Introduction to Computational Thinking

Program analysis;
Loop pattern: Sentinel input/Fencepost loops

Qiao Xiang, Qingyu Song
https://sngroup.org.cn/courses/ct-xmuf25/index.shtml
11/23/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

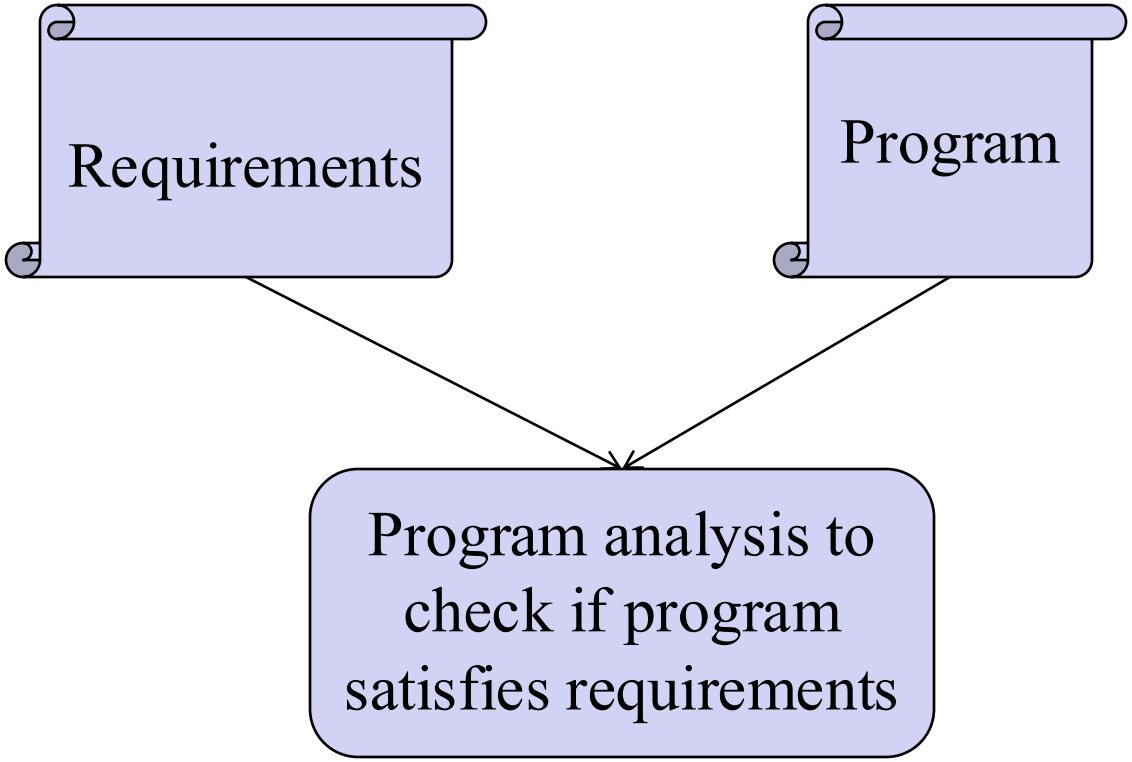
Foundational Programming Concepts

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do-while)
  - Common indefinite loop pattern: search loop
  - Program analysis

Program Analysis

- A useful tool to understand flow control



Foundation of Program Analysis: Logical Assertions

- **Assertion**: A statement that we focus on whether it is true, false, or sometime true/sometime false (unknown).

  Examples:

  - Yale was founded in 1701.
  - The capital of Connecticut is New Haven.
  - Prof. Yang met a donkey this morning.
  - int x;

    …

    x = x+10;

    x divided by 2 equals 7. *(depends on the value of x)*

Logical Assertions on Program Variables

- One can make assertions on program variables at each point of a program
  - For example, right after a variable is initialized, its value is known, and we can make true/false statement:

```
int x = 3;
// is x > 0?  True
```

- A common confusion: An assertion is not part of your program; it is an external claim/property of your program

Difficulty of Making Assertions

- The value of a variable may become unknown after certain operations (hence leading to "unknown/sometimes" assertions), e.g.,
  - reading from a `Scanner`
  - assigned a number from a random number
  - a parameter's initial value to a method

```
public static void mystery(int a, int b)
{
// is a == 10?  UNKNOWN
```

Control Structure Establishes Assertions

- A key function of a control structure (e.g., `if`, `while`, `break`) is that it establishes assertions:

```
public static void mystery(int a, int b)
{
    if (a < 0) {
        // is a == 10?    FALSE
        ...
    }
}
```

We know a < 0, which implies a != any positive number.

