Java

Repetition, Program Design and Testing

Lecture objectives

To be able to understand the following fundamental concepts of the Java programming language:

- repetition statements
  - while,
  - do,
  - for
- program design pseudocode
- testing theory

Repetition Statements

- *Repetition statements* or “loops” allow us to execute a statement or block multiple times

- Like `if` statements, they are controlled by boolean expressions
  - ie. they cause a single statement or block to be executed repeatedly while an expression is true

Types of Loops

- Java has three kinds of repetition statements:
  - the *while loop*,
  - the *do loop*,
  - the *for loop*

- The programmer needs to consider the right kind of loop for the situation

The while Statement

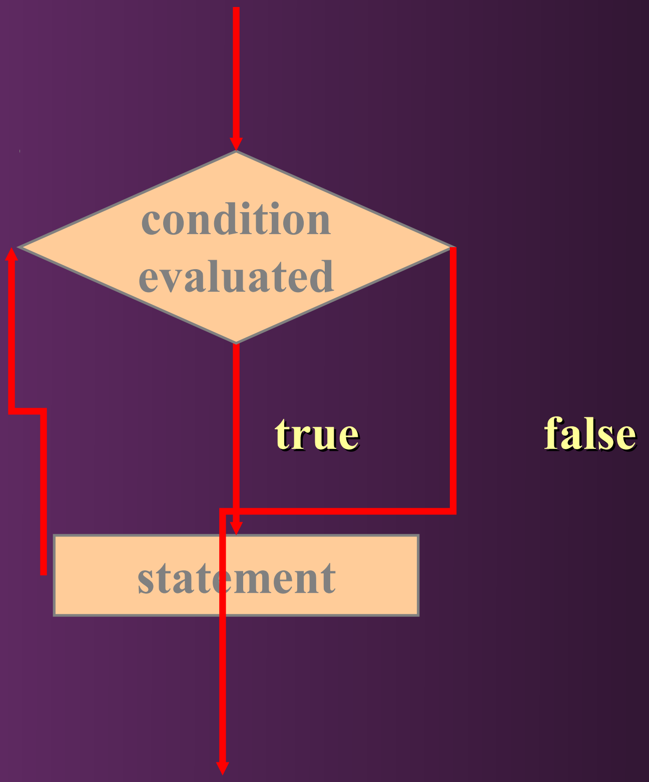
- The *while statement* has the following syntax:

```
while ( condition )
   statement;
```

If the condition is true, the statement is executed.
Then the condition is evaluated again.

The statement is executed repetitively until
the condition becomes false.

Logic of a while loop



The `while` Statement

- The condition in a `while` statement must return a boolean
- If the condition of a `while` statement is false initially, the statement is never executed
  - so the body of a `while` loop will execute zero or more times
- Something in the body of a while loop must alter the value of the control condition to stop the loop iterations

- The repetition of a loop can be
  - Count controlled or
  - Event controlled

Count controlled `while` loop

```
int   count ;
count  =  1;                                   // initialise loop variable

while ( count <= 3 )                           // test expression (loops 3 times)
{

    System.out.println( “count is “ + count );    // repeated action

           count = count + 1 ;                 // update loop variable
}

System.out.println( “Done” );
```

```
int   count ;
count  =  1;

while ( count <= 3 )
{

    System.out.println
           ( “count is “ + count );

     count =count + 1;


}

System.out.println( “Done” );
```

count

OUTPUT

```
int   count ;
count  =  1;
while ( count <= 3 )
{
     System.out.println
              ( “count is “ + count );
     count =count + 1;
}
System.out.println( “Done” );
```

count

1

OUTPUT

```
int   count ;

count  =  1;

while ( count <= 3 )              TRUE
{

    System.out.println
           ( “count is “ + count );

     count =count + 1;


}

System.out.println( “Done” );
```

count

1

OUTPUT

```
int   count ;

count  =  1;

while ( count <= 3 )
{
     System.out.println
          ( “count is “ + count );

     count =count + 1;


}

System.out.println( “Done” );
```

count

1

OUTPUT

```
count is 1
```

```
int   count ;

count  =  1;

while ( count <= 3 )
{

    System.out.println
           ( “count is “ + count );

    count =count + 1;


}

System.out.println( “Done” );
```

