# **Programming Concepts**

Part 2: A language for algorithms

# **Topics**

# **Reading Suggestions**

- · Dowek. Chapter 1
- Harel. Chapters 1-3

# Purpose of algorithms

- Algorithms are designed for person-to-person communication
- Programming languages are designed for person-to-computer communication
- Nevertheless algorithms are meant to be implemented on computers

# Purpose of algorithms

- Algorithms are designed for person-to-person communication
- Programming languages are designed for person-to-computer communication
- Nevertheless algorithms are meant to be implemented on computers
- · Algorithms sit halfway between
  - excruciating detail needed for person-to-computer communication
  - informal English used for person-to-person communication

# Purpose of algorithms

- Algorithms are designed for person-to-person communication
- Programming languages are designed for person-to-computer communication
- Nevertheless algorithms are meant to be implemented on computers
- · Algorithms sit halfway between
  - excruciating detail needed for person-to-computer communication
  - informal English used for person-to-person communication

### We need to eliminate vagueness:

- make a note of ...
- proceed through the list of records . . .
- find record of boss in list . . .

# **Essential features of algorithms**

### Data storage and manipulation

- make a note of number 0
- · make note of name of boss
- · add salary to noted number
- add up resulting numbers
- · increase the counter

#### **Control structures**

- proceed through the employee list .....
- if salary of boss is less than . . . then . . .
- when the end of the list is finished . . . . .

Pseudocode = a human-readable way to write algorithms using exactly these features

### Data storage: VARIABLES or little boxes

### Examples:

- "noted number" is a variable
- "increase counter" "counter" is a variable

### Data storage: VARIABLES or little boxes

### Examples:

- "noted number" is a variable
- "increase counter" "counter" is a variable

### In pseudo code

- Variables are used to store data
- Variables can be updated:
  - Counter  $\leftarrow 0$
- Variables can be interrogated:
  - TOTAL  $\leftarrow$  SALARY + INCREMENT
  - COUNTER ← COUNTER + 1
  - SALARY ← SALARY \* 5

### Data storage: VARIABLES or little boxes

### Examples:

- · "noted number" is a variable
- "increase counter" "counter" is a variable

### In pseudo code

- Variables are used to store data
- Variables can be updated:
  - Counter  $\leftarrow 0$
- Variables can be interrogated:
  - $TOTAL \leftarrow SALARY + INCREMENT$
  - COUNTER ← COUNTER + 1
  - SALARY ← SALARY \* 5

What operations can be performed with contents of variables?

 If I put something in a box (SUM ← SALARY), do I still have hold of SALARY after?

- If I put something in a box (SUM ← SALARY), do I still have hold of SALARY after?
- When I use SALARY is the box empty afterwards?

- If I put something in a box (SUM ← SALARY), do I still have hold of SALARY after?
- When I use SALARY is the box empty afterwards?
- What if I put two things in the box?

- If I put something in a box (SUM ← SALARY), do I still have hold of SALARY after?
- When I use SALARY is the box empty afterwards?
- · What if I put two things in the box?

These are choices: a semantic description is needed

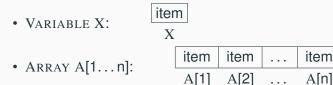
- "proceed through the list of records"
- "search records for boss of current employee"

- "proceed through the list of records"
- · "search records for boss of current employee"

• VARIABLE X:  $\frac{\text{item}}{X}$ 

- "proceed through the list of records"
- "search records for boss of current employee"

- "proceed through the list of records"
- "search records for boss of current employee"



### characteristics of arrays:

- Length of an array is always known:
  - A[1 . . . n] has n boxes
- Each box in an array is directly accessible, via index:
  - A[3] ← 27
  - $A[7] \leftarrow B[2] + A[1]$

item

• Direct sequencing:

"do A then do B then do C then do ..."

Direct sequencing:

"do A then do B then do C then do ..."

Conditional sequencing:

"if something is true then do A" and "if something is true then do A otherwise do B"

Direct sequencing:

"do A then do B then do C then do ..."

· Conditional sequencing:

"if something is true then do A" and "if something is true then do A otherwise do B"

· Bounded iteration:

"do something exactly *n* times" number of repetitions is known in advance

Direct sequencing:

"do A then do B then do C then do ..."

· Conditional sequencing:

"if something is true then do A" and
"if something is true then do A otherwise do B"

· Bounded iteration:

"do something exactly *n* times" number of repetitions is known in advance

Conditional iteration:

"do something so long as some condition is true" number of repetitions depends on evaluation of "condition"

· Direct sequencing:

"do A then do B then do C then do ..."