Assertions and Controls

- At the start of an `if` or loop's body, the <test> or what can be implied by the <test> must be `true`, e.g.,

```
while (y < 10) {
    // is y < 10?   TRUE
    ...
}
```

- In the `else` or after a loop w/o break, the <test> must be `false`:

```
while (y < 10) {
    ...
}
// is y < 10?       FALSE
```

- Note: Inside a loop's body, the loop's test may become `false`:

```
while (y < 10) {
    y++;
    // is y < 10?   UNKNOWN
    // if y < 12    TRUE
}
```

Using Assertions to Understand Program

```
public static double getNonNegDouble(Scanner console) {
    System.out.print("Type a nonnegative number: ");
    double number = console.nextDouble();

    while (number < 0.0) {
        // ASSERTION: number < 0
        System.out.print("Error: Negative; try again: ");

        number = console.nextDouble();
    }
    // ASSERTION: number >= 0.0
    return number;
}
```

Is the following statement about the program above correct:

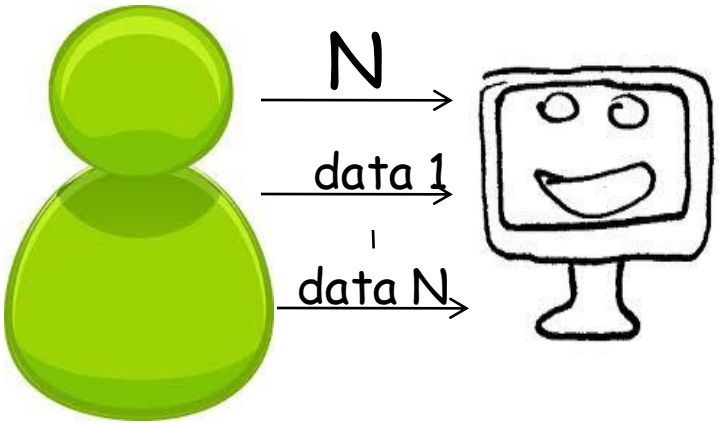
- it prints an error message only if user `number` is negative
- it returns `number` only if non-negative

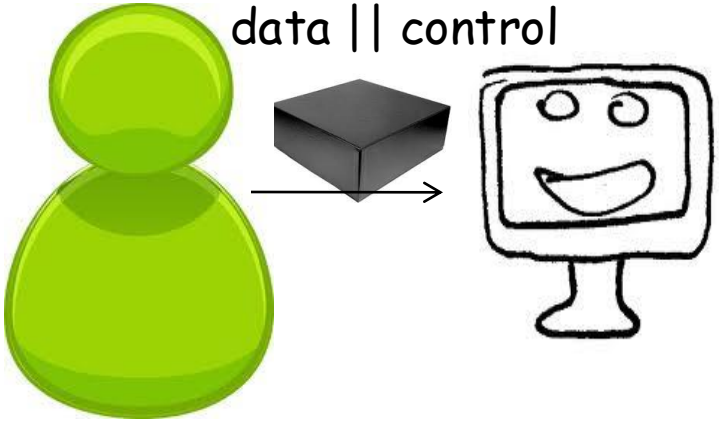
Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing

User Input Protocol

- Two input styles
  - Header control protocol
    - User first specifies the number of data items (e.g., TaskMan)
  - In-band control protocol
    - User finishes input by entering a sentinel value
      - e.g., -1 to signal the end of input grades; "quit" to finish the program
    - Why in-band sentinel: flexibility.
    - Complexity of in-band sentinel: a data item just read can be either a real data item or the signaling sentinel





Sentinel Values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for exam grade until the user types -1, then outputs the average grade.
  - (In this case, the value -1 is the sentinel value.)

```
Type a grade (-1 to exit): 100
Type a grade (-1 to exit): 90
Type a grade (-1 to exit): -1
The average is 95.
```

Design question: for, while, or do loop?

