PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- We have looked at how to allocate arrays at run-time.
- There is also the reverse of this, freeing memory.
- It is always good practice to free memory when it is not needed. Using less memory is always a good thing.
- This should be practised even when the array is needed for the whole of the run-time.
- As we shall see, sloppiness in this regard can lead to memory leaks.
- There is also reallocating arrays when we find they are too small.

Freeing Memory

- Consider the following code.

```
int *p;

p = (int *) malloc(10 * sizeof(int));
// pretend we do stuff with *p
p = (int *) malloc(10 * sizeof(int));
// pretend we do some other stuff with *p
p = (int *) malloc(10 * sizeof(int));
```

- Three blocks of memory have been allocated. Yet, only the third is still accessible.
- The first two blocks are allocated, yet unreachable. They are *lost* memory that will only be freed on termination of the program.
- Dynamically allocated memory must be *freed* when no longer needed.

Freeing Memory

- The `free()` function frees memory previously allocated by `malloc()`.
- Unless explicitly freed, dynamically allocated memory remains allocated for the entire program execution.

```
int *p;

p = (int *) malloc(10 * sizeof(int));
// [...]
free(p);
p = (int *) malloc(10 * sizeof(int));
// [...]
free(p);
p = (int *) malloc(10 * sizeof(int));
// [...]
free(p);
```

- Though memory is freed for the system, `p` still points to the same area which is no longer reserved.
- Assigning NULL to a "freed pointer" can avoid doing anything dangerous with the memory.

Freeing Memory

FORTRAN

- Unlike C, local dynamically allocated memory is freed when returning from a program unit.
- If the array is in a "module" then it will only be freed at the end of the program (by default).
- However it is good practise to deallocate any dynamically allocated arrays, after use.

```
integer (kind=4), allocatable :: iarr(:)
integer (kind=4) :: n,error

n = 1000
allocate(iarr(n),stat=error)  ! Allocate

if (allocated(iarr)) then     ! Test to see if allocated
  deallocate(iarr,stat=error) ! Deallocate
endif
```

Reallocating Memory

C

- Sometimes, you may want to increase the size of a memory block..
- The `realloc()` function will do it for you.
- It will allocate the memory block requested, copy the contents of the old block at the beginning of the new one, free the old one, and return a pointer to the beginning of the new block.

```c
int *p;

p = (int *) malloc(100 * sizeof(int));
// do something with p, fill it up, etc
p = (int *) realloc(p, 200 * sizeof(int)); // p now has 200 '
```

Reallocating Memory

FORTRAN

- Sadly there is no equivalent of the realloc() function. Some systems may have extensions that include realloc().
- Below is a function that does the job.

```
MODULE realloc_mod
CONTAINS
  FUNCTION reallocate(p, n)  ! realloc REAL 1D array
    REAL, POINTER, DIMENSION(:) :: p, reallocate
    INTEGER, intent(in) :: n
    INTEGER :: nold, ierr
    ALLOCATE(reallocate(1:n), STAT=ierr)
    IF(ierr .NE. 0) STOP "allocate error"
    IF(.NOT. ASSOCIATED(p)) RETURN
    nold = MIN(SIZE(p), n)
    reallocate(1:nold) = p(1:nold)
    DEALLOCATE(p)
  END FUNCTION REALLOCATE
END MODULE realloc_mod
```

Reallocating Memory

FORTRAN

- Below is an example of the use of the function.
- In FORTRAN the only way of expanding an existing array is to destroy it and allocate it again.
- Thus if there are any associated pointers, they must be reassociated.

```
PROGRAM realloc_test
USE realloc_mod
IMPLICIT NONE
REAL, POINTER, DIMENSION(:) :: p
  ALLOCATE(p(2))
  p => reallocate(p, 10000)     ! note pointer assignment
END PROGRAM realloc_test
```