Repetition

Objectives

- To understand that algorithms often involve the repetition of statements.
- To understand the form of test and termination in looping.
- To understand the form and function of flowchart elements for repetition.
- To understand computational state of repetition in algorithms.
- To understand use of counter and sentinels in loops.
- To learn to write algorithms using loops, as well as using selection statements within loops.

A Repetitive Task

✓ *In everyday life we must often repeat a sequence of actions in order to achieve some greater outcome.*

- You *paint a wall*, by repeatedly dipping your brush into the paint can and applying the paint to the wall.
- You *construct a fence*, by repeatedly digging a hole and inserting a fence post.
- You even *sing a song* by repeatedly singing a verse followed by the chorus.

✓ A *repetitive task* is a repeated set of actions/instructions.

Eg. Read books on the bookshelf

✓ *This is a situation where I am doing a similar thing over and over again.*

- The repetitive task is:
  1. Pick a book that you have not read.
  2. Read the book.
  3. Put the book back.

✓*Do you think the description is complete?*

✓*Does the description include when I should continue and when I should stop?*

The Termination Condition

✓ *It is not enough to give the instructions to be repeated.*

- The instructions on when to stop (*the termination condition*) should be given.
- We most commonly phrase this question the other way round in computing,
  - we describe the situations in which *we should keep going.*
- For the example here, the termination condition is:

  Continue provided there are books on the shelf you have not read.

✓ *Can you put these together as an algorithm?*

The algorithm is:

while

there are books on the shelf you have not read

do the following repeatedly

a. Pick a book that you have not read.

b. Read the book.

c. Put the book back.

Search for an article in a pile of magazines

- *I wish to find an article I remember reading about "Sabre-tooth Wombats".*
- *I think the article was in the magazine Dinosaurs Today, but am not sure.*
- *If it was, then it will be in the pile of Dinosaurs Today in my loft (I have every one).*
- The repetitive task is:
  1. Take the next *Dinosaurs Today* from the pile.
  2. Check whether the article is in it.
  3. Add it to a discard pile.

✓ *What is the condition for continuing to search?*

✓ *Write the algorithm putting all this together.*

What is the condition for continuing to search?

- The termination condition would be:

  I am not at the bottom of the pile and I have not found it yet.

- When either of the above are not true I would stop.

- Note, that this is just one of our true/false Boolean questions.

Putting them together we have the following:

while

I am not at the bottom of the pile and I have not found it yet

do the following repeatedly

a. Take the next *Dinosaurs Today* from the pile.

b. Check whether the article is in it.

c. Add it to a discard pile.

The question: *"Do I continue?"*

- The termination condition that tells us when to continue and when to stop can thus be thought of as a question meaning "*Do I continue*".
- In the first example, the question is:

  "*Are there still books on the shelf I have not read?*"

  - If the answer to this question is *Yes* then

    I must continue.

  - If the answer is *No* then

    I stop.

- Similarly in the second example we are asking the question:

  "*Am I not yet at the bottom of the pile and have not yet found it?*"

  ✓*We ask this question - "Do I continue?" - once every time we have done the repetitive task, to see if we need to do it again.*

An algorithm for treating heart stroke

(BMA, 1990)

1. Move the victim to a cool place.
2. Remove clothing.
3. Place the victim in the sitting position.
4. Support the head and shoulders using pillows.
5. Cover the victim with a wet sheet.
6. Fan the victim with a magazine until the temperature drops to 38C.
7. Seek medical help immediately.

- ✓ *Can you identify the loop in it and say what the repeated action is and what is the termination condition?*
- ✓ *Can you rewrite the algorithm using a loop?*

The loop in Treating heatstroke - algorithm

- The loop is in line 6.
- What are we told to do repeatedly?

  "*Fan the victim with a magazine*".
- What is the condition we have to keep checking to determine if we can stop?

  "*The temperature drops to 38C*".

✓ *Rewriting line 6 we get.*

While the temperature is above 38C **do the following repeatedly**

Fan the victim with a magazine.

Eg. A maze problem

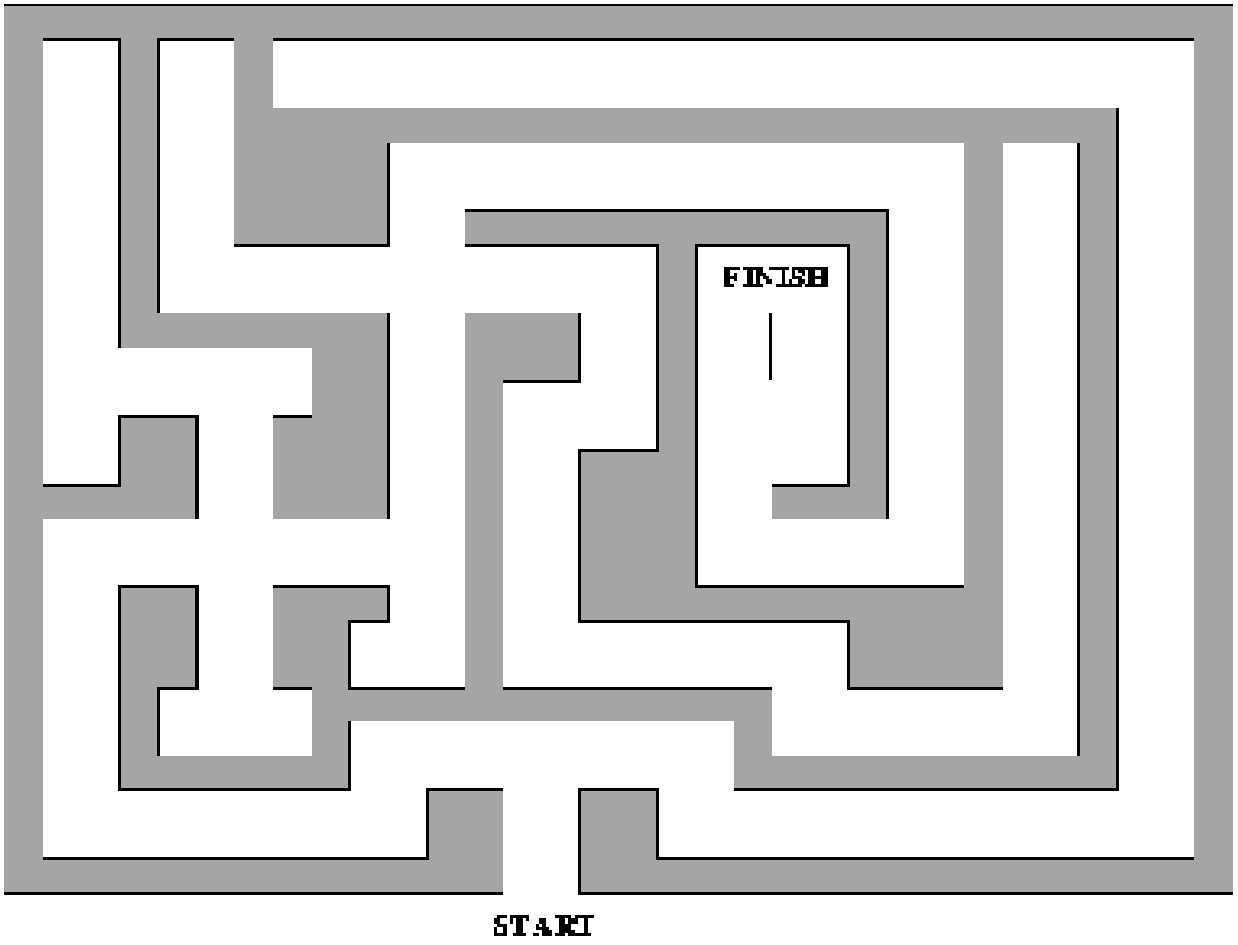


Figure: Original Maze

✓*Solve the maze by drawing a line.*

✓*Can you write an algorithm to get to the centre of the maze?*

To get to the centre of the maze

1. Enter the maze by the gate following the path ahead.
2. Turn left following the path ahead.
3. Go straight across the junction following the path ahead.
4. Turn left following the path ahead.
5. Go straight across the junction following the path ahead.
6. Stop at the cross.

✓ *Why write the same instructions out several times if we can avoid it?* .....

Is it worthwhile avoiding repeatedly writing the same instructions?

- This maze is small so the amount of repetition in the instructions is not great –
  - but imagine if the maze was much bigger with more junctions, then if there was repetition it would be very worthwhile avoiding repeatedly writing the same instructions.

- Consider that you are learning them for a race, it will be easier to learn if we can make the instructions simpler.

✓ *We can simplify this algorithm using a loop*.

