

Goals for This Lecture:

- Introduce loop constructs
- Understand the DO loop
- Understand how to use the EXIT statement
- Understand how to design algorithms utilizing looping constructs
- Introduce the DO WHILE loop

Loop Constructs

- The capability to repeatedly execute certain blocks of code is highly desirable
- Virtually all modern high level languages provide this capability
- This type of construct is generically referred to as looping
- In FORTRAN 90 & 95 the loops take on two forms:
- WHILE loops which execute while some condition is true
- Iterative loops which execute a specified number of times

The WHILE loop construct

- The WHILE loop construct allows looping to occur until the loop is exited with an EXIT statement
- The EXIT statement terminates execution of the loop and transfers execution to the first executable line following the ENDDO statement

Form:

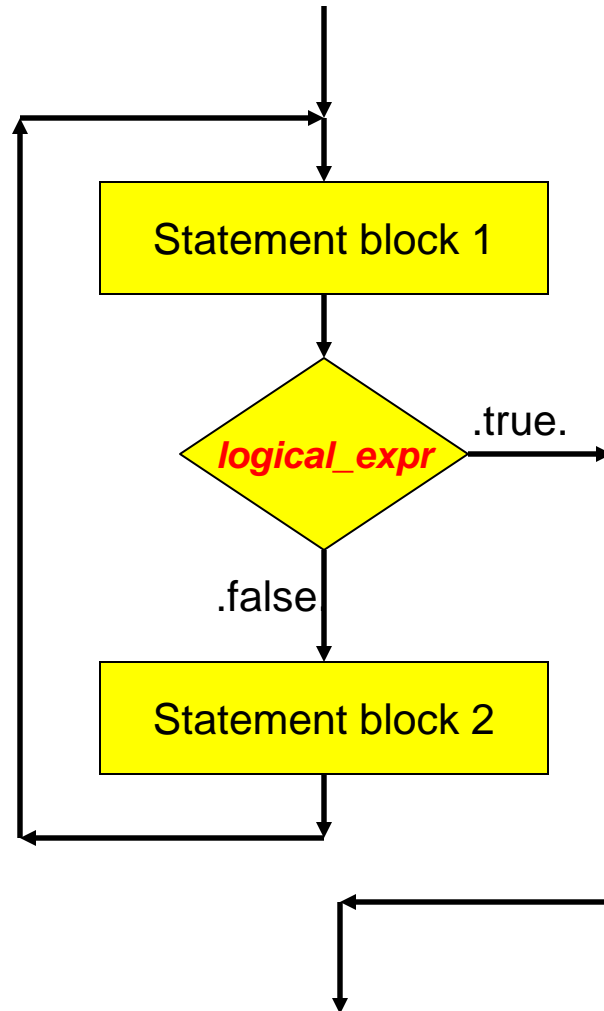
```
do
  statement block1
  ...
  if(logical_expr) exit
  statement block2
  ...
enddo
```

Example of use:

```
x = 0.0
do
  f_of_x = 2.0*(x**2)+3.0
  write(*,*) \ x,f = \,x,f_of_x
  if( x >= 1.0) exit
  x = x+0.01
enddo
```

- As many **IF(logical_expr) EXIT** statements as needed can be included

Execution flow for WHILE loop construct



Example of Top Down Design

- **Step 1. State the problem:** create a Kelvin to Fahrenheit calculator that repeatedly prompts the user for a temperature until told to stop
- **Step 2. List the inputs and outputs:**
 - Outputs: Prompt the user to enter temperature
Prompt the user to continue-or-not
Temperature in Fahrenheit
 - Inputs: Temperature in Kelvin
Continue-or-not response from user

Step 3: Design an algorithm

Algorithm:

1. Start the program
2. Initiate the While loop
3. Prompt the user for a temperature in Kelvin
4. Read in the temperature in Kelvin
5. Convert the Temperature to Fahrenheit
6. Output the temperature in Fahrenheit
7. Ask the user if they want to continue
8. Read in the response
9. If the response is negative exit the loop
10. Terminate the loop
11. Stop

