PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- This lecture we will talk about generic routines and how to construct them.
- We can do something similar in C but it is a bit of a hack. It is more for interest.
- However the modern version of C, C++ does have a better solution.

Generic Subroutines

FORTRAN

- A generic function or subroutine is one that works on different variable types *e.g.* integer and real variables.
- In fact most of the intrinsic functions are generic.
- The cot function is constructed below.

```
interface cot
  function fcot(x)
     real (kind=4) :: fcot,x
  end function fcot
  function dcot(x)
    real (kind=8) :: dcot,x
  end function dcot
  function icot(x)
    integer (kind=4) :: x
    real (kind=4) :: icot
  end function icot
end interface cot
```

Cot Function

- Here is the rest of the program.

```
program fexample
   integer (kind=4) :: i
   real (kind=4) :: x
   real (kind=8) :: dx

   i = 1; x = 1.0; dx = dble(1.0);
   write(6,*) cot(i),cot(x),cot(dx)
end program fexample

real (kind=4) function icot(x)
   integer (kind=4) :: x
   icot = 1.0/tan(real(x))
end function icot
real (kind=4) function fcot(x)
   real (kind=4) :: x
   fcot = 1.0/tan(x)
end function fcot
real (kind=8) function dcot(x)
   real (kind=8) :: x
   dcot = dble(1.0)/dtan(x)
end function dcot
```

Dereferencing

- Before dealing with the C equivalent, we need to have a solid understanding of pointers.
- The syntax of pointers can be a bit confusing in C.
- When declaring a pointer you use *type*$*$

```
int *p;     // pointer to int
float *px; // pointer to float
int i;      // ordinary variables
float x;
```

- Elsewhere in the code (not a declaration) the $*$ means something else, it is called the dereferencing operator.

```
*p = 1;    // Sets r-value of memory address
*px = 1.0;
```

- Finally the & opertor gets the address of a variable or its *l − value*.

```
p = &i;  // p points to i
px = &x; // px points to x
&px;     // even pointers have l-values
```

Generic Functions in C

- Below is a similar example to the previous *cot* function.
- As we do not know the variable type the argument's $l-value$ is passed as "void *".
- Within the function the variable type is reestablished, with the help of the other argument "type".
- The tan function takes a double as input and returns a double. Thus the variable y (within the function) has its $r-value$ set to that of the input argument.
- The function then returns a $r-value$ of type double.

```
double cot(void *px, int type);
int main(void) {
  int a; double dx,cotval;
  a =1; dx = 1.0;
  cotval = cot(&a,1);
  cotval = cot(&dx,3);
  return 0;
}
double cot(void *px, int type) {
   double y;
   switch (type) {
   case (1):
      y = (double) *(int *) px;
      break;
   case (2):
      y =  (double) *(float *) px;
      break;
   case (3):
      y = *(double *) px;
      break;
   }
   return 1.0/tan(y);
}
```

Summary

- This week we discussed:
  1. casting,
  2. pointers,
  3. passing by reference,
  4. generic routines.