CSC 1051 Algorithms & Data Structures I

Data, Variables & Expressions

DATA & VARIABLES

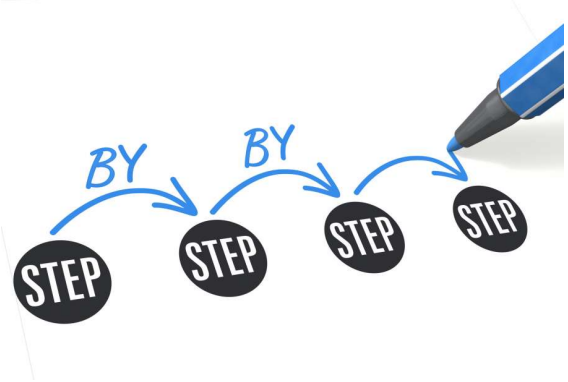
What is an Algorithm?

An algorithm is a step-by-step procedure for solving a problem.

It's a set of instructions designed to perform a specific task.

How an algorithm works:

- You're in some initial state (the input)
- Do something with it (the algorithm)
- The result comes out (the output)



Everyday Algorithms

Some everyday things that are algorithms:

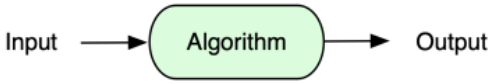
- Baking a cake
- Reading a book
- Planning a party
- A puppy fetching a ball
- Finding something to watch on Netflix
- Driving from home by the shortest route
- Making a peanut butter and jelly sandwich

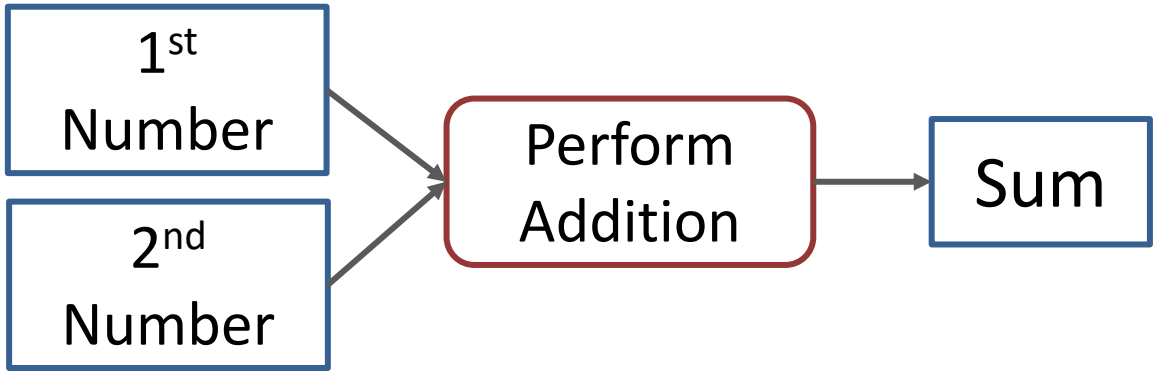
Try writing down the algorithm steps yourself!

Try It: Simple Calculator Algorithm

Design the algorithm for a two number addition calculator.

Draw a flowchart inspired by the generic algorithm flowchart:





How does the computer remember these numbers?

Variables

A variable is a name used to refer to data stored in memory

A Java variable must be declared before it can be used

A variable declaration establishes the variable's data type and may initialize the value

```
int count;
int index = 1;
int quantity, minimum = 0, result = -1;
```

The data type of these variables is int, which means they each store an integer value

Variables

A variable name, like any other name you make up in a program, is an identifier

Identifier names can be composed of letters, digits, the underscore character (_) and the dollar sign ($)

An identifier cannot begin with a digit

Variable names should be written in so-called camelCase

```
currentScore
finalLetterGrade
```

Variables

There are eight Java primitive data types

4 integer types:	byte, short, int, long
2 floating-point types:	float, double
1 character type:	char
1 boolean type:	boolean

The numeric types differ by how much memory they use, which dictates the range of values they can store

```
int count = 5;
double price = 2.99;
char initial = 'K';
boolean flag = true;
```

Assignment Statements

An assignment statement stores a value in a variable using the assignment operator (=)

The right-hand side of the assignment operator can be a simple value or an expression

```
sum = 0;
capacity = 100;
area = length * width;
max = measuredValue + delta;
```

A variable can appear on both sides of the assignment operator

```
count = count + 1;
capacity = capacity * 2;
```

