Java Programming

U Hou Lok

Department of Computer Science and Information Engineering,
National Taiwan University

Java 273
22 Aug.–2 Sep., 2016

```
1  class Lecture4 {
2
3  "Loops"
4
5  }
6
7  // keywords:
8  while, do, for, break, continue
```

Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

- For example, a program outputs "Hello, Java." for 100 times.

```
1  ...
2  System.out.println("Hello, Java.");
3  System.out.println("Hello, Java.");
4  .
5  . // copy and paste for 100 times
6  .
7  System.out.println("Hello, Java.");
8  ...
```

```
1  ...
2  int cnt = 0;
3  while (cnt < 100) {
4  System.out.println("Hello, Java.");
5  cnt++;
6  }
7  ...
```

- This is a simple example to show the power of loops.
- In practice, any routine which repeats couples of times[1] can be done by folding them into a loop.

[1] I'd like to call them "patterns."

- Loops provide significant computational power.
- Loops bring an efficient way of programming.
- Loops could consume a lot of time.[2]

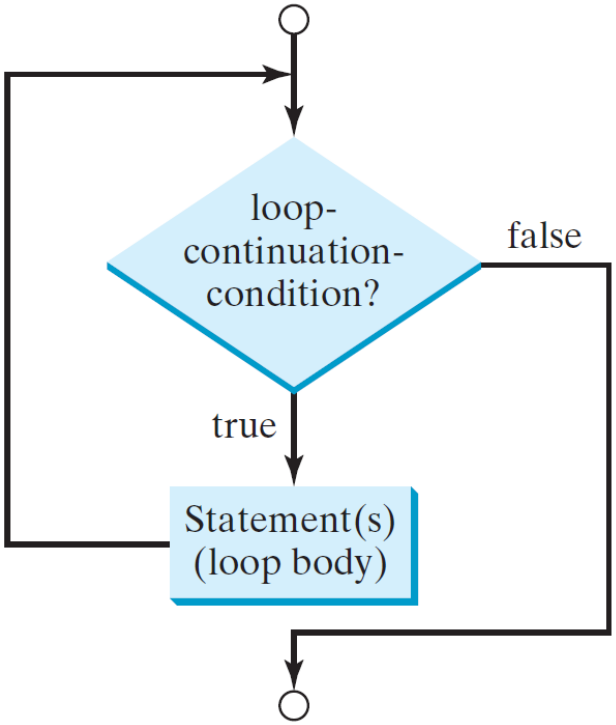
[2]We will visit the analysis of algorithms in the end of this lecture.

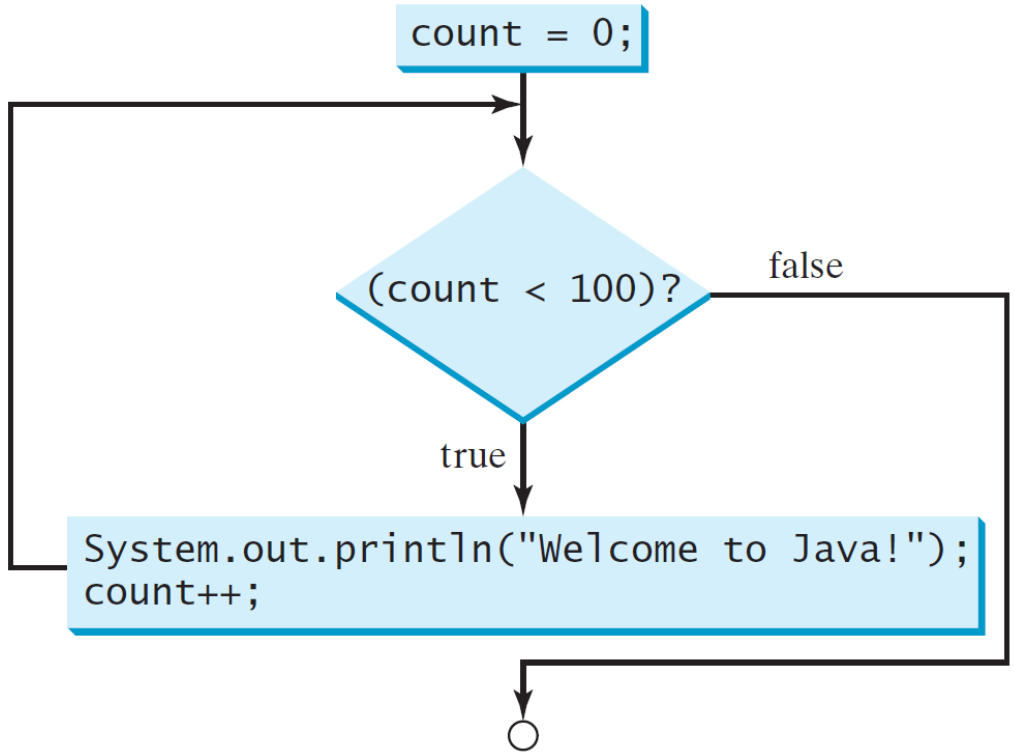
while Loops

A while loop executes statements repeatedly while the condition is true.

```
1  ...
2  while (condition) {
3  // loop body
4  }
5  ...
```

- The condition is a Boolean expression which controls the execution of the body.
- It is evaluated each time to determine if the loop body is executed.
- If true, the loop body is executed.
- Otherwise, the entire loop terminates.


loop-
continuation-
condition?
false
true
Statement(s)
(loop body)


count = 0;
(count < 100)?
false
true
System.out.println("Welcome to Java!");
count++;

Example

Write a program which sums up all integers from 1 to 100.