count

2

OUTPUT

```
count is 1
```

```
int   count ;

count  =  1;

while ( count <= 3 )                TRUE
{

    System.out.println
            ( “count is “ + count );

    count =count + 1;


}

System.out.println( “Done” );
```

count

2

OUTPUT

```
count is 1
```

```
int   count ;

count  =  1;

while ( count <= 3 )
{
      System.out.println
            ( “count is “ + count );

      count = count + 1 ;


}

System.out.println( “Done” );
```

count

2

OUTPUT

```
count is 1
count is 2
```

```
int   count ;

count  =  1;

while ( count <= 3 )
{

    System.out.println
           ( “count is “ + count );

    count = count + 1 ;


}

System.out.println( “Done” );
```

count

3

OUTPUT

```
count is 1
count is 2
```

```
int   count ;

count  =  1;

while ( count <= 3 )            TRUE
{

    System.out.println
           ( “count is “ + count );

     count =count + 1;


}

System.out.println( “Done” );
```

count

3

OUTPUT

```
count is 1
count is 2
```

```
int   count ;

count  =  1;

while ( count <= 3)
{
      System.out.println
            ( “count is “ + count );

    count =count + 1;


}

System.out.println( “Done” );
```

count

3

OUTPUT

```
count is 1
count is 2
count is 3
```

```
int   count ;

count  =  1;

while ( count <= 3 )
{

    System.out.println
          ( “count is “ + count );

    count =count + 1;

}

System.out.println( “Done” );
```

count

4

OUTPUT

```
count is 1
count is 2
count is 3
```

```
int   count ;

count  =  1;

while ( count <= 3 )          FALSE
{

    System.out.println
            ( “count is “ + count );

     count =count + 1;


}

System.out.println( “Done” );
```

count

4

OUTPUT

```
count is 1
count is 2
count is 3
```

```
int   count ;

count  =  1;

while ( count <= 3 )
{

    System.out.println
          ( “count is “ + count );

    count =count + 1;


}
System.out.println( “Done” );
```

count

4

OUTPUT

```
count is 1
count is 2
count is 3
Done
```

Event controlled loop

- Used when the number of iterations is unknown
- Again, something in the body of the loop causes the condition to be false
- Often we use a variable called a *sentinel*
  - “one who keeps watch…a sentry”   Chambers Dictionary
- Keep looping until the value of the sentinel indicates that processing should stop

Using a sentinel

- Requires initialising the *sentinel* before entering the loop
- Requires reviewing the *sentinel* as the last statement in the loop

```
int sum = 0, count = 0, value;                  //value is the sentinel
double average;
String valueStr;

 valueStr = JOptionPane.showInputDialog ("Enter an integer (0 to quit): ");
 value = Integer.parseInt(valueStr);

  while (value != 0)                                // sentinel value of 0 to terminate loop
  {
        count = count  + 1;
        sum = sum + value;

         valueStr = JOptionPane.showInputDialog ("Enter an integer (0 to quit): "); //final
                                                              opportunity to reset sentinel
         value = Integer.parseInt(valueStr);
  }

  average = (double)sum / count;
  System.out.println ("The average is " + average);
```

Infinite Loops

- The body of a `while` loop must eventually make the condition false
- If not, it is an *infinite loop*, which will execute until the user interrupts the program
- Ensure that your loops will terminate normally

```
//  Forever.java       Author: Lewis and Loftus
//  Demonstrates an INFINITE LOOP.  WARNING!!
//**************************************************************************
public class Forever
{
   public static void main (String[] args)
  {
      int count = 1;

      while (count <= 25)
      {
         System.out.println (count);
         count = count - 1;
     }

     System.out.println ("Done");       // this statement is never reached
  }
}
```

