **p4.f90**
- Make things faster: examples **swap.f90** and **p_swap.f90**
- Use **allocate(pointer)** to create a new data object

Linked list

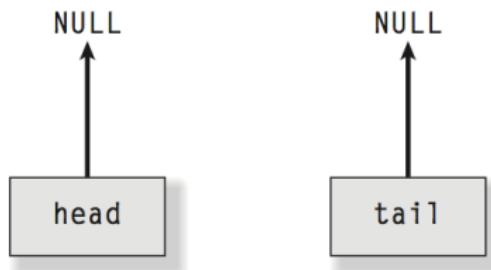
- Derived data type: ordinary variable (integer, real, etc.) + pointer



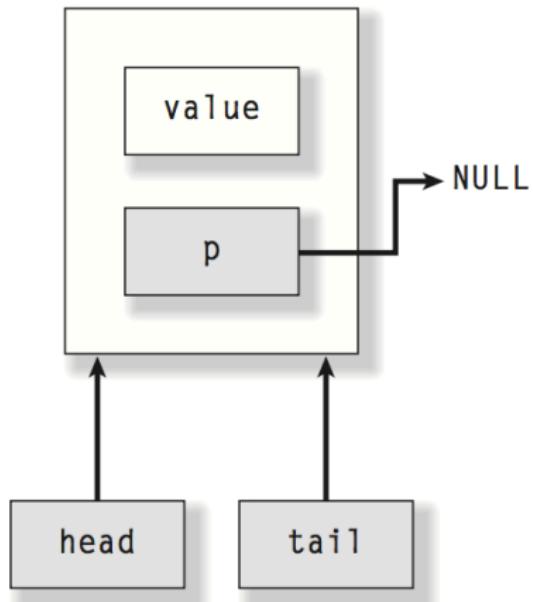
- Linked list



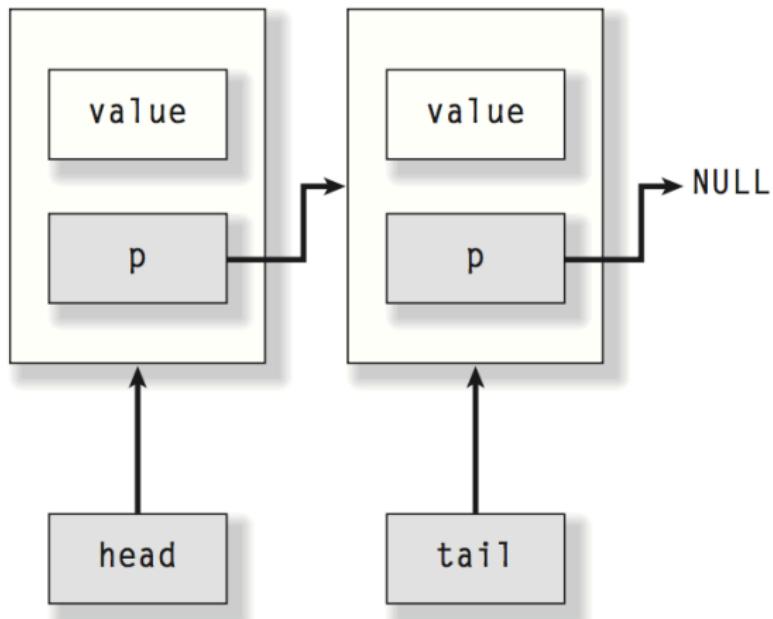
Linked list: first step



Linked list: second step



Linked list: third step



Example `list.f90`