The map_any_type module

Neil N. Carlson

May 2013
Version 1.0

Abstract

The `map_any_type` module defines a map data structure (or associative array) which stores (key, value) pairs as the elements of the structure. The keys are unique and are regarded as mapping (or indexing) to the value associated with the key. In this implementation keys are character strings but the values may be a scalar value of any intrinsic or derived type. An associated iterator derived type is also defined which gives a sequential access to all elements of the map structure.

1 Synopsis

Usage

```
use :: map_any_type
```

Derived Types

`map_any`, `map_any_iterator`

2 The map_any derived type

The derived type `map_any` defines a map data structure which stores (key, value) pairs. The keys are unique character strings that map (or index) to the value associated with the key. Values may be a scalars of any intrinsic or derived type; see *Limitation on values* below. The derived type has the following properties:

- Finalization occurs for `map_any` objects; when a map object is deallocated, or otherwise ceases to exist, all allocated data associated with the object is automatically deallocated.
- Scalar assignment is defined for `map_any` objects with the expected semantics. The contents of the lhs map are first cleared, and then the lhs map defined with the same (key, value) pairs as the rhs map, becoming an independent copy of the rhs map; see *Limitation on values* below.
- The structure constructor `map_any()` evaluates to an empty map, and `map_any` variables come into existence as empty maps.

2.1 Type bound subroutines

`insert(key,value)`
: adds the specified key and associated value to the map. If the mapping already exists, its value is replaced with the specified one. `Key` is a character string and `value` may be a scalar of any intrinsic or derived type. A copy of `value` is stored in the map; see *Limitation on values* below.

`remove(key)`
: removes the specified key from the map and deallocates the associated value. If the mapping does not exist, the map is unchanged.

`clear()`
: removes all elements from the map, leaving it with a size of 0.

2.2 Type bound functions

`mapped(key)`
: returns true if a mapping for the specified key exists.

`value(key)`
: returns a `class(*)` pointer to the mapped value for the specified key, or a null pointer if the map does not contain the key.

`size()`
: returns the number of elements in the map.

`empty()`
: returns true if the map contains no elements; i.e., `size()` equals 0.

2.3 Limitation on values

The values contained in a map are copies of the values passed to the `insert` method, but they are shallow copies as created by sourced allocation. For intrinsic types these are genuine copies. For derived type values, the literal contents of the object are copied, which for a pointer component means that a copy of the pointer is made, but not a copy of the target; the original pointer and its copy will have the same target. This also applies to map assignment where the values in the lhs map are sourced-allocation copies of the values in the rhs map. Thus great caution should be used when using values of derived types that contain pointer components, either directly or indirectly.

3 The map_any_iterator derived type

The values in a map can be accessed directly, but only if the associated keys are known. The derived type `map_any_iterator` provides a means of iterating through the elements of a `map_any` object; that is, sequentially visiting each element of a map once and only once. Once initialized, a `map_any_iterator` object is positioned at a particular element of its associated map, or at a pseudo-position *the end*, and can be queried for the key and value of that element. The derived type has the following properties:

- `map_any_iterator` variables must be initialized with the map they will iterate through. The structure constructor `map_any_iterator(map)` evaluates to an iterator positioned at the the initial element of the specified map, or the end if the map is empty.
- Scalar assignment is defined for `map_any_iterator` objects. The lhs iterator becomes associated with the same map as the rhs iterator and is positioned at the same element. Subsequent changes to one iterator do not affect the other.

3.1 Type bound subroutines

`next()`
: advances the iterator to the next element in the map, or to the end if there are no more elements remaining to be visited. This call has no effect if the iterator is already positioned at the end.

3.2 Type bound functions

`key()`
: returns the character string key for the current map element. The iterator must not be positioned at the end.

`value()`
: returns a `class(*)` pointer to the value of the current map element. The iterator must not be positioned at the end.

`at_end()`
: returns true if the iterator is positioned at the end.

4 Example

```
use map_any_type

type(map_any) :: map, map_copy
type(map_any_iterator) :: iter
class(*), pointer :: value

type point
  real x, y
end type

!! Maps come into existence well-defined and empty.
if (.not.map%empty())  print *, 'error: map is not empty!'

!! Insert some elements into the map; note the different types.
call map%insert ('page', 3)
call map%insert ('size', 1.4)
call map%insert ('color', 'black')
```

```
call map%insert ('origin', point(1.0, 2.0))

!! Replace an existing mapping with a different value of different type.
call map%insert ('size', 'default')

!! Remove a mapping.
call map%remove ('color')
if (map%mapped('color')) print *, 'error: mapping not removed!'

!! Retrieve a specific value.
value => map%value('origin')

!! Write the contents, using an iterator to access all elements.
iter = map_any_iterator(map)
do while (.not.iter%at_end())
  select type (uptr => iter%value())
  type is (integer)
    print *, iter%key(), ' = ', uptr
  type is (real)
    print *, iter%key(), ' = ', uptr
  type is (character(*))
    print *, iter%key(), ' = ', uptr
  type is (point)
    print *, iter%key(), ' = ', uptr
  end select
  call iter%next
end do

!! Make a copy of the map.
map_copy = map

!! Delete the contents of map; map_copy is unchanged.
call map%clear
if (map%size() /= 0) print *, 'error: map size is not 0!'
if (map_copy%empty()) print *, 'error: map_copy is empty!'
```

5 Issues

Intel compiler versions $\geq$ 13.1.1 are required due to compiler bugs in earlier versions.
Bug reports and improvement suggestions should be directed to `neil.n.carlson@gmail.com`