Algorithms (more examples…)

- **Supplementary Notes:**
  1. For your reference…
     (esp. those new to programming)
  2. More and simpler examples given…
- **Readings: [SG] Ch. 2 & 3**
- **If you are new to algorithms**
  - read the textbook
  - TRY out the algorithms
  - do the exercises

Overview…

- After this "extra lecture/notes", you can expect to be able to do…
  - Read a set of operations presented to you.
  - Determine if the set is an algorithm or not.
  - If so, determine whether it solves the problem or not.
  - Also, determine what happens if changes are made to algorithms we have studied.
  - If changes are made and the algorithm is no longer correct, what must be done to make it correct.

Notes about Algorithm Design…

- To design an algorithm to solve a problem,
  - you must FIRST know how to solve it,
  - Figure out the steps involved,
  - Organize these steps into steps
  - Express them as algorithms

- To FIRST know how to solve the problem
  - Suggest you work out some cases
  - As many cases as it takes…

Pseudo-Code to express Algorithms

- **Pseudo-Code**
  - **Mixture of computer language and English**
    - *Somewhere in between*
    - *precise enough to describe what is meant without being too tediuos*
  - **Examples:**
    - *Let c be 0;*
    - *c ← 0;*
    - *Sort the list A of numbers in increasing order;*

Variables and Arrays…

- Computers work with data (numbers, words, etc)
- Data must be stored (in variables)
- Each variable is assigned a storage "box"
  - can store one number at any time
  - eg: sum, j, carry
- Arrays:
  - Often deal with many numbers
  - Such as $A_1, A_2, A_3, \ldots, A_{100}$
  - Store as an "array" A[1], A[2], … , A[100]
    - *we treat each of them as a variable,*
    - *each is assigned a storage "box"*

Algorithms

- Three types of operations
  - Sequential Operations…
  - Conditional Operations…
  - Iterative Operations….

Examples of Sequential Operations/Statements

- **Assignment statements**
  - Set count to 0;
  - Assign X the value of (A+B)/2;
  - Let Interest be rate*Principle*Duration;
  - Let A[3] be 3;
  - Let Smallest be A[i+3];
- **Another way to express these…**
  - Count ← 0;
  - X ← (A+B)/2;
  - Interest ← rate*Principle*Duration;
  - A[3] ← 3;
  - Smallest ← A[i+3];
- **Note: These statements are executed one-by-one**

More Sequential Operations/Statements

- **Input / Output Statements;**
  - Get the value of N;
  - Read in the value of A[1], A[2], A[3], A[4];
  - Print the string “Welcome to my Intelligent Agent”;
  - Print “Your IQ is”, A, “ but your EQ is”, A/3;
- **Another way of expressing them…**
  - Read (N);
  - Read (A[1], A[2], A[3], A[4]);
  - Print “Welcome to my Intelligent Agent”;
- **Note: These statements are executed one-by-one**

Tracing (exercising) an algorithm…

```
Sample Algorithm

1. J ← 3;
2. X ← 14;
3. J ← X + 2*J;
```

J	X
?	?
3	?
3	14
20	14

- Given an algorithm (above left), to *exercise* it means
  - **to "trace" the algorithm step-by-step; and**
  - **observe the value of each variable after each step;**
  - **Good to organize as a "table" as shown above (right)**

Algorithms (using sequential stmts)

- Problem
  - Given: Starting mileage, ending mileage, amount of gas used for a trip;
  - Calculate average "km per litre" for the trip
- Example:
  - StartMiles = 12345; EndMiles = 12745; Petrol = 40 litre
  - Average = (12745 – 12345) / 40 = 400/40 = 10 (km/litre)

```
ALGORITHM
1. Get values for StartMiles, EndMiles, GasUsed
2. Let Distance be (EndMiles - StartMiles);
3. Let Average be Distance / GasUsed;
4. Print the value of Average
5. Stop
…
```

Algorithms (using sequential stmts)

- Remarks…
  - **Algorithm below must work for all valid values of StartMiles, EndMiles, and GasUsed;**
  - **Do not need to change the algorithm for different data**
- Can also express algorithm (more concisely) as…

```
ALGORITHM
1. Read ( StartMiles, EndMiles, GasUsed );
2. Distance ← (EndMiles - StartMiles);
3. Average ← Distance / GasUsed;
4. Print Average;
5. Stop
…
```