Assignment Statements

The result of the expression must be type compatible with the variable to which it's assigned

For example, you can't assign a boolean value to an integer, or vice versa

Numeric values can be assigned if there is no risk of losing information

```
short shortVal = 1000;
int num = shortVal;
double amount = i;
```

Assignment Statements

Converting to a larger type is called a widening conversion – going the other way is a narrowing conversion

To perform a narrowing conversion, you have to use type casting

A cast is expressed as a type within parentheses, placed in front of the value to be converted

```
double amount = 138.756;
int num = (int)amount;
```

A cast explicitly causes a value of one type to be treated like another, even if data is lost

That assignment stores the value 138 in the variable num, but the value in amount remains the same

Assignment Statements

Casting is powerful and should be used with care

But it's very helpful at times

If two integers are divided, the result is an integer (the fractional part is discarded)

If you want the fractional part, you can cast one of the operands as a double

```
average = (double)sum / count;
```

The cast causes the value of sum to be treated as a double for the purposes of the expression

Constants

A constant is similar to a variable, except that its value cannot be changed

```
final int PINTS_PER_GALLON = 8;
```

Once it has been given a value, the compiler will complain if later code attempts to change it:

```
PINTS_PER_GALLON = 12;
```

By convention, constant names are in all UPPERCASE LETTERS with words separated by underscores, so they are OBVIOUS.

Constants

Three reasons to use constants:

1. Constants convey more meaning that literals

   MAX_OCCUPANCY vs. 650

2. They prevent inadvertent programming errors

   A change to the value must be explicit

3. They make maintenance tasks easier

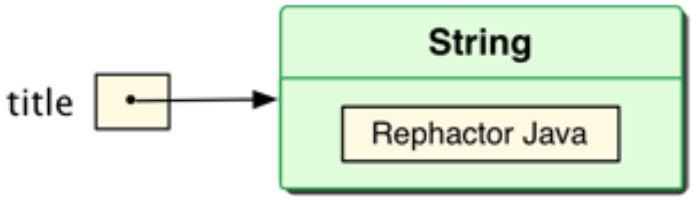
   If the value does change, it only needs to be changed in one place

Strings

A character string is a group of ordered characters

Character strings are objects in Java, defined by the String class

So a String variable is a reference to an object



The String class is part of the java.lang package, so does not need to be imported

Strings

Strings can be created with the new operator, but a double-quoted string literal is already an object

```
String name = new String("James Gosling");
String title = "Rephactor Java";
```

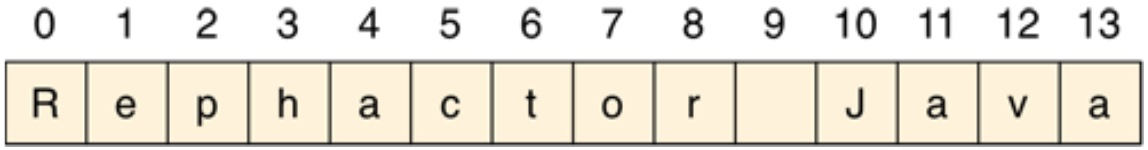
The plus operator (+) is used to perform string concatenation

```
System.out.println("Without " + name +
    ", there would be no " + title + ".");
```

```
Without James Gosling, there would be no Rephactor Java.
```

Strings

Strings are managed so that you can refer to individual characters by a numeric index, which starts at 0



The String class has many methods to help manage strings

For example, the length method returns the number of characters in the string (14 in this case)

The charAt method returns the character at a particular index

```
int num = title.length();
char letter = title.charAt(10);
```

Strings

The substring method returns a new String that contains a subset of characters copied from another string

