Introduction to Algorithms

CELEN086

Seminar 2
(w/c 14/10/2024)

Outline

In this seminar, we will study and review on following topics:

- IF and nested IF structures
- Compounded conditional statements in IF structure
- Luhn algorithm
- Sub-algorithm

You will also learn useful Math/CS concepts and vocabularies.

Activity -1

Write the header (DO NOT write the body!) for the algorithms below:
In your header you should

- **Give the algorithm a suitable name**
- **Make sure it has the correct arguments**
- **give more information about the arguments in < requires; >**
- **clearly describe the result expected of the algorithm in < returns : >**

(i) An algorithm to calculate the score of one team of a basketball match (assume 5 players in the team) given the scores of the individual players.

(ii) An algorithm isTriangle which takes the lengths of three metal rods: len1, len2, and len3 and determines whether or not the rods can be arranged in a triangle.

(iii) An algorithm which calculates the amount of concrete required for a new airport runway.

Activity -1

Write the header (DO NOT write the body!) for the algorithms below:
In your header you should

- **Give the algorithm a suitable name**
- **Make sure it has the correct arguments**
- **give more information about the arguments in < requires; >**
- **clearly describe the result expected of the algorithm in < returns : >**

(i) An algorithm to calculate the score of one team of a basketball match (assume 5 players in the team) given the scores of the individual players.

Algorithm : Bscore(a,b,c,d,e)
Requires: Five positive numbers a,b,c,d and e.
Returns: a positive number i.e. score sum of all team members.

Activity -1

Write the header (DO NOT write the body!) for the algorithms below:

In your header you should

- **Give the algorithm a suitable name**
- **Make sure it has the correct arguments**
- **give more information about the arguments in < requires; >**
- **clearly describe the result expected of the algorithm in < returns : >**

(ii) An algorithm isTriangle which takes the lengths of three metal rods: len1, len2, and len3 and determines whether or not the rods can be arranged in a triangle.

Algorithm: isTriangle(len1,len2,len3)
Requires: Three positive numbers len1, len2 and len3 three sides length of triangle.
Returns: a Boolean variable true if triangle possible otherwise false.

Activity -1

Write the header (DO NOT write the body!) for the algorithms below:
In your header you should

- **Give the algorithm a suitable name**
- **Make sure it has the correct arguments**
- **give more information about the arguments in < requires; >**
- **clearly describe the result expected of the algorithm in < returns : >**

(iii) An algorithm which calculates the amount of concrete required for a new airport runway.

Algorithm: Voconcrete(l , w,h)
Requires : three positive numbers representing length l , width w and height h of runway.
Returns: a positive number i.e. volume of concrete required to make desired runway.

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1
      return z
   else
      return x
   endif
else
   if ? //Condition 2
      ? //Statement 3
   else
      ? //Statement 4
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1   z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2
      ? //Statement 3
   else
      ? //Statement 4
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1    z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2     y<z
      ? //Statement 3
   else
      ? //Statement 4
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1   z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2    y<z
      ? //Statement 3   return y
   else
      ? //Statement 4
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1   z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2    y<z
      ? //Statement 3   return y
   else
      ? //Statement 4   return z
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1   z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2    y<z
      ? //Statement 3   return y
   else
      ? //Statement 4   return z
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

Trace min3(4, 5, 6)

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if x<y
   if  ? //Condition 1   z<x
      return z
   else
      return x
   endif
else
   if ? //Condition 2    y<z
      ? //Statement 3   return y
   else
      ? //Statement 4   return z
   endif
endif
```

Complete the algorithm with the missing conditional statements 1&2, and statements 3&4.

Trace min3(4, 5, 6)

Trace min3(5, 5, 6)

(Algorithm using Nested IF and simple conditional statements)

Minimum of 3 numbers

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
     return x
elseif (y<=x) && (x<=z)
     return y
else
     return z
endif
```

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.      return x
3. elseif (y<=x) && (x<=z)
4.      return y
5. else
6.      return z
7. endif
```

Does it look good?

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
     return x
elseif (y<=x) && (x<=z)
     return y
else
     return z
endif
```

Does it look good?

Trace min3(5, 2, 8)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
     return x
elseif (y<=x) && (x<=z)
     return y
else
     return z
endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Trace min3(2, 6, 4)

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.       return x
3. elseif (y<=x) && (x<=z)
4.       return y
5. else
6.       return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.      return x
3. elseif (y<=x) && (x<=z)
4.      return y
5. else
6.      return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)   T && F = False

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.      return x
3. elseif (y<=x) && (x<=z)
4.      return y
5. else
6.      return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)   T && F = False   Go to Line 3

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.      return x
3. elseif (y<=x) && (x<=z)
4.      return y
5. else
6.      return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)   T && F = False   Go to Line 3

Line 3: (6<=2) && (2<=4)

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.      return x
3. elseif (y<=x) && (x<=z)
4.      return y
5. else
6.      return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4) T && F = False Go to Line 3

Line 3: (6<=2) && (2<=4) F && T = False

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.       return x
3. elseif (y<=x) && (x<=z)
4.       return y
5. else
6.       return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)	T && F = False	Go to Line 3
Line 3: (6<=2) && (2<=4)	F && T = False	Go to Line 5 and Line 6, return 4.