- In math, the question can be:

$$\text{sum} = 1 + 2 + \cdots + 100.$$

- But the form is not doable in the machine.
- Try computational thinking.[3]

[3] We are familiar with mathematical thinking since we learn math before we learn how to code.

- Normally, the computer executes the instructions sequentially.[4]
- So one needs to decompose the math equation into several steps, like:

```
1  ...
2  int sum = 0;
3  sum = sum + 1;
4  sum = sum + 2;
5  .
6  .
7  .
8  sum = sum + 100;
9  ...
```

- Cons: not efficient, not general (what if sum up to 10^{10}?)

[4] If we are talking about the parallel computing, then it is a different story.

- Using a while loop, the program looks like this:

```
1  ...
2  int sum = 0;
3  int i = 1;
4  while (i <= 100) {
5  sum = sum + i;
6  ++i;
7  }
8  ...
```

- Make sure that the condition eventually becomes false so that the loop will terminate.
- It is really easy to make an infinite loop.

```
1  ...
2  while (true);
3  ...
```

- Besides, replacing 100 by n determined by the user makes this program more general.

```
1  ...
2  Scanner input = new Scanner(System.in);
3  int n = input.nextInt();
4  int sum = 0;
5  int i = 1;
6  while (i <= n) {
7  sum = sum + i;
8  i = i + 1;
9  }
10 ...
```

- In practice, the number of loop steps is unknown until the input data is given.

Example

Write a program which sums two random integers and lets the user repeatedly enter a new answer until it is correct.

```
1   ...
2   Scanner input = new Scanner(System.in);
3   int x = (int) (Math.random() * 10);
4   int y = (int) (Math.random() * 10);
5   int ans = x + y;
6
7   System.out.println(x + " + " + y + " = ? ");
8   int z = input.nextInt();
9
10  while (z != ans) {
11  System.out.println("Try again? ");
12  z = input.nextInt();
13  }
14  System.out.println("Correct.");
15  input.close();
16  ...
```

Loop Design Strategy

Writing a correct loop is not an easy task for novice programmers.
Consider 3 steps when writing a loop:

- Find the pattern: identify the statements that need to be repeated.
- Wrap: put these statements in a loop.
- Set the continuation condition: translate the criteria from the real world problem into computational conditions.[5]

[5] Not unique.

Sentinel-Controlled Loop

Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a sentinel value, signifies the end of the loop.
- For example, the operating systems and the GUI apps.

Example: Cashier Problem

Write a program which sums positive integers from the input except for -1 to exit, and displays the sum.

```
1   ...
2   Scanner in = new Scanner(System.in);
3   System.out.println("Enter a positive integer (-1 to exit): ");
4   int x = in.nextInt();
5   int sum = 0;
6   while (x != -1) {
7   if (x > 0) sum += x;
8   System.out.println("Enter a positive integer (-1 to exit): ");
9   x = in.nextInt();
10  }
11  System.out.println("Sum = " + sum);
12  in.close();
13  ...
```

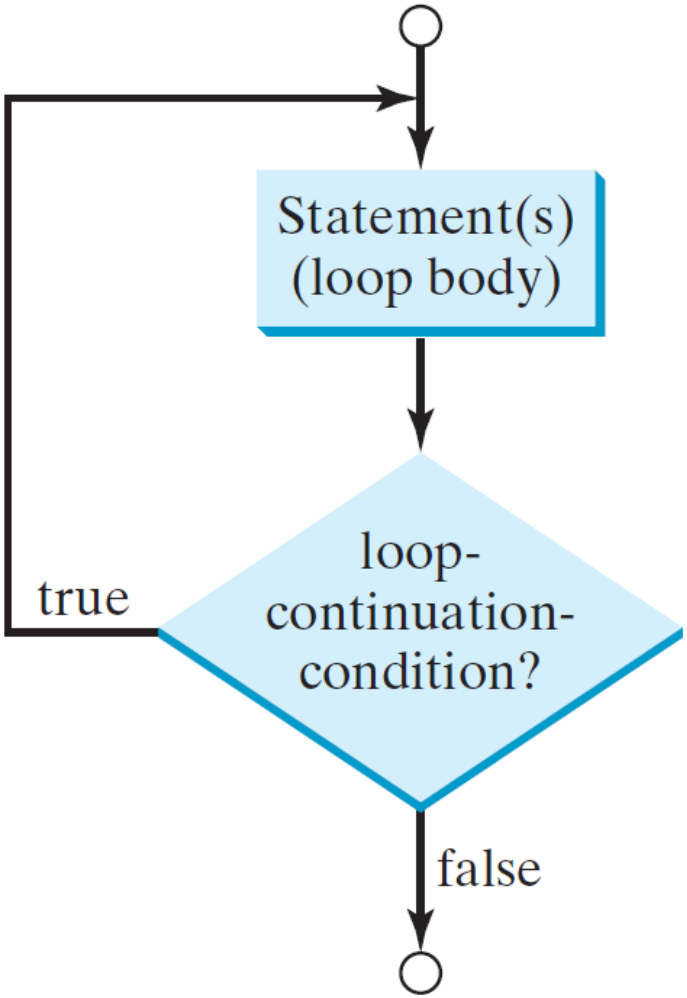
- Line 8 and 9 are the recurrence of Line 3 and 4?!