Algorithms (with better output)

- To obtain a better report, use more print statements;
  - **Print out Details in nice report format;**

```
ALGORITHM
1. Read ( StartMiles, EndMiles, GasUsed );
2. Distance ← (EndMiles - StartMiles);
3. Average ← Distance / GasUsed;
4. Print "Trip Report"
5. Print " Your StartMiles =", StartMiles;
6. Print " Your EndMiles   =", EndMiles;
7. Print " Gas Used        =", GasUsed;
8. Print " Average km/litre=", Average;
9. Print "End of Trip Report";
5. Stop
…
```

To exchange the value of two variables

- Given two values stored in A and B;
- Wanted: An algorithm to exchange the values stored;
- Example:
  - Input: A = 15; B = 24;
  - Required Output: A = 24; B = 15;
- Two Incorrect Algorithms

ALG 1:	A B
1. A ← B; 2. B ← A;	15 24

ALG 2:	A B
1. B ← A; 2. A ← B;	15 24

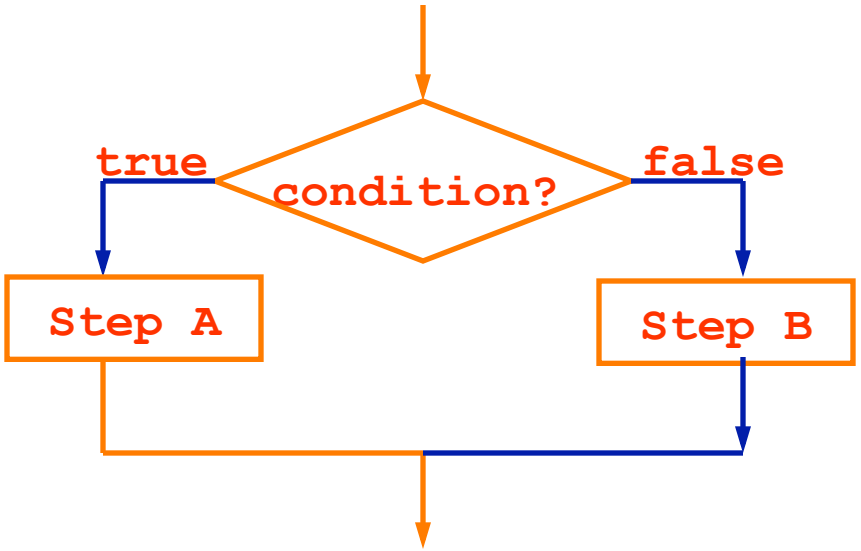
- Error: One of the values was over-written;
- HW: What is a correct algorithm to swap A & B?

Conditional Operations (statements)

- **if statement**
  - to take different actions based on condition
- **Syntax**

```
if (condition)
  then (Step A)
  else (Step B)
endif

if (condition)
  then (Step A)
endif
```



- **Semantics**

Conditional (an example…)

- Problem (continue from AverageMileage Problem)
  - **Suppose we consider good petrol consumption to be Average that is >= 12 km / litre**
  - **Determine if petrol consumption for trip is Good!**
- Example:
  - **Average = 10.0, then "Not good petrol consumption"**
  - **Average = 13.6, then "Good petrol consumption"**

```
ALGORITHM
1. Get Average;
2. if (Average >= 12)
3.    then Print "Good Petrol Consumption";
4.    else Print "Not good petrol comsumption";
5. Endif
6. Stop
…
```

AverageMileage Problem

- Can combine the two parts into one algorithm

```
ALGORITHM
1. Read ( StartMiles, EndMiles, GasUsed );
2. Distance ← (EndMiles - StartMiles);
3. Average ← Distance / GasUsed;
4. Print "Average Mileage is", Average;
5. if (Average >= 12)
6.    then Print "Good Petrol Consumption";
7.    else Print "Not good petrol comsumption";
8. Endif
9. Stop
…
```

If Statement (example…)

- Alg to read in a mark and print out if student pass.
  - Let's say that the passing mark is 40;
- Examples:
  - mark = 25; Expected Output is "Student fail"
  - mark = 45; Expected Output is "Student pass"
  - mark = 99; Expected Output is "Student pass"

```
Algorithm:
1. Read (mark);  (*get value of mark*)
2. if (mark < 40)
3.   then (print "Student fail")
4.   else (print "Student pass")
5. endif
…
```

If Statement (another example…)

```
Algorithm:
1. Read (mark); (* Get value of mark *)
2. if (mark < 40)
3.   then (print "Student fail")
4.   else (print "Student pass")
5. endif
…
```

- Try some cases:
  - When mark = 30; Output is "Student fail"
  - When mark = 42; Output is "Student pass"
  - When mark = 95; Output is "Student pass"
- Note: in the above,
  - either 3 or 4 is executed; *not both*
- Q: What about the different grades of passes?