Eg. A simple maze

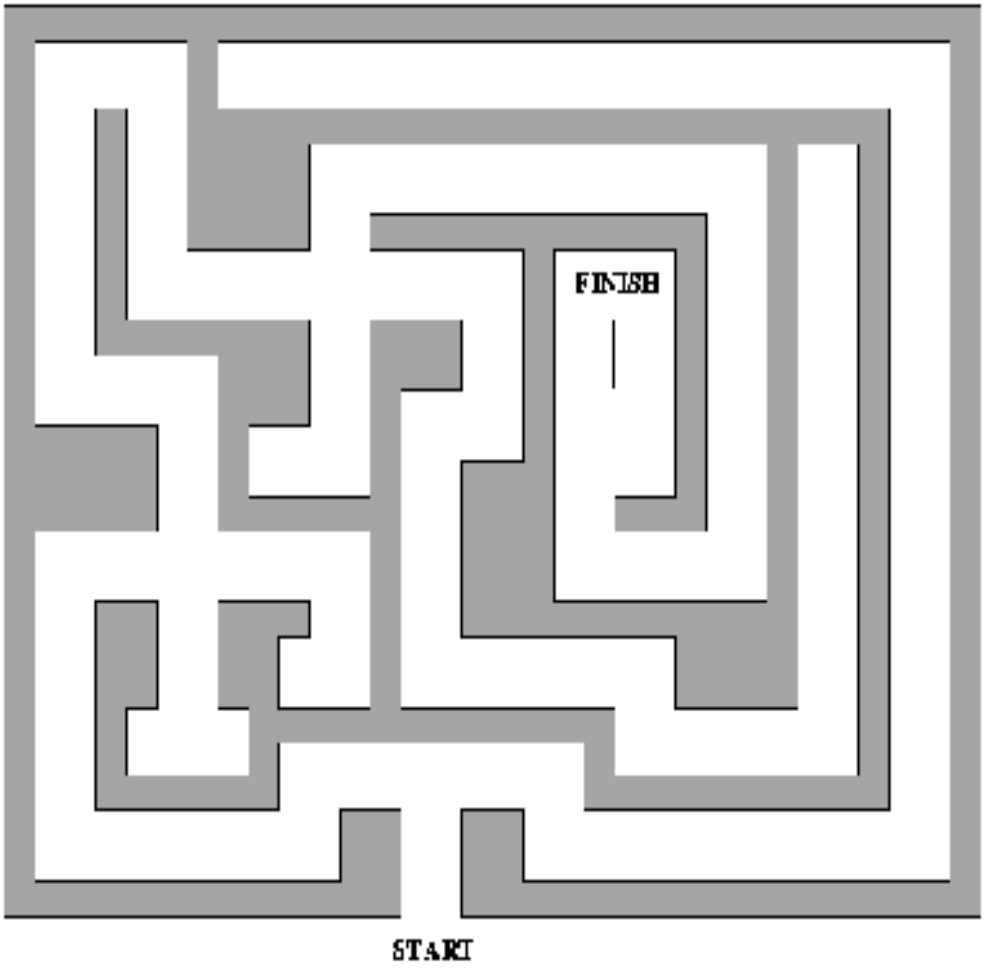


Figure: Simple Maze

To get to the centre of the maze:

1. Enter the maze by the gate following the path ahead.
2. Turn Left following the path ahead.
3. Turn Left following the path ahead.
4. Turn Left following the path ahead.
5. Stop at the cross.

✓*Notice any pattern?*

"*At every junction I must turn left and then follow the path ahead*"----

Devising the loop

We must ask ourselves *two questions* to devise the loop:

- "*How do we know when to continue*?"

  We continue as long as we are not at the cross.

- "*What in the above is being repeated*?"

  It is the instruction: 1. Turn Left following the path ahead.

- We can write the algorithm for *simple maze* using loop as:

  1. Enter the maze by the gate following the path ahead.
  2. **While** not at the cross **do the following repeatedly**
     a. Turn left following the path ahead.

✓ *What does the algorithm say?*

What does the algorithm say?

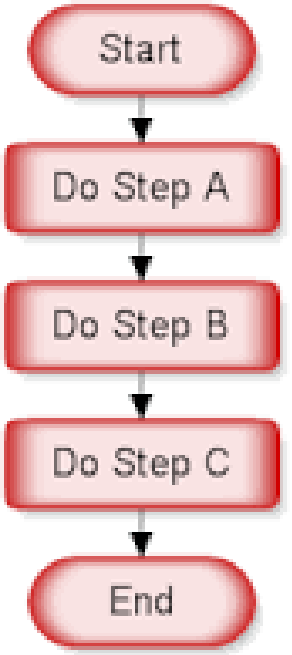
- It says that what we should do is first enter the maze and follow the path ahead.
- We then must follow the loop instruction.
- If we follow this we expect to repeatedly turn left and follow the path ahead.
- However the first thing it tells us to do is to check if we are at the cross because if so we have finished.
- If not then we turn left and follow the path ahead.

If not at the cross of the maze, then what?

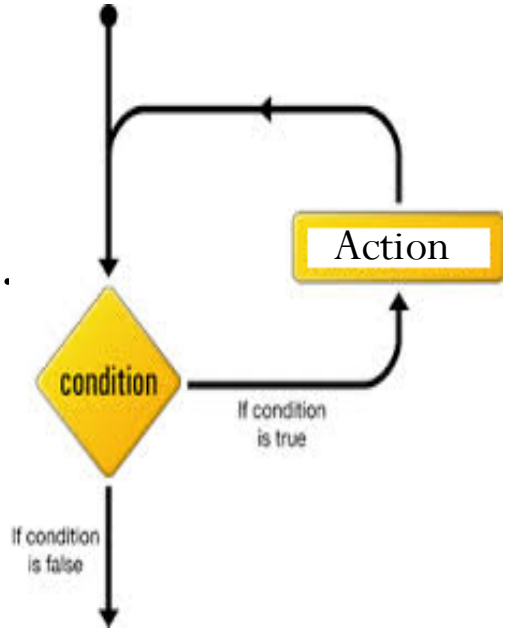
- If not we will need to continue and follow the instructions in the loop again.
- We therefore check if we are at the cross to decide whether we have finished – we go back to the top of the loop.
- Only if we are at the cross do we carry on with the rest of the algorithm.
- This algorithm would get us to the centre of any maze where you must turn left at any junction, irrespective of the number of junctions.

Sequencing and looping

- In sequencing, we just follow instructions down the page, stopping when we get to the last one.



- With a loop we repeatedly do the instruction that is inside the loop.
- In this case:
  - "*Turn left following the path ahead*".
  - On getting to the bottom of the loop we go back to the top and ask the question again.
  - Perhaps we are at the cross in which case we could stop.



The Original Maze revisited

- In it you sometimes had to turn left and sometimes go straight on.
- *What is the thing that we want to do repeatedly?*
  - This time it is two instructions: turning left then going straight on at the next junction.
- *When do we stop*?
  - If we are at the cross.
- The algorithm using *loop* is:

1. Enter the maze by the gate following the path ahead.
2. **While** not at the cross **do the following repeatedly**
   a. Turn left following the path ahead.
   b. Go straight across the junction following the path ahead.

Eg. A T-junction maze

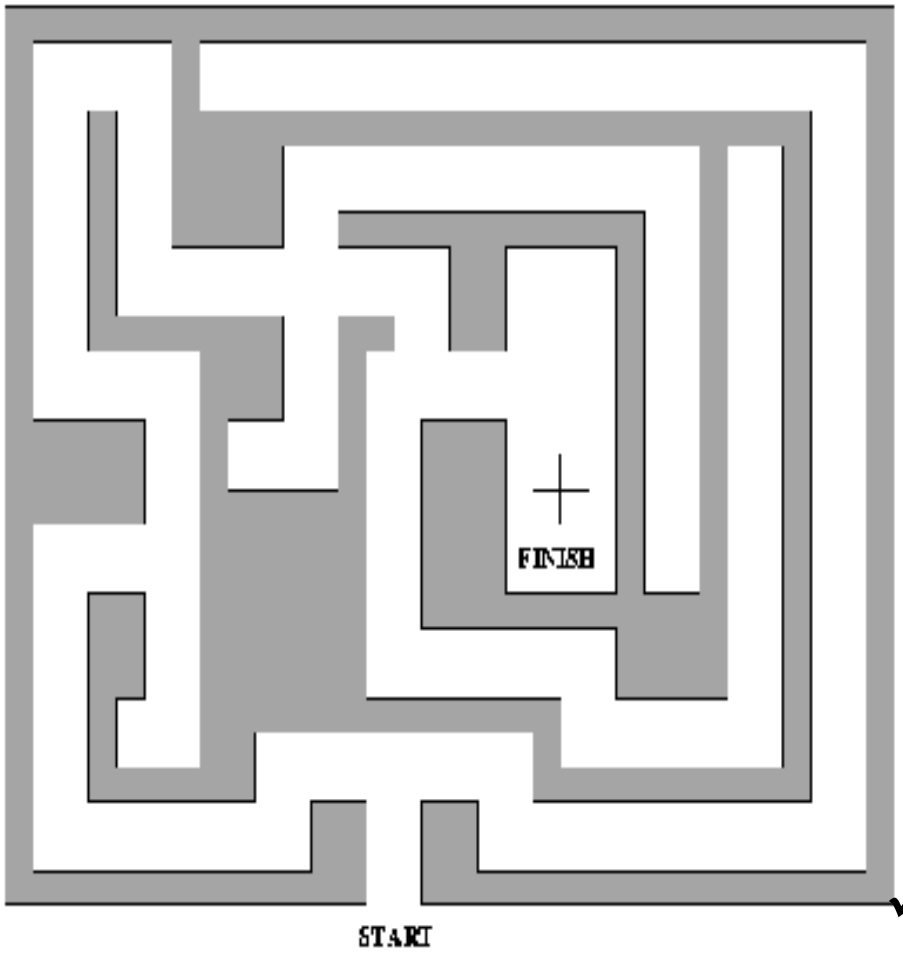


Figure: T-junction Maze

To get to the centre of the maze:

1.Enter the maze by the gate following the path ahead.