Step 4: Convert Algorithm to Code

```
! Purpose: Convert Kelvin to Fahrenheit
! Author:  F. Douglas Swesty
! Date:    9/23/2005
Program k_to_f
implicit none      ! Turn off implicit typing
real :: t_kelvin  ! Temperature in Kelvin
real :: t_faren   ! Temperature in Fahrenheit
logical :: cflag  ! Continuation flag
do                ! Initiate loop
  write(*,*) "Enter T (in Kelvin) : " ! Prompt the user for temperature
  read(*,*) t_kelvin                 ! Read in temperature
  t_faren = 1.8*(t_kelvin-273.0)     ! Convert Kelvin to Fahrenheit
  write(*,*) " T (deg. Fahrenheit) = ",t_faren
  write(*,*) "Continue: (.true. or .false.)" ! Prompt the user about continuation
  read(*,*) cflag                    ! Read in continuation "flag"
  if(.not.cflag) exit                ! If continuation flag is false then Exit from loop
enddo  ! Terminate loop
stop                                       ! Stop execution of the program
end program k_to_f
```

Step 5: Test the code

Step 6: Repeat Step 3: Improve algorithm if possible

- Current code requires two inputs per temperature conversion: T in Kelvin and continuation flag
- Makes program inconvenient. Too much typing!
- Can we reduce this?
- Use an invalid temperature to signal completion!
- $T < 0$ can be used as a flag to signal it's time to stop

Step 6: Refine code to reflect new algorithm

```
! Purpose: Convert Kelvin to Fahrenheit
! Author:  F. Douglas Swesty
! Date:    9/23/2005
Program k_to_f
implicit none      ! Turn off implicit typing
real :: t_kelvin  ! Temperature in Kelvin
real :: t_faren   ! Temperature in Fahrenheit

do                ! Initiate loop
    ! Prompt the user for
    temperature
    write(*,*) "Enter T (in Kelvin, T < 0 aborts) :",t_kelvin
    read(*,*) t_kelvin
    ! Read in temperature
    if(t_kelvin < 0.0) exit
    ! Exit from loop
    t_faren = 1.8*(t_kelvin-273.0) ! Convert Kelvin to Fahrenheit
    write(*,*) " T (deg. Fahrenheit) = ",t_faren
enddo            ! Terminate loop
stop              ! Stop execution of the program
end program k_to_f
```

The DO WHILE loop construct

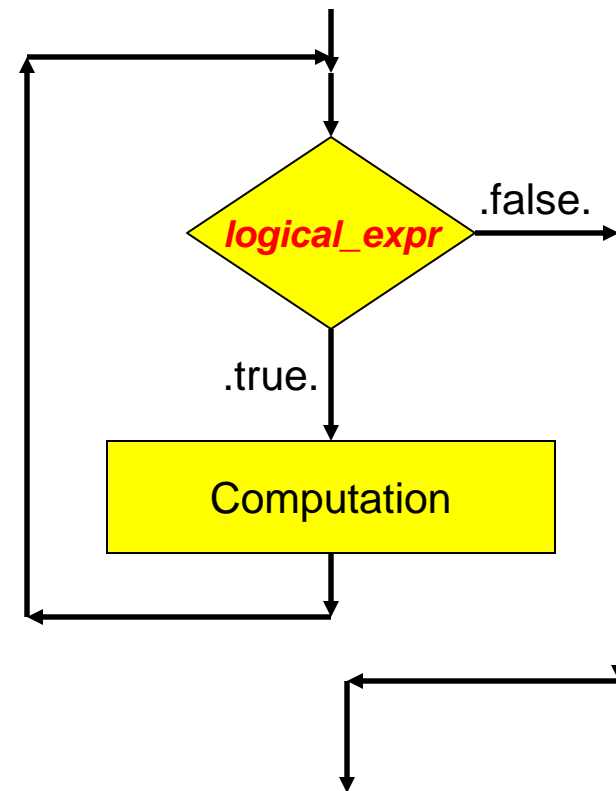
- The DO WHILE construct allows looping to occur as long as some logical expression has a value of true
- If the logical expression does not change it's value to false as a consequence of some executable statement contained in the body of the loop, the loop will execute forever!

