COMP1002
Computational Thinking and Problem Solving

Lecture 3
Computation I

Lecture 3

- Computational Steps
  - Sequential
  - Iteration/Repetition
  - Conditional/Selection
- Iterations
  - Definite and Indefinite
  - Nested Loops

Random Number Generation Revisited

- Generate a random list of unique numbers from 0 to 99:

```
Repeat
    Generate first digit 0 to 9 using a 10-faced die
    Generate second digit 0 to 9 using a 10-faced die
    If number already exists, try again
    If number does not exist, write it down on a piece of paper
Until all 100 numbers are generated
```

- This is a piece of pseudo-code that solves a problem. It indicates the computation steps needed
- Human can execute English-like pseudo-code
- Computer can execute programs derived from pseudo-code

Computational Steps

- How many "types" of steps can you find in this pseudo-code, i.e., "program" in English?
  1. Step-after-step execution
     - Sequential
  2. Repeat…until loop
     - Iteration/Repetition
  3. If *number exists* test
     - Conditional/Selection

```
Repeat
    Generate first digit 0 to 9 using a 10-faced die
    Generate second digit 0 to 9 using a 10-faced die
    If number already exists, try again
    If number does not exist, write it down on a piece of paper
Until all 100 numbers are generated
```

Computational Steps

- There are three types of computational steps:
  - Sequential: statements in a program are executed sequentially from top to bottom
    - A statement may be an arithmetic expression
    - A statement may be an input or output statement
  - Conditional: some conditions can be checked on the program state (indicated by variable or memory content) and actions would be taken if condition is true (or false)
  - Iteration: some part of the code will be repeated, also called a loop
    - Definite iteration: you know before an iteration starts on how many times to repeat
    - Indefinite iteration: you do not know in advance the number of times to repeat in an iteration

Computational Steps

› Definite iteration

```
set P = 1
for each number n in list L
        multiply P by n
return the product P
```

› Indefinite iteration

```
Repeat
   Generate first digit 0 to 9 using a 10-faced die
   Generate second digit 0 to 9 using a 10-faced die
   If number already exists, try again
   If number does not exist, write it down on a piece of paper
Until all 100 numbers are generated
```

Computational Steps

- We say that these three types of computational steps are "functionally complete"
- Example
  - sine and cosine are sufficient to express the other 4 trigonometric functions tangent, cotangent, secant and cosecant. Even cosine can be expressed using sine!
- Other types of computational steps only exist to make program development easier and more convenient
  - For example,
    - Function and Procedure

Iterations

- Iterations are often called loops
- Examples
  - Definite iteration: print 1 to 4

```
for i in [1…4]  do
      print i
```

```
for i in range(1,5):
    print(i)
```

  - How many times are executed?
  - Indefinite iteration: print 1 to 4

```
j = 1
repeat
      print j
      j = j + 1
until j > 4
```

  - How many times are executed?

Iterations

- Definite and indefinite iterations in code:
  - Definite iteration is easier, since you will know how many times it is executed, and the (loop) variable is well understood
  - Indefinite iteration is harder, since you need to make reasoning on number of times of execution, and the (loop) variable may change in various ways
  - How can you know that an indefinite iteration is correct?
    - Total number of times executed
    - Values produced in each iteration

Iterations

› We can unfold a loop to see behavior better
  – Definite loop

```
for i in [1…4]  do
       print i
```

```
i = 1
print i
i = 2
print i
i = 3
print i
i = 4
print i
```

Iterations

› Unfolding an indefinite loop

```
j = 1
repeat
    print j
    j = j + 1
until j > 4
```

```
j = 1
print j
j = j + 1
if j > 4 then stop repeating
print j
j = j + 1
if j > 4 then stop repeating
print j
j = j + 1
if j > 4 then stop repeating
print j
j = j + 1
if j > 4 then stop repeating
. . .
```

Iterations

- Indefinite iteration 1

```
j = 1
repeat
    print j
    j = j + 1
until j >  4
```

- How many times are executed?
- What are printed?

- Indefinite iteration 2

```
j = 1
repeat
    print j
    j = j + 1
until j >= 4
```

- How many times are executed?
- What are printed?

Iterations

› Indefinite iteration 3

```
j = 0
repeat
    j = j + 1
    print j
until j >  4
```

› How many times are executed?