do-while Loops

A do-while loop is the same as a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```
1  ...
2  do {
3  // loop body
4  } while (condition); // Do not miss the semicolon!
5  ...
```

- Note that there is a semicolon at the end the do-while loop.
- The do-while loops are also called posttest loop, in contrast to while loops, which are pretest loops.


Statement(s)
(loop body)
loop-
continuation-
condition?
true
false

Example (Revisted)

Write a program which sums positive integers from the input except for -1 to exit, and displays the sum.

```
1   ...
2   Scanner in = new Scanner(System.in);
3   int sum = 0;
4   int x = 0;
5
6   do {
7   System.out.println("Enter a positive integer (-1 to exit): ");
8   x = in.nextInt();
9   if (x >= 0) sum += x;
10  } while (x != -1);
11  System.out.println("Sum = " + sum);
12  in.close();
13  ...
```

Exercise

Write a program which allows the user to enter positive integers except for -1 to exit, and displays the maximum.

```
Please enter a real number (-1 to exit):
5
Max = 5.0
Please enter a real number (-1 to exit):
2
Max = 5.0
Please enter a real number (-1 to exit):
7
Max = 7.0
Please enter a real number (-1 to exit):
-1
```

for Loops

A for loop generally uses a variable to control how many times the loop body is executed and when the loop terminates.

```
1  ...
2  for (init; condition; increment) {
3  // loop body
4  }
5  ...
```

- *init*: declare and initialize a variable
- *condition*: a continuation criterion
- *increment*: how the variable changes after each iteration
- Note that the three terms are separated by semicolons.

Example

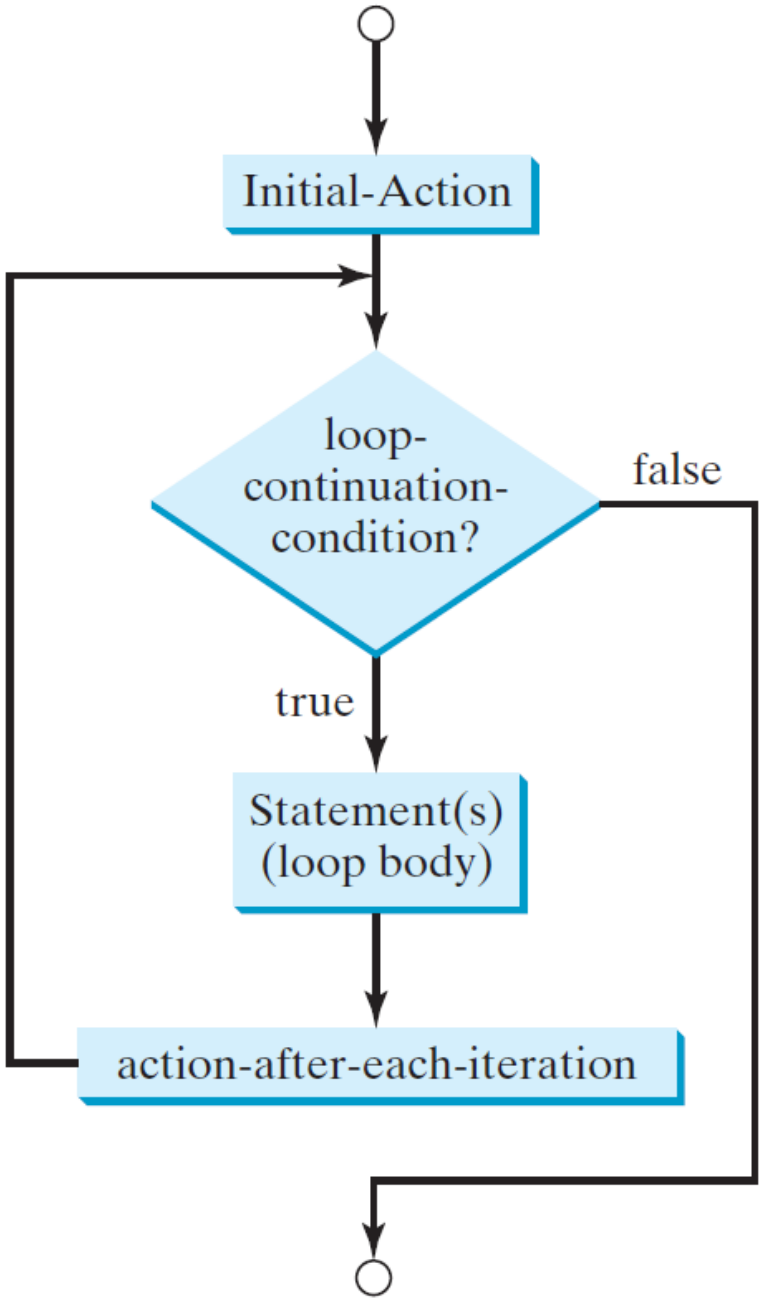
Sum from 1 to 100

Write a program which sums from 1 to 100.