Two If Statements (one after another)…

```
1. Read (mark); (* Get value of mark *)
2. if (mark < 40)
3.    then (print "Student fail")
4. endif;
5. if (mark >= 40) and (mark < 50)
6.    then (print "Grade D")
7. endif;
…
```

- Try some cases:
  - When mark = 30; Output is "Student fail"
  - When mark = 42; Output is "Grade D"
  - When mark = 95; What is output?
- Where is the "error"?

"Nested" If Statements (one inside another)…

```
1. Read (mark); (* Get value of mark *)
2. if (mark < 40)
3.    then (print "Student fail")
4.    else if (mark < 50)
5.      then (print "Grade D")
6.      else (print "Grade C or better")
7.    endif
7. endif;
…
```

- **Try some cases:**
  - When mark = 30; Output is "Student fail"
  - When mark = 42; Output is "Grade D"
  - When mark = 95; Output is "Grade C or better"

Complicated If Statement

```
read in mark (*from the terminal*)
if (mark < 40) then (Grade ← "F")
  else if (mark < 50) then (Grade ← "D") endif
  else if (mark < 60) then (Grade ← "C") endif
  else if (mark < 70) then (Grade ← "B") endif
  else if (mark < 80) then (Grade ← "A") endif
  else (Grade ← "A+")
endif
print "Student grade is", Grade
```

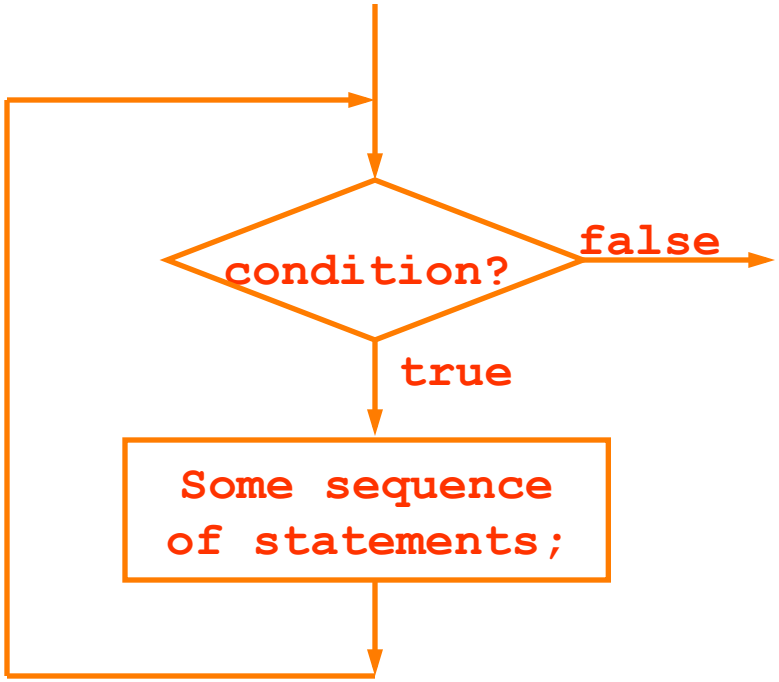
- **This is a complicated if statement;**
  - **Study it carefully to make sure you understand it;**
  - **Can you come up with this algorithm yourself?**

Looping Operations – while-loop

- the while-loop
  - loop a "variable" number of times
- Syntax

```
while (condition) do
    (some sequence
    of statements)
endwhile
```

- Semantics…



"Exercising a while loop"

```
j ← 1;
while (j <= 3) do
  print j;
  j ← j + 1;
endwhile
print "--- Done ---"
```

```
Output:
  1
  2
  3
--- Done ---
```

```
(* General Loop *)
Read(n);
j ← 1;
while (j <= n) do
  print j, A[j];
  j ← j + 1;
endwhile
print "--- Done ---"
```