2. Turn left following the path ahead.

3. Turn left following the path ahead.

4.Go straight across the junction following the path ahead.

5. Turn left following the path ahead.

6. Stop at the cross.

✓ *What do you observe?*

✓ *Can we use this observation to rewrite the instructions using the if statement?*

✓ *Algorithm using the if-control structure is :*

✓ *Notice any pattern? Now, go for the if statement in a loop.*

What do you observe?

- It is obvious what the repetition is.

- The secret of this maze is that whenever you get to a T-junction you turn left and whenever you get to a crossroads you go straight on.

- In other words, if you are at a T-junction turn left else go straight on.

Algorithm using the if-control structure

1. Enter the maze by the gate following the path ahead.
2. **If** at a T-junction
   then turn left following the path ahead
   else go straight across the junction following the path ahead.
3. **If** at a T-junction
   then turn left following the path ahead
   else go straight across the junction following the path ahead.
4. **If** at a T-junction
   then turn left following the path ahead
   else go straight across the junction following the path ahead.
5. **If** at a T-junction
   then turn left following the path ahead
   else go straight across the junction following the path ahead.
6. Stop at the cross.

✓*Notice any pattern? …….*

Now, go for the *if statement* in a loop

1. Enter the maze by the gate following the path ahead.
2. **while** not at the cross **do the following repeatedly**
   if at a T-junction
      then turn left following the path ahead
      else go straight across the junction following the path ahead.

✓ *You check if you are the cross and if not check if you are at a T-junction taken the specified action depending on the answer.*

✓ *Then you do it all again, stopping if you are at the centre of the maze.*

The *while* loop

- Just as many real-life activities require *repetition*, computers often must also execute a sequence of actions repeatedly in order to achieve some greater outcome.
- A *loop* is a control structure that repeatedly executes a sequence of actions.
- A *while loop* is a type of loop where a sequence of actions is repeated as long as some *logical condition holds*.

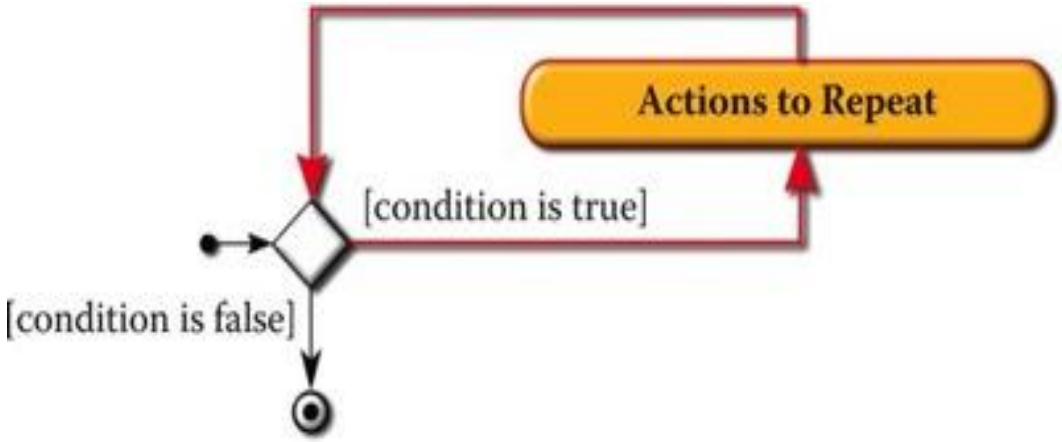


Figure: Flowchart structure for a *while loop*

Reading the flowchart structure

- When the *while loop is first encountered*, we must determine whether to perform some action or sequence of actions at least one (more) time.
- This decision is made by asking a yes/no question.
- If the *question is answered affirmatively*, then the actions are performed after which we again ask whether we must repeat the sequence at least one more time.
- The repeating loop is highlighted by the red arrow in the Figure: Flowchart structure for a *while loop*.

Syntactic pattern of *while* loop

- The *condition* and the *actions* contain elements that make the loop meaningful.
- In particular, the *condition* must be an expression that produces a logical value and the *actions* is a meaningful sequence of actions.
- The sequence of actions of a loop is often referred to as the *loop body*.
- The condition controls how many times the loop body actions will be executed.

Eg. Grill a steak

- We might describe a repeated sequence of actions with a phrase similar to
  - "*while the steak has not yet reached a temperature of 135°F, cook for three minutes*".
- The logical condition is "*the steak has not yet reached 135°F*".
- The repeated action is "*cook for three minutes*."
- A flowchart that models this process is :

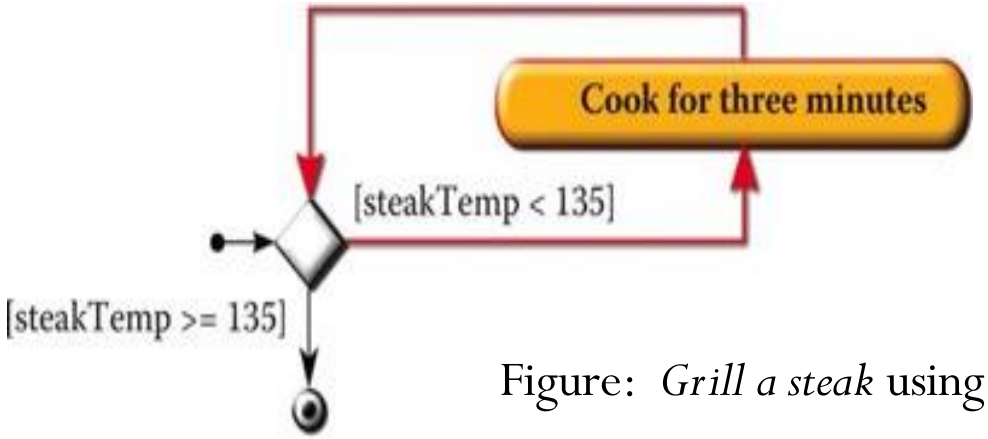


Figure: *Grill a steak* using while loop

Modeling the temperature of a steak as it is grilled

- Assume that the steak is placed on the grill *at a room temperature of* 75° and that for *every three minutes* the steak remains on the grill the *temperature increases* by 13°.
- An algorithm for the same is:

```
1. steakTemp ⟵ 75
2. while steakTemp < 135 do
3.    steakTemp ⟵ steakTemp + 13
4. endwhile
```

✓ *What do you observe?*

What do you observe?

- The first line binds the value 75 to the name *steakTemp*;
- It is a variable that models the current temperature of the steak.
- Of course, the current temperature of the steak will increase with the amount of time that the steak remains on the grill.
- The loop is controlled by the logical condition

  "*steakTemp* < *135*"

- The repeated action is expressed as

  "*steakTemp* ← *steakTemp* + *13*"

- In other words, the loop repeatedly increases the steak temperature in increments of 13 degrees until the minimum desired temperature is attained.