```
1  ...
2  int sum = 0;
3  for (int i = 1; i <= 100; ++i)
4  sum = sum + i;
5  ...
```

- Compared to the while version,

```
1  ...
2  int sum = 0;
3  int i = 1;
4  while (i <= 100) {
5  sum = sum + i;
6  ++i;
7  }
8  ...
```


Initial-Action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration

Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where k is an integer.

- You may use modular operator (%).

```
...
for (int i = 1; i <= 100; i++) {
if (i % 2 == 0)
System.out.println(i);
}
...
```

- You may consider this alternative:

```
...
for (int i = 2; i <= 100; i += 2) {
System.out.println(i);
}
...
```

- How about odd numbers?

Jump Statements

The keywords, break and continue, are often used in loop structures to provide additional controls.

- break: the loop is terminated right after a break statement is executed.
- continue: the loop skips this iteration right after a continue statement is executed.
- In practice, jump statements are placed within selection structures in loops. (Why?)

Example

isPrime problem

Write a program which determines if the input integer is a prime number.

- Recall that a prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- Let x be any natural number.
- The most naive approach is to divide x by all natural numbers smaller than x.
- A better approach is to divide x by all natural numbers smaller than $\sqrt{x}$. (Why?)

```
1   ...
2   Scanner input = new Scanner(System.in);
3   System.out.println("Please enter an integer: ");
4   int x = input.nextInt();
5   boolean isPrime = true;
6   in.close();
7
8   for (int i = 2; i <= Math.sqrt(x); i++) {
9   if (x % i == 0) {
10  isPrime = false;
11  break;
12  }
13  }
14
15  if (isPrime) {
16  System.out.println("Prime");
17  } else {
18  System.out.println("Composite");
19  }
20  ...
```

Exercise (Revisited)

- Redo the cashier problem by using an infinite loop with a jump statement.

```
...
while (true) {
System.out.println("Enter a positive integer (-1 to exit): ");
x = input.nextInt();
if (x >= 0) {
sum += x;
} else if (x == -1) {
System.out.println("Sum = " + sum);
break;
} else {
System.out.println("Try again.");
}
}
in.close();
...
```

Equivalence: while and for Loops

Compounding problem

Consider that one saves $10,000$ NTD in a bank. Write a program which determines the number of years n such that the compounding amount of saving exceeds $15,000$ NTD as r is specified.

- We may need the following variables:
  - *principal*: the starting amount
  - *currAmount*: the current balance of saving
  - *goalAmount*: the amount we set
  - r: interest rate, herein, assuming constant
- Recall that the compounding formula.
- Stopping criterion? Continuation criterion?

```
1   ...
2   public static void main(String[] args) {
3   Scanner inpunt = new Scanner(System.in);
4
5   System.out.println("Interest rate (%) = ?");
6   double r = input.nextDouble();
7   System.out.println("Principal = ?");
8   int principal = input.nextInt();
9   System.out.println("Goal = ?");
10  int goalAmount = input.nextInt();
11  int currAmount = principal;
12  in.close();
13
14  int years = 0;
15  while (currAmount < goalAmount) { // continuation criterion
16  currAmount *= (1 + r / 100);
17  years++;
18  }
19
20  System.out.println("Years = " + years);
21  System.out.println("Amount = " + currAmount);
22  }
23  ...
```

