Review of Programming & Mathematical Concepts

Computational Thinking with Algorithms

Roadmap

- Mathematical operators
- Order of operations
- Variables & data types
- Common operators in programming
- Functions/methods/procedures
- Control structures
- Data structures

Mathematical operators

Operator	Description	Examples
+	Additive operator (also string concatenation)	2 + 1 = 3 “abc” + “_” + 123 = “abc_123”
-	Subtraction operator	25 – 12 = 13
*	Multiplication operator	2 * 25 = 50
/	Division operator	35 / 5 = 7
%	Remainder operator	35 % 5 = 0, 36 % 5 = 1

Order of operations - BEMDAS

- Brackets
- Exponents
- Multiplication
- Division
- Addition
- Subtraction

Multiplication/division and addition/subtraction may always be worked out in the same step

Exponents

- Exponents indicate that a quantity is to be multiplied by itself some number of times
- In general: x^n specifies that a number x is to be multiplied by itself n times
- 4^3 (pronounced "4 to the power of 3"), therefore evaluates to:
  - 4^3 = 4 * 4 * 4 = 64

Order of operations example

Evaluate: 4^3 * (-2 / 4 + 3 * 4), following BEMDAS

- Step 1: 4^3 * (-0.5 + 12)
- Step 2: 4^3 * 11.5
- Step 3: 64 * 11.5
- Step 4: 736

Variables

- A variable is simply a storage location and associated name which we can use to store some information for later use
- Some different types of variables:
    - Integer (whole numbers e.g. 1)
    - Floating point (real numbers e.g. 1.123543)
    - String (a collection of characters e.g. “test”)

Data types

- Numeric data
  - Integers, i.e. whole numbers, e.g. 1, 0, -127
  - Floating point, i.e. real numbers, e.g. 2.12, 3.1415, -127.01
- Character data
  - Can contain any valid character symbols, e.g. @, !, 4, K, abcd1234
  - String data type in Java
  - Enclosed in quotes, e.g. “the quick brown fox jumped over the lazy dog”
- Boolean data
  - true or false

Strongly and weakly typed

- Programming languages are often classified as being either **strongly typed** or **weakly typed**
- **Strongly typed** languages will generate an error or refuse to compile if the argument passed to a function does not closely match the expected type
- **Weakly typed** languages may produce unpredictable results or may perform implicit type conversion if the argument passed to a function does not match the expected type

Strongly and weakly typed

Java

```
int myInt = 2;

float myFloat = 2.3456f;

String myString = “test”;
```

JavaScript

```
var myInt = 2;

var myFloat = 2.3456f;

var myString = “test”;
```

Common operators in programming

Operator	Description	Examples
=	Assignment operator	int number = 23; string myWord = “apple”;

++	Increment operator; increments a value by 1	int number = 23;
		number++;
		System.out.println(number); // prints 24

--	Decrement operator; decrements a value by 1	int number = 23;
		number--;
		System.out.println(number); // prints 22

+=	Assignment
	(shorthand for number = number + value)
		int number = 23;
		number += 2;
		System.out.println(number); // prints 25