Computational state

Line number	Comment	State after execution
1	Bind 75 to steakTemp	{steakTemp=75}
2	The condition steakTemp < 135 is true (therefore repeat)	{steakTemp=75}
3	steamTemp = steakTemp + 13	{steakTemp=88}
2	The condition steakTemp < 135 is true (therefore repeat)	{steakTemp=88}
3	steamTemp = steakTemp + 13	{steakTemp=101}
2	The condition steakTemp < 135 is true (therefore repeat)	{steakTemp=101}
3	steamTemp = steakTemp + 13	{steakTemp=114}
2	The condition steakTemp < 135 is true (therefore repeat)	{steakTemp=114}
3	steamTemp = steakTemp + 13	{steakTemp=127}
2	The condition steakTemp < 135 is true (therefore repeat)	{steakTemp=127}
3	steamTemp = steakTemp + 13	{steakTemp=140}
2	The condition steakTemp < 135 is **false** (therefore stop)	{steakTemp=140}
	Since the condition is false we are done with the program. The variable steakTemp is bound to the value 140.	{steakTemp=140}

Figure: The computational state of *steak grill algorithm*

Further insight into the meaning of a while loop

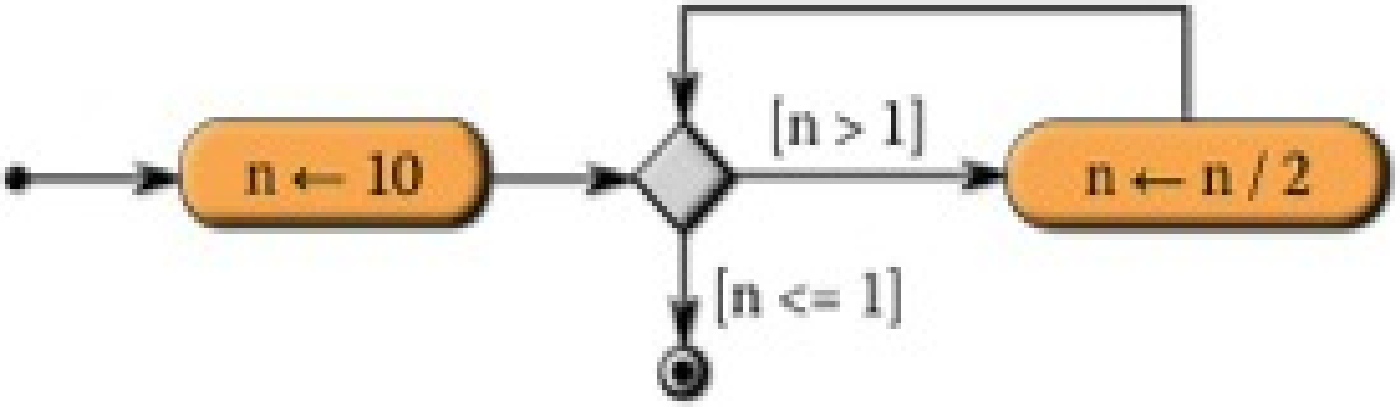
- The figure gives a list of each action the computer takes when executing the algorithm and also shows the computational state that results from each individual action.
- The steak temperature is initialized to the value 75.
- After this initial action, the algorithm repeatedly decides whether to leave the steak on the grill for three more minutes (the result being to increase the steak temperature by precisely 13 degrees).
- Since, at this point in time, the steak temperature is less than 135, the algorithm decides to grill for three more minutes; and thus the steak temperature changes from 75 to 88.

✓ *Any more insight?*

Insight *contd*----

- After several times through this process, the steak temperature finally increases to a temperature that is not less than 135.
- In that case, the algorithm decides that the steak should not be grilled for another three minutes and the loop terminates.
- The algorithm for grilling a steak computes the final temperature of the steak at the conclusion of grilling.
- The final temperature may not be exactly 135 but, as the example illustrates, may be higher than 135 depending upon the starting temperature and the amount by which the temperature rises within a three-minute span.

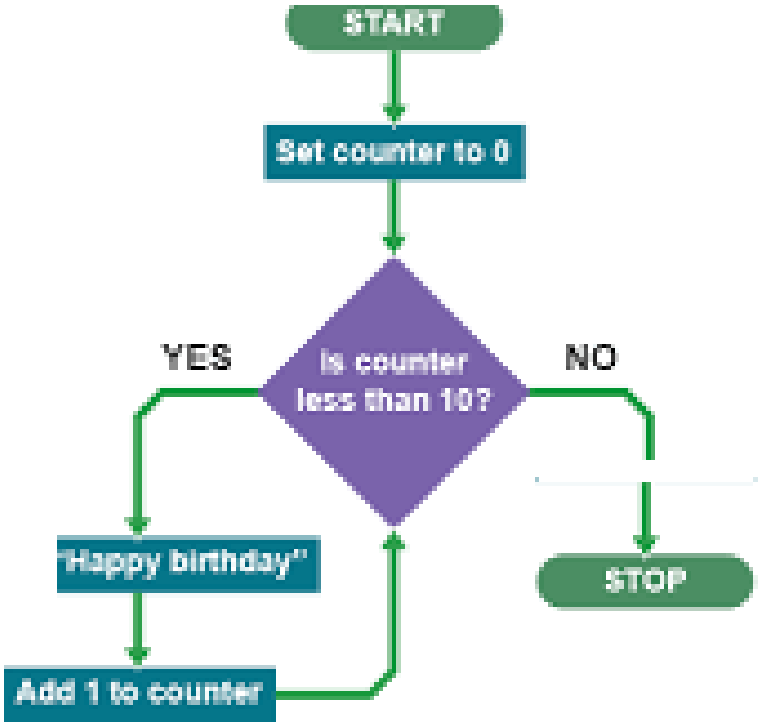
✓ *Indicate the computational state for the flow chart below:*



- List action(s).
- Indicate the computational state that follows from each action.

Counter-controlled loops

- One of the most common forms of loop occurs where you *know in advance how many times* the repetition must be done.
- Doing a task a fixed number of times whilst keeping a count is known as a *counter-controlled loop*.



Eg. Imposition

- A teacher giving a pupil *a punishment line* "*I must not pick my nose in class*"*to be written 100 times* is an example.
- The pupil *knows exactly how many times* they must write the sentence before they start.
- *Keeps a count of how many times* they have written the sentence: perhaps keeping a tally (making marks in groups of five each time they write a line).
- When the *tally gets to a 100* they stop.

✓ Without a *counter* of some kind they would have no way of knowing when to stop.

Imposition algorithm

- *What do we do repeatedly?*

  We write the punishment line, but we also make another tally mark.

- *What is the termination condition?*

  We continue while the tally is not 100.

- The algorithm using a while loop is:

while the tally is not showing 100 **do the following repeatedly**

a. Write "*I must not pick my nose in class*".

b. Add one to the count by making another tally mark.

Eg. Hide and seek

- The child doing the seeking must cover their eyes ,*count to 10* then shout "*Coming, ready or not*" then look for the other children.

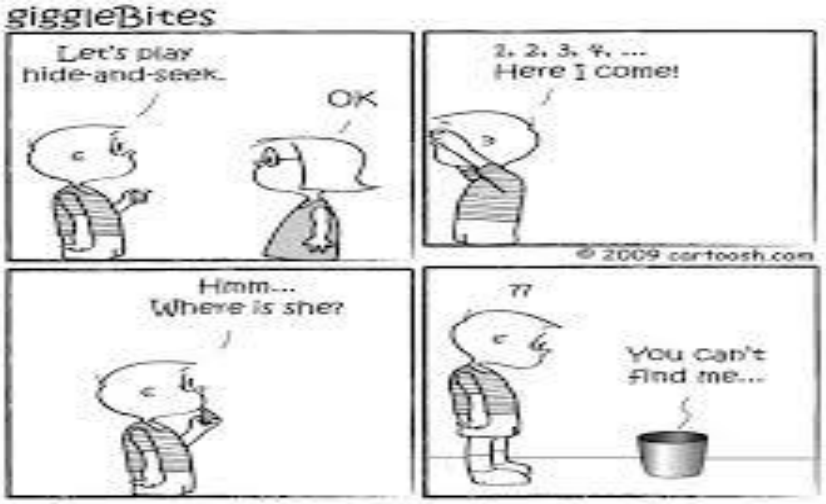
- The *counting part* involves adding one to the last number you thought of.

✓*Write the algorithm.*

while the number you are thinking is not 10 **do the following repeatedly**

a. Add one to the number you are thinking of.

What number to start with?



- *One* presumably.
- We ought to say that in our instructions (as otherwise a devious child might count "9, 10").
- We need to say *what its first value* is.
- In fact strictly we should have done this in the imposition example too – *make sure our tally was zero when we started writing lines*.

✓ *Write the full instructions for hide and seek.*

✓ *What do you observe?*

The full instructions for hide and seek

1. Think of the number 1.
2. **While** the number you are thinking is not 10 **do the following repeatedly**
    a. Add one to the number you are thinking of.
3. Shout "Coming, ready or not".
4. Look for the other children.

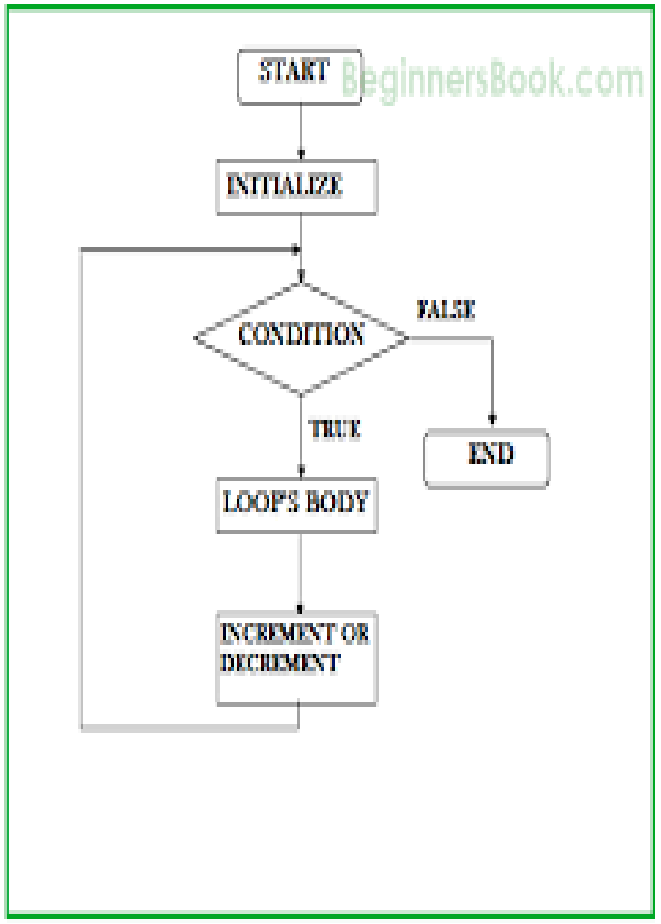
✓ *In fact strictly we should have done this in the imposition example too – make sure our tally was zero when we started writing lines.*

What do you observe?