```
1    ...
2    int years; // should be declared; scope issue
3    for (years = 1; true; n++) {
4    currAmount *= (1 + r / 100);
5    if (currAmount >= goalAmount) // stopping criterion
6    break;
7    }
8    ...
```

- A for loop can be an infinite loop by setting true or simply leaving empty in the condition statement.
- An infinite for loop with an if-break statement is equivalent to a normal while loop.

Equivalence: while and for Loops (Concluded)

You can use a for loop, a while loop, or a do-while loop, whichever is convenient.

- In general, a for loop may be used if the number of repetitions is known in advance.
- If not, a while loop is preferred.

Nested Loops

A loop can be nested inside another loop.

- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, and started anew.

Example

Multiplication table

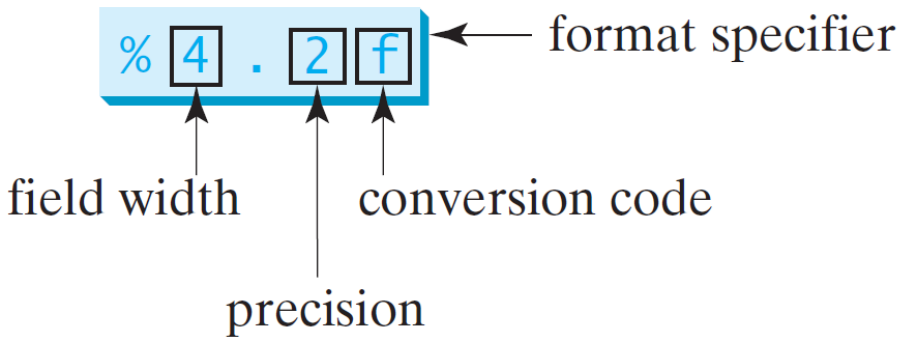
Write a program which displays the multiplication table.

```
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Formatting Console Output

You can use *System.out.printf*() to display formatted output on the console.

```
1    ...
2    double amount = 1234.601;
3    double interestRate = 0.00528;
4    double interest = amount * interestRate;
5    System.out.printf("Interest = %4.2f", interest);
6    ...
```

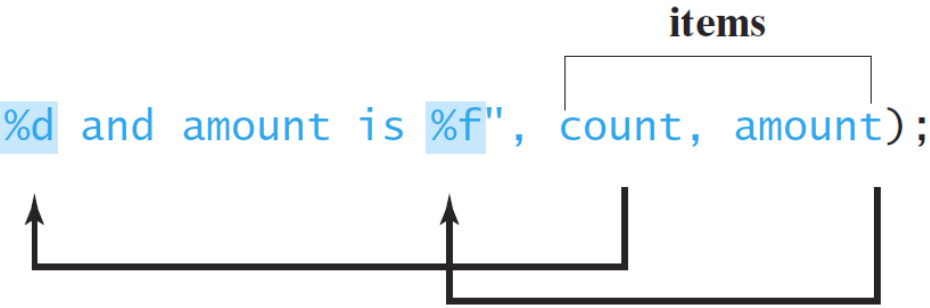


Format Specifier	*Output*	*Example*
%b	a Boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"

- By default, a floating-point value is displayed with 6 digits after the decimal point.

Example: Multiple Items

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

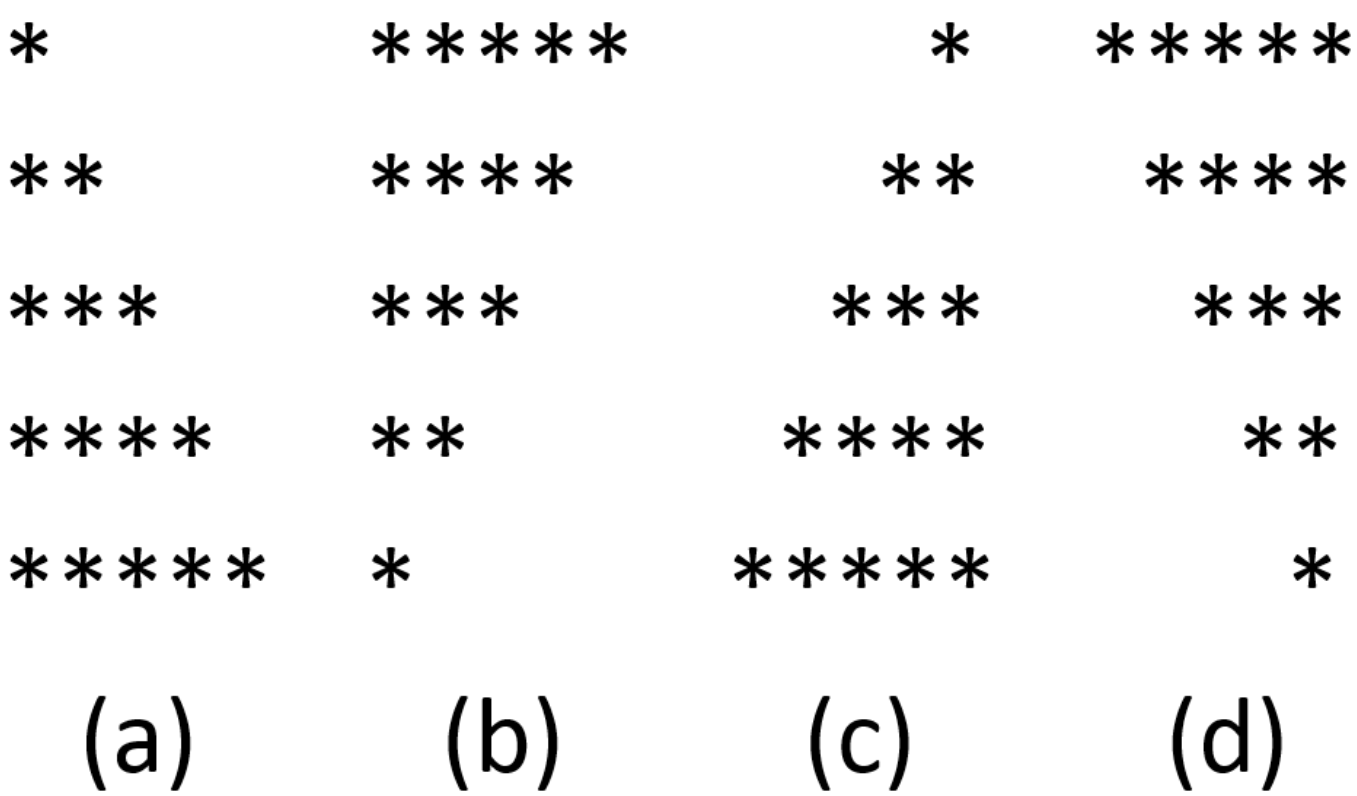



```
display                count is 5 and amount is 45.560000
```

