Lesson 21: Iterative Algorithms #1 (W06D3)

Balboa High School

Michael Ferraro

September 23, 2015

Do Now

For the algorithm below:

1. Write down the predicted output of the execution.
2. Test the code in a new class' `main()` in new project `Lesson21`.

```
int i = 0;
int j = 1;
int a = 10;

while ( i < j ) {
    System.out.println(a);
    i++;
    j++;
    a--;
}
```

Aim

Students will learn to trace iterative algorithms and write iterative algorithms to determine the sum of squares.

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	`i`	`j`	`a` (printed)

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	`i`	`j`	`a` (printed)
0	0	1	10
1	1	2	9

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10
1	1	2	9
2	2	3	8

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10
1	1	2	9
2	2	3	8
3	3	4	7

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10
1	1	2	9
2	2	3	8
3	3	4	7
...	...	...	...

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10
1	1	2	9
2	2	3	8
3	3	4	7
...	...	...	...
50	50	51	-40

Tracing an Iterative Algorithm

Using a table is an excellent way to keep track of what's happening in an iterative algorithm.

Do Now Algorithm: `while(i<j) { i++; j++; a--; }`

iteration	i	j	a (printed)
0	0	1	10
1	1	2	9
2	2	3	8
3	3	4	7
...	...	...	...
50	50	51	-40
∞	inifinite loop!		

What is +=?

You've seen "+=" before, but what does it do?

- Ex. 1:

```
String s = "Hello there...";
s += "Am I late?";
System.out.println(s);
```

What is +=?

You've seen "+=" before, but what does it do?

- Ex. 1:

```
String s = "Hello there...";
s += "Am I late?";
System.out.println(s);
```

→ `Hello there...Am I late?`

What is +=?

You've seen "+=" before, but what does it do?

- Ex. 1:

```
String s = "Hello there...";
s += "Am I late?";
System.out.println(s);
```

→ Hello there...Am I late?

- Ex. 2:

```
int a = 9;
a += 5;
System.out.println("a is " + a);
```

What is +=?

You've seen "+=" before, but what does it do?

- Ex. 1:

```
String s = "Hello there...";
s += "Am I late?";
System.out.println(s);
```

→ `Hello there...Am I late?`

- Ex. 2:

```
int a = 9;
a += 5;
System.out.println("a is " + a);
```

→ `a is 14`

What is +=?

You've seen "+=" before, but what does it do?

- Ex. 1:

```
String s = "Hello there...";
s = s + "Am I late?";
System.out.println(s);
```

→ `Hello there...Am I late?`

- Ex. 2:

```
int a = 9;
a = a + 5;
System.out.println("a is " + a);
```

→ `a is 14`

Sum of ints

- Download the sum_of_ints Scratch program, save to desktop
- Start cloud-based Scratch:
  https://scratch.mit.edu/projects/editor/
- File → Upload from your computer...,
  choose the downloaded program file
- Click the green flag at the top right to start the program
- The intended result: sum = 1 + 2 + 3 = 6
- Fix the error!

Sum of `ints`

- Let's modify the program so that

  `sum = 1 + 2 + 3 + ... + 10`

Sum of `ints`

- Let's modify the program so that

  `sum = 1 + 2 + 3 + ... + 10`

- How about a program that computes `sum` in this manner —

  `sum = 1 + 2 + 3 + ... +` n

  — where n is chosen by the user?

Sum of Squares

- Determine the sum of squares from 1 to `n`:

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + ... + n^2$$

[1] condition we require be true before an algorithm runs

Sum of Squares

- Determine the sum of squares from 1 to `n`:

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + ... + n^2$$

- For example:
  - $n = 1$: $1^2 = \mathbf{1}$

[1] condition we require be true before an algorithm runs

Sum of Squares

- Determine the sum of squares from 1 to `n`:

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + ... + n^2$$

- For example:
  - $n = 1$: $1^2 = \mathbf{1}$
  - $n = 2$: $1^2 + 2^2 = \mathbf{5}$

[1] condition we require be true before an algorithm runs

Sum of Squares

- Determine the sum of squares from 1 to `n`:

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + ... + n^2$$

- For example:
  - $n = 1$: $1^2 = \mathbf{1}$
  - $n = 2$: $1^2 + 2^2 = \mathbf{5}$
  - $n = 3$: $1^2 + 2^2 + 3^2 = \mathbf{14}$

