

Chapter 4: Looping Structures

39

- used in situations where you need to repeat an operation numerous times.

Example

- compute the sum of 5 positive integers

```
READ(*,*) I1, I2, I3, I4, I5
```

```
total = I1 + I2 + I3 + I4 + I5
```

```
WRITE(*,*) 'sum is = ', total
```

- This works, but there is a conceptual problem.

- * only works for 5 integers.

- * in principle, the idea can be extended to any number of integers.

- * would be impractical for 100 integers and useless for a million.

- we would like a way to do this that works for any number of integers

Example

Program to add up an unspecified number of positive integers

- we will use an uncounted loop to do this
- You use this type of loop in situations where you don't know ahead of time how many times the loop must execute.
- This is also an example of an accumulation operation

INTEGER :: rvalue, total

total = 0

important! Always set the value of the accumulation variable to a known value before starting the loop. Never assume that variables are automatically set to zero at the start of the program.

```

loop1: DO
  WRITE (*,*) 'input next value'
  READ (*,*) rvalue
  IF (rvalue < 0) THEN
    EXIT loop1
  END IF

  total = total + rvalue
ENDDO loop1

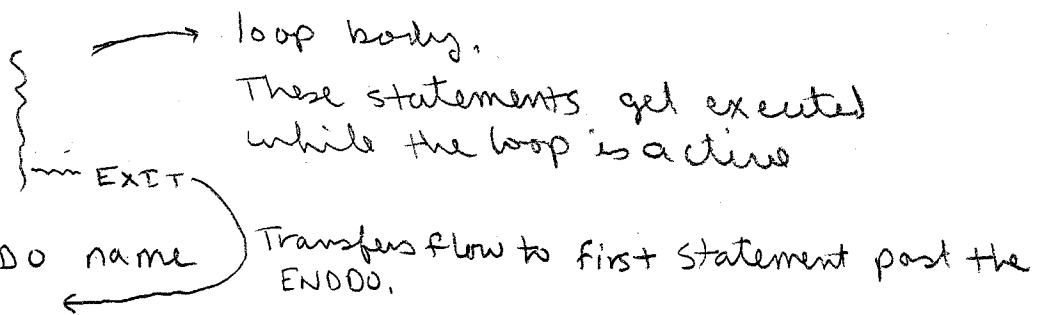
WRITE (*,*) 'sum is = ', total

```

uncounted DO Loop

- Syntax

name: DO



- * name is an optional loop name
- * loop must have an exit condition within the body that eventually triggers, otherwise the loop will never terminate
- * used in situations where you don't know prior to the start of the loop how many times the loop needs to execute
- * This is a variation on what is called a WHILE loop.

WHILE loop

- Syntax

name: DO WHILE (condition)

{ body

ENDDO name

- * This loop executes until the logical condition becomes false
- * you need to be certain the condition becomes false at some point in the body otherwise the loop becomes infinite
- * Bodies of DO loops should be indented.