Potential Solution

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;

do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();

  sum += grade; count ++;
} while (grade != -1);

System.out.println(”The average is “ +
                   (count > 0? sum /count : 0) );
```

Potential Solution: Test

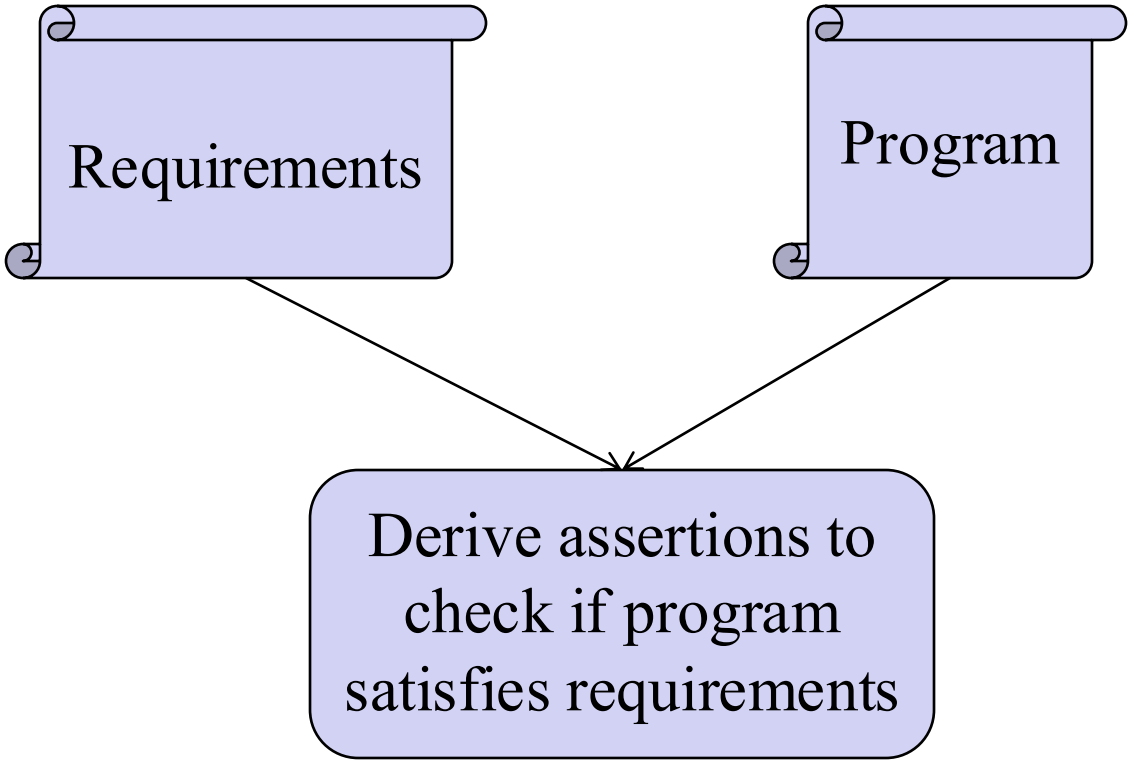
```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();

  sum += grade; count ++;
} while (grade != -1);

System.out.println(”The average is “ +
                    (count > 0? sum /count : 0) );
```

```
Type a grade (-1 to exit): 100
Type a grade (-1 to exit): 90
Type a grade (-1 to exit): -1
```

Output: `The average is 63`

The output is incorrect!

Program Analysis



Requirements Specification

The program reads in a sequence of grades, adds each non-sentinel grade to sum/count, stops when sees sentinel.

reads in a non-sentinel `grade` => add it to sum/count

reads in sentinel `grade` => not add to sum/count; read loop must stop

Equivalent cond for safety:

add to sum/count => grade is not sentinel

Program Analysis

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  // assertion grade != -1 must be established
  sum += grade; count ++;
} while (grade != -1);

System.out.println("The average is " +
                    (count > 0? sum /count : 0) );
```

Condition to guarantee safety:

add to sum and count => grade != -1

Program Revision to Establish Assertion

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;
  }
} while (grade != -1);

System.out.println(”The average is “ +
                    (count > 0? sum /count : 0) );
```

Version 2: do/while -> while

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
while (grade != -1) {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;

  }
}

System.out.println("The average is " +
                    (count > 0? sum /count : 0) );
```

Final Version 2: do/while -> while

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade = 0; // any value other than sentinel
while (grade != -1) {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;

  }
}

System.out.println(”The average is “ +
                    (count > 0? sum /count : 0) );
```

Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing
    - Basic design using assertion to view and fix bug
    - An alternative view of the bug: a design pattern

A "Simpler" Problem…

- Revisit the `countDown` method that prints from a given maximum (>=1) to 1, separated by commas.

  For example, the call:

  ```
  countDown(5)
  ```

  should print:

  ```
  5, 4, 3, 2, 1
  ```

Previous "Solution"

5, 4, 3, 2, 1

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(i + ", ");
    }
    System.out.println();  // to end the line of output
}
```

- Output from `countDown(5)`: 5, 4, 3, 2, 1,

Previous "Solution"

5, 4, 3, 2, 1

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(i + ", ");
    }
    System.out.println();  // to end the line of output
}
```

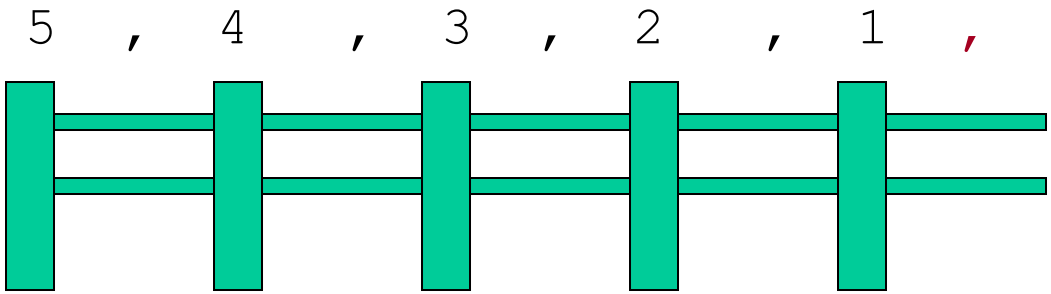
- Output from `countDown(5)`: 5, 4, 3, 2, 1,

```
public static void countDown(int max) {
    for (int i = max; i >= 1; i--) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line of output
}
```

- Output from `countDown(5)`: , 5, 4, 3, 2, 1

Problem: Fence Post Analogy

- We print *n* numbers but need only *n* - 1 commas.

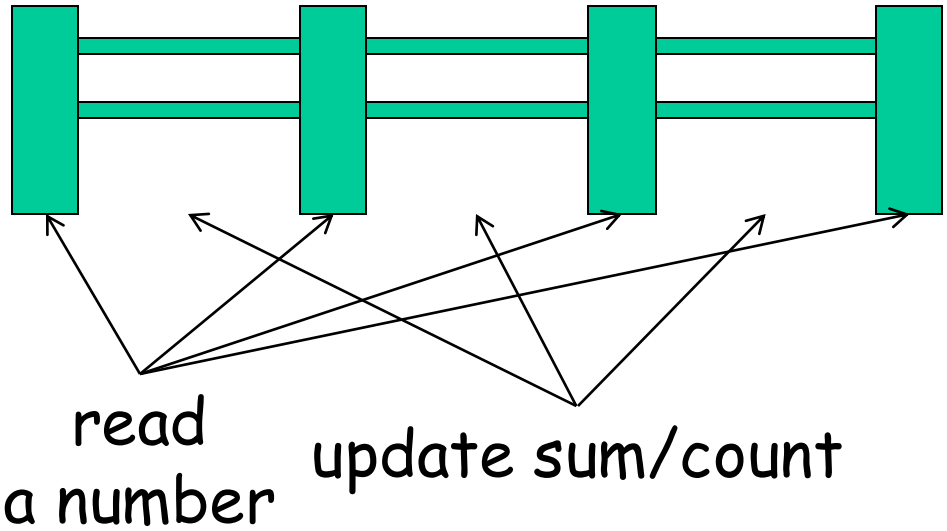


- If we use a loop algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