[1] condition we require be true before an algorithm runs

Sum of Squares

- Determine the sum of squares from 1 to `n`:

$$\sum_{a=1}^{n} a^2 = 1^2 + 2^2 + ... + n^2$$

- For example:
  - $n = 1$: $1^2 = \mathbf{1}$
  - $n = 2$: $1^2 + 2^2 = \mathbf{5}$
  - $n = 3$: $1^2 + 2^2 + 3^2 = \mathbf{14}$
- **precondition**[1] for algorithm: $n \geq 1$

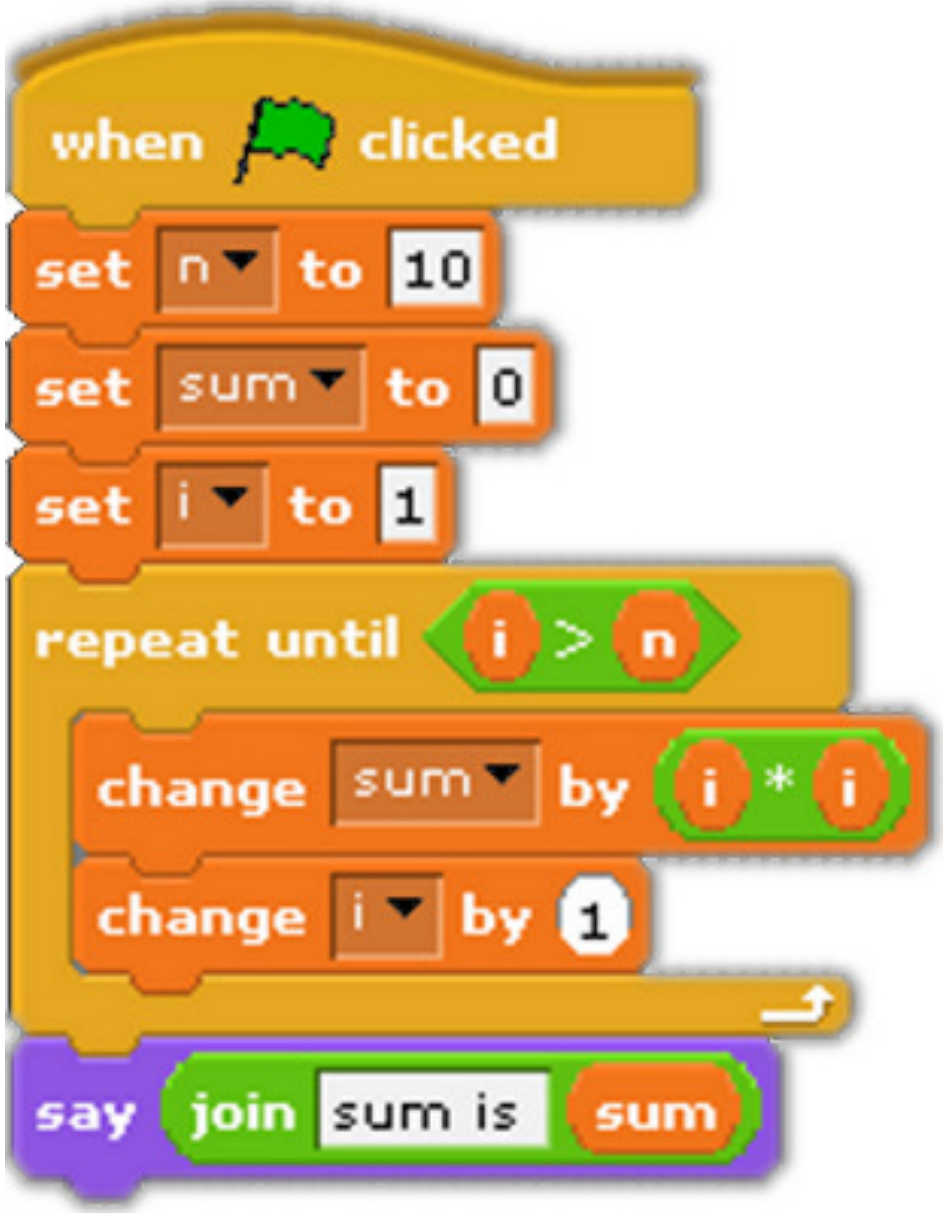
[1] condition we require be true before an algorithm runs

Sum of Squares

How can our Scratch program be modified to compute a sum of squares from 1 to n?

