Fortran pointers

Victor Eijkhout and Charlie Dey

spring 2017

Pointers are aliases

- Pointer points at an object
- Access object through pointer
- You can change what object the pointer points at.

```
real,pointer :: point_at_real
```

Setting the pointer

- You have to declare that a variable is pointable:

  ```
  real,target :: x
  ```

- Set the pointer with => notation:

  ```
  point_at_real => x
  ```

- Now using `point_at_real` is the same as using `x`.

Pointer example

```
real,target :: x,y
real,pointer :: that_real

x = 1.2
y = 2.4
that_real => x
print *,that_real
that_real => y
print *,that_real
y = x
print *,that_real
```

1. The pointer points at `x`, so the value of `x` is printed.
2. The pointer is set to point at `y`, so its value is printed.
3. The value of `y` is changed, and since the pointer still points at `y`, this changed value is printed.

Assign pointer from other pointer

```
real,pointer :: point_at_real,also_point
point_at_real => x
also_point => point_at_real
```

Now you have two pointers that point at x.

Very important to use the =>, otherwise strange memory errors

Exercise 1

Write a routine that accepts an array and a pointer, and on return has that pointer pointing at the largest array element.

Linked list

- Linear data structure
- more flexible for insertion / deletion
- . . . but slower in access

Linked list datatypes

- Node: value field, and pointer to next node.
- List: pointer to head node.

```
type node
   integer :: value
   type(node),pointer :: next
end type node

type list
   type(node),pointer :: head
end type list

type(list) :: the_list
nullify(the_list%head)
```

List initialization

First element becomes the list head:

```
allocate(new_node)
new_node%value = value
nullify(new_node%next)
the_list%head => new_node
```

Attaching a node

Keep the list sorted: new largest element attached at the end.

```
allocate(new_node)
new_node%value = value
nullify(new_node%next)
current%next => new_node
```

Inserting 1

Find the insertion point:

```
current => the_list%head ; nullify(previous)
do while ( current%value<value &
     .and. associated(current%next) )
   previous => current
   current => current%next
end do
```

Inserting 2

The actual insertion requires rerouting some pointers:

```
allocate(new_node)
new_node%value = value
new_node%next => current
previous%next => new_node
```