The do Statement

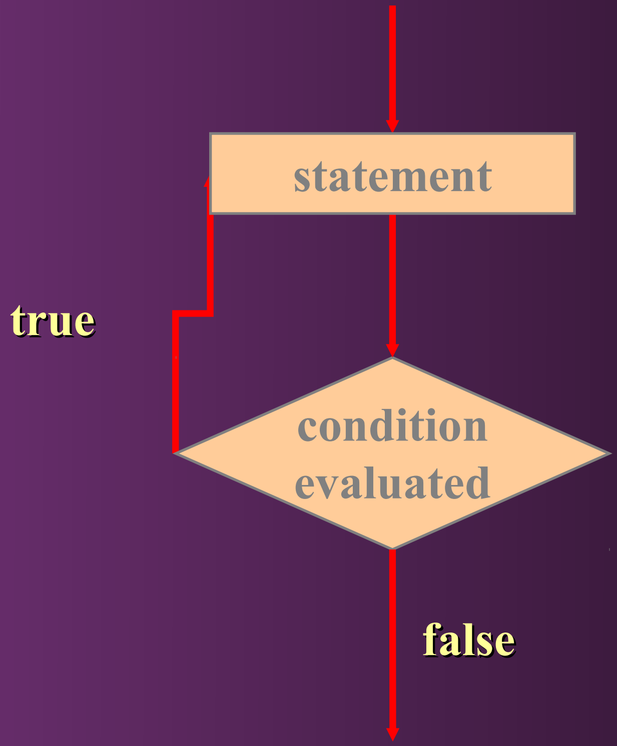
- The *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition )
```

•The statement is executed once initially, then the condition is evaluated

The statement is repetitively executed until the condition becomes false

Logic of a do loop



The do Statement

- A do loop is similar to a while loop, except that the condition is evaluated after the body of the loop is executed

- Therefore the body of a do loop will execute at least one time

```
public class DemoWhileLoop
{
    public static void main (String[] args)
    {
        final int LIMIT = 3;
        int count = 0;

        do
        {
            count = count + 1;
            System.out.print (count + “ “);