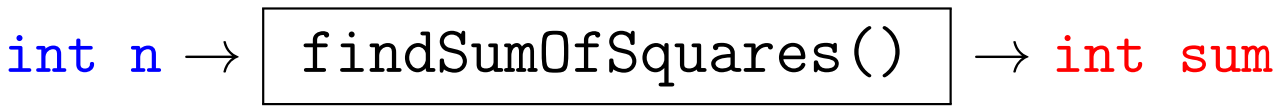
Sum of Squares

Let's analyze the design of this algorithm so that we may translate it into Java:



Sum of Squares: Plan of Attack

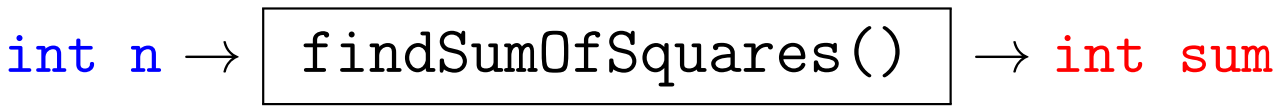
Viewed as a procedure, with input and output values:



[2] Ask about `static` later.

Sum of Squares: Plan of Attack

Viewed as a procedure, with input and output values:



Viewed as a Java method signature: [2]

```
public static int findSumOfSquares(int n) {
   ...
}
```

[2] Ask about `static` later.

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total
  - stop looping after we've added the n^{th} term

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total
  - stop looping after we've added the n^{th} term
- Return that "running total"

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total
  - stop looping after we've added the n^{th} term
- Return that "running total"

```
public static int findSumOfSquares(int n) {

    ...

}
```

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total
  - stop looping after we've added the n^{th} term
- Return that "running total"

```
public static int findSumOfSquares(int n) {
    int sum = 0; // accumulator var
                 // (running total)
    ...

}
```

Sum of Squares: Plan of Attack

- How can we *build up* the sum over some iterations of a loop?
  - each iteration, add another term (e.g., 3^2) to our running total
  - stop looping after we've added the n^{th} term
- Return that "running total"

```
public static int findSumOfSquares(int n) {
    int sum = 0; // accumulator var
                 // (running total)
    ...

    return sum;
}
```

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    ...
    return sum;
}
```

- We need a loop!

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator

    while ( ? ) {
        ...
    }

    return sum;
}
```

- We need a loop!

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator

    while ( ? ) {
        ...
    }

    return sum;
}
```

- We need a loop!
- Need to keep track of current term's base (i.e., the number to be squared)

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( ? ) {
        ...
    }

    return sum;
}
```

- We need a loop!
- Need to keep track of current term's base (i.e., the number to be squared)

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( <stopping condition> ) {
        ...
    }

    return sum;
}
```

- Your task:
  - figure out the *stopping condition*

Sum of Squares: Plan of Attack

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( <stopping condition> ) {
        <tasks for each iteration>
    }

    return sum;
}
```

- Your task:
  - figure out the *stopping condition*
  - come up with what *each iteration* should do

Steps for Building `SumOfSquares`

1. Create class `SumOfSquares` in `Lesson21`, include a `main()` method
2. Insert the code for the `findSumOfSquares()` method below — *but not inside* `main()`*!*
3. Come up with *stopping condition* and code for `while()` loop
4. Have `main()` call `findSumOfSquares()` with different values of n, printing out the result; e.g., `System.out.println(findSumOfSquares(5));`

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( ... ) {
        ...
    }

    return sum;
}
```

Sum of Squares: One Solution

```
public static int findSumOfSquares(int n) {
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( i <= n ) {
        sum += i*i;
        i++;
    }

    return sum;
}
```

Sum of Squares: Solution w/Comments

```
/**
  * This procedure returns sum of squares, from
  * 1^2 to n^2; for example, when n=4, return 30:
  * 1^2 + 2^2 + 3^2 + 4^2  =  1 + 4 + 9 + 16  =  30
  *
  * @param n highest pos. integer whose square is included
  * @return sum of squares
  */
public static int findSumOfSquares(int n) {
    //precondition:  n > 0
    int sum = 0; //accumulator
    int i = 1;   //term to be squared

    while ( i <= n ) {
        sum += i*i;
        i++;
    }

    return sum;
}
```

HW

- Make sure you finish the `SumOfSquares` class and that it works; get help if necessary!
- Complete PS #4a, §4, #1: Copy the flowchart for Euclid's GCF[3] algorithm from Litvin.

[3] Greatest Common Factor