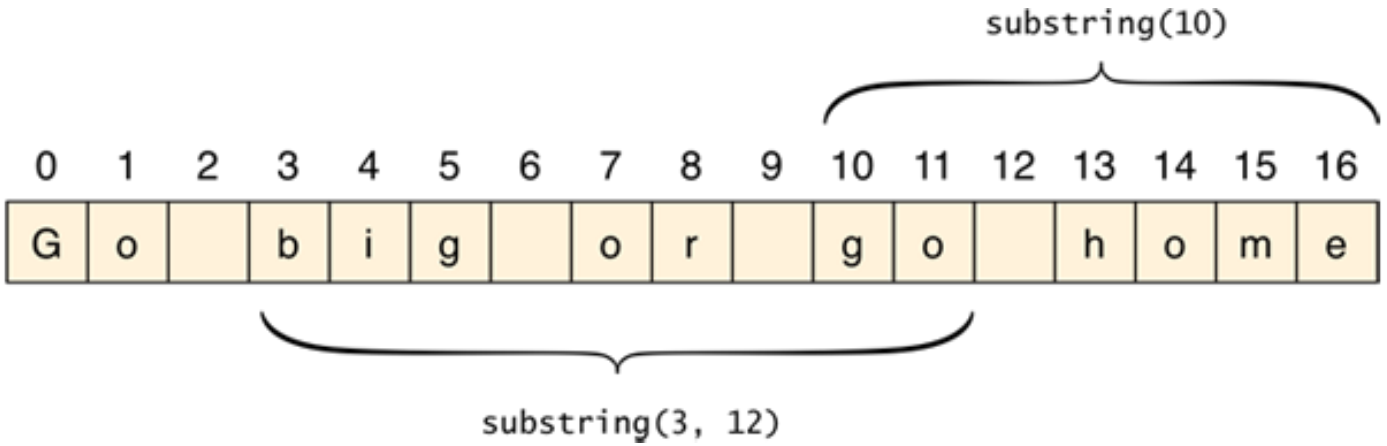
There are two versions: one that takes one index as an argument and one that takes two

```
str.substring(5)
str.substring(7, 12)
```

If one index is specified, the substring runs from that index to the end of the string

If two indexes are specified, the substring runs from the first index up to *but not including* the second index

Strings




```
String goBig = "Go big or go home";
String sub1 = goBig.substring(10);
String sub2 = goBig.substring(3, 12);

System.out.println("Original string: " + goBig);
System.out.println("substring(10): " + sub1);
System.out.println("substring(3, 12): " + sub2);
```

The print and println Methods

The System.out object prints output to the console window

The println method moves to the next line after it prints its output

The print method does not

```
System.out.println("One");
System.out.print("Two");
System.out.println("Three");
```

```
One
TwoThree
```

The print and println Methods

```
System.out.print("One, ");
System.out.print("Two, ");
System.out.println("Buckle my shoe.");
System.out.println();
System.out.print("Three, ");
System.out.print("Four, ");
System.out.println("Close the door.");
```

No argument required

```
One, Two, Buckle my shoe.

Three, Four, Close the door.
```

Result: blank line

The print and println Methods

The print and println methods can print any type of data

```
System.out.println(25);
```

This is an example of method overloading – a method accepting different types of data

Expressions in the arguments are evaluated and the results are sent to the method

```
System.out.println(38 + 31);
```

```
69
```

The print and println Methods

Character strings cannot be broken across lines:

```
System.out.println("No word in the English language
rhymes with the words month or orange");
```

The plus sign can be used to perform string concatenation

```
System.out.println("No word in the English language " +
"rhymes with the words month or orange");
```

The two strings are joined into one long string which is passed to the method

The print and println Methods

The plus sign is an overloaded operator – it operates on different types of data

It determines which operation to perform based on the types of its operands

The plus operator is evaluated left to right

```
System.out.println("Concatenated: " + 123 + 456);
```

```
Concatenated: 123456
```

```
System.out.println("Added: " + (123 + 456));
```

```
Added: 579
```

Escape Sequences

An escape sequence is a technique for representing a character

It's used when the traditional representation is problematic or less convenient

For example, suppose you wanted to print double quotes as part of your output

```
"Tell the truth and then run."
```

You can't just include them in the string – that will confuse the compiler

```
System.out.println(""Tell the truth and then run."");
```

Escape Sequences

An escape sequence begins with a backslash (\), which tells the compiler to treat what follows in a special way

To represent a double quote, use the \" escape sequence

```
System.out.println("\"Tell the truth and then run.\"");
```

```
"Tell the truth and then run."
```

An escape sequence can be used wherever needed

```
System.out.println("She said \"Hi\" to me.");
```

```
She said "Hi" to me.
```

Escape Sequences

It's often convenient to include a newline character in a string, which is represented with the \n escape sequence

It causes output to move to the next line

```
System.out.println("One\nTwo\nThree");
```

```
One
Two
Three
```

```
System.out.println("Batman\n\nRobin");
```

```
Batman

Robin
```

Escape Sequences

Summarizing the Java escape sequences:

Escape Sequence	What it Represents
\"	double quote
\'	single quote
\\	backslash
\t	horizontal tab
\n	newline
\b	backspace
\r	carriage return
\f	form feed
\uXXXX	a Unicode character

ARITHMETIC EXPRESSIONS

Numeric Expressions

An expression is a combination of one or more operators and operands that typically perform a calculation.