· Conditional sequencing:

"if something is true then do A" and
"if something is true then do A otherwise do B"

· Bounded iteration:

"do something exactly *n* times" number of repetitions is known in advance

· Conditional iteration:

"do something so long as some condition is true" number of repetitions depends on evaluation of "condition"

All available in all programming languages

### General format:

do A

do B

do C

### General format:

do A

do B

do C

## Examples

 $CURR \leftarrow 1$ 

 $\text{last} \leftarrow 10$ 

 $MIDDLE \leftarrow$ 

(CURR + LAST) div 2

### General format:

do A

do B

do C

## Examples

$$TEMP \leftarrow A[i]$$
$$A[i] \leftarrow A[j]$$
$$A[j] \leftarrow TEMP$$

### General format:

do A do B

do C

## Examples

```
// interchange A[i] with A[j] TEMP \leftarrow A[i] A[i] \leftarrow A[j] A[j] \leftarrow TEMP
```

# **Conditional sequencing**

#### General formats

- 1.
  - if some condition is true then
    - | do something
- 2.
- if some condition is true then
  - do something

#### else

do some other thing

# Conditional sequencing

### General formats

```
if some condition is true then

do something

if some condition is true then

do something

else

do some other thing
```

```
e.g.

if sales have decreased
then
| lower price by 10%
e.g.

if price > limit then
| pay x
else
| pay y
```

```
if item is taxable then

if price > limit then

| pay x

else
| pay y

else
| pay z
```

```
if item is taxable then

if price > limit then

pay x

else

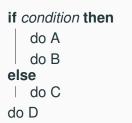
pay y

else

pay z
```

```
if item is taxable then
if price > limit then
pay x
else
pay y
```

```
if condition then
do A
do B
else
do C
do D
```



```
if condition then
do A
do B
else
do C
do D
```

## **Bounded iteration**

Do something an exact number of times

General format:

#### **Bounded iteration**

### Do something an exact number of times

#### General format:

i: the iterator.Available to use within the something

**start:** value at which the iterator starts

**finish:** value at which the iterator ends

# **Bounded iteration: Example**

Summing the first *n* positive numbers:

```
Input: positive number n
Output: sum of first n positive numbers sum \leftarrow 0
for i \leftarrow 1 to n do
| sum \leftarrow sum + i
return sum
```

# **Bounded iteration: Example**

Summing the first *n* positive numbers:

```
Input: positive number n
Output: sum of first n positive numbers
SUM \leftarrow 0
for i \leftarrow 1 to n do
| SUM \leftarrow SUM + i
return SUM
```

### Explanation:

Input: description of expected inputOutput: description of desired outputreturn value: an operation in pseudo-code

#### **Conditional iteration**

Perform *something* repeatedly so long as some *condition* remains true

#### **General format:**

while condition do | something

#### **Conditional iteration**

Perform *something* repeatedly so long as some *condition* remains true

#### **General format:**

while condition do

something

### **Explanation:**

- Executing something can affect the value of condition
- If condition is false something is not executed
- If condition is true something is executed and ... evaluation repeats itself
- The value of condition may remain true forever

#### **Conditional iteration**

Perform *something* repeatedly so long as some *condition* remains true

#### **General format:**

while condition do

something

### **Explanation:**

- Executing something can affect the value of condition
- If condition is false something is not executed
- If condition is true something is executed and ... evaluation repeats itself
- The value of condition may remain true forever

## Conditional iteration: Example

· Summing numbers:

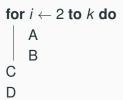
```
Input: positive number n
Output: sum of first n positive numbers sum \leftarrow 0
ITER \leftarrow 1
while ITER \leq n do
sum \leftarrow sum + iter
ITER \leftarrow iter + 1
return sum
```

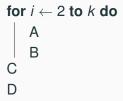
### Conditional iteration: Example

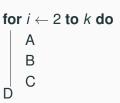
Summing numbers:

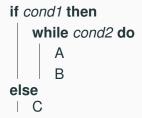
```
Input: positive number n
Output: sum of first n positive numbers
SUM \leftarrow 0
ITER \leftarrow 1
while ITER \leq n do
SUM \leftarrow SUM + ITER
ITER \leftarrow ITER + 1
return SUM
```

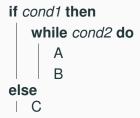
- Note:
  - · ITER needs to be explicitly managed
  - ITER is automatically furnished by for-loop construct

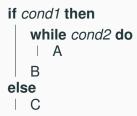












#### **Pseudocode**

- · Data storage: variables, arrays
- Control structures: direct sequencing, bounded iteration, conditional sequencing.
- return to give back a value at the end

Don't forget: any algorithm must come with a valid specification! How do you implement these in Python / Java?

