Recitation Class 03 for VG101

Date: 2012 / 10 / 08

Wang Qian

hwk & lab feedback

- HWK 01
  - Wrong names (may lead to failure in compiling)
  - Wrong units
  - Usage of semi-colon (;)
  - Wrong answer in Problem 3 and Problem 4
  - No steps in Problem 1
- LAB 01
  - Two methods in Problem 1 (vector or for loop)
  - Both strategies can work in LAB 02
  - About the optional problem

String

- Single quotation marks.
- Use sprintf() to record a formatted data in a string.
  - Placeholder & Conversion Specification (see the next slide)
  - Similar to the usage of fprintf()!
- Try the following functions!
  - str2num(), num2str()
  - hex2dec(), dec2hex()
  - size(), length()
  - strcmp(), strcat()
  - strfind(), findstr()

String

- Placeholder
  - str = sprintf('%d %o %x',1234,1234,1234)
  - str = sprintf('%f %e',0.000001234,0.000001234)
  - str = sprintf('%c%c%c%c%c%c', 'H', 'e', 'l', 'l', 'o', '! ')
- Conversion Specification

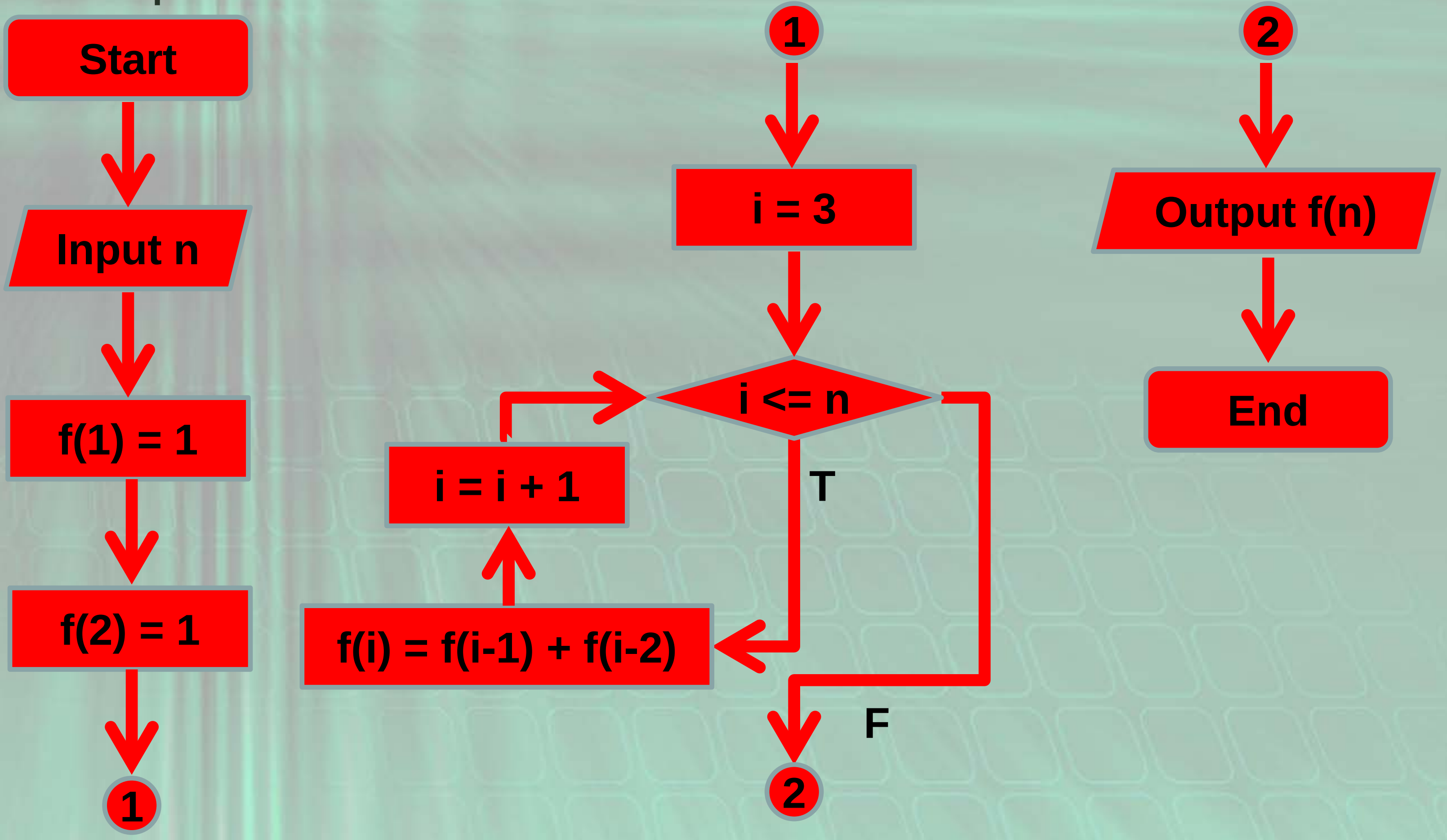
– \n	new line
– \\	back slash
– ''	single quotation mark
– %%	percent sign