› What are printed?

› Indefinite iteration 4

```
j = 0
repeat
    j = j + 1
    print j
until j >= 4
```

› How many times are executed?

› What are printed?

Iterations

- Indefinite iteration 5

```
j = 1
while j <= 4 do
    print j
    j = j + 1
```

- How many times are executed?
- What are printed?

- Indefinite iteration 6

```
j = 1
while j < 4 do
    print j
    j = j + 1
```

- How many times are executed?
- What are printed?

Iterations

- Indefinite iteration 7

```
j = 0
while j <= 4 do
    j = j + 1
    print j
```

- How many times are executed?
- What are printed?

- Indefinite iteration 8

```
j = 0
while j < 4 do
    j = j + 1
    print j
```

- How many times are executed?
- What are printed?

Iterations

› How about this one?

```
j = 0
while j > 4 do
    j = j + 1
    print j
```

Iterations

- Points to notice for indefinite loops:
  - Initial loop variable value
  - Loop termination conditions: < vs <= or > vs >= (perhaps != vs = in some rare cases)
  - Moment loop variable value is changed
- When debugging, write the variables down on a piece of paper
  - Of course, in the future, a program, called *debugger*, can help

Iterations

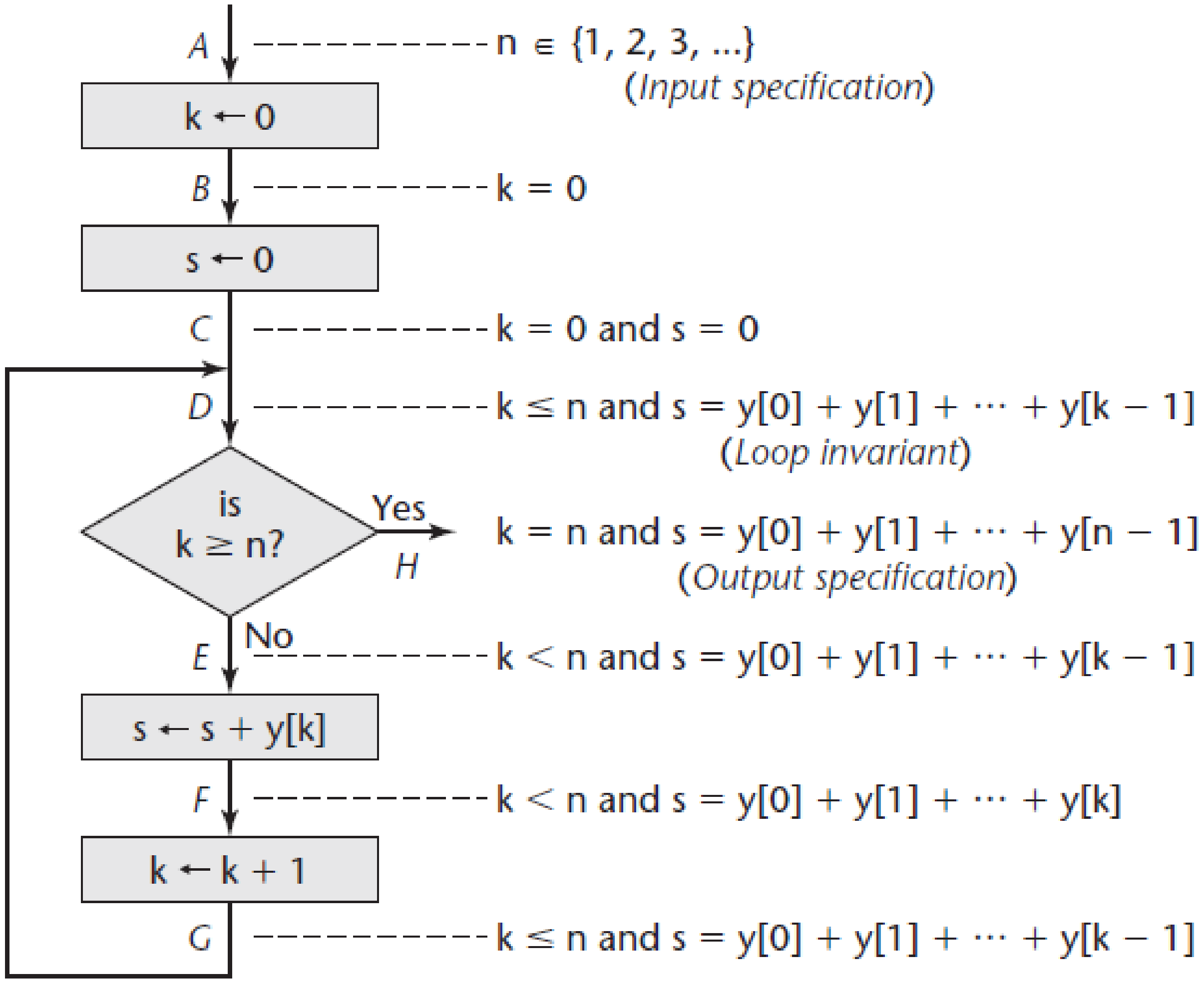
- Human are not good at remembering a lot of data/values at the same time
- Always write down the values or use the program's *print* facility
- If you realize there is a problem of your program, do not just run the program (and for many many times)!
  - Insert some *print* statements to help you know the values of various variables at a particular moment of execution