- Items must match the format specifiers in order, in number, and in exact type.
- If an item requires more spaces than the specified width, the width is automatically increased.
- By default, the output is right justified.
- You may try the plus sign (+), the minus sign (-), and 0 in the middle of format specifiers.
  - Say %+8.2f, %-8.2f, and %08.2f.

```
1    ...
2    public static void main(String[] args) {
3    for (int i = 1; i <= 9; ++i){
4    for (int j = 1; j <= 9; ++j)
5    System.out.printf("%3d", i * j);
6    System.out.println();
7    }
8    }
9    ...
```

Exercise: Coupled Loops



```
1     public class PrintStarsDemo {
2     public static void main(String[] args){
3     // case (a)
4     for (int i = 1; i <= 5; i++) {
5     for (int j = 1; j <= i; j++){
6     System.out.printf("*");
7     }
8     System.out.println();
9     }
10    }
11    }
```

Analysis of Algorithm In A Nutshell

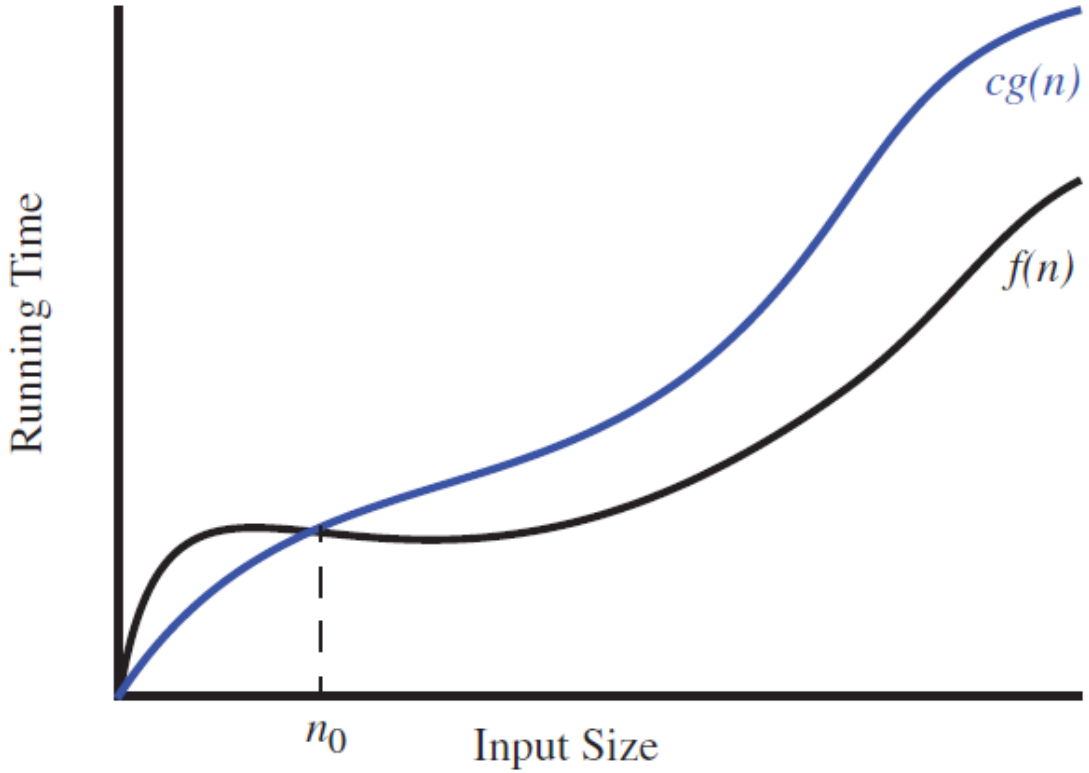
- First, there may exist some algorithms for the same problem, which are supposed to be correct.
- Then we compare these algorithms.
- The first question is, Which one is more efficient? (Why?)
- We focus on the growth rate of the running time or space requirement as a function of the input size n, denoted by $f(n)$.