Note: regard them as characters

Flow Chart

- A diagram showing the steps and procedures of a certain algorithm
- Basic symbols:
  - Start / end: round rectangle
  - Input / output: parallelogram
  - Condition: diamond
  - General steps: rectangle
  - Flow: arrow
  - Connector: circle

Flow Chart

- Example: calculate the *n*-th Fibonacci number
- 

for statement

- Let’s have a review at first.
- for $<$ scalar $>$ = $<$ vector $>$

        expression

    end
- It can also be nested.
- for $<$ scalar $_1>$ = $<$ vector $>$

        for $<$ scalar $_2>$ = $<$ vector $>$

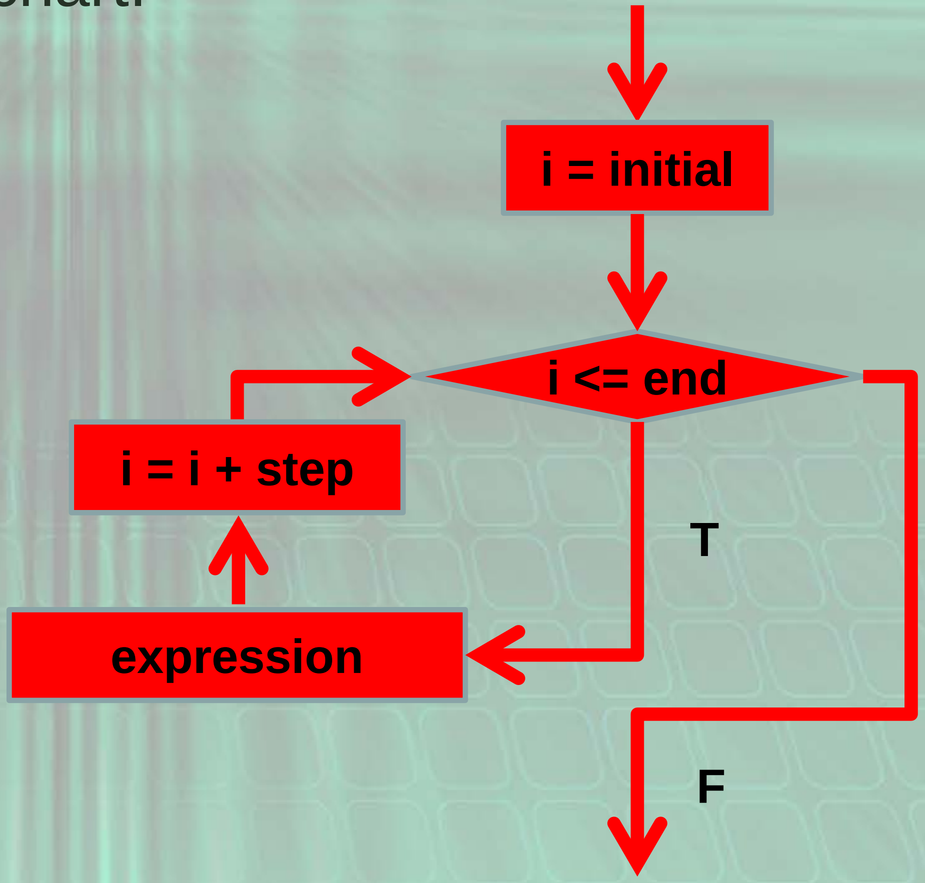
            expression

        end % the end of the inner loop

    end % the end of the outer loop

for statement

- Flow chart:



if statement

- if (condition $_1$)

        expression$_1$

  elseif (condition $_2$) % optional

        expression$_2$

  ……

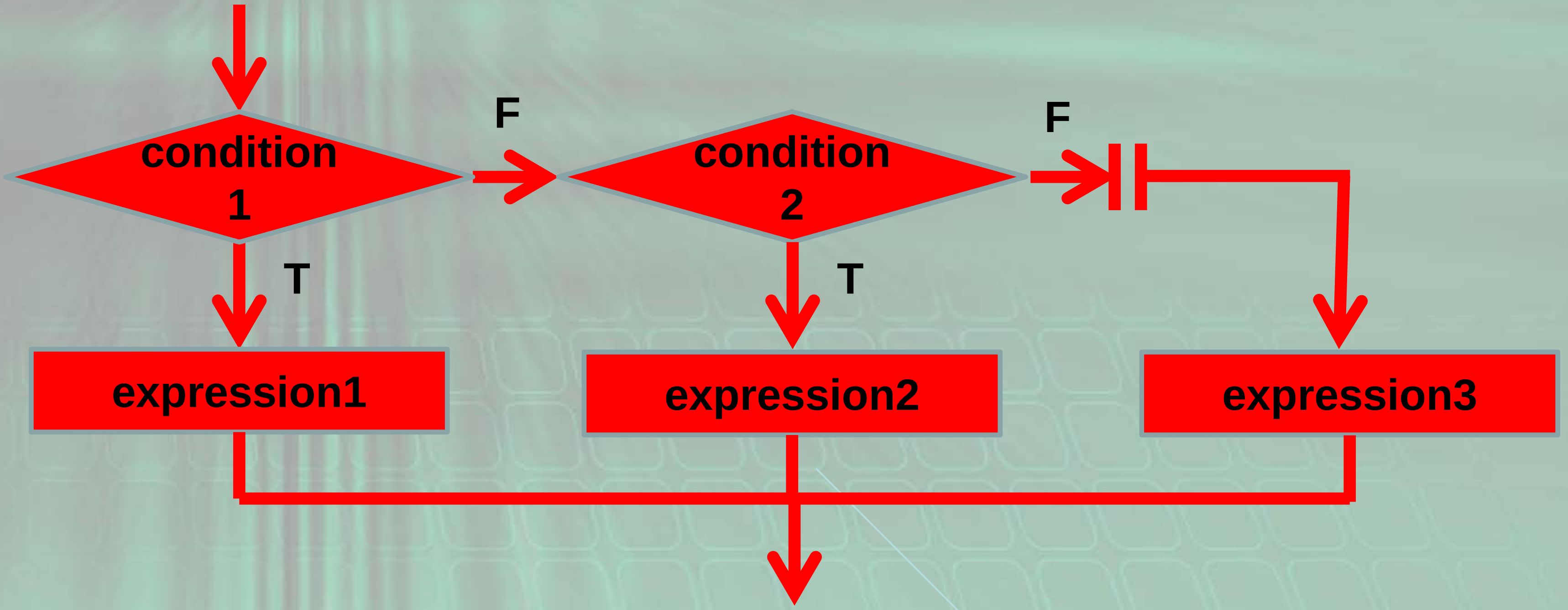
  else (condition $_3$) % optional

        expression$_3$

  end
- Pay attention to the structure of nested *if* statement.

