Control structures

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

Control Structures

If-then-else statements

You will eventually need to your program to make a decision, i.e. `if A is true do something, if A is false do something else. If A and B is true, do something completely different, else if A or C is true, exit the program.`

We do this through a combination of relational and logical expressions.

Control Structures

❖ Relational Expressions

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
/=	not equal to

❖ Logical Expressions

`.not.`	not
`.or.`	or
`.and.`	and

if statement

```
[label0:] if (logical expression) then
               {if-block}
          [else if (logical expression) then
               {else-if-block} ]
          [else
               {else block} ]
          end if [label0]
```

labels are optional, they just make your code easier to read.

if statement

```
program test

implicit none
integer :: a=3, b=4, c=5

if (a < b .and. b < c) then
   print *, "c is the biggest of them all!"
else if (a < b and b > c) then
   print *, "b is the biggest of them all!"
else if (b > c) then
   print *, "even though this is true, else condition is never hit"
end if

demo_or: if (a > b .or. c > b)
             print *, "one of the conditions is .true."
         else
             print *, "none of the condtions are .true."
         end if demo_or

end program
```

`else` blocks are also optional, but very useful to cut down on the number of if-block cycles. Once a condition within a `if-elseif` block is hit, the block is exited

if statement

```
program test

implicit none
integer :: a=3, b=4, c=5

if (a < b .and. b < c) then
   print *, "c is the biggest of them all!"
else if (a < b .and. b > c) then
   print *, "b is the biggest of them all!"
else if (b > c) then
   print *, "even though this is true, else condition is never hit"
end if

demo_or: if (a > b .or. c > b) then
             print *, "one of the conditions is .true."
         else
             print *, "none of the conditions are .true."
         end if demo_or

end program
```

What's different from C/C++?

- no `{ }`'s
- `then` statement
- `/=` vs `!=`
- `.and.` vs `&&`
- `.or.` vs `||`
- `end if`
- `labels`

Control Structures – Exercise 1

FizzBuzz

Read in an integer.

If it's a multiple of three print 'Fizz'; if it's a multiple of five print 'Buzz'.

It it is a multiple of both three and five print 'FizzBuzz'.

Otherwise print nothing.

NOTE: `mod(A,P)` computes the remainder of the division of A by P where A and P are both integers. Try writing your code **without** using the `mod()` function

select-case statement

```
[label:] select case (expression)
          [case selector 1
             block]
          [case selector 2
             block]
          [case selector 3
             block]
          [case selector 4
             block]
          [case default
             block]
          end select [label]
```

labels are optional, they just make your code easier to read.

`expression` may be an **integer** or character or **logical**

`selector` is a list of **non-overlapping** values

`default` is selected when no cases are valid.

sometimes, `select case` blocks might be useful in place of multiple `if-elseif` statement but for a **single** logical expression.

if statement

```
program test

implicit none
integer :: a=3, b=4, c=5
!select case in place of an if block
select case (b > a)
   case (.true.)
      print *, "TRUE!"
   case (.false.)
      print *, "FALSE!"
end select


end program
```

Silly to use instead of if-block in this case.

if statement

```
program test

implicit none
integer :: a=3, b=4, c=5
!select case in place of an if block
select case (a)
   case (1)
      ...
   case (2)
      ...
   case (3)
      ...
   case default
      ...default
end select


end program
```

However, when we have an expression and need to make different choices for multiple cases...

if statement

```
...
select case (n)
case (:-1)     ! Range from smallest
               ! integer to -1
   print*, "n should be positive"
   exit
case (0:)      ! Range from 0 to largest
integer
   factorial=1
   <factorial code>
end select
...
```

You may also do ranges...

Do Loops

Just like in other programming languages, you will need to repeat a statement or a block of statements a number of times.

That's where the loop comes in. A loop has a counter, called a loop variable or index, which (usually) ranges from a lower bound to an upper bound.

`[label:] do variable=expr1, expr2[, expr3]`
`    block`
`end do [label]`
	`variable` is a scalar integer variable
	`expr1`, `expr2` & `expr3` are integer expressions
	
	The Do Loop advances variable from `expr1` to `expr2` by counts of `expr3`
	
	Similar in style and execution to the `for loop` from C/C++, but the "test" condition is `variable >=` (or `<=`) `expr2` vs. C/C++ where the test condition can be any logical expression.

The Do Loop

```
integer :: i

do i = 0, 5
   print *, i
end do
```

`variable` is a scalar integer variable
`expr1`, `expr2` & `expr3` are integer expressions

The Do Loop advances variable from `expr1` to `expr2` by counts of `expr3`

Similar in style and execution to the `for loop` from C/C++, but the "test" condition is `variable >=` (or `<=`) `expr2` vs. C/C++ where the test condition can be any logical expression.

Control Structures – Exercise 2

Loops

Read an integer value
print ‘Hello world’ that many times.

The Infinite Do Loop

`do`
`...`
`end do`
	This will loop forever

Exiting the loop

`do`
`…`
`   if (expr1) then`
`      exit`
`   end if`
`…`
`end do`
	`expr1` is a logical expression
	`exit` will exit the `do loop`

Skipping

<pre>do
 	
…
   if (expr1) then
      cycle
   end if
…
 	
end do</pre>
	`expr1` is a logical expression
	`cycle` will skip the current iteration of the `do loop`

The Do While Loop

<pre>[label:] do while (expr1)
            block
         end do [label]</pre>
	`expr1` is a logical expression
	
	Similar in style and execution to the `while loop` from C/C++

Control Structures – Exercise 3

Loops

Find all triples of integers u,v,w under 100 such that $u^2 + v^2 = w^2$.

Make sure you omit duplicates of solutions you have already found.

Control Structures – Exercise 4

Loops

One bank account has 100 dollars and earns a 5 percent per year interest rate.
Another account has 200 dollars but earns only 2 percent per year.

In both cases the interest is deposited into the account.

After how many years will the amount of money in both accounts be the same?

Control Structures – Exercise 5

Loops

$$u_{n+1} = \begin{cases} u_n/2 & \text{if } u_n \text{ is even} \\ 3u_n + 1 & \text{if } u_n \text{ is odd} \end{cases}$$

leads to the Collatz conjecture: no matter the starting guess u1,

the sequence n □⟶ un will always terminate.

For u1 < 1000 find the values that lead to the longest sequence: every time you find a sequence that is longer than the previous maximum, print out the starting number.

Control Structures Projects 1

Divisors

Read two numbers and print a message like:

```
3 is a divisor of 9
```

if the first is an exact divisor of the second, and another message

```
4 is not a divisor of 9
```

if it is not.

Control Structures Projects 2

Loops

Read an integer and determine whether it is prime by testing for the smaller numbers whether they are a divisor of that number.

Print a final message:

```
Your number is prime
```

or

```
Your number is not prime: it is divisible by ....
```

where you report just one found factor.