        }
        while (count < LIMIT);              // note the relational operator to loop 3 times

        System.out.println ("Done");
    }
}
```

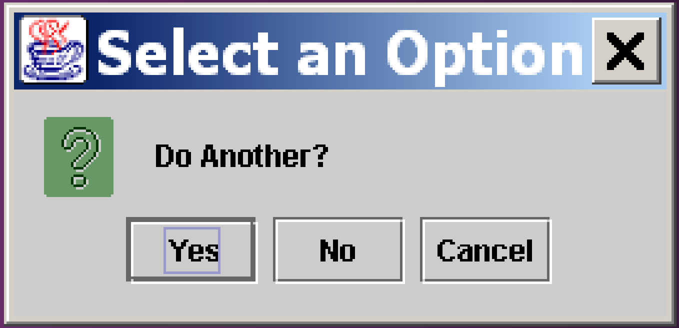
```
1  2   3  Done
```

Displaying a question

- To display a dialog box containing a specified question and Yes/No/Cancel button options.

- For example

```
again = JOptionPane.showConfirmDialog (null, "Do Another?");
```



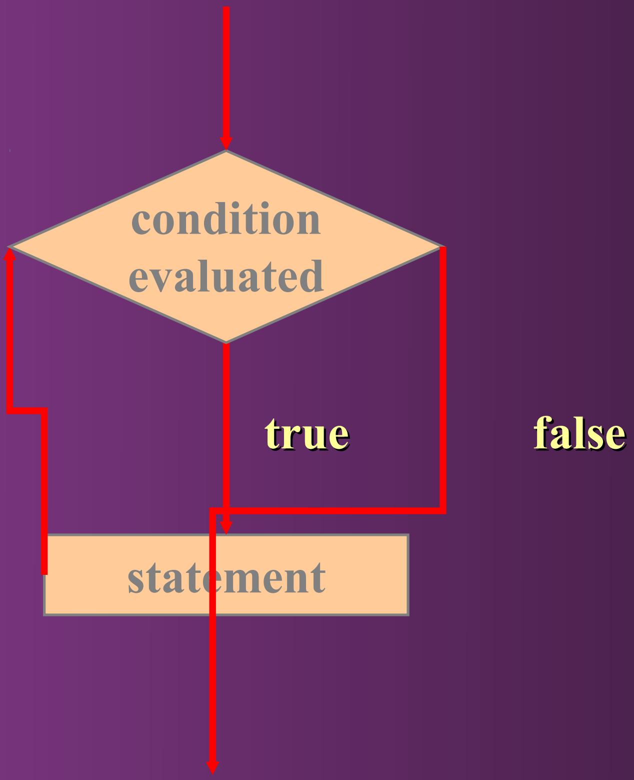
Do Another?

- Returns constants YES_OPTION, NO_OPTION, CANCEL_OPTION,

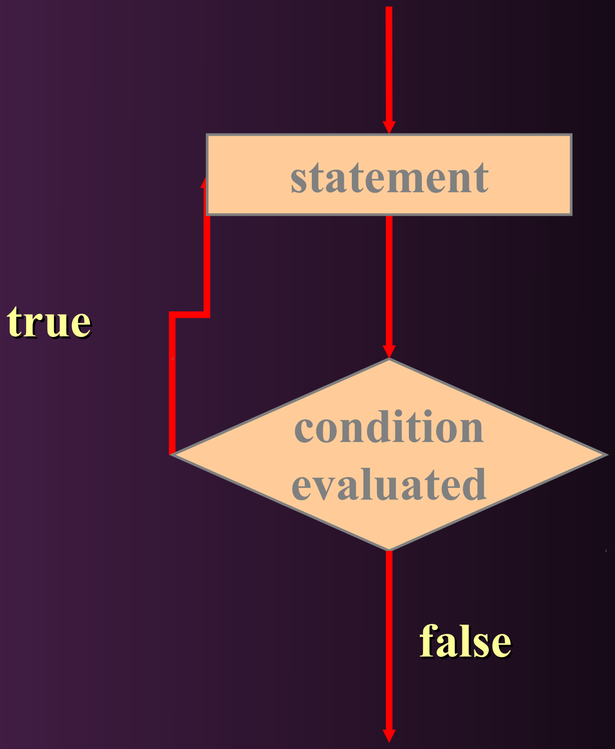
```
import javax.swing.JOptionPane;
public class EvenOdd
{
   public static void main (String[] args)
   {
      String numStr, result;
      int num, again;

