Procedures and Conditionals

22.901 Introduction to Computer Programming for Nuclear Engineers

January 23, 2012

Outline

- Subroutines
- Functions
- Logical Operators

Sub-Dividing the Problem

- Most programs are thousands of lines!
- You may notice you have to have similar code in multiple places
- Want to have good organization

use procedures

Massachusetts Institute of Technology

So what can we do?

- There must be a single main program
- Subroutines and functions are called procedures
- For now we will list them in one file

Subroutines::

- Code that exists somewhere else
- You can pretty much do anything

Functions::

- Purpose is to return a result

Calling a subroutine

- use the call statement, `call <sub name>(args...)`

The subroutine is defined somewhere else:

- `subroutine <name>(args)`
- `use ...`
- `implicit none`
- variable declarations
- executable statements
- `end subroutine <name>`

Dummy Arguments

- Variable names in a procedure, only exist in that procedure
- Fortran is **pass-by-reference**
- The memory location are only passed between procedures (copies are not made!)
- Therefore in arguments list only order matters, not the name!
- Each argument must match in type and rank

Functions

- Use when the result is a single value
- For example to return the maximum value of a vector
- **Important:** The function name defines the variable in the function
- The function name is returned at the end of a function

```
a = maxval(vec)
function maxval(v,n)  integer ::  n !  length of
vector
 real ::  maxval !  the maximum value of a vector
 real ::  vec(n) !  the vector
 ...
end function maxval
```

The Intent Attribute

- You can make arguments read-only, write-only or read-write(default)
- This should be used to avoid overwriting a variable
- It is an attribute so it is listed before the ::
- Example::
  - `integer, intent(in) :: avar ! read-only`
  - `integer, intent(out) :: avar ! write-only`
  - `integer, intent(inout) :: avar ! read-write`
  - `integer :: avar ! read-write`

Massachusetts Institute of Technology

Internal Procedures

- use the contains command
- this will be use a lot more for modules and object-oriented programming
- included procedure are internal and are private to other routines
- they may not contain their own internal subprograms

```
program test
...
contains
subroutine atest
...
end subroutine atest
end program test
```

Calculating zeta I

```
program zeta

  implicit none

  real :: zeta_fun   ! zeta function
  real :: zeta_val   ! zeta variable
  real :: val=2.0    ! random value

  ! call zeta subroutine
  call zeta_sub(val,zeta_val)

  ! print results
  print *,'From subroutine:',zeta_val
  print *,'From function:',zeta_fun(val)

end program zeta

subroutine zeta_sub(x,zeta)
```

Massachusetts Institute of Technology

Calculating zeta II

```
  real, intent(in)  :: x     ! an input  value
  real, intent(out) :: zeta  ! the answer

  zeta = 3.14159/x

end subroutine zeta_sub

function zeta_fun(x)

  real, intent(in) :: x          ! an input value
  real             :: zeta_fun   ! the answer

  zeta_fun = 3.14159/x

end function zeta_fun
```

Decision Making - Conditionals

- Almost every time we code, decisions need to be made from with in the code

- Conditionals allow us to execute certain parts of code

- We will go over the following constructs:
  - `if` statement
  - `if-then` statement
  - `if-else` statement
  - `if-elseif-else` statement

Massachusetts Institute of Technology

The 'if' statement

- Oldest and simplest control statement
- `if (logical expression) <execute statement>`
- If the LHS is `.true.`, the RHS is executed
- This is only useful if there is a simple decision and one execute statment
- For anything more complicated, we need to go to block statements

```
if (x < a) x = a
```

The 'if-then' block statement

- A block if-then statement is more flexible than the if statment
- It has the following form:

```
if (logical expression) then
   execute statements
   more execute statements
end if
```

- If the logical expression is `.true.` then the block is executed

```
if (coremap(i,j,k) == 99999) then
    beta = eval_albedo(i,j,k)
    D = beta(2)/beta(1)
    Dhat = D/dx*(flux(i,j,k))
end if
```

Massachusetts Institute of Technology

The 'if-else' block statement

- A block if-then statement is more flexible than the if statment
- It has the following form:

```
if (logical expression) then
   execute statements
else
   execute statements
end if
```

- If the logical expression is `.true.` then the first block is executed. Otherwise, the second block is omitted

```
if (coremap(i,j,k) == 99999) then
    beta = eval_albedo(i,j,k)
else
    beta = 1.0
end if
```

Including the 'else if'

- You can use as many `else if` statements as you want
- There is only 1 `end if` per `if` (none for else if)
- All else if statements must be listed before the `else`
- They are checked in order until the first `.true.` is evaluated
- You don't have to have the `else` statement

Logical Operators

Order of logical operation is used

- `()` - inside out, and function references
- `.not.` - left to right
- `.and.` - left to right
- `.or.` - left to right
- `==,/=,>,<,>=,<=` - left to right

Square root I

```
program squareroot

  implicit none

  real :: valin  ! the val to be square rooted
  real :: valout ! the output value

  ! read in val from user
  print *, 'Enter a real number:'
  read *, valin

  ! determine if the square root is real or complex
  if (valin > 0.0) then

    ! take the sqrt
    valout = sqrt(valin)

    ! print the result
```

Massachusetts Institute of Technology

Square root II

```
    print *,'The square root is also real:',valout

  else if (valin < 0.0) then

    ! make the valin positive and take sqrt
    valout = sqrt(-1.0*valin)

    ! print the result
    print *,'The square root is complex: +/-',valout,'i'

  else

    print *, 'The square root is obviously 0.0!'

  end if

end program squareroot
```

End of class/External Assignment

- Write a program that solves the quadratic equation

$$y = ax^2 + bx + c$$

- Have the user enter coefficients a, b and c
- Depending on the sign of the discriminant you will have 3 rules:
  - Imaginary, Real, or Double real root
- Calculate appropriate number of solutions
- Print result to user

Massachusetts Institute of Technology