- Notice that these instructions do not tell you to shout "*Coming, ready or not*" repeatedly.
- That instruction is not part of the loop.
- It comes after the loop and is not in the body.
- It is the instruction you follow after the loop has finished: after the termination condition allowed you out of the loop.
- You will only shout when the number you are thinking of gets to 10.

The counter-controlled loop

- A general flow:
  Set the *counter* to its initial value.
  while the *counter* is not the final value
  do the following repeatedly
    a. Do the tasks that are to be repeated the known number of times.
    b. Change the *counter*.
- For example,
  Set the counter to zero.
  while the counter is not 100 **do the following repeatedly**
    a. Write "*I must not point my fingers at Jimmy in class*".
    b. Add 1 to the counter.



Counting *upwards* or *downwards*?

- In the above examples we always added one to the counter when we changed it.
  - ✓ *That makes us count upwards*.
- We could equally well count down too.
  - ✓ *Then we would set the counter to the number we wanted to count down from.*
  - ✓ *We would subtract one from our count each time and the number we would stop at would be 0.*
- For example,

  Set counter to 10

  While counter is not 0 **do the following repeatedly**

  a. Write "*I must not set fire to Sally's hair in class*".

  b. Subtract one from counter.

A Nursery rhyme

- This rhyme is just describing a series of actions done in a given order.

  There were ten in the bed and the little one said,"Roll over, Roll over"

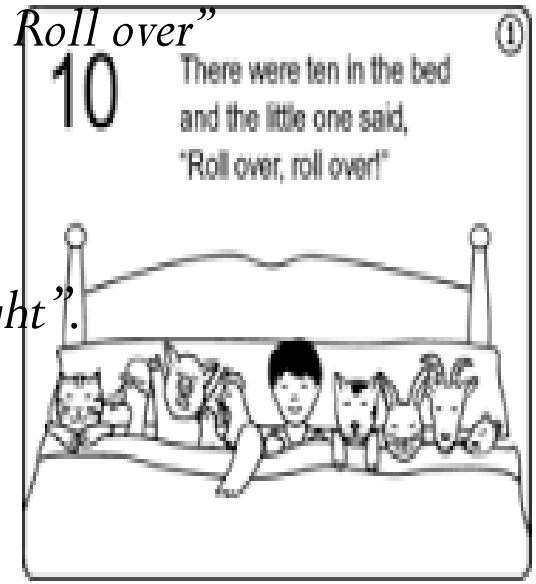
  So they all rolled over and one fell out.

  There were nine in the bed and the little one said,"Roll over, Roll over"

  So they all rolled over and one fell out.

  etc

  There were none in the bed and the little one said"Good Night".



✓*Re-interpret it as an algorithm and write it as a loop.*

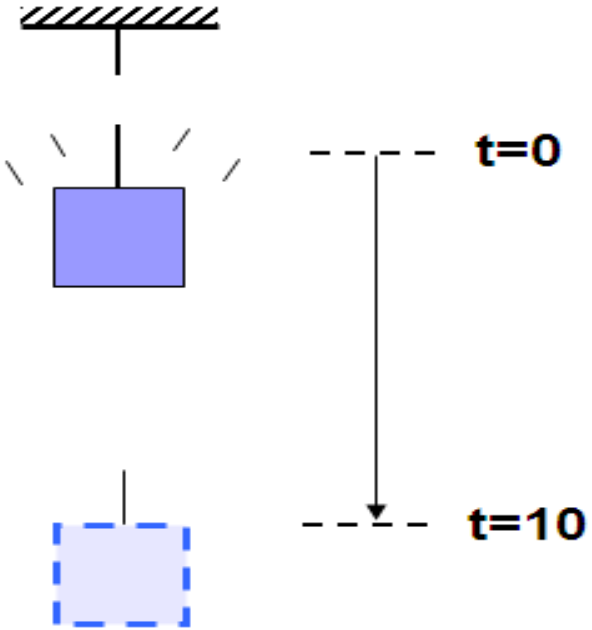
Nursery rhyme- algorithm

1. Ten get into bed.
2. **while** there are any in the bed **do the following repeatedly**
   a. The little one says "Roll over".
   b. They all roll over.
   c. One falls out.
3. The little one says "Good night".

- ✓ *We initialize the number in the bed.*
- ✓ *Our termination condition is that this number is not zero.*
- ✓ *Some actions occur repeatedly and after each time we reduce the number (in the bed) by one.*
- ✓ *It is a counter controlled loop that counts down.*

Repeating calculations

- Consider an object dropped from a height at t = 0 sec.
- You are asked to calculate the distance traveled each second by the object in a standard gravitational field over the next 10 seconds.



$$d = \frac{1}{2}gt^2 + v_0 t + d_0$$

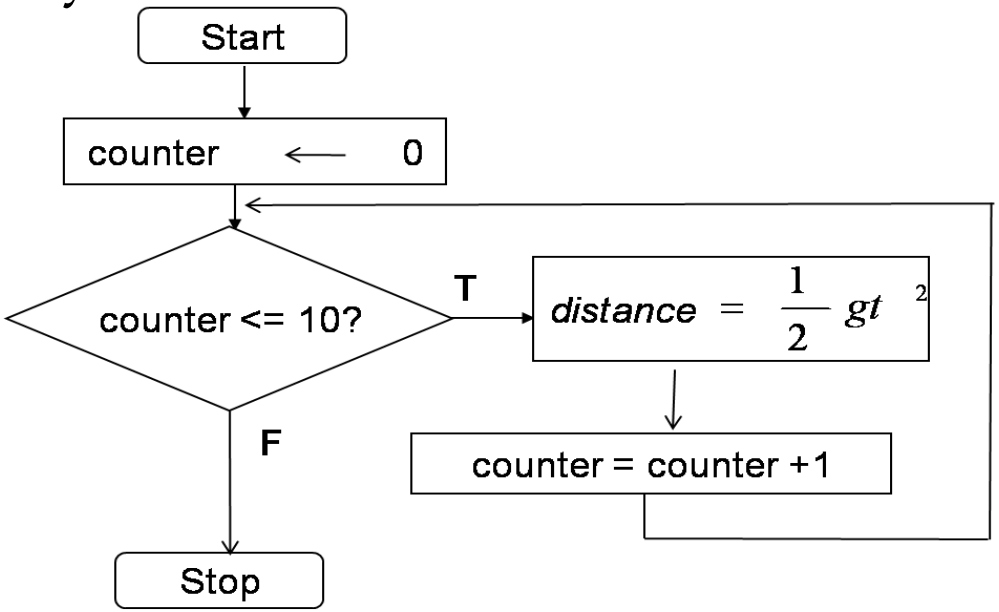
Simplify: Assume v_0 and d_0 are both zero

$$d = \frac{1}{2}gt^2$$

✓ *Give the flowchart and algorithm.*

Algorithm and flowchart

1. Set the counter to its initial value
2. **While** counter is less than or equal to 10 **do the following repeatedly**
   a. calculate distance
   b. increment the counter by one



Playing darts

- When playing darts, each person has three darts.

- For each of their turns, they have to get the highest score they can with just three darts.

✓*Write the instructions for throwing darts as a counter-controlled loop.*

Eg. Grill a steak

- Perhaps the chef is cooking a large meal that includes steak, baked potatoes, and steamed broccoli.
- The chef needs to *know the time required for grilling* the steak in order to ensure that all three dishes are completed at about the same time.
- A small addition to the algorithm for modeling the temperature of a steak as it is grilled will allow us *to compute additional information* that a grill chef might require.

Length of time required

- To *compute the length of time required*, we can introduce a new variable that represents the number of minutes of grilling that have elapsed at the conclusion of every three-minute span of time.
- This variable should obviously be initialized to zero prior to entering the loop since, at that point in the process, the steak has not yet been put on the grill.
- Each execution of the loop should increase the variable by three minutes since our model states that the temperature is checked in three minute intervals.
- Once the loop terminates, the value of the variable will then contain the length of time required to grill the steak.