Minimum of 3 numbers

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
1. if (x<=y) && (y<=z)
2.       return x
3. elseif (y<=x) && (x<=z)
4.       return y
5. else
6.       return z
7. endif
```

Does it look good?

Trace min3(5, 2, 8)

Caution!
You need to trace your algorithm for different sets of input arguments (test cases) for its validation.

Trace min3(2, 6, 4)

x=2, y=6, z=4

Line 1: (2<=6) && (6<=4)	T && F = False	Go to Line 3
Line 3: (6<=2) && (2<=4)	F && T = False	Go to Line 5 and Line 6, return 4.

min3(2, 6, 4) = 4?

Practice

Correct the algorithm or design your own algorithm for min3(x,y,z)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Practice

Correct the algorithm or design your own algorithm for min3(x,y,z)

Then trace min3(2, 6, 4)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Practice

Correct the algorithm or design your own algorithm for min3(x,y,z)

Then trace min3(2, 6, 4)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Practice

Correct the algorithm or design your own algorithm for min3(x,y,z)

Then trace min3(2, 6, 4)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
if (x<=y) && (x<=z)
      return x
elseif (y<=x) && (y<=z)
      return y
else
      return z
endif
```

(Algorithm using simple IF and compounded conditional statements)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest value of x, y, z

```
if (x<=y) && (y<=z)
      return x
elseif (y<=x) && (x<=z)
      return y
else
      return z
endif
```

Luhn algorithm

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c′ = LuhnDouble(c)
2. let a′ = LuhnDouble(a)
3. let s = a′+ b+ c′+ d
4. return (s mod 10) == 0
```

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.     return 2*x-9
3. else
4.     return 2*x
5. endif
```

Is 3874 a valid mini card number?

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
let c' = LuhnDouble(c)
let a' = LuhnDouble(a)
let s = a'+ b+ c'+ d
return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
if 2*x>9
     return 2*x-9
else
     return 2*x
endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
let c' = LuhnDouble(c)
let a' = LuhnDouble(a)
let s = a'+ b+ c'+ d
return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
if 2*x>9
      return 2*x-9
else
      return 2*x
endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

a=3, b=8, c=7, d=4

[main]	[sub]

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]	[sub]
a=3, b=8, c=7, d=4	
1: c'=LuhnDouble(7)	

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.       return 2*x-9
3. else
4.       return 2*x
5. endif
```

Is 3874 a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c'=LuhnDouble(7)

[sub]

x=7

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.       return 2*x-9
3. else
4.       return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]	[sub]
a=3, b=8, c=7, d=4	
1: c’=LuhnDouble(7)	x=7 2*x>9? True. Return 5.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is 3874 a valid mini card number?   Trace(3,8,7,4)

[main]	[sub]
a=3, b=8, c=7, d=4	
1: c’=LuhnDouble(7) =5	x=7 2*x>9? True. Return 5.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is 3874 a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c'=LuhnDouble(7) =5

2: a'=LuhnDouble(3)

[sub]

x=7 2*x>9? True. Return 5.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.       return 2*x-9
3. else
4.       return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c’=LuhnDouble(7) =5

2: a’=LuhnDouble(3)

[sub]

x=7 $2 * x > 9$? True. Return 5.

x=3

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is 3874 a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c'=LuhnDouble(7) =5

2: a'=LuhnDouble(3)

[sub]

x=7 2*x>9? True. Return 5.

x=3 2*x>9? False. Return 6.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c'=LuhnDouble(7) =5

2: a'=LuhnDouble(3) =6

[sub]

x=7 2*x>9? True. Return 5.

x=3 2*x>9? False. Return 6.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is 3874 a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c’=LuhnDouble(7) =5

2: a’=LuhnDouble(3) =6

3: s=6+8+5+4=23

[sub]

x=7 2*x>9? True. Return 5.

x=3 2*x>9? False. Return 6.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c’ = LuhnDouble(c)
2. let a’ = LuhnDouble(a)
3. let s = a’+ b+ c’+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c’=LuhnDouble(7) =5

2: a’=LuhnDouble(3) =6

3: s=6+8+5+4=23

4: return (23 mod 10)==0

[sub]

x=7 2*x>9? True. Return 5.

x=3 2*x>9? False. Return 6.

Luhn algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a'+ b+ c'+ d
4. return (s mod 10) == 0
```

