

Goals for This Lecture:

- Understand Top-down design
- Describe five steps to building programs with top-down design principles
- Understand what an algorithm is
- Using pseudocode to describe an algorithm
- Using a flowchart to describe an algorithm
- Define structured programming
- Introduce the block IF construct
- Introduce the ELSE clause
- Introduce the ELSEIF clause

Top-down design principles

- Start at the top and break the problem into smaller tasks through the following steps:
 1. Clearly define your problem
 - What should the program do?
 2. List inputs needed by program and outputs to be produced by program
 - Specify units for inputs
 3. Design algorithm to solve your problem
 - Develop the logic for accomplishing what the program must do
 4. Turn algorithm into code statements
 - Compile & fix compilation (compile-time) errors
 5. Test the program
 - Correct run-time errors as necessary

Example of top-down design based on assignment 1

1. Define the problem

- Calculate the position of a particle (with zero initial velocity and which experiences acceleration a) @ time t

2. List inputs and outputs

- Inputs: acceleration, time
- Outputs: position in meters, position in feet

3. Design algorithm

- Prompt user for time & acceleration
- Read in time & acceleration
- Calculate position with formula $x(t) = (\frac{1}{2}) a t^2$
- Write out position, x , in meters
- Calculate position, x_{feet} , from position in meters, x , by formula $x_{\text{feet}} = 3.208 x$
- Write out position, x_{feet} , in feet
- Stop

4. Translate the algorithm into code

```
! Purpose: Calculate position as a function of time
! Author: F. Douglas Swesty
! Date: 9/19/2005
program position
implicit none                                ! Turn off implicit typing
real :: accel, time                          ! Define acceleration & time variables
real :: xmetres, xfeet                       ! Variables to hold position
                                              ! Prompt the user for inputs
write(*,*) "Enter accel. & time (in MKS units):"
read(*,*) accel, time                        ! Read in the acceleration and time

xmetres = 0.5e0*accel*(time**2) ! Calculate position in meters
write(*,*) ' position (in meters) = ',xmetres ! Output position in meters
xfeet = 3.208e0*xmetres           ! Calculate position in feet
write(*,*) ' position (in feet)=',xfeet      ! Output the position in feet
stop                                         ! Stop execution of the program
end program position
```