The operands used in the operations might be literals, constants, variables, or other sources of data.

```
int result = 14 + 8 / 2;
```

numeric literals and arithmetic operators

Numeric Expressions

Basic Java arithmetic operators

Operator	Name	Example	Result
+	Addition	25 + 17	42
–	Subtraction	18.92 – 12	6.92
*	Multiplication	5 * 7.3	36.5
/	Division	7.65 / 3.4	2.25
%	Remainder	15 % 6	3

Operands and operators combine to form potentially complex expressions

Numeric Expressions

If either or both of operands to the division operator (/) are floating-point values, the result is a floating-point value

But it performs integer division if both operands are integers

The result is an integer and any fractional part is discarded

5.0 / 2.0 → 2.5

5 / 2 → 2

The remainder operator (%) computes the remainder left over after dividing one operand into another

25 % 8 → 1

25 % 10 → 5

Numeric Expressions

Integer division and remainder

a	b	a / b	a % b
10	5	2	0
7	4	1	3
4	7	0	4
-5	2	-2	-1
5	-2	-2	1
-5	-2	2	-1
5346	7	763	5

The result of the remainder operator takes the sign of the dividend (the first operand)

Numeric Expressions

One number is evenly divisible by another if the remainder is 0

```
if (total % 5 == 0)
    System.out.println("evenly divisible by 5");
```

So, to determine if a number is even or odd:

```
if (num % 2 == 0)
    System.out.println(num + " is even");
else
    System.out.println(num + " is odd");
```

Numeric Expressions

Integer division and remainder often work well together

```
int seconds = 2172;

int mins = seconds / 60;
int secs = seconds % 60;

System.out.println(seconds + " seconds is " + mins
    + " minutes and " + secs + " seconds.");
```

```
2172 seconds is 36 minutes and 12 seconds.
```

Increment and Decrement Operators

Incrementing the value of a variable is normally done like this:

```
count = count + 1;
```

With the increment operator, this does exactly the same thing:

```
count++;
```

The decrement operator subtracts 1 from the variable:

```
count--;
```

Increment and Decrement Operators

Increment and decrement operators have two forms:

The postfix form uses the value before it increments it. If count is 15, after the assignment total is 15 and count is now 16.

```
total = count++;
```

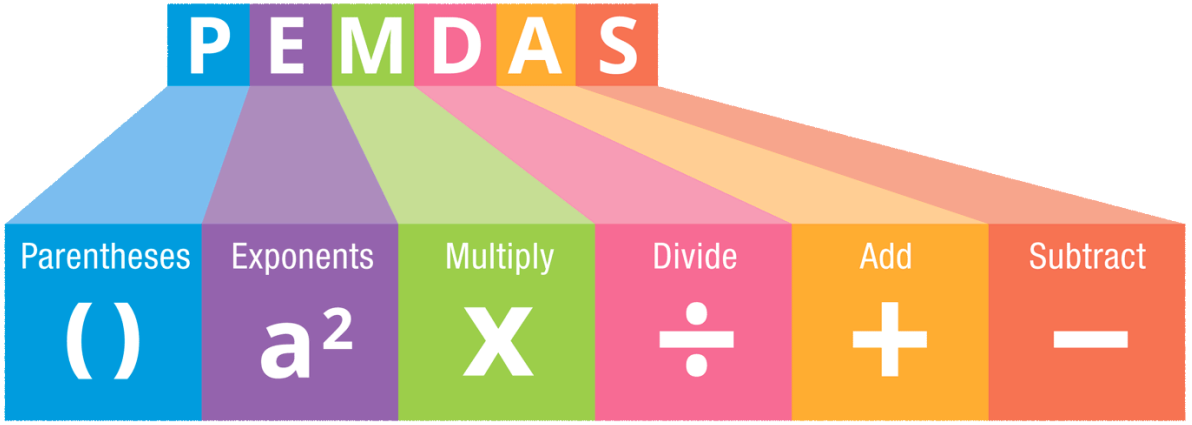
The prefix form increments the value first and then uses it. If count is 15, after the assignment total is 16 as is count.

```
total = ++count;
```

Always be careful when using increment and decrement operators. They can be very concise **and** very tricky.

Operator Precedence

The order in which operations are performed relies on their precedence.



In grade school, you may have learned the PEMDAS mnemonic for remembering order of operations. This approach is the inspiration for how programming languages handle order of operations… but it's a little more complicated.