Time to grill-algorithm

1. steakTemp ← 75
2. minOnGrill ← 0
3. while steakTemp < 135 do
   a. steakTemp ← steakTemp + 13
   b. minOnGrill ← minOnGrill + 3
4. endwhile

Figure: To compute time taken to grill a steak

- You will quickly recognize that the loop is effectively *a counting loop*.
- A *counting loop* is one that contains a variable to keep track of the number of times the loop is actually executed.

The counting loop

- The variable *minOnGrill* is initialized to zero in line 2 and incremented by three in line b.
- *minOnGrill* effectively counts the number of times that the loop executes, although the count is maintained as a multiple of three.
- Prior to executing the loop body, the variable is initialized to zero, thus denoting that the loop has been executed zero times.
- After the loop body is executed for the first time, the *minOnGrill* variable holds the value 3, denoting that the loop has been executed 1 time (1 multiple of 3).
- After the loop is executed for the second time, the *minOnGrill* variable holds the value 6, denoting that the loop has been executed 2 times (2 multiples of 3).

Well-written loops

1. *Initialization*
   - Every variable that occurs in the loop must hold the correct value prior to entering the loop.
   - The initialization of the loop in Figure: To compute time taken to grill a steak, is performed on lines 1 and 2.
   - We state that the *steakTemp* immediately prior to the grilling process is 75 degrees and that the steak has been on the grill for 0 minutes.

2. *Condition*
   - The condition that determines when to repeat the loop must be precise.
   - Since the chef requires the steak to reach 135 degrees, the condition, given on line 3, indicates that the process will continue as long as *steakTemp* < *135*.

Well-written loops *contd*---

3. *Progress*

- The actions that are repeatedly executed must in some way make progress that allows the loop to terminate.

- Since the condition implies that the loop will terminate only when the *steakTemp* is not less than 135, the loop makes progress toward termination by increasing the *steakTemp* by 13 with every repetition.

- This repeated action ensures that the loop will eventually terminate since the *steakTemp* must eventually exceed 135.

✓ *Write an algorithm to calculate the sum of n numbers.*

✓ *Modify the above algorithm to calculate the average.*

✓ *Can you try writing an algorithm for generating addition and multiplication tables?*

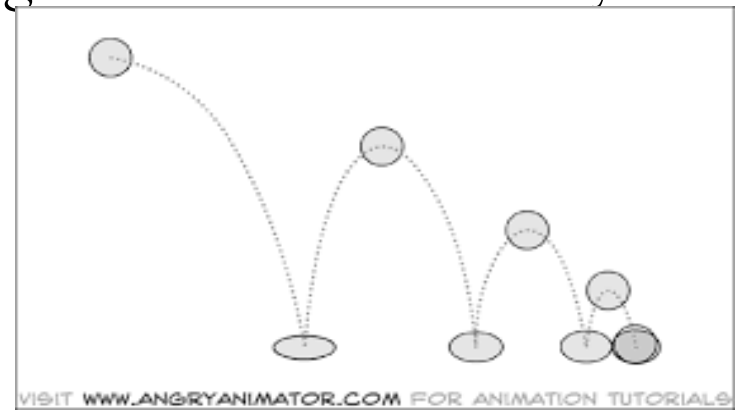
The sum of *n* numbers

1. Prompt and read in the number of numbers to be summed.
2. Initialize sum for zero numbers.
3. While less than *n* numbers have been summed repeatedly do
   (a) read in next number,
   (b) compute current sum by adding the number read to the most recent sum.
4. Write out sum of *n* numbers.

✓ *How about calculating the average?…….*

Bouncing ball

- Consider a ball that when dropped, it bounces to a height one-half to its previous height.
- We seek an algorithm which displays the number of the bounce and the height of the bounce, until the height of the bounce is very small.



✓ *What do you require*?

Height of bounce (current height) , threshold for height of bounce (the very small height).

✓*Write an algorithm for this.*

Bouncing the ball -algorithm

1. Initialize / set *bounce* to 0
2. Read / get value for *height*
3. Display original *height* value with label
4. While *height* >= *SMALL_NUM* do the following repeatedly

   // if *height* < *SMALL_NUM*, terminate loop

   a. replace *height* with *height* / 2

   b. add 1 to *bounce*

   c. display *bounce* and *height*

Endwhile

Remember

- Phases of loop execution
  - Loop entry ⟶ flow of control reaches first statement inside loop.
  - Iteration ⟶ each pass through the loop.
  - Loop test ⟶ condition tested before each iteration.
  - Loop exit ⟶ when termination condition occurs.
- The counter / loop control variable must be:
  - Initialized
  - Incremented /Decremented
  - Inspected

Sentinel-controlled loops

- Often the number of times a loop is executed is not predetermined.
- You may then use a value (*sentinel value*) to signify the end of the loop.
- To give an algorithm with *sentinel controlled repetition* we need to indicate that:
  - some instructions need to be repeated.
  - what the question is.
  - which instructions exactly we want to be repeated while the answer to the question means carry on.

Eg. Dice cricket

Dice Cricket

- A special Dice has on its sides the numbers 0,1,2,4 and 6 and on the last side is written "*OUT*".
- One player acting as the batsman repeatedly rolls the dice, adding up the scores obtained which correspond to the number of runs scored on that bowl.
- If they roll "*OUT*" then that batsman is out and it is the end of their turn.

✓ Here the termination question is:

"*Is the dice roll OUT*?".

✓ The *sentinel value* is "*OUT*" showing on the dice.

Dice cricket algorithm

while

the dice is not showing OUT

do the following repeatedly

a. Add whatever is showing on the dice to the current score.

b. Roll the dice.

- ✓ *We roll the dice.*
- ✓ *If the dice shows OUT we stop.*
- ✓ *If it does not, then we add the dice value to the score and roll again.*
- ✓ *We then repeat this all again and keep doing so until the dice does show stop, at which point we have finished the while instruction and carry on with any subsequent instructions.*

✓ *If we tried to play the game following the above rules exactly, would it work ?*

Can you see what is wrong?

- ✓ *Do I start with a score of 100 or of 0?*
- ✓ Understanding the game I know that the score starts at 0, but the algorithm does not say that, so somebody who was trying to learn the rules from my instructions would be confused.
- ✓ In fact if we were playing a series of rounds of the game then we might actually want the score to be its previous value.
- ✓ For now we will assume we are only playing one game so zero is the correct initial score.
- ✓ We need to add an extra instruction "*Set the score to 0*".
- ✓ Notice this is something that we only want to happen once so we need to make it clear that it is not part of the repetition, but comes first.

Set the initial score……..

1. Set the score to zero.
2. **while** the dice is not showing OUT **do the following repeatedly**

   a. Add whatever is showing on the dice to the current score.

   b. Roll the dice.

- ✓ *This still has a problem however.*
- ✓ *We set the score to zero, then go into the loop, asking if we should do the instructions to be repeated or stop straight away: is the dice showing OUT?*
- ✓ *However, we have not rolled the dice yet!*
- ✓ *We have not got to the instruction that tells us to do that yet!*

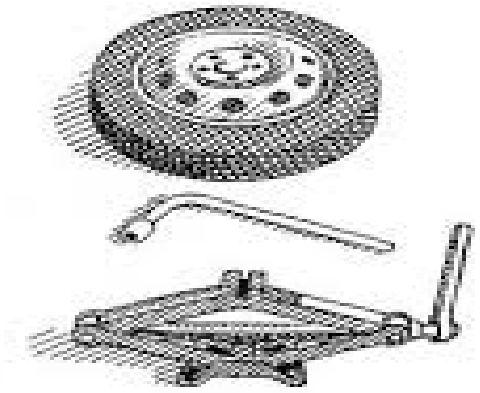
Roll the dice once first before we do the repetitive part, to get us started…….

1. Set the score to zero.
2. Roll the dice.
3. **while** the dice is not showing OUT **do the following repeatedly**
    a. Add whatever is showing on the dice to the current score.
    b. Roll the dice.

Eg. Motor racing

- Drivers repeatedly lap a circuit.
- On each circuit the pit crew hold up boards to give information to the driver.
- One of the situations in which a driver can be made to stop is if the information on the board tells him to come into the pits for a tyre change.
- Here the termination condition is:

"*Does the board on this lap say Change Tyres*



Eg. Throwing darts

- There each player takes turns throwing 3 darts and subtracting the score from their total.
- The game (and so the repetition) ends when the score hits exactly 0.
- The termination condition is thus:

  "*Is the score of the current player 0*?"

Termination condition

✓ *These termination conditions have the same form:*

They all ask "*Has a particular value arisen*?".

- The particular value being checked for differs completely in each case but the form of the question is the same.
- Because this kind of termination condition arises frequently the corresponding kind of loop is given a special name: a *sentinel-controlled loop*.
- The special value that if it arises causes the repetition to stop is called the *sentinel value*.