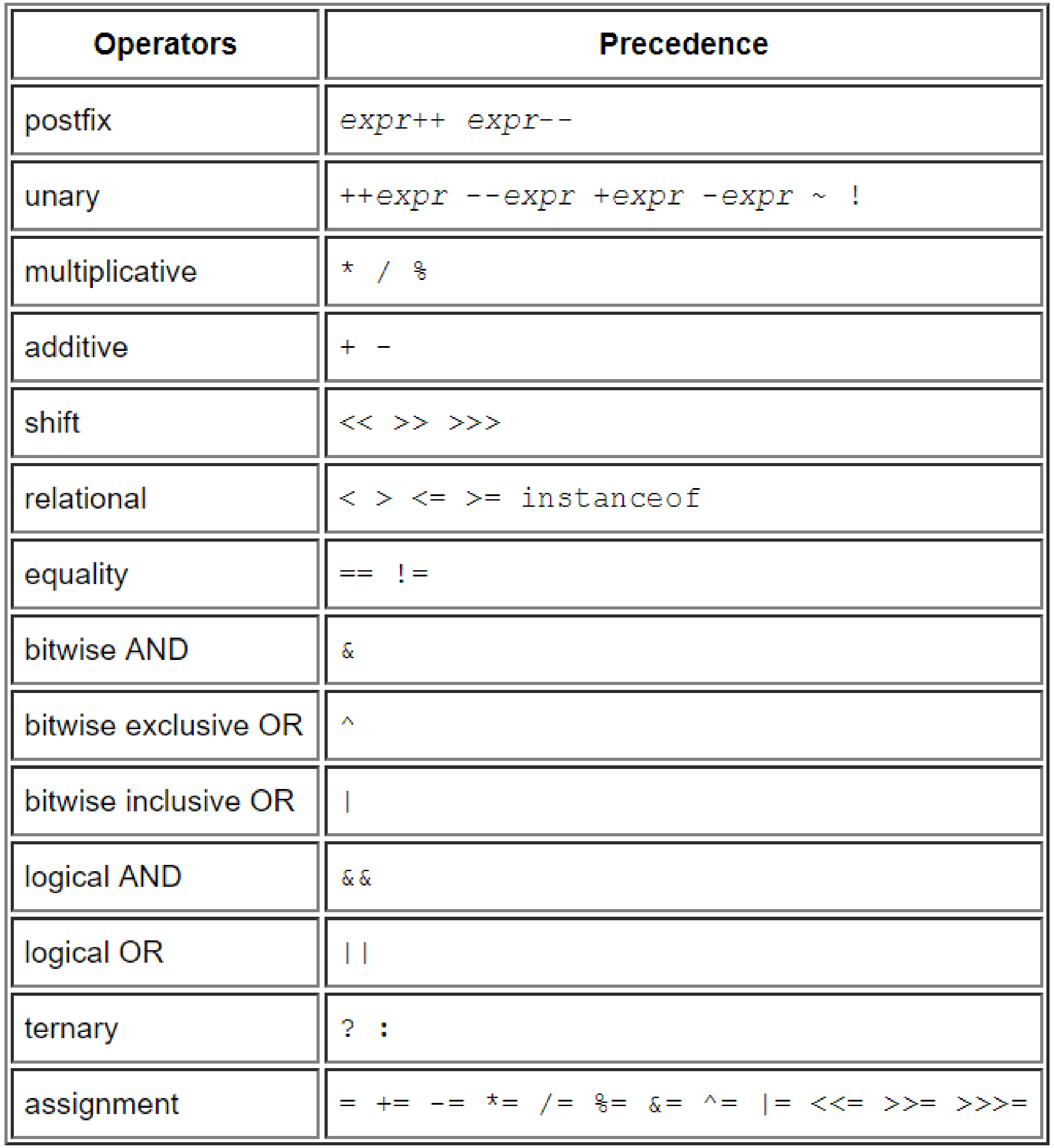
-=	Assignment
	(shorthand for number = number - value)
		int number = 23;
		number -= 2;
		System.out.println(number); // prints 21

Common operators in programming

Operator	Description	Examples
==	Equality, tests if two values are equal	System.out.println(2==1); // prints false
!=	Equality, tests if two values are not equal	System.out.println(2!=1); // prints true
&&	Logical AND	System.out.println(2==1 && 1==1); // prints false
||	Logical OR	System.out.println(2==1 || 1==1); // prints true
!	Logical complement operator, inverts the value of a boolean	boolean success = false; System.out.println(!success); // prints true
>	Relational, greater than	System.out.println(1>1); // prints false
>=	Relational, greater than or equal to	System.out.println(1>=1); // prints true
<	Relational, less than	System.out.println(1<1); // prints false
<=	Relational, less than or equal to	System.out.println(1<=1); // prints true

Operator precedence in Java

Source: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html



Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Functions

- A function is a block of code designed to perform a particular task
- A function is executed when "something" invokes it (calls it)
- C/C++/JavaScript – function
- Java/C# – method
- Psuedocode – procedure