      do
      {
           numStr = JOptionPane.showInputDialog ("Enter an integer: ");
           num = Integer.parseInt(numStr);
           if (num%2 == 0)
                result = "That number is even“;
            else
                 result = "That number is odd";
           JOptionPane.showMessageDialog (null, result);
           again = JOptionPane.showConfirmDialog (null, "Do Another?");
       }
       while (again == JOptionPane.YES_OPTION);
   }
}
```

Comparing the while and do loops

while loop

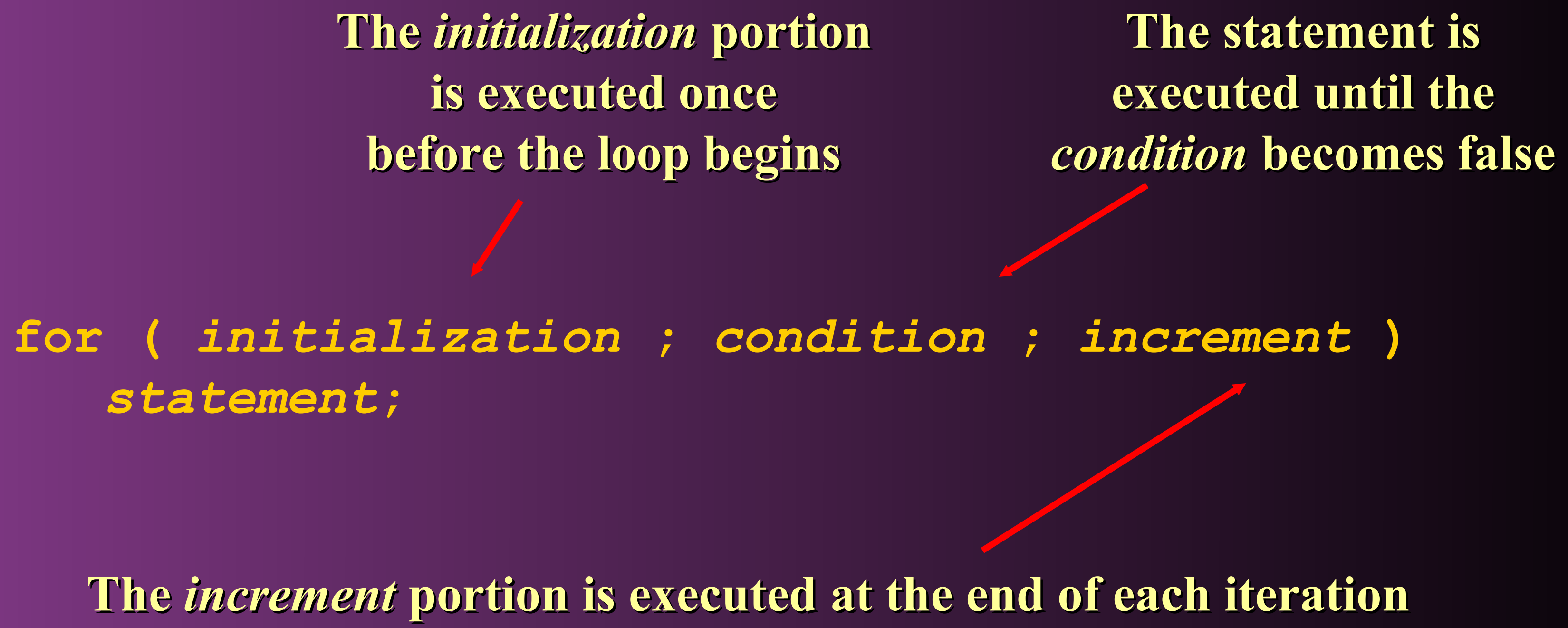


do loop



The for Statement

- The *for statement* has the following syntax:



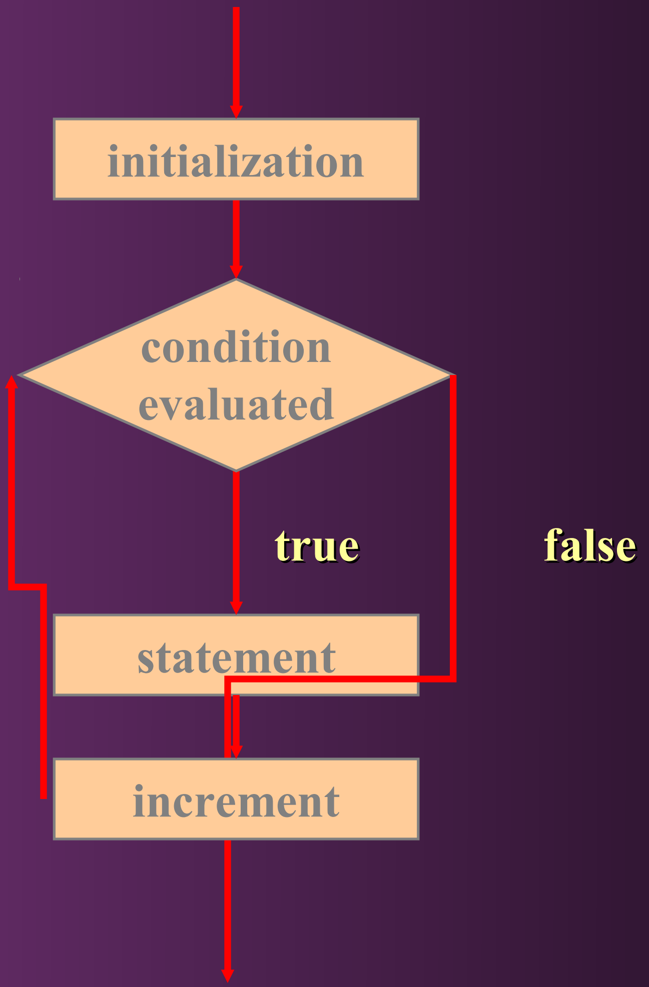
The *initialization* portion is executed once before the loop begins

The statement is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop



```
//***********************************************************
//  Prints integer values from 1 to a specific limit.
//***********************************************************

public class DemoDoLoop
{
    public static void main (String[] args)
   {
       final int LIMIT = 3;

       for (int count=1; count <= LIMIT; count++)
           System.out.print (count + “ “);

       System.out.println ("Done");
    }
}
```

```
1   2    3    Done
```

The for Statement

- A `for` loop is equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

- Therefore you never need to use a `for` loop - but programmers like them

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop could contain another loop
- For each single time through the outer loop, the inner loop will go through its entire set of iterations

Nested loop

```
while (outer loop condition)
{
     . . .
     . . .

     while (inner loop condition)
     {
          . . .
          . . .
     }

     . . .

}
```

Nested loop

- Say we want to write a program that prints out a multiplication table

```
1 2 3 4 5            ie 1 * 1,2,3,4,5
2 4 6 8 10              2 *
3 6 9 12 15             3 *
```

```
public class NestedLoop
{
    public static void main(String [] args)
    {
        for (int x = 1; x <= 3; x++)
        {
            for (int y = 1; y <= 5; y++)
            {
                int z = x * y;
                System.out.print(" " + z);
            }
            System.out.println();
        }
    }
}
```

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
```

Which Loop to use?

- `for` loop
  - If the number of repetitions is known

- `while` loop
  - If the number of repetitions is not known

- `do-while` loop
  - Use instead of while if the loop body has to be executed before the continuation condition is tested

Finding logic errors

- Be careful of one-off errors ie. the loop executes one to few or one too many times

- If you are having problems debugging, insert `System.out.println()` statements into your code
  - to print the value of a loop counter variable,
  - a sentinel
  - or any other relevant variables that will help you track each iteration

BlueJ Debugger

- Demonstrate BlueJ debugger (eg *OddEven*)
- A debugger is an essential tool for finding logic errors
- What functions does it provide?
  - Setting breakpoints
    - This stops program execution at this point and displays the code
    - Click in the area to the left of the text in the text editor
  - Stepping through the code
    - *Step* line by line
    - *Step into* a method
  - Inspecting variables
    - These are automatically displayed

Test Cases

- Carefully develop a variety of test cases, then

static test

- Test the design (pseudocode) using the test cases

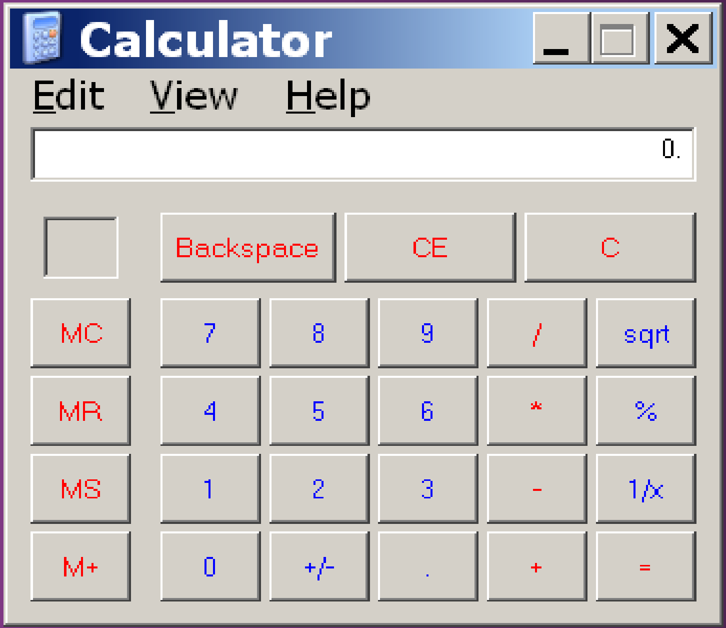
dynamic test

- executing the compiled program using the test cases

Testing

- Software is written by people – it is not perfect
- Testing is far more complex than running a program to see if it works
- Requires careful planning and discipline
- A program should be executed multiple times with various input in an attempt to find errors