Precedence and Associativity

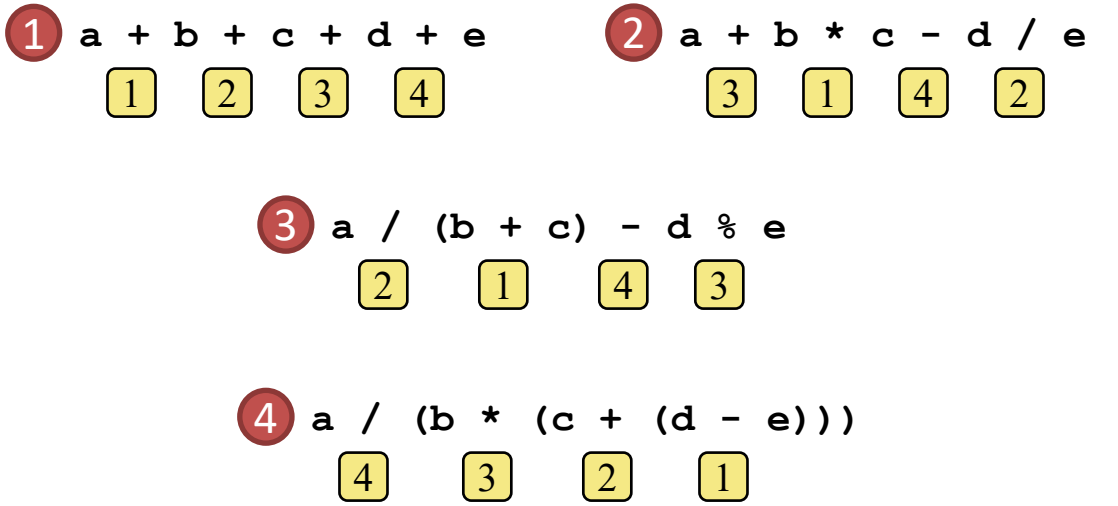
This is the first row from the Java precedence table. In addition to parentheses (the P in PEMDAS), also at this top level are of few other operators.

Precedence	Operator	Operation	Associativity
1	[] . () ++ --	array index member access method call postfix increment, decrement	Left

In addition to precedence, the table shows associativity. That's how you know whether operators with the same precedence are evaluated from left-to-right or right-to-left.

Operator Precedence - Example

In what order are the operators evaluated in the following expressions?



Assignment Operator Precedence

The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4     1   3     2
```

Then expression result is stored in the variable on the left-hand side

Shortcut Assignment Operators

It's common to update a variable's value using its current value:

```
balance = balance + deposit;
```

Java provides a shortcut operator that combines the calculation and the assignment:

```
balance += deposit;
```

Those two statements are functionally equivalent – they accomplish the same thing

Shortcut Assignment Operators

There are shortcut operators corresponding to each arithmetic operator:

Operator	Example	Equivalent To
+=	total += 6.98;	total = total + 6.98;
-=	gap -= step;	gap = gap - step
*=	capacity *= 2;	capacity = capacity * 2;
/=	max /= factor;	max = max / factor;
%=	index %= length;	index = index % length;

The left hand variable is always on the left-hand side of the operator in the expanded expression

Shortcut Assignment Operators

The right-hand side doesn't have to be a single value – it could be a more complex expression:

```
x *= y + z / 2;
```

The entire right-hand expression is evaluated, then the shortcut operator is applied

So that statement is equivalent to this:

```
x = x * (y + z / 2);
```

INTERACTIVE PROGRAMMING

Interactive Programs

To be really useful, a program should be able to interact with a user, or it may do the same thing every time!

An interactive program accepts input directly from the user and does something in response.

Reading Input

A Scanner object is used to read and parse input in a program. It typically reads data from a keyboard or file.

The Scanner class is part of the java.util package in the Java API. To use it, you should include this import statement:

```
import java.util.Scanner;
```

Then, to create the Scanner object, use the new operator:

```
Scanner scan = new Scanner(System.in);
```

Interactive Input

Here's how to use a Scanner to read in a String and an integer:

```
Scanner scan = new Scanner(System.in);

System.out.print("Who are you? ");
String name = scan.nextLine();

System.out.print("How many fingers do you see? ");
int count = scan.nextInt();

System.out.println();
System.out.println("You say you are " + name);
System.out.println("I held up " + count + " fingers.");
```

Explore the Scanner Class Rephactor topic to discover the many ways to use this powerful and versatile feature of Java.