O-notation

- In math, O-notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.
- **Definition** (O-notation) $f(n) \in O(g(n))$ as $n \to \infty$ if and only if there is a positive constant c and a real number n_0 such that

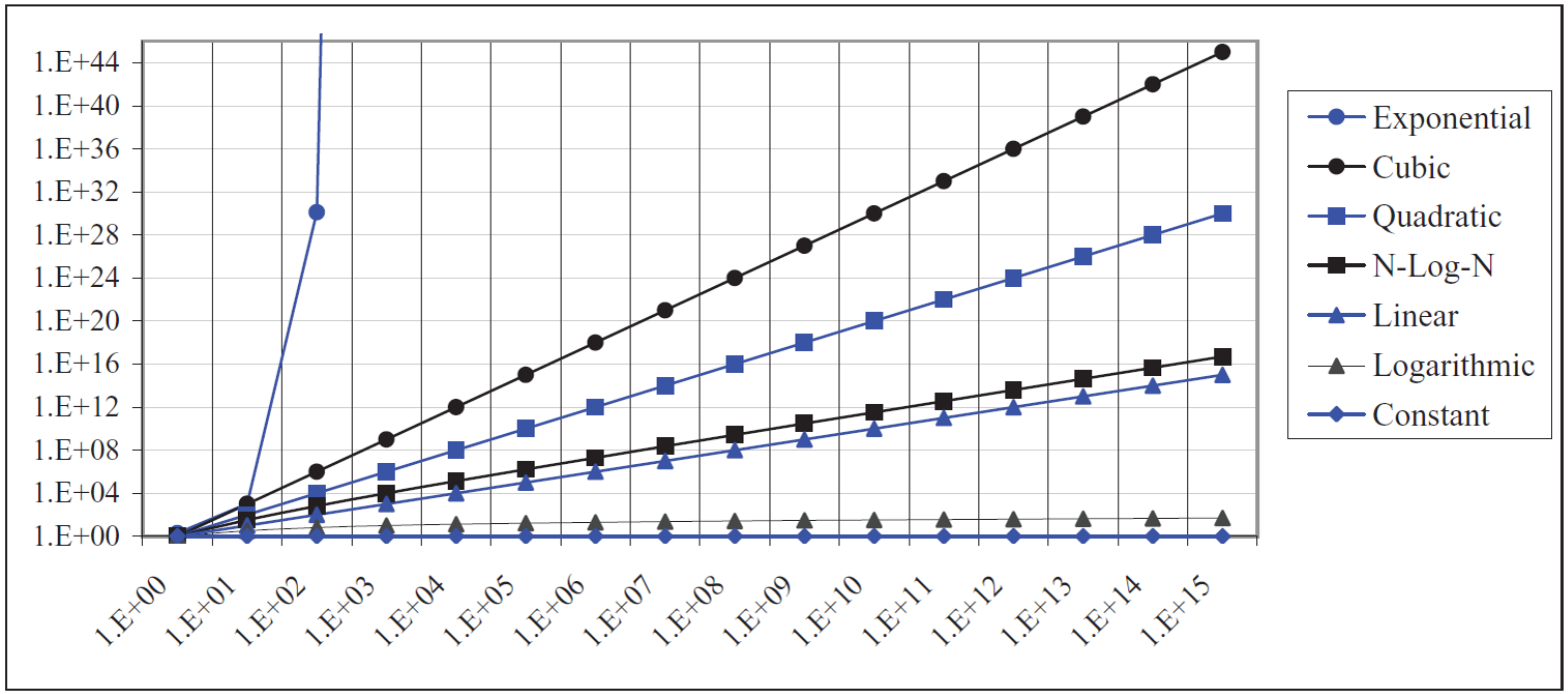
$$|f(n)| \leq c|g(n)| \quad \forall n \geq n_0. \tag{1}$$

- Note that $O(g(n))$ is a set featured by some simple function $g(n)$.
- Hence $f(n) \in O(g(n))$ is equivalent to say that $f(n)$ is one instance of $O(g(n))$.



- For example, $8n^2 - 3n + 4 \in O(n^2)$ (tight bound).
- We say $8n^2 - 3n + 4 \in O(n^3)$ and $8n^2 - 3n + 4 \notin O(n)$.