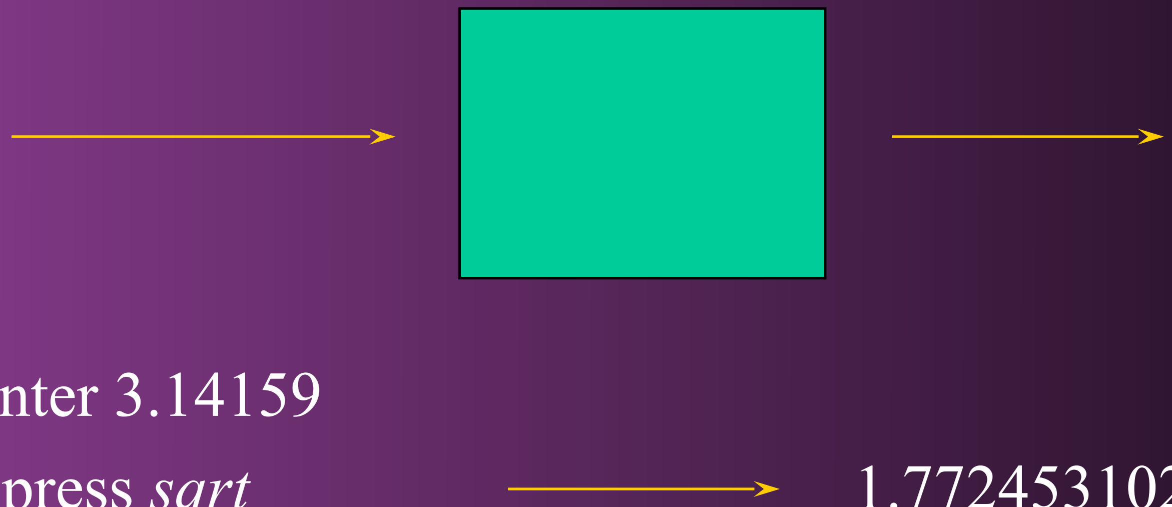
The Windows Calculator



- What tests would we do to exhaustively test it, given that it will take up to 32 digits?
- 1+1 ….1+2 ….1+9999999999999999999999999999 (largest)
- what else?

Functional testing

- Testing with reference to the requirements specification
- inputs are verified against outputs



inputs

outputs:

compare to
expected results

eg enter 3.14159
and press *sqrt*

1.772453102341

Selecting Test Data

- Most important task that software testers do
- Using equivalence classes
  - a systematic means for selecting test data
  - a set of test cases that tests the same thing, or reveals the same bug
  - reduces infinite set of possible test cases to a manageable but equally effective set
  - eg for the calculator software:
    - You have added 1+1, 1+2, 1+3, 1+4 successfully
    - Do you need to do 1+5 and 1+6?
    - What about 1+9999999999999999999999999999?

Selecting Test Data

- More errors occur at boundaries
  - 1+9999999999999999999999999999
  - The addition of 1 to the maximum value will be handled differently by the software ie. a different equivalence class

- Include test data which is
  - at the limit of that allowable
  - just within the beyond the limit
  - Just beyond the limit
- Include valid and invalid cases for all inputs

Test Plan

- Document all test cases prior to testing including expected outputs
- Compare expected to actual outputs after testing
- eg to test an account code that is valid from 1 – 9999
- What data might you use?

Test Plan

Input variable name or prompt	Input data	Expected output	Actual output
Enter account number:	1234	valid account	√
	-1	invalid account	√
	0	invalid account	√
	1	valid account	√
	9999	valid account	√
	10000	invalid account	√
	102.9	invalid account	√
	abc	invalid account	√

Pseudocode revisited

What does a computer program do?

- Receive information
- Do something to the information
  - Perform arithmetic
  - Assign a value to a variable
  - Compare 2 variables and select one of two alternative actions
  - Repeat a group of actions
- Put out information

Receive information

When the information is being received from the keyboard we need to prompt the operator to enter the data

```
Prompt operator for studentName
```

Put out information

- When a program is required to supply information to an output device, use `Display`, `Print`, or `Write`
- Display
  - if the output is to be written to the screen

```
Display studentGrade
```

- For straightforward output it is sufficient to say

```
Display output as per specification
```

Perform arithmetic

- Use either the mathematical symbols

```
total = total + number
```

- Or the words for those symbols

```
Add number to total
```

Assign a value to a variable

- Use *Initialise*, *Set,* =

```
 Set assignmentMark to 0
 Initialise customerCount to 0
totalPrice = costPrice + profitMargin
```

If-else

- Use IF for the condition
- Use ELSE for the false option
  - use separate lines and indentation
- Use END to close the operation

```
IF (student is partTime)
     add 1 to partTimeCount
ELSE
     add 1 to fullTimeCount
END
```

Repeat a group of actions

- WHILE
  - establishes the condition for the repetition of a group of statements (indented not using { })
- END
  - closes the repeated statements

```
WHILE (patientID is valid)
    finalPatientExpense =  patientExpense + tax
    Display finalPatientExpense
END
```

For Loop

- Say we wanted to loop 12 times to collect monthly rainfall figures and accumulate them.
- The following pseudocode would be acceptable

```
For i = 1 to 12 loop
   Prompt operator for month i rainfall
   total = total + rainfall
END
```

Lecture Outcomes

Today we have covered:

- repetition statements
  - while, do, for
- Pseudocode
- Testing theory

- Questions?