School of Computing

Algorithm Design
(Algorithms as High-Order-Primitives)
Video 6.3e

Hon Wai Leong
Department of Computer Science
National University of Singapore

Email, FB: leonghw@comp.nus.edu.sg

Algorithm is Cool. Learn Algorithms.

Quick review

Problem-1: Algorithm to Compute the sum of (1 + 2 + 3 + … + 99 + 100)

Give no-brainer calculated algorithm (BAD-Sum-to-Hundred)

Evolved to first algorithm with **a loop** (**Sum-1-to-100**) (given in flowchart and in pseudo-code)

Algorithm Sum-1-to-100

Problem-1: Algorithm to Compute the sum of
(1 + 2 + 3 + … + 99 + 100)

ALGORITHM Sum-1-to-100;

1. Let Sum ← 0 ;
2. Let k ← 1 ;
3. While (k ≤ 100) repeat Steps 4-6
4.   Sum ← Sum + k
5.   k ← k + 1
6. end-of-while-block;
7. Print out the value of Sum
8. End

Algorithm Sum-1-to-100
(in pseudo-code)

An algorithm is a new *primitive*

So, Sum-1-to-100 is a new *primitive*
(aka module, API, etc)

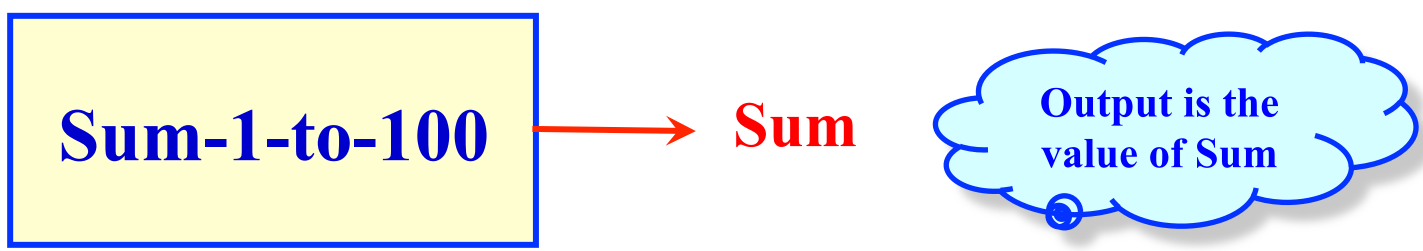
Anyone can use Sum-1-to-100 to solve Prob-1.

If implemented in code, aka **module**, **component**, or **API**, ***sharable*** with others.

Define new primitive carefully

So, we have an algorithm for computing
sum of integers from 1 to 100.
We call it **Sum-1-to-100**

Definition: A new primitive called Sum-1-to-100
The (high-level) primitive **Sum-1-to-100** computes the sum of integers 1 to 100 and returns the total via variable sum.



Analysis of Sum-1-to-100 primitive

Definition: A new primitive called Sum-1-to-100
The (high-level) primitive **Sum-1-to-100** computes the sum of integers 1 to 100 and returns the total via variable sum.

Primitive Sum-1-to-100 has no input… WHY?

If I execute/run Sum-1-to-100 many time,
it always produce the same answer. (5050)
Does not matter what "input" you give to it.

Occam's Razor:
Fewest parameters rules.
In this case, 0 parameters!

Abstraction

How to make this more useful?

Reformulate the Problem

Evolve to Problem-2

Problem-2: Want to compute the sum of integers from 1 to ***n***?
Namely, calculate (1 + 2 + 3 + … + ***n***)

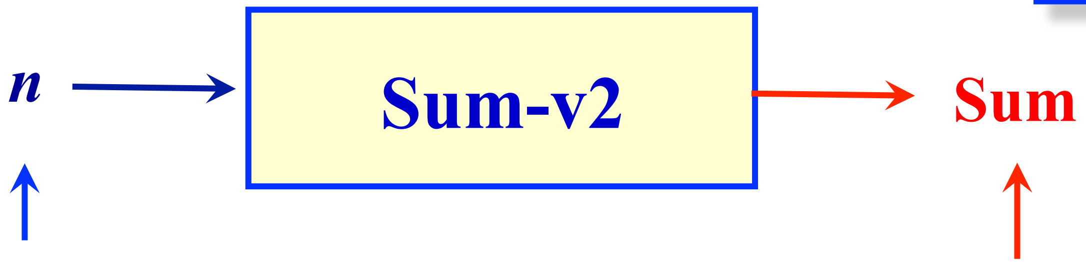
(But we want to specify ***n*** later!)