Growth Rates for Fundamental Functions[6]



constant	*logarithm*	*linear*	*n-log-n*	*quadratic*	*cubic*	*exponential*
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

[6] See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.

- O-notation is used to classify algorithms by how they respond to changes in input size.[7]
  - Time complexity
  - Space complexity
- In short, O-notation describes the asymptotic[8] upper bound of the algorithm, equivalently, the worst case as $n \to \infty$.

[7] Actually, there are $\Theta, \theta, o, \Omega$, and ω which classify algorithms.

[8] The asymptotic sense is that the input size n grows toward infinity.

Digress: Analysis of Algorithms

Once an algorithm is given for a problem and decided to be correct, the next step is to determine how much time in use.[9] (Why?)

- How to estimate the time required for a program? (Real time cost?)
- How to reduce the running time of a program?
- Faster =?= better

[9] Sometime we also consider the space required for a program. But time is scarcer than space in common cases.

Problem Set

Exercise 4.1 (Greatest common divisor)

Write a program which receives two positive integers and returns the greatest common divisor which is the largest positive integer that divides the numbers without a remainder.

Exercise 4.2 (Find all prime numbers smaller than 1000)

Write a program which displays all prime numbers smaller than 1000.

Exercise 4.3 (π estimated by Monte Carlo)

Write a program which estimates π by Monte Carlo Simulation.

Exercise 4.4 (Find the two highest scores)

Write a program that prompts the user to enter the number of students and each student's name and score, and finally displays the student with the highest score and the student with the second-highest score.

Exercise 4.5 (Find numbers divisible by 5 and 6)

Write a program that displays all the numbers from 100 to 1000 that are divisible by 5 and 6. Numbers are separated by exactly one space.

Exercise 4.6 (Continued from 4.6)

Write a program that displays all the numbers from 100 to 200, ten per line, that are divisible by 5 or 6, but not both. Numbers are separated by exactly one space.

Exercise 4.7

Write a program that finds the smallest n such that $n^2 > 12000$ using a while loop.

Exercise 4.8

Write a program which finds the largest n such that $n^3 < 12000$ using a while loop.

Exercise 4.9 (Display pyramid)

Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid, as shown in the following sample run:

```
Enter the number of lines: 7 ↵Enter
                    1
                  2 1 2
                3 2 1 2 3
              4 3 2 1 2 3 4
            5 4 3 2 1 2 3 4 5
          6 5 4 3 2 1 2 3 4 5 6
        7 6 5 4 3 2 1 2 3 4 5 6 7
```

Exercise 4.10 (Display numbers in a pyramid pattern)

Write a nested for loop that prints the following output:

```
                    1
                  1 2 1
                1 2 4 2 1
              1 2 4 8 4 2 1
            1 2 4 8 16 8 4 2 1
          1 2 4 8 16 32 16 8 4 2 1
        1 2 4 8 16 32 64 32 16 8 4 2 1
      1 2 4 8 16 32 64 128 64 32 16 8 4 2 1
```

Exercise 4.11 (Display four patterns using loops)

Use nested loops that display the following patterns in four separate programs:

```
Pattern A
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

```
Pattern B
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
Pattern C
          1
        2 1
      3 2 1
    4 3 2 1
  5 4 3 2 1
6 5 4 3 2 1
```

```
Pattern D
1 2 3 4 5 6
  1 2 3 4 5
    1 2 3 4
      1 2 3
        1 2
          1
```

Exercise 4.12 (Display leap years)

Write a program that displays all the leap years, ten per line, in the twenty-first century (from 2001 to 2100), separated by exactly one space.

Exercise 4.13 (Compute π)

You can approximate by using the following series:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^{i+1}}{2i-1}\right)$$

Write a program that displays the value for $i = 10000, 20000, \ldots$ and 100000.

Exercise 4.14 (Perfect number)

A positive integer is called a perfect number if it is equal to the sum of all of its positive divisors, excluding itself. For example, 6 is the first perfect number because $6 = 3 + 2 + 1$. The next is $28 = 14 + 7 + 4 + 2 + 1$. There are four perfect numbers less than 10000. Write a program to find all these four numbers.

Exercise 4.15 (Game: scissor, rock, paper)

Exercise 3.17 gives a program that plays the scissor-rock-paper game. Revise the program to let the user continuously play until either the user or the computer wins more than two times.

Exercise 4.16 Histogram for math grades

Write a program which allows the user to enter the math grades in order (-1 to exit), and makes a histogram.

```
Enter (-1 to exit): 40
Enter (-1 to exit): 90
Enter (-1 to exit): 86
Enter (-1 to exit): 56
Enter (-1 to exit): 78
Enter (-1 to exit): 64
Enter (-1 to exit): -1
End of Input...
```

```
Total: 6
 90 ~ 100: *
 80 ~  89: *
 70 ~  79: *
 60 ~  69: *
  0 ~  59: **
```