Consider iteration 5:

```
j = 1
while j <= 4 do
    print j
    j = j + 1
```

Iterations

- Formal approach
  - program correctness verification, by means of loop-invariant analysis



Iterations

› Comparing the use of < and <= in the conditions used in while and repeat loops

  – It seems that using <= would have the loop executed one more time

  – Is that true?

	1	2	7	8

Pseudo-code	j = 1
	repeat
	    print j
	    j = j + 1
	until j > 4
		j = 1
		repeat
		    print j
		    j = j + 1
		until j >= 4
			j = 0
			while j <= 4 do
			    j = j + 1
			    print j
				j = 0
				while j < 4 do
				    j = j + 1
				    print j

Output	1 2 3 4	1 2 3	1 2 3 4 5	1 2 3 4

Iterations

- Besides counting up, we often need to count down
- Examples
  - Definite iteration: print 4 to 1 (reversed)

```
for i in [4…1]  do
    print i
```

```python
for i in range(4,0,-1):
    print(i)
```

  - How many times are executed?
  - Indefinite iteration: print 4 to 1 (reversed)

```
j = 4
repeat
      print j
      j = j - 1
until j < 1
```

  - How many times are executed?

Exercises

	1	2	3	4

Pseudo-code	j = 4
	repeat
	    print j
	    j = j - 1
	until j > 1
		j = 4
		repeat
		    j = j - 1
		    print j
		until j > 1
			j = 4
			repeat
			    print j
			    j = j + 1
			until j < 1
				j = 4
				repeat
				    j = j + 1
				    print j
				until j < 1

Output				

	5	6	7	8

Pseudo-code	j = 4
	while j <= 1 do
	    print j
	    j = j - 1
		j = 4
		while j <= 1 do
		    j = j - 1
		    print j
			j = 4
			while j >= 1 do
			    print j
			    j = j + 1
				j = 4
				while j >= 1 do
				    j = j + 1
				    print j

Output				

Iterations

› Iterations can be nested:

```
for i in [1…3]  do
   for j in [1…3]  do
      print i, j
```

› Here, j is the inner loop and i is the outer loop
  – Inner loop (j) will be completed once for each value of the outer loop (i), for three times here

› Can you unfold it?

Iterations

```
for i in [1…3]  do
    for j in [1…3]  do
        print i, j
```

- That will give you all combinations of i and j values (3 x 3 = 9 pairs), but what order?
  - 1 1 / 1 2 / 1 3 / 2 1 / 2 2 / 2 3 / 3 1 / 3 2 / 3 3
  - 1 1 / 2 1 / 3 1 / 1 2 / 2 2 / 3 2 / 1 3 / 2 3 / 3 3

$$\begin{array}{cc} i & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \\ & j \end{array} \qquad a_{i,j}$$

- Row major order
  - i x j matrix

Iterations

› Iterations can be further nested:

```
for i in [1..3]  do
    for j in [1..3]  do
        for k in [1..2]  do
            print i, j, k
```

› That will give you all combinations of i, j and k values (3 x 3 x 2 = 18 tuples).

› What is the output?

Summary

- Computational Steps
  - Sequential
  - Iteration/Repetition
  - Conditional/Selection
- Iterations
  - Definite and Indefinite
  - Nested Loops