```
loop (length of fence-wire pair) {
    place a post.        // e.g., <number>
    place some wire.     // e.g., ,
}
```

Problem: Fence Post Analogy

The sentinel input loop pattern is also a fencepost problem: Must read *N* grades, but sum/count only the first *N*-1.



Problem: Fence Post Analogy

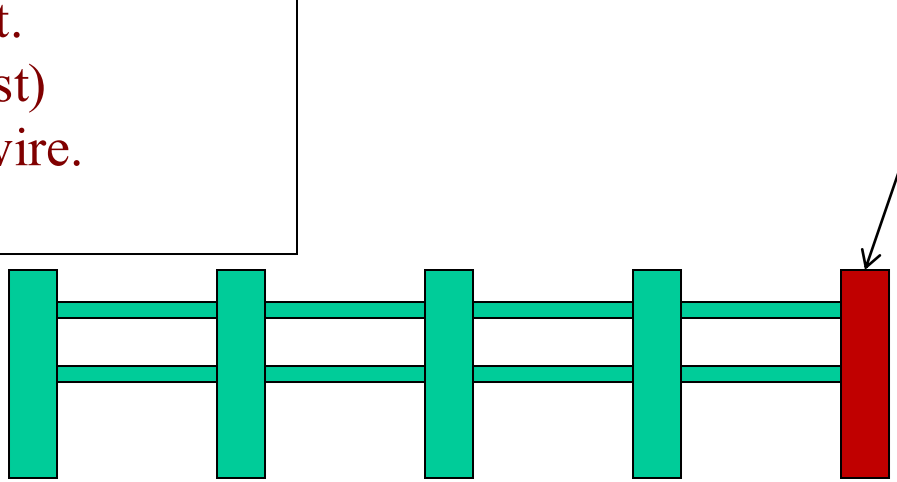
The sentinel input loop pattern is also a fencepost problem: Must read *N* grades, but sum/count only the first *N*-1.

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();


  sum += grade; count ++;

} while (grade != -1);

System.out.println(”The average is “ +
                   count > 0? sum /count : 0);
```

Solve the Fencepost Problem: Design I

```
loop (length of fence) {
    place a post.
    if (! last post)
        place a wire.
}
```

Detect if it is the last post in the loop, if it is, do not place the wire



Revisit Previous Program

We test grade != -1 twice

```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {
      sum = sum + grade;
    }
}
```

Q: What does the if() do?

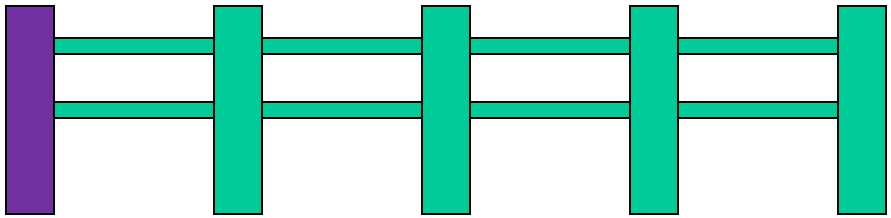
Alternative: Sentinel Loop with `break`

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade == -1)
    break; // break stops the containing loop
  // Do we have assertion grade != -1?
  sum += grade; count ++;

} while (grade != -1);

System.out.println("The average is " +
                    (count > 0? sum /count : 0) );
```

Design Pattern II

- Add a statement outside the loop to place the initial "post." Also called a "loop-and-a-half" solution.

```
place first post.
loop (length of fence - 1) {
    place some wire.
    place a post.
}
```



Fencepost Method Solution

```
public static void countDown(int max) {
    System.out.print(max);     // first post
    for (int i = max-1; i >= 1; i--) {
        System.out.print(", " + i); // wire + post
    }
    System.out.println();      // to end the line
}
```

- Alternate solution: Either first or last "post" can be taken out:

```
public static void countDown(int max) {
    for (int i = max; i >= 2; i--) {
        System.out.print(i + ", "); // post + wire
    }
    System.out.println(1);  // last post
}
```

Fencepost Sentinel Loop: Grade

```
public static final int SENTINEL = -1;
public static final String PROMPT = "Type a grade ("
                                    + SENTINEL + " to exit): ";
public static GradeAnalyzer() {
  Scanner console = new Scanner(System.in);
  int sum = 0; int count = 0;

  // pull one prompt/read ("post") out of the loop
  int grade = getInt(PROMPT, console);

  while ( grade != SENTINEL ) {
    // Do we have assertion grade != -1 before updates?
    sum  += grade; count++;                    // wire
    grade = getInt(PROMPT, console);           // post
  }

  if (count > 0)
     System.out.println("Avg: "  1.0 * sum / count);
}

public static String getInt(String prompt, Scanner console)
{   System.out.print(prompt);
    return console.nextInt();
}
```

Comparison

```
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int grade = console.nextInt();

while ( grade != -1 ) {
    sum = sum + grade;

    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
}
```

Duplicated code

```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
        sum = sum + grade;
    }
}
```

Duplicated code
Duplicate execution

Offline Exercise

- Design loops without duplicates