Looping Primitive – for-loop

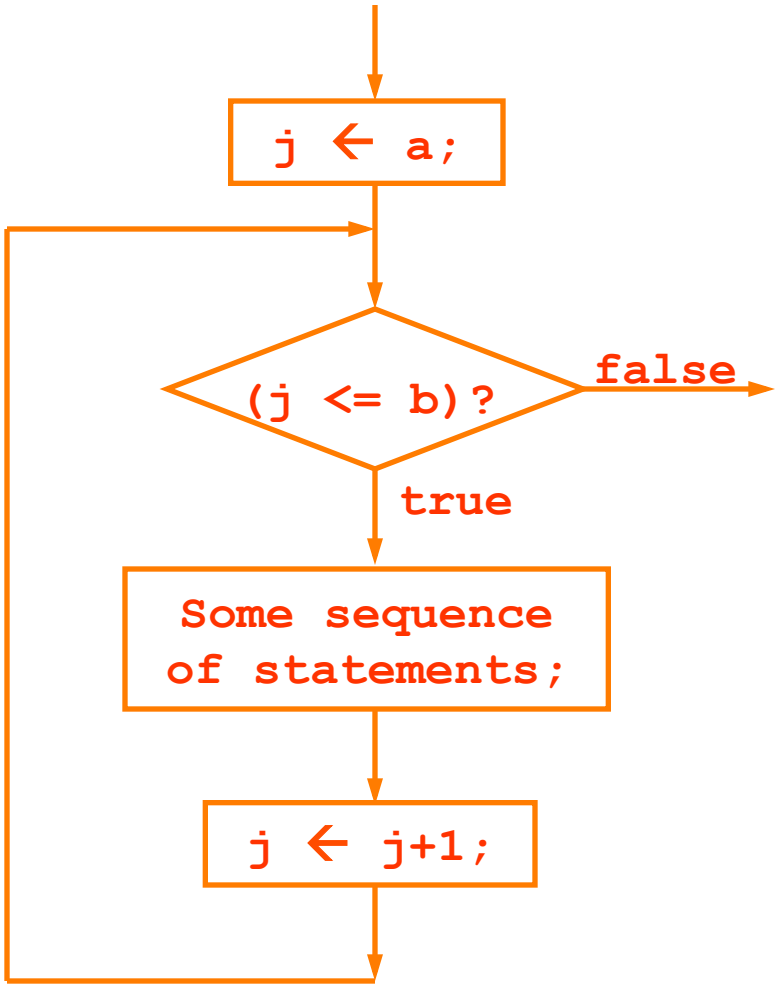
- **First, the for-loop**
  - loop a "fixed" or (pre-determined) number of times
- **Syntax**

  for j ← a to b do

  (some sequence

  of statements)

  endfor
- **Semantics…**



“Exercising the alg”: for

```
for j ← 1 to 3 do
  print j;
endfor
print “--- Done ---”
```

```
Output:
  1
  2
  3
--- Done ---
```

"Exercising the alg": for and while

```
for j ← 1 to 4 do
  print 2*j;
endfor
print "--- Done ---"
```

```
Output:
  2
  4
  6
  8
--- Done ---
```

```
j ← 1;
while (j <= 4) do
  print 2*j;
  j ← j + 1;
endwhile
print "--- Done ---"
```

```
Output:
  2
  4
  6
  8
--- Done ---
```

Simple iterative algorithm: Sum

- **Given: List of numbers: A_1, A_2, A_3, …., A_n**
- **Output: To compute the sum of the numbers**

Note: Store numbers in array A[1], A[2], … , A[n]

```
Sum(A, n);
begin
  Sum_sf ← 0;
  k ← 1;
  while (k <= n) do
    Sum_sf ← Sum_sf + A[k];
    k ← k + 1;
  endwhile
  Sum ← Sum_sf;
  Print "Sum is", Sum
end;
```

Exercising Algorithm Sum:

Input:

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	n=6
2	5	10	3	12	24	

Processing:

k	Sum-sf	Sum
?	0	?
1	2	?
2	7	?
3	17	?
4	20	?
5	32	?
6	56	?
6	56	56

Output:

```
Sum is 56
```

Remarks about the iterative algorithm…

- **Note the three stages:**
  1. **Initialization**
     - ***Set some values at the beginning***
  2. **Iteration**
     - ***This is the KEY STEP***
     - ***Where most of work is done***
  3. **Post-Processing or Cleanup**

- **Can use this setup for other problems**
  - **Calculating average, sum-of-squares**
  - **Finding max, min; Searching for a number,**

Another Example of Algorithm (with loops)

PROBLEM:

Start with a collection of names $N_1, N_2, \ldots, N_{10000}$, and corresponding telephone numbers $T_1, T_2, \ldots, T_{10000}$.