Form:

```
do while(logical_expr)  
  statement block  
  ...  
enddo
```

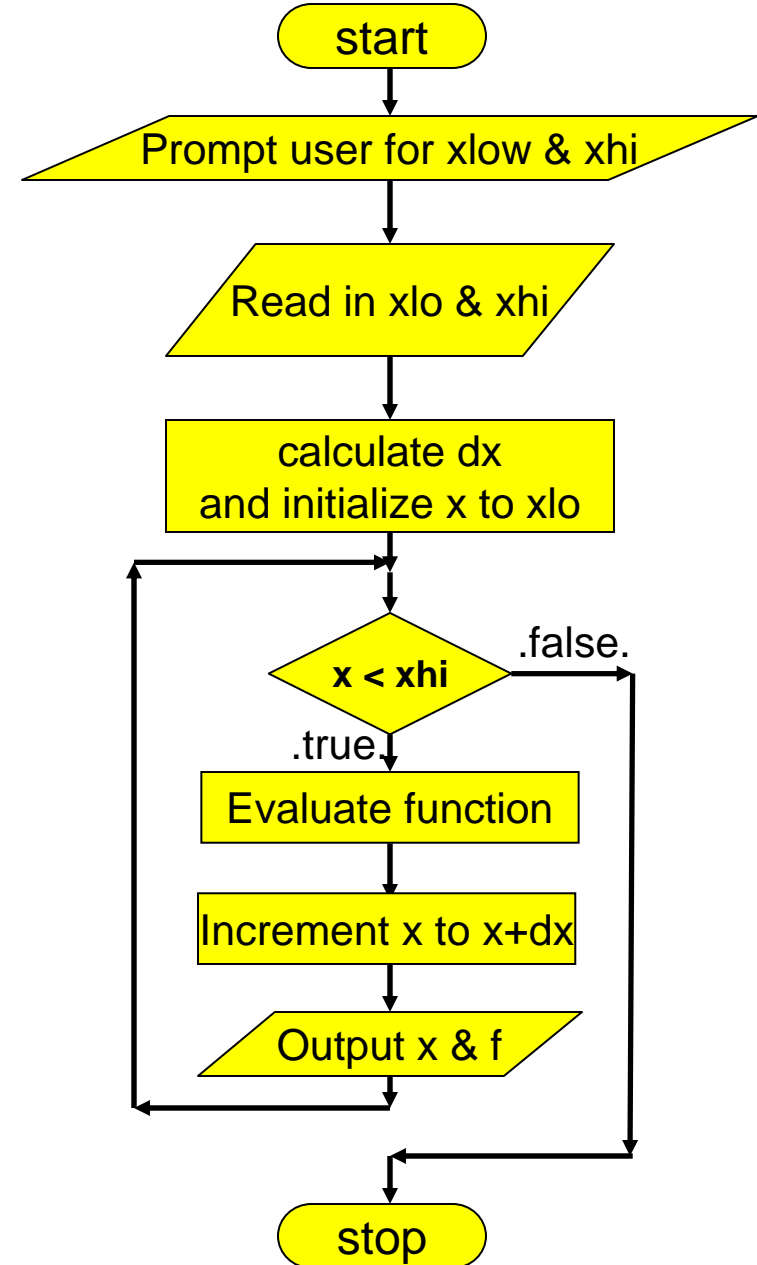
Example of use:

```
x = 0.0  
do while(x < 1.0)  
  f_of_x = 2.0*(x**2)+3.0  
  write(*,*) \ x,f = \,x,f_of_x  
  x = x+0.01  
enddo
```



Algorithm for function evaluation

1. Start execution
2. Prompt user for low & hi end of range
3. Read in low & high end of range
4. Calculate step size delta-x
5. Initialize x to xlow
6. While $x < xhi$
 1. Evaluate $f(x)$
 2. Write out x and $f(x)$
 3. Increment x to $x+dx$
7. End of while loop
8. Stop execution



Example of DO WHILE loop use

```
! Purpose: Evaluate a function over the range of  $x_{low} < x < x_{hi}$ 
! Author:  F. Douglas Swesty
! Date:    9/21/2005
Program eval_function
implicit none      ! Turn off implicit typing
real :: xlow, xhi ! Variables for low & high bounds on range
real :: x, dx, f   ! Variables for x, delta-x, and function
                  ! Prompt the user for hi & low values
write(*,*) "Enter xlow & xhi:"
read(*,*) xlow,xhi      ! Read in xlow & xhi

dx = (xhi-xlow)/100.0   ! Calculate step size
x = xlow                ! Initialize value of x
do while(x < xhi)      ! Loop while  $x < x_{hi}$ 
  f = 2.0*(x**3)+3.0   ! Evaluate the function
  write(*,*) " x, f = ",x,f
enddo
stop                   ! Stop execution of the program
end program eval_function
```

Reading Assignment

- Read Sections 4.1,4.3