✓*Can you write an algorithm that reads and calculates the sum of an unspecified number of integers?*

Remember

Three questions to determine whether loops will be required in the general algorithm.

1. *Are there any steps I need to repeat to solve a problem? If so, which ones?*
2. If the answer to question 1 is yes, *do I know in advance how many times to repeat the steps?*
3. If the answer to question 2 is no, *how do I know how long to keep repeating the steps?*

Non-terminating loops

- ✓ *Say, we wanted to give precise instructions to* ***do nothing for ever and ever****.*
  - *How would we do it?*
  - *Is it possible using one of our loop constructs?*

- It clearly is a repetitive thing we want to be done.

- *What is it that we wish to be done over and over?*
  - Nothing!
- *Under what circumstances do we keep doing it?*
  - Always!

Eg. Do not mourn

"Don't mourn for me now, don't grieve for me never, For I'm going to do nothing for ever and ever". (quoted in Lovric 2000)

- Our loop will be something like:

  While true **do the following repeatedly**

  a. Do nothing.

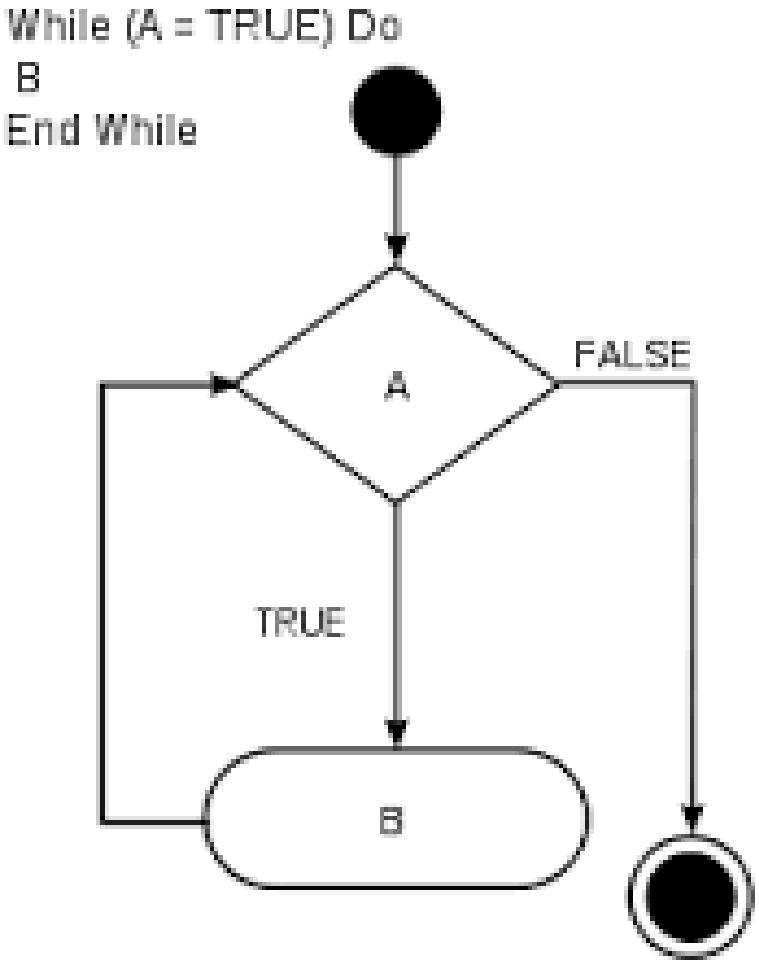
How does this work?

- We needed a test to mean "*always*".
- Well we want to always keep going and the loop keeps going when the test is true.
- So if we always want it to keep going we just use "*true*" as the test.
- We are basically seeing

  "***While true is true do the following repeatedly***".
- Since "*true*" is always true the answer to the question is always yes.

✓ This kind of loop with a test that is always true is known as a *non-terminating loop*.

How does this work? Contd…



Are such loops program bugs?

- Usually non-terminating loops are bugs.
- They are run-time errors in the program.

- However sometimes you do never want a program to stop.
  - e.g. it might be useful if a program controlling a heart pace maker would keep going forever.

Eg. Grill a steak… revisited…

- ✓ We know that the steak is done when it reaches a temperature of 135 degrees.
- ✓ Let us decide to reformulate the terminating condition to ensure that we continue to cook the steak as long as the temperature of the steak *is not equal to 135 degrees*.
- This change is reflected on line 3 of the algorithm.

1. steakTemp ⟵ 75
2. minOnGrill ⟵ 0
3. while steakTemp ≠135 do
   a. steakTemp ⟵ steakTemp + 13
   b. minOnGrill ⟵ minOnGrill + 3
4. endwhile

Figure: To compute time taken to grill a steak

✓ *Any comments?*

Logical flaw

- Notice that the loop will only terminate if the steak temperature is precisely 135.
- Our algorithm, however, is written in such a way that….
  - the steak temperature will never have a value of precisely 135.
  - The steak temperature will instead take on the values of 75, 88, 101, 113, 126, 139, 152, 175, and so forth.

✓ *The steak temperature will therefore increase without limit; becoming computationally hotter than any object in the universe!*

✓ *In addition, the time required for grilling the steak will also grow without bound.*

✓ *The algorithm is logically flawed.*

✓ *The loop does not make progress toward termination.*

✓ *This is an example of an infinite loop….*

Infinite loops

- An *infinite loop* is a loop that will never terminate because the loop never makes progress toward termination.

- When we consult the condition of the loop to determine whether the loop should repeat, the answer is always

  "*yes … let's do it again*!"

- An infinite loop is *almost always a logical error* since it violates the requirement that a program halt in a finite number of steps.

- Infinite loops are the source of many mistakes in real-world software.