5. Now test the program

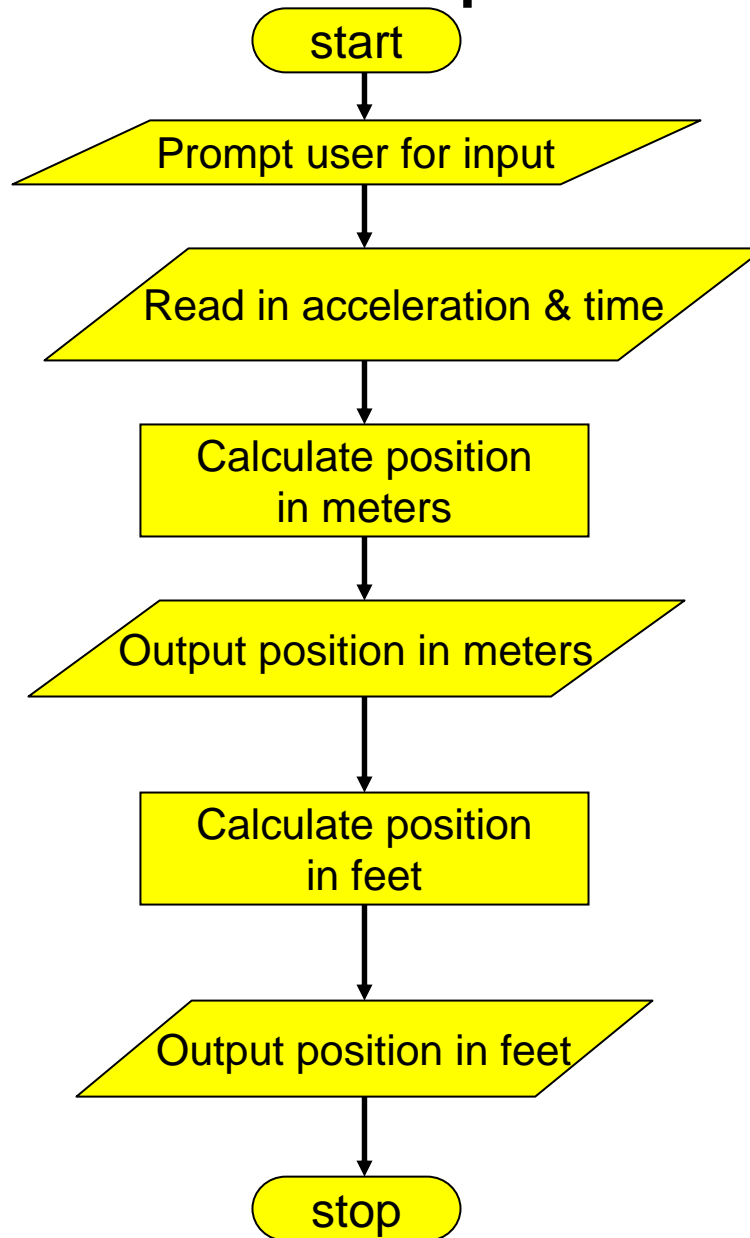
Designing your Algorithm

- An *algorithm* is a a step-by-step process for accomplishing a task
- Look for portions of the problem that can be broken down into smaller (and hopefully less daunting) sub-tasks
- This process is referred to as *decomposition*
- Repeat this process, further decomposing the sub-tasks until the smallest tasks are clear and easy to accomplish in at most a few lines of code
- This recursive process can be characterized as *refinement*

Pseudo-code description of algorithm

- One way of describing an algorithm is via *pseudocode*
- Create a written description of the algorithm in a series of steps
- Example from previous slide:
 - Prompt user for time & acceleration
 - Read in time & acceleration
 - Calculate position with formula $x(t) = (\frac{1}{2}) a t^2$
 - Write out position, x , in meters
 - Calculate position, x_{feet} , from position in meters, x , by formula $x_{\text{feet}} = 3.208 x$
 - Write out position, x_{feet} , in feet
 - Stop

Flow-chart description of algorithm



Common flow-chart symbols

Start or stop



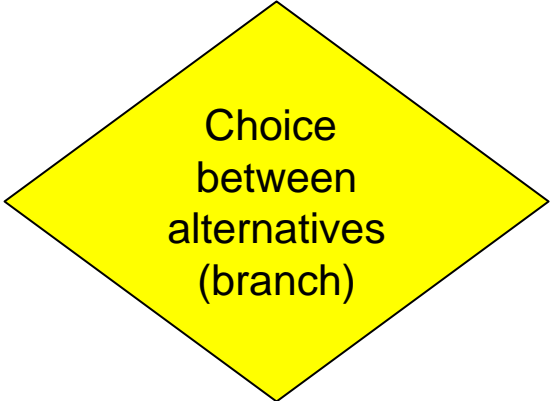
Input or output



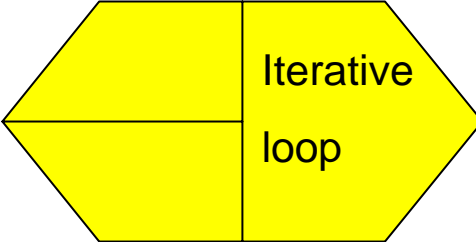
Computation



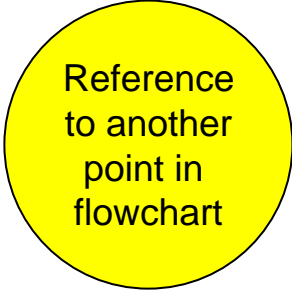
Choice
between
alternatives
(branch)



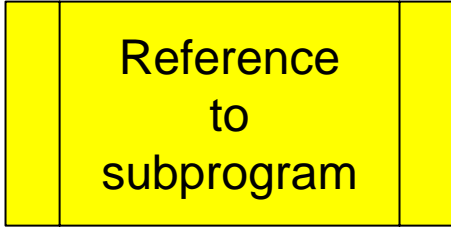
Iterative
loop



Reference
to another
point in
flowchart



Reference
to
subprogram



Flow Control & Structured Programs

- It would be highly desirable to be able to execute portions of a program either repeatedly or selectively
- This is accomplished via *constructs* which control the flow of the program
- Complex algorithms can be assembled out of such constructs
- This is called *structured programming*
- Selective execution of portions of the code based on some condition is referred to as *branching* or *conditional execution*
- Repeated execution of a section of code a specified number of times or while some condition is true is called *looping*

Block IF construct

- The conditional execution of a some piece of code, or a *block*, based on a logical condition can be accomplished via the block-if construct