Why is this useful?

Call it primitive Sum-v2(n)

- Sum-v2(n) is a *high-level primitive* with an input parameter n

Abstraction



Input to **Sum-v2**: variable n

Output is variable Sum

Definition: Sum-v2 (n)
The high-level primitive Sum-v2 takes in as input a integer n, and it computes and returns the sum of $(1 + 2 + \ldots + n)$.

Sum-v2(*n*) more useful?

We use it to compute many different sums

- just by sending different value of n

Examples:

- Sum-v2(100) gives (1+2+3+…+100)
- Sum-v2(24) gives (1+2+3+…+24)
- Can also call Sum-v2(1024)

SO, **Sum-v2(*n*)** is more general that Sum-1-to-100.

How to design
an algorithm for
Sum-v2(n) ?

Appeal to Polya Step-2

PQ: Have we seen a similar problem?
PQ: Can we reuse the result (black box reuse) **or the method** (white box reuse).

Answer: Can reuse Sum-1-to-100 ?

White box reuse: Reuse the algorithm (method) for Sum-1-to-100 with minor changes

Black box reuse: Reuse (result) without any changes to the primitive Sum-1-to-100

Hon Wai Leong, SoC, NUS

White box reuse (small change)

Algorithm Sum-1-to-100

ALGORITHM Sum-1-to-100;

1. Let Sum ← 0 ;
2. Let k ← 1 ;
3. While (k ≤ 100) repeat Steps 4-6
4. Sum ← Sum + k
5. k ← k + 1
6. end-of-while-block;
7. Print out the value of Sum
8. End

Algorithm Sum-v2(n)

ALGORITHM Sum-v2(n);

1. Let Sum ← 0 ;
2. Let k ← 1 ;
3. While (k ≤ n) repeat Steps 4-6
4. Sum ← Sum + k
5. k ← k + 1
6. end-of-while-block;
7. Print out the value of Sum
8. End

Black box reuse (no change)

Q: Can we reuse Sum-1-to-100 as black box to help solve Problem-2 ?

Answer: NO.

Every time we use (run) **Sum-1-to-100, it always gives 5050.**

Problem Reformulation again.

Reformulate again…

Reformulate the Problem

Evolve to Problem-3

Problem-3: We want Sum-Range(p, q) that computes the sum of integers from ***p*** to ***q***. Namely, (p + (p+1) + … + q)

Eg: Sum-Range(25,100) = (25 + 26 + … + 100)

PQ: Can we reuse the result (black box reuse) **or the method** (white box reuse).

White box reuse

White box reuse: Reuse the algorithm (method) for Sum-v2(n) with minor changes to get algorithm for Sum-Range(p,q)

ALGORITHM Sum-v2(n);

1. Let Sum ← 0 ;
2. Let k ← 1 ;
3. While (k ≤ n) repeat Steps 4-6
4. Sum ← Sum + k
5. k ← k + 1
6. end-of-while-block;
7. Print out the value of Sum
8. End

Your DIY HW.
(you remember what is HW?)

Black box reuse of Sum-v2(n)

Black box reuse: Reuse without any changes to the primitive Sum-v2(n)

Example: Sum-Range(25,100)

- Sum-v2(100) gives (**1+2+…+24**+25+…100)
- Sum-v2(24) gives (**1+2+…+24**)

So, Sum-v2(100) – Sum-v2(24) gives Sum-Range(25,100)

COOL! Did not change any code!

Sum-Range(p,q) = Sum-v2(q) – Sum-v2(p–1)

Summary of 6.3d, 6.3e

- Design first algorithm with a loop;
  - Magic power of *loop* (*iterations*)
- Algorithms for 3 related problems
- Learned how to re-use algorithms
  - White box reuse (modify method slightly)
  - Black box reuse (cannot change method)
  - Different thinking skills are needed

(End of video 6.3e)

If you want to contact me,

Email: leonghw@comp.nus.edu.sg

NUS National University of Singapore

School of Computing