✓ *Study the loops given below.*

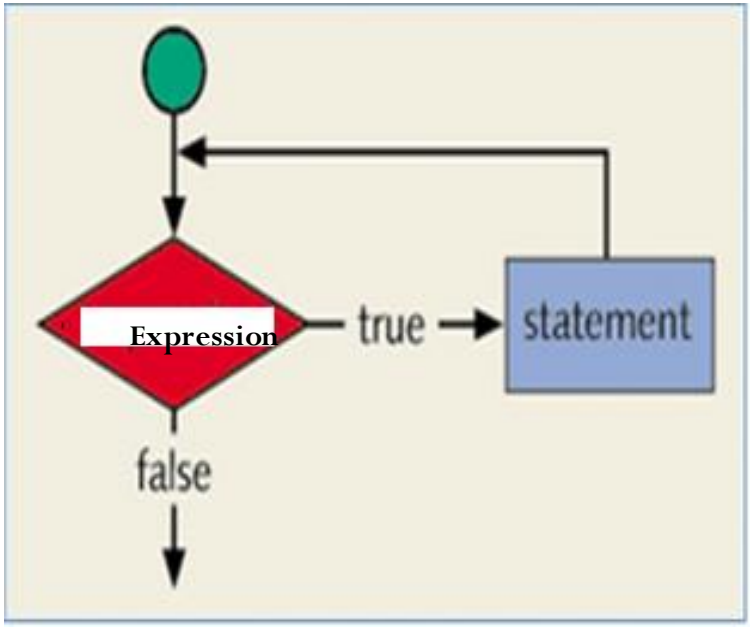


Figure: Loop A

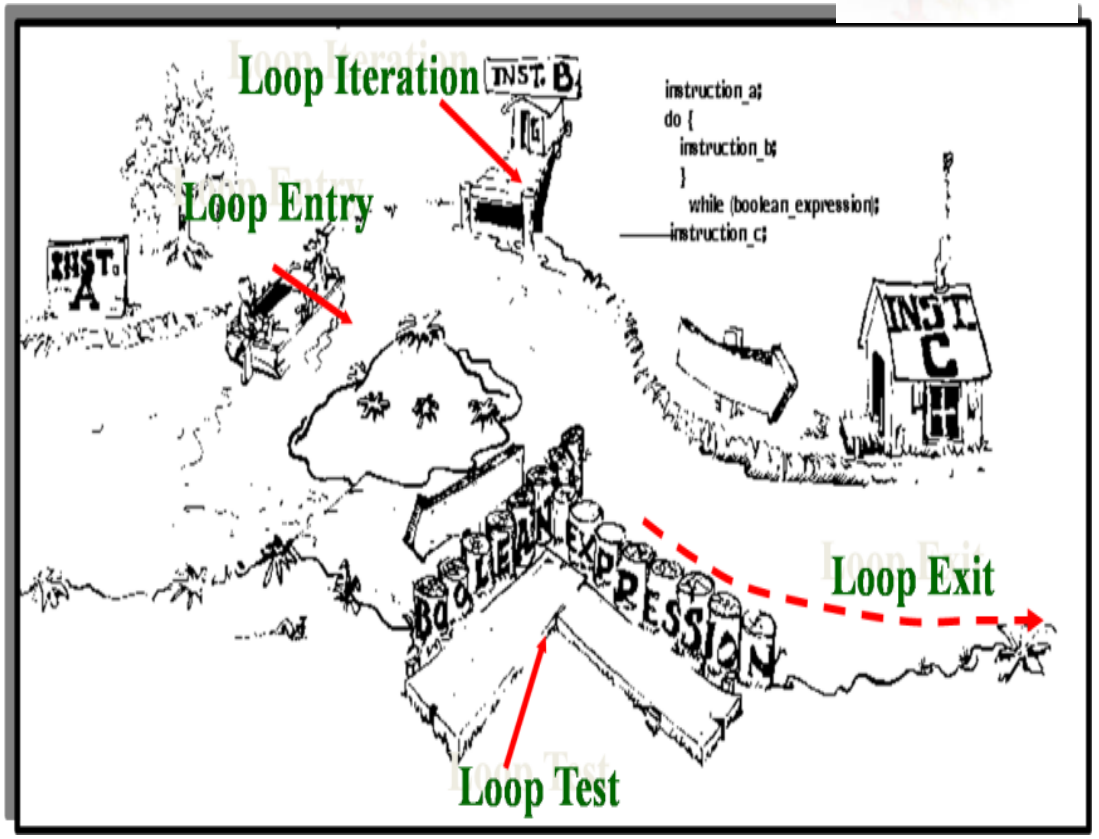


Figure: Loop B

✓ *Did you notice the differences?*

✓ *Draw the flowchart for loop B.*

The differences

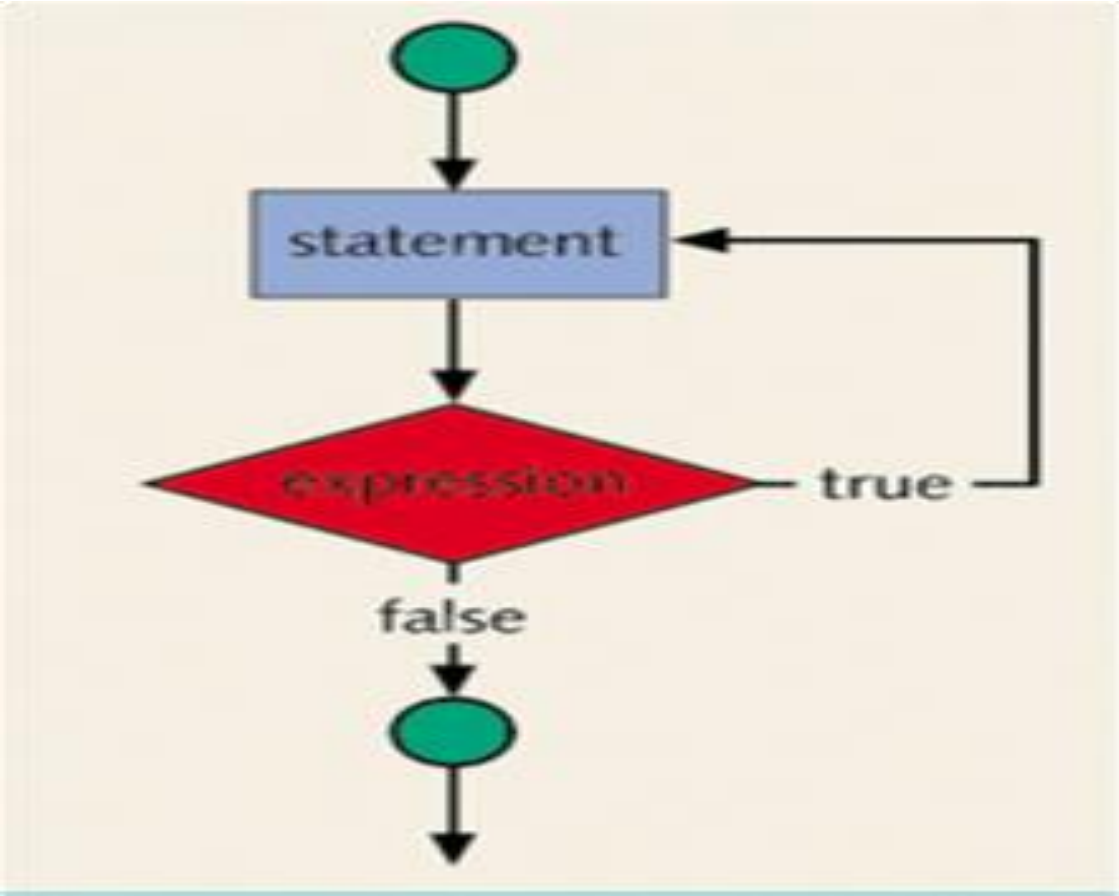
Loop A

- You are instructed to perform the loop test.
- You can do the action(s) iff the looping condition is met (True).
- If the loop test fails, you exit the loop without performing the action(s) even once.
- It is a pre tested loop.

Loop B

- You are instructed to perform action A; a looping activity follows A.
- On entry into the loop it is required to perform action B.
- This is followed by a loop test.
- On exiting the loop action C has to be performed.
- It is post tested loop

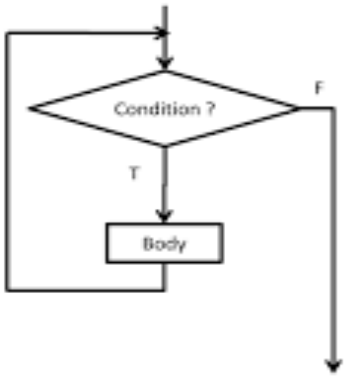
Flowchart for loop B



Post and pre-tested loops

While loop

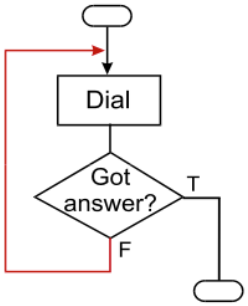
- Pre-tested loop.

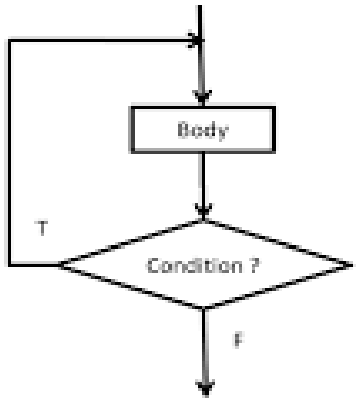


- We perform the loop test and then do the action(s) iff the looping condition is met (*True*).
- It is possible that we may exit the loop without performing the action(s) even once; this happens when the loop test fails.

Do-while loop

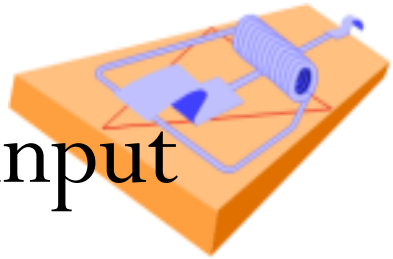
- Post-tested loop.





- We perform the action(s) and then do the loop test.
- The action(s) is performed once anyway.
- We exit the loop only after performing the action(s) at least once even when the loop test fails.

Eg. To trap for a valid input



- The task is to read only valid values ie. values within the given **x..y** range .

- The algorithm is given as:

  Display range x..y

  Do

    a. Read value

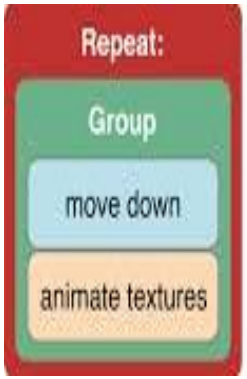
  Repeat Step 'a' if value is within the range x..y

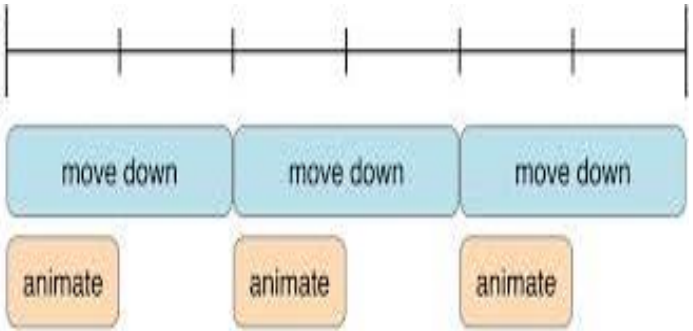
  Display invalid value

  Exit

What has been described?

- Loops as ways of expressing algorithms and computational processes.
- The form and function of flowchart elements for repetition.
- How the computational state of an algorithm captures the current values of all data as the algorithm is executed.





Credits

- *Computing without computers, Paul Curzon; http://www.eecs.qmul.ac.uk/~pc/research/education/puzzles/reading/*
- *cs.calvin.edu/books/java/intro/1e/PPSlides/chap08.pp*
- *www.letu.edu/people/stevearmstrong/.../Chapter%205%20Malik.ppt*
- *www.engr.sjsu.edu/bjfurman/courses/.../lecture_4_repetition_structures.ppt.*
- *Google images*