Given a name, aName, find a telephone number T_k for that name if it matches with N_k occurs; otherwise, print "Not Found".

Note the use of subscripts: $N_1, N_2, N_3, N10000, N_k$, etc.

Given a problem, there are often many ways to provide an algorithm for solving the problem.

Note: You must first know how to solve the problem by hand in order to write an algorithm for the solution!!!

A FIRST Attempt at a Solution to the Telephone Search Problem

1. Get values for N_1, N_2, ..., N_{10000}, T_1, T_2, ,,,, T_{10000}, and Name.
2. if Name is N_1, then print T_1 ; Stop endif;
3. if Name is N_2, then print T_2; Stop; endif;
4. If Name is N_3 then print T_3; Stop; endif;

. . . .

{a lot of tedious writing here that is being skipped}

. . . .

10001. If Name is N_{10000}, then print T_{10000} ; Stop; endif
10002. Print "Not found"
10003. Stop.

Method works!
But… Will you do it?
But is extremely tedious.
Can ONLY be used if we have 10000 names!

A SECOND Attempt at a Solution to the Telephone Search Problem

1. Get values for N_1, N_2, ..., N_{10000}, T_1, T_2, … , T_{10000}, and Name.
2. Set the value of k to 1 and the value of Found to NO.
3. Repeat steps 4 through 8 until (Found is Yes)
4. If Name is equal to N_k, then
5. Print the telephone number T_k
6. Set the value of Found to Yes

   Else
7. Add 1 to the value of k
8. Endif
9. Stop.

Better Method
Uses a Loop (repeat loop), more general
ALMOST works.
WHY almost?

ANOTHER ATTEMPT AT A SOLUTION TO THE TELEPHONE SEARCH PROBLEM

1. Get values for N1, N2, ..., N10000, T1, T2, ,,,, T10000, and Name.
2. Set the value of k to 1 and the value of Found to NO.
3. Repeat steps 4 through 8 until (Found is Yes) or (k > 10000)
4. If Name is equal to Nk, then
5. Print the telephone number Tk
6. Set the value of Found to Yes

   Else
7. Add 1 to the value of k
8. endif;
9. end-Repeat
8. If (Found is No) then
9. Print "Not found"
10. Stop.

Solution to Telephone Search Problem (Using a while loop)

Get values for N_1, N_2, ..., N_{10000}, T_1, T_2, ,…, T_{10000}, and Name.
Set the value of i to 1;
Set the value of Found to "NO";
While (Found = "No") and (i <= 10000) do
    If (Name = N_i) then
        Print the telephone number T_i ;
        Set the value of Found to "Yes";
    Else
        Add 1 to the value of i;
Endwhile
If (Found = "No") then
    Print "Not found";

FIND LARGEST ALGORITHM

PROBLEM: **Given n, the size of a list, and a list of n numbers, find the largest number in the list.**

```
Get a value for n and values A1, A2, ..., An for the list items.
Set the value of Largest-so-far to A1.
Set the Location to 1.
Set the value of i to 2.
While ( i <= n) do
        If Ai > Largest-so-far then
                Set Largest-so-far to Ai
                Set Location to i
        Add 1 to the value of i.
Endwhile
Print the values of Largest-so-far and Location.
```

Finally…

- If you are new to algorithms
  - read the textbook
  - try out the algorithms
  - do the exercises

… The End …

Algorithm: A = B + C (in pseudo-code)

We can re-write the C=A+B algorithm as follows:

```
Alg. to Compute C = A + B:
(*sum two big numbers*)
carry ← 0;
for i ← 1 to m do
   x[i] ← a[i] + b[i] + carry ;
   if (x[i] < 10)
      then ( c[i] ← x[i];  carry ← 0; )
      else ( c[i] ← x[i] - 10; carry ← 1; )
endfor;
c[m+1] ← carry;
Print c[m+1], c[m], …., c[1]
```