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.      return 2*x-9
3. else
4.      return 2*x
5. endif
```

Is <u>3874</u> a valid mini card number? Trace(3,8,7,4)

[main]

a=3, b=8, c=7, d=4

1: c'=LuhnDouble(7) =5

2: a'=LuhnDouble(3) =6

3: s=6+8+5+4=23

4: return (23 mod 10)==0 False.

[sub]

x=7 2*x>9? True. Return 5.

x=3 2*x>9? False. Return 6.

Practice

1. Check if the following mini card numbers are valid.

- 4291
- 8947

2. Determine the check number (?) such that 665(?) is a valid card number.

Practice

Answers:

1. Check if the following mini card numbers are valid.

- 4291

- 8947

2. Determine the check number (?) such that 665(?) is a valid card number.

Practice

Answers:

1. Check if the following mini card numbers are valid.

- 4291 — Valid
- 8947

2. Determine the check number (?) such that 665(?) is a valid card number.

Practice

Answers:

1. Check if the following mini card numbers are valid.

- 4291 — Valid
- 8947 — Invalid

2. Determine the check number (?) such that 665(?) is a valid card number.

Practice

Answers:

1. Check if the following mini card numbers are valid.

- 4291 — Valid
- 8947 — Invalid

2. Determine the check number (?) such that 665(?) is a valid card number.

0

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z)
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

Algorithm: min(x,y)
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
if x<y
      return x
else
      return y
end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine (sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4

Line 1: F, go to Line 3 and then
Line 4: return 4

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4

Line 1: F, go to Line 3 and then
Line 4: return 4

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4

Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4

Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine (sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4
Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

x=2, y=4
Line 1: T, go to Line 2 and return 2

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4
Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

x=2, y=4
Line 1: T, go to Line 2 and return 2

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine
(sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.       return x
3. else
4.       return y
5. end
```

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4
Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

x=2, y=4
Line 1: T, go to Line 2 and return 2

= 2

Minimum of 3 numbers (alternative ver.)

Algorithm: min3(x,y,z) **[main]**
Requires: three numbers x, y and z
Returns: the smallest of x, y, z

```
1. return min(x, min(y,z))
```

call subroutine (sub-algorithm/function)

Algorithm: min(x,y) **[sub-algorithm]**
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.     return x
3. else
4.     return y
5. end
```

Note: x,y (or x,y,z) are local variables that are used only within each algorithm, e.g., sub (or main).

Trace min3(2, 6, 4)

Line 1: return min(2, min(6,4))

x=6, y=4

Line 1: F, go to Line 3 and then
Line 4: return 4

= min(2, 4)

x=2, y=4

Line 1: T, go to Line 2 and return 2

= 2

Practice

Write an algorithm that takes three numbers and returns the sum of two largest numbers.

Practice

Write an algorithm that takes three numbers and returns the sum of two largest numbers.

Algorithm: sum2largest(a,b,c)
Requires: three numbers a, b, c
Returns: sum of the two largest

Practice

Write an algorithm that takes three numbers and returns the sum of two largest numbers.

Algorithm: sum2largest(a,b,c)
Requires: three numbers a, b, c
Returns: sum of the two largest

---

1. return (a+b+c)-min(a, min(b,c))

Practice

Write an algorithm that takes three numbers and returns the sum of two largest numbers.

Algorithm: sum2largest(a,b,c)
Requires: three numbers a, b, c
Returns: sum of the two largest

```
return (a+b+c)-min(a, min(b,c))
```

Algorithm: min(x,y)
Requires: two numbers x and y
Returns: the minimum of x and y

```
if  x<y
        return x
else
        return y
endif
```

Practice

Write an algorithm that takes three numbers and returns the sum of two largest numbers.

Algorithm: sum2largest(a,b,c)
Requires: three numbers a, b, c
Returns: sum of the two largest

---

1. return (a+b+c)-min(a, min(b,c))

Algorithm: min(x,y)
Requires: two numbers x and y
Returns: the minimum of x and y