## Calculating salary bill

Legal inputs: any list of employee records; each record

contains their salary

Required output: the total salary bill

### **Algorithm**

- (1) Make a note of number 0
- (2) Proceed through the employee list, each time
  - adding salary to noted number
- (3) When end of list is reached
  - · output noted number

Name	Salary
Tom Jones	12000
Mary Clark	17000
Shaun Collins	16000
Lisa	23000

# Pseudo-code for calculating salary bill

**Input**: an array E[1 ... n] of employee details **Output**: total salary of all employees

**Assumptions**: array elements contain *salary* field

```
TOTAL \leftarrow 0

PTR \leftarrow 1

while PTR \leq n do

TOTAL \leftarrow TOTAL + salary(E[PTR])

PTR \leftarrow PTR + 1

return TOTAL.
```

# Pseudo-code for calculating salary bill

**Input**: an array  $E[1 \dots n]$  of employee details

Output: total salary of all employees

Assumptions: array elements contain salary field

```
TOTAL \leftarrow 0

PTR \leftarrow 1

while PTR \leq n do

TOTAL \leftarrow TOTAL + salary(E[PTR])

PTR \leftarrow PTR + 1

return TOTAL.
```

Somewhat easier with for-loop construct

# Pseudo-code for calculating salary bill

**Input**: an array  $E[1 \dots n]$  of employee details

Output: total salary of all employees

Assumptions: array elements contain salary field

```
TOTAL \leftarrow 0

PTR \leftarrow 1

while PTR \leq n do

TOTAL \leftarrow TOTAL + salary(E[PTR])

PTR \leftarrow PTR + 1

return TOTAL.
```

Somewhat easier with for-loop construct

Details of salary extraction ignored

### Happy employees

### **Counting happy employees**

**Legal inputs:** any list of employee records; each record contains their salary and name of boss

**Required output:** number of employees earning more than their boss

### Algorithm?

- (1) Make a note of counter 0
- (2) Proceed through the employee list, each time
  - (a) Note name of boss, and salary of current employee
  - (b) Find record of boss in list
  - (c) If salary of boss is less than that of current employee, increase the counter
- (3) When end of list is reached, output value of counter

Name	Salary	Boss
Tom	12000	James
Mary	17000	Cindy
Shaun	16000	Tom
Lisa	23000	Mary

### Happy employees

### **Counting happy employees**

**Legal inputs:** any list of employee records; each record contains their salary and name of boss

**Required output:** number of employees earning more than their boss

### Algorithm?

- (1) Make a note of counter 0
- (2) Proceed through the employee list, each time
  - (a) Note name of boss, and salary of current employee
  - (b) Find record of boss in list
  - (c) If salary of boss is less than that of current employee, increase the counter
- (3) When end of list is reached, output value of counter

Name	Salary	Boss
Tom	12000	James
Mary	17000	Cindy
Shaun	16000	Tom
Lisa	23000	Mary

## Towards pseudo-code

**Input**: an array  $E[1 \dots n]$  of employee details **Output**: number of employees earning more than their boss **Assumptions**: array elements contain *name*, *salary*, *boss* fields. Every employee has a *boss* 

```
HAPPY \leftarrow 0

for i \leftarrow 1 to n do

BOSS \leftarrow boss(E[i])

SALARY \leftarrow salary(E[i])

find PTR satisfying BOSS = name(E[PTR])

if salary(E[PTR]) < SALARY then

HAPPY \leftarrow HAPPY + 1
```

return HAPPY

# Pseudo-code for counting happy employees

**Input**: an array E[1 ... n] of employee details **Output**: number of employees earning more than their boss **Assumptions**: array elements contain *name*, *salary*, *boss* fields. Every employee has a *boss* 

```
HAPPY \leftarrow 0

for i \leftarrow 1 to n do

BOSS \leftarrow boss(E[i])

SALARY \leftarrow salary(E[i])

PTR \leftarrow 1

while BOSS \neq name(E[PTR]) do

PTR \leftarrow PTR + 1

if salary(E[PTR]) < SALARY then

HAPPY \leftarrow HAPPY + 1
```

return HAPPY

### **Summary**

- A fixed language. Powerful enough to write all algorithms.
- Data storage: Variables and Arrays. Care with indexing.
- Difference between bounded and conditional iteration.
- Layout is very important!

#### To Do:

- Exercise sheet 1: check the solutions.
- Homework 1.
- · Exercise sheet 2.

Ask any questions at the Helpdesk/Exercise sessions.