Functions

Java

```java
int myMethod (int p1, int p2) {
   return p1*p2;
   // This method returns the product of p1 and p2
}
```

JavaScript

```javascript
function myFunction(p1, p2) {
   return p1 * p2;
   // This function returns the product of p1 and p2
}
```

Control Structures

- Sequential
- Selection
- Iteration

Sequential

- This is the default control structure
- Statements are executed line by line in the order that they appear

Selection

- Selection statements allow different blocks of code to be executed based on some condition
- Examples:
    - if
    - if/else if/else
    - switch

if/else if/else

Java

```
int i=0;
if (i==1) {
   // do something
}
else if (i==2) {
   // do something else
}
else {
   // do something different
}
```

JavaScript

```
var i=0;
if (i===1) {
   // do something
}
else if (i===2) {
   // do something else
}
else {
   // do something different
}
```

Switch

Java

```
int i=0;
switch (i) {
   case 0:
      // do something
      break;
   case 1:
      // do something else
      break;
   default:
      // default code block
}
```

JavaScript

```
var i=0;
switch (i) {
   case 0:
      // do something
      break;
   case 1:
      // do something else
      break;
   default:
      // default code block
}
```

Iteration

- Iteration structures repeatedly execute a series of statements as long as the condition stated in parenthesis is true
- Important to ensure that the loop condition will eventually become false to prevent infinite looping
- Examples of iteration structures:
  - for loops
  - while loops
  - do/while loops

For loop

Java

```
int i=0;
for (i=0; i<10; i++) {
   // do something
}
```

JavaScript

```
var i=0;
for (i=0; i<10; i++) {
   // do something
}
```

For loop (infinite looping)

Java

```
int i=0;
for (i=0; i<10; i--) {
   // do something
}
```

JavaScript

```
var i=0;
for (i=0; i<10; i--) {
   // do something
}
```

While loop

Java

```
int i=0;
while (i<10) {
   // do something
   i++;
}
```

JavaScript

```
var i=0;
while (i<10) {
   // do something
   i++;
}
```

Data structures

- A data structure is a way of storing and organising data in a program
- Example – an array
- Arrays have an **index** which allows us to access the information stored at a particular position in the array
- In most programming languages, arrays are **zero-indexed** (i.e. the first element is indexed with the number 0)
- We will assume that arrays are zero-indexed throughout this course
- Loops are extremely useful when working with arrays

Array example

- If we wanted to store 10 names, we could make 10 different variables to store them, e.g. (in Java):
    - string contact1 = “John Smith”;
    - string contact2 = “Jane Doe”;
    - string contact3 = “Jim Doe”;
    - etc.
- This is a **terrible** way of storing this kind of information
    - We must write a large amount of text, what if we had 1000 names?
    - This code isn’t flexible - we would have to rewrite it if we want to add another name in the future

Array example

- It makes much more sense to use a data structure such as an array to store large quantities of related values
- Depending on the programming language, array like constructs typically have useful built in functions, e.g.
    - Sorting array elements (more on this later!)
    - Counting the number of elements in the array

Array example

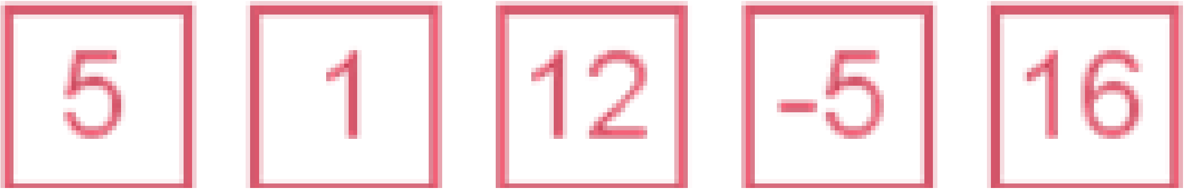
Java

- int [] numbers = new int [] {5,1,12,-5,16};

JavaScript

- var numbers = [5,1,12,-5,16];

Array with 5 elements



5	1	12	-5	16
index 0	index 1	index 2	index 3	index 4