if statement

- Flow chart:



Boolean calculation

- Only two values involved.
  - 1 represents TRUE
  - 0 represents FALSE
  - Relation operators may also be useful
  - **> , < , >= , <= , == , ~=**
- Boolean Operator

A	B	A & B	A | B	~A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Boolean calculation

- Difference between & and &&, | and ||?
  - Elementwise V.S. Short-circuit
  - Try [1 2 3 4] & [0 1 -1 0.9]
  - All numbers other than 0 means true.
- They have different priorities.

  ~ > **relation operator** > **&** > **|** > **&&** > **||**
- A useful strategy: use brackets!
- In C or C++: ^ means xor (more details in the future)

Naive Primality Test

- Determine whether *n* is prime or not.
- Other faster tests exist (Miller-Rabin or AKS)
- Naive primality test is the easiest
- We only use *for* and *if* here
- Strategy:
  - Check whether *n* is divisible by any integer not smaller than 2 and not larger than n-1
  - How to optimize this algorithm?

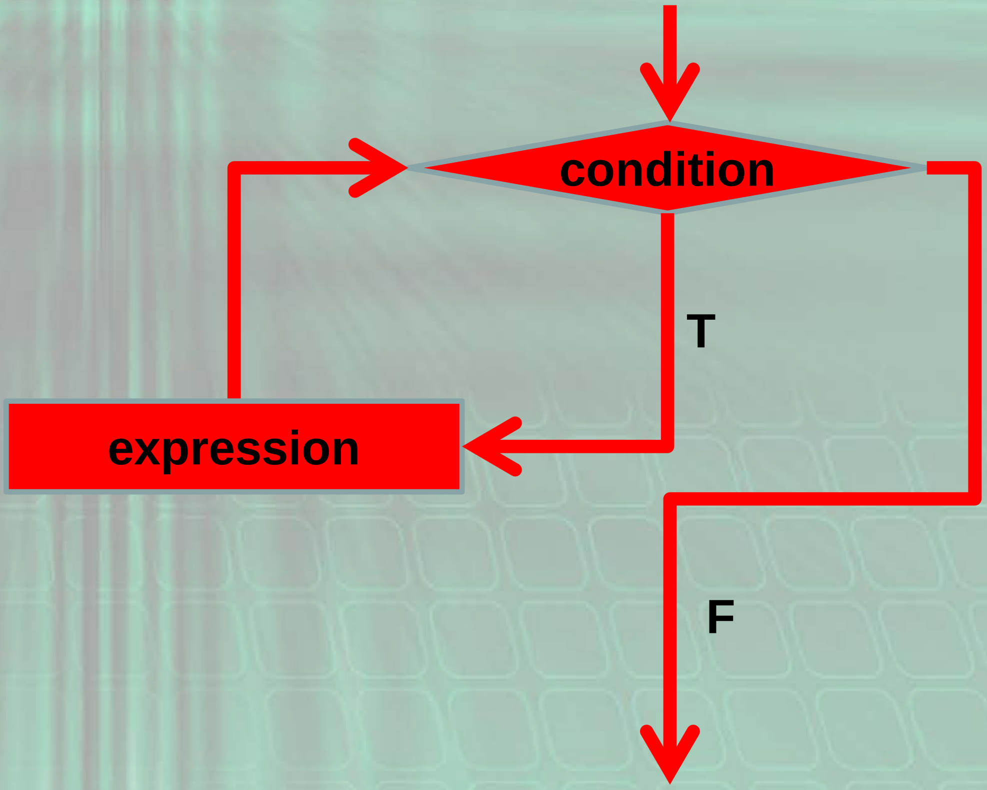
while loop

- while (condition)

  ```
      expression
  end
  ```
- The same as *for* loop, it can also be nested.
- Useful commands: break / continue / return
- Comparison with *for* loop:
  - *for* loop: discrete and given iteration
  - *while* loop: unclear iteration

while loop

- Flow chart:



Greatest Common Divisor

- GCD is very useful in the math world, for example:
  - Multiplicative inverse and RSA algorithm
  - Chinese Remainder Theorem
- Euclid's Algorithm:
  - Given two positive integers *a* and *b*.
  - We have: gcd(*a*,*b*) = gcd(*b*,mod(*a*,*b*))
  - When *b* is equal to 0, gcd(*a*,*b*) = *a*
- Binary Algorithm (get rid of division, efficient for big num):
  - When *a* is equal to *b*, gcd(*a*,*b*) = *a*
  - When a and b are both even, gcd(*a*,*b*) = gcd(*a*/2,*b*/2)
  - When a is even, b is odd, gcd(*a*,*b*) = gcd(*a*/2,*b*)
  - When a and b are both odd, gcd(*a*,*b*) = gcd(*a*-*b*,*b*)

switch statement

- Useful condition: variables representing discrete values
  - total number of the students V.S. current temperature
- switch $< \text{scalar} >$

```
    case {$< \text{value}_{11} >$, $< \text{value}_{12} >$, $< \text{value}_{13} >$}
        $\text{expression}_1$
    case {$< \text{value}_{21} >$, $< \text{value}_{22} >$, $< \text{value}_{23} >$}
        $\text{expression}_2$
    ……
    otherwise
        $\text{expression}_3$
end
```