```
1. if x<y
2.      return x
3. else
4.      return y
5. endif
```

Alternative solutions are available/acceptable.

Fake coin problem

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8

4 | 4

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8

4 | 4

2 | 2

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8

4 | 4

2 | 2

1 | 1

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8

4 | 4

2 | 2

1 | 1

8

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8	8	
4 | 4	3 | 3	2
2 | 2		
1 | 1		

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8	8	
4 | 4	3 | 3	2
2 | 2	1 | 1 1	
1 | 1		

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8	8		
4 | 4	3 | 3		2
2 | 2	1 | 1	1	1 | 1
1 | 1			

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

8	8		
4 | 4	3 | 3		2
2 | 2	1 | 1	1	1 | 1
1 | 1			

2-way division

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

2-way division	3-way division	
8	8	
4 | 4	3 | 3	2
2 | 2	1 | 1 1	1 | 1
1 | 1		

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

2-way division		3-way division	
8		8	
4 | 4		3 | 3	2
2 | 2		1 | 1 1	1 | 1
1 | 1	(3 steps)		

Fake coin problem (Cont'd)

Suppose we have a total of 8 coins (with 1 fake coin).

- What could be a good way of finding the fake one?
- How many steps are there?

2-way division		3-way division	
8		8	
4 | 4		3 | 3	2
2 | 2		1 | 1 1	1 | 1
1 | 1	(3 steps)		(2 steps)

Practice

- If the total number is 24, how many steps are required for:

Practice

- If the total number is 24, how many steps are required for:
  - 2-way division
  - 3-way division

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division

    4 steps

  - 3-way division

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division

    4 steps

  - 3-way division

    3 steps

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

- Estimate how many steps are needed for a binary division method, if the problem size is 1000.

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

- Estimate how many steps are needed for a binary division method, if the problem size is 1000.

  $\log_2 1000 \approx 9.97$

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

- Estimate how many steps are needed for a binary division method, if the problem size is 1000.

  $\log_2 1000 \approx 9.97$

  $(2^9 = 512, 2^{10} = 1024)$

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $\log_2 24 \approx 4.58$

  - 3-way division

    3 steps

    $\log_3 24 \approx 2.89$

- Estimate how many steps are needed for a binary division method, if the problem size is 1000.

$\log_2 1000 \approx 9.97$

About 9~10 steps.

$(2^9 = 512, 2^{10} = 1024)$

Practice

- If the total number is 24, how many steps are required for:

  - 2-way division (binary division)

    4 steps

    $$\log_2 24 \approx 4.58$$

  - 3-way division

    3 steps

    $$\log_3 24 \approx 2.89$$

- Estimate how many steps are needed for a binary division method, if the problem size is 1000.

$$\log_2 1000 \approx 9.97$$

$$(2^9 = 512, 2^{10} = 1024)$$

About 9~10 steps.

Much better than a one-by-one comparison method.

Further discussion

Further discussion

- In the coin problem, the repeating "dividing – weighing – dividing" process is featured by recursive algorithms.

Further discussion

- In the coin problem, the repeating "dividing – weighing – dividing" process is featured by recursive algorithms.

- The binary division idea will be an key element in designing searching algorithms in this module.

Further discussion

- In the coin problem, the repeating "dividing – weighing – dividing" process is featured by recursive algorithms.

  (Lecture 3)

- The binary division idea will be an key element in designing searching algorithms in this module.

  (Lecture 5)

Practice

1. Write an algorithm to calculate the perimeter of a rectangle given its height and width.

   Hint perimeter = 2 (width+ height)

2. Write an algorithm that takes an integer as the number of minutes and outputs the total hours and minutes.

   For example (90 minutes = 1 hour 30 minutes).

3. Write an algorithm that converts Celsius to Fahrenheit.

   Hint $F = \frac{9}{5}(C + 32)$

4. Write an algorithm to calculate the volume of a sphere.

   Hint Volume of sphere = $\frac{4}{3}\pi r^3$

Practice

1. Write an algorithm to calculate the perimeter of a rectangle given its height and width.

   Hint perimeter = 2 (width+ height)

Algorithm: Area(w, h)
Requires: two positive numbers width w and height h.
Returns : a positive number P, i.e. perimeter of rectangle.

1. Let $P = 2 * (w + h)$
2. Return P

Practice

2. Write an algorithm that takes an integer as the number of minutes and outputs the total hours and minutes.

   For example (90 minutes = 1 hour 30 minutes).

Algorithm: Timecal(M)
Requires: a positive number M i.e. total number of minutes.
Returns : a pair of positive numbers i.e. h total hours and m remaining minutes.

1. Let h = M/60
2. Let m = M mod 60
3. returns[h,m]

Practice

3. Write an algorithm that converts Celsius to Fahrenheit.

   Hint $F = \frac{9}{5}(C + 32)$

Algorithm: TemConv(C)
Requires: a positive number C ,i.e. temperature record in celcius.
Returns: a positive number F i.e. temperature in Fahrenheit.

1. let $F = \frac{9}{5}(C + 32)$
2. return F

Practice

4. Write an algorithm to calculate the volume of a sphere.

Hint Volume of sphere = $\frac{4}{3}\pi r^3$

Algorithm: Volumeofsphere(r)
Requires: a positive number r i.e. radius of sphere.
Returns : a positive number V i.e. volume of sphere having radius r given.

1. Let $\pi = 3.14$
2. Let $V = \frac{4}{3} * \pi * r * r * r$
3. Return V