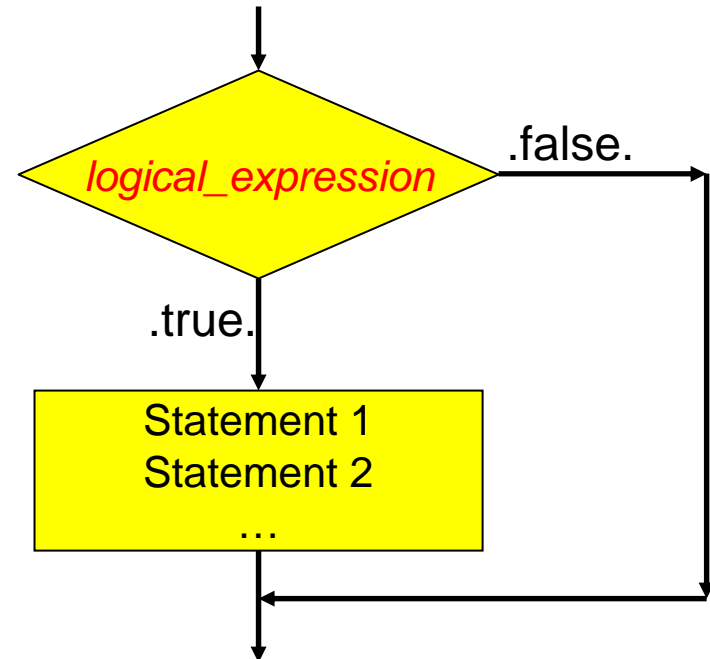
- Form:

```
if(logical_expression) then
    statement 1
    statement 2
    ...
endif
```

- Example:

```
if(x > 0.0) then
    y = log(x)
endif
```

- If the logical expression is true block of statements is executed otherwise it is skipped



ELSE clause

The block if statement can be extended, with the ELSE clause, to allow execution of one block of code if the logical expression is true and another if it is false

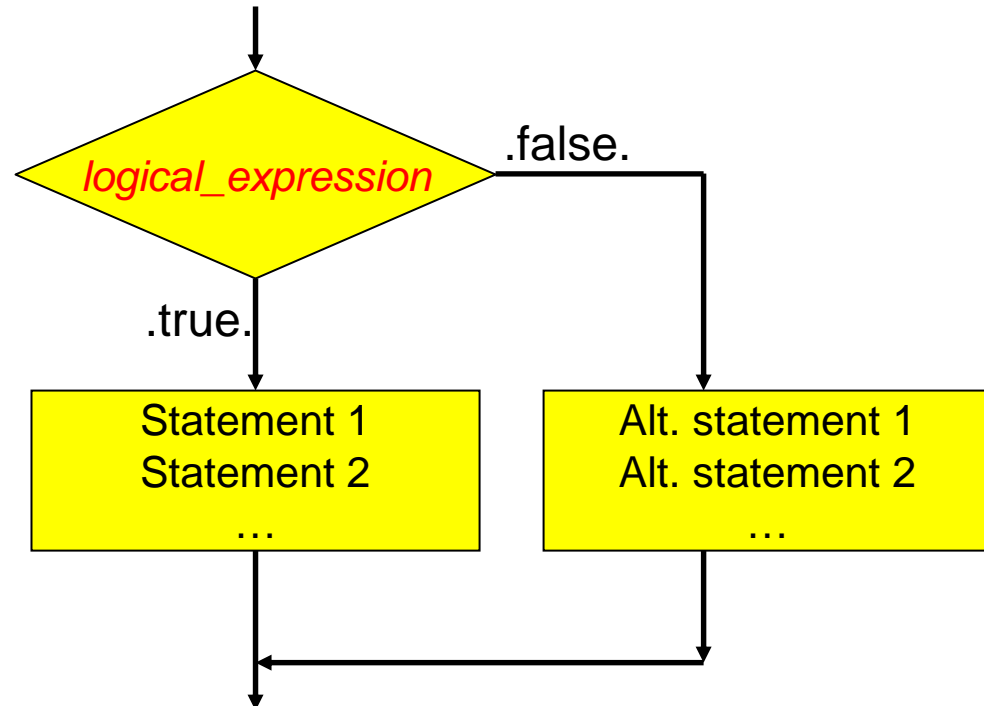
Form:

```
if(logical_expression) then
    statement 1
    statement 2
    ...
else
    alternative statement 1
    alternative statement 2
    ...
endif
```

Example:

```
if(x > 1.0) then
    f = x**2
else
    f = x
endif
```

If the logical expression is true
first block of statements is executed
otherwise the second block is executed



ELSEIF clause

- Arbitrary numbers of blocks of statements can be executed based on a whole series of conditions using ELSEIF clauses

Form:

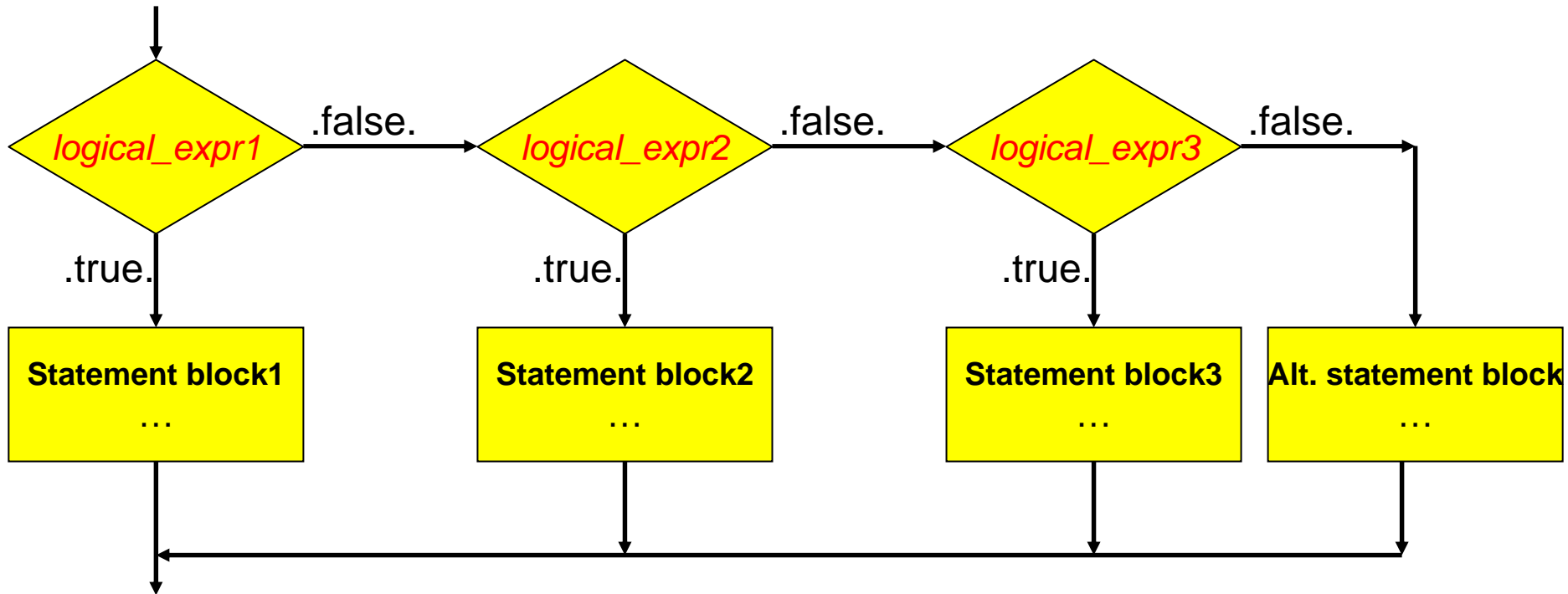
```
if(logical_expr1) then
  statement block 1
  ...
elseif(logical_expr2)
  then
  statement block 2
  ...
elseif(logical_expr3)
  then
  statement block 3
  ...
else
  alternative statement block
  ...
endif
```

Example:

```
if(x > 2.0) then
  f = 2.0e0*(x**2)+3.0e0
elseif((x <= 2.0).and.(x > 0.0)) then
  f = x
elseif((x <= 0.0).and.(x > -1.0)) then
  f = -1.0*x
else
  f = -1.0*(x**2)
endif
```

- If the first logical expression is true, the first block is executed and the program jumps to the first executable statement after the endif.
- Each subsequent ELSEIF block is tried in turn. If the expression is true the block is executed and the program jumps to the first executable statement following the ENDIF statement.
- If none of the logical expressions are true, then the block of statements following the ELSE clause (if it is present) is executed.

Block IF-ELSEIF-ELSE construct flowchart



Example of IF-ELSEIF-ELSE Construct

```
! Purpose: Evaluate a discontinuous function
! Author:  F. Douglas Swesty
! Date:    9/19/2005
program function_eval
implicit none                                ! Turn off implicit typing
real :: x                                    ! Input variable
real :: f_of_x                               ! Function-value variable
write(*,*) "Enter x:"                       ! Prompt the user to enter x
read(*,*) x                                  ! Read in x

if((x>2.0).and.(x<3.0)) then
  f_of_x = 3.0+x**2                          ! Evaluate f(x) when 2 < x < 3
elseif((x> 0.0).and.(x <= 2.0)) then
  f_of_x = log10(x)                          ! Evaluate f(x) when 0 < x <= 2
else
  write(*,*) ' warning: x must be in the range 0 < x < 3' ! Otherwise warn the user
endif

write(*,*) ' f('x,') = ',f_of_x             ! Output the value of the function
stop                                        ! Stop execution of the program
end program function_eval
```

Programming Style Tip:

Indent the statement blocks to make your code more readable

Reading Assignment

- Re-read Sections 3.3 paying close attention to detail