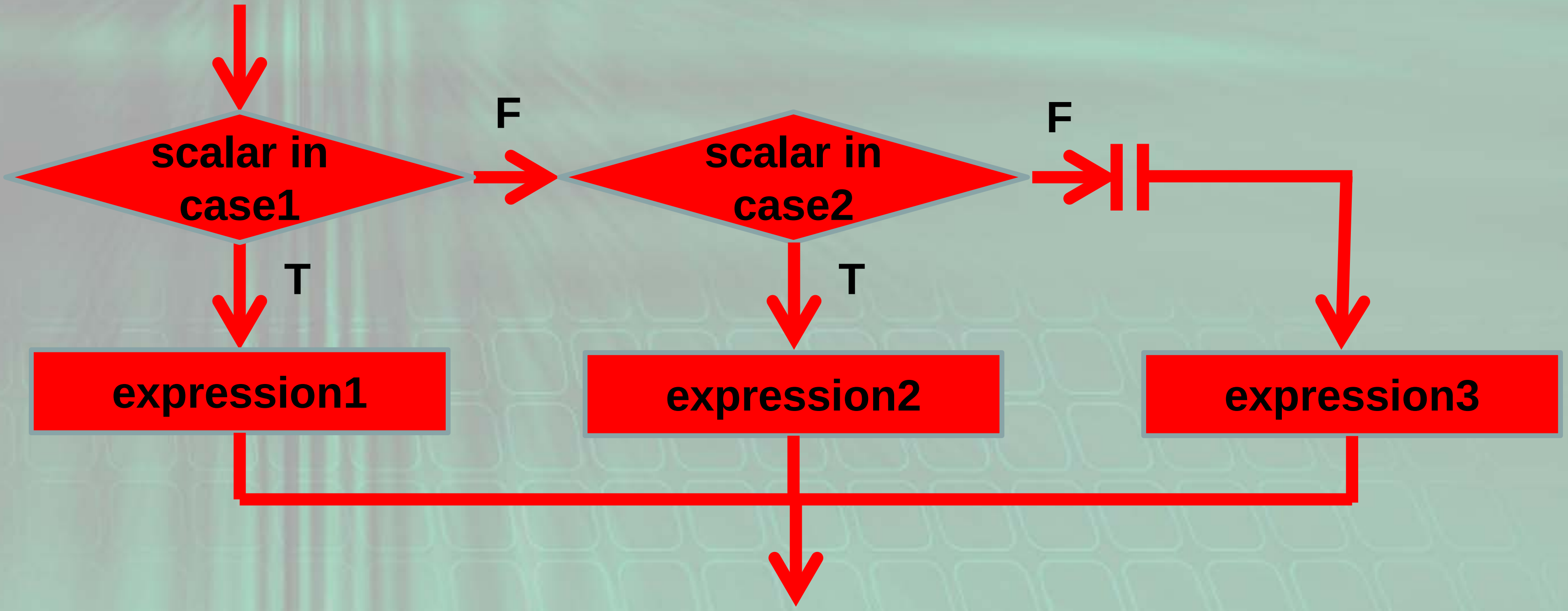
switch statement

- Useful condition: variables representing discrete values
  - total number of the students V.S. current ✗mperature
- switch $<\mathrm{scalar}>$

```
    case {$<\mathrm{value}_{11}>, <\mathrm{value}_{12}>, <\mathrm{value}_{13}>$}
        $\mathrm{expression}_1$
    case {$<\mathrm{value}_{21}>, <\mathrm{value}_{22}>, <\mathrm{value}_{23}>$}
        $\mathrm{expression}_2$
    ……
    otherwise
        $\mathrm{expression}_3$
end
```

switch statement

- Flow chart (different from that in C or C++):



Function

- We have already see the power of the built-in functions.
- function [return values] = name(arguments)

  expression

  return
- Lifespan of variables (will be illustrated later soon)
- e.g.
  - flag = prime_check(n);
  - n = encrypt('Hello! VG101!');
  - x = gcd(a,b);
  - [x,y] = gcd_lcm(a,b);

Debug

- Not only useful in MATLAB
- Tips:

  As a beginner, I recommend you to record all kinds of mistakes which you have made together in a list. This list can be a reminder for you. I listed some common mistakes from my experience.

  – Always initialize the variables
  – Never compare real numbers directly
  – Check the constants
  – 1==a instead of a==1
  – Similar variable “i” and “j”
  – Zero denominator
  – Save different versions
  – ……

Debug

Run:	F5
Halt:	shift + F5
Break point:	F12
Step over:	F10
Step into:	F11
Step out:	shift + F11

- Show the variables on the command window
- Also use “%” instead of deleting the commands

Debug

- The following is only my habit of debug.
  1. Static debug
  2. Compile the code
  3. Check some easy test cases
  4. Make use of the debugger (if mistakes found in step 3)
  5. Check some special test cases
  6. Make use of the debugger (if mistakes found in step 5)
  7. Delete the temporary variables and command for debug
- Example: bubble sort and debug
- How to optimize the bubble sort so that it can halt as soon as the list is in the expected order?

Function

- Easy problem:
- Design a function for bubble sort and optimized bubble sort. This is a very important algorithm.
  - If you are interested in sorting algorithms, you can try to build functions for insertion sort, selection sort, bucket sort and counting sort. You may find their principles easily online, since they are all very classic. Also, they are all among the easiest sorting algorithms.
- Design functions for naive primality test, gcd algorithms, introduced in the previous slides.
- Design a function to find the lcm of two positive integers.
  - Notice the fact that lcm(a,b) * gcd(a,b) = ab

Function

- Challenging problem: (Beyond this course)
- Sieve of Eratosthenes is an algorithm enable us to find all the primes not larger than *n*.
  - Label all the integers as primes.
  - Check *m* from 2 to sqrt(n).
  - For each *m*, label *m***m*, *m**(*m*+1), … , *m**[*n*/*m*] as composite number.
- Design a function for sieve of Eratosthenes.