Computer Programming I - CSC111

Chapter 2 – Basic Computation

Dr. Mejdl Safran

mejdl@ksu.edu.sa

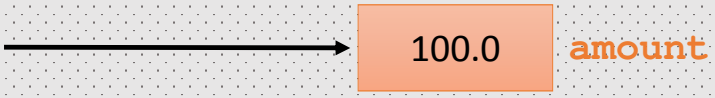
Outline

- Type casting
- Arithmetic operators
- Assignment operators
- Order of precedence for arithmetic operators
- Increment/decrement operators
- Relational operators
- Logical operators
- Order of precedence for arithmetic operators
- Vending machine change example
- The class `String`
- Documentation and style

Assignment compatibilities

- Java is said to be *strongly typed.*
  - You cannot, for example, assign a floating point value to a variable declares to store an integer
- Sometimes conversions between numbers are possible.

```
double amount = 100 ;
```



is possible even if `amount` is of type `double`, for example.

Assignment compatibilities

- A value of one type can be assigned to a variable of any type further to the right

`byte` → `short` → `int` → `long` → `float` → `double`

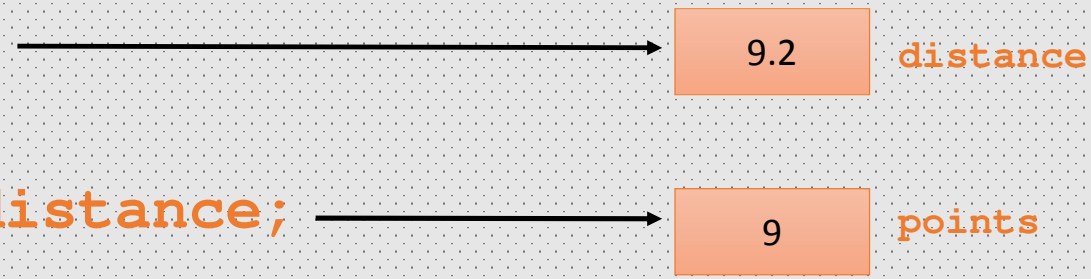
  - But not to a variable of any type further to the left

- You can assign a value of type char to a variable of type `int`

Type casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.
- For example,

```
double distance;
distance = 9.2;
int points;
points = (int)distance;
```



- Illegal without `(int)`

After casting

Java operators

- Operators are special symbols used for:
  - Mathematical functions
  - Assignment statements
  - Logical comparisons
- Examples:
  - 3 + 5
  - 14 + 5 – 4 * (5 – 3)
- Expressions: can be combinations of variables and operators the result in value.
  - `avg = (x1+x2+x3) / 3;`

Group of operators

- Arithmetic operators
- Assignment operator
- Increment/decrement operators
- Relational operators
- Logical operators

Arithmetic operators

- Addition +
- Subtraction −
- Multiplication *
- Division /
- Remainder (Modulus) %

Arithmetic expressions

- They can be formed using *arithmetic operators* together with variables or numbers referred to as *operands*.
  - When both values are of the same type, the result is of that type.
  - When one of the values is a floating-point type and the other is an integer, the result is a floating point type.

Examples

```
int x = 20 , y = 3 ;
double z = 3.0;
```

Java Operator	Example	Value
+	x + y	23
-	x - y	17
*	x * y	60
/	x / y	6
/	x / z	6.666
%	x % y	2

integer division where fractional part is truncated

Arithmetic operations

- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  - The result of one step produces one of the operands to be used in the next step.
- example

```
balance + (balance * rate)
```

Arithmetic operations

- If at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.

- The result is the rightmost type from the following list that occurs in the expression.

```
byte → short → int → long → float → double
```

Division operator

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.

```
10.0 / 3 → 3.33333
```

- When both operands are integer types, the result is truncated, not rounded.

```
10 / 3 → 3
```

Modulus (remainder)

- To obtain the remainder after integer division
- 14 divided by 4 is 3 *with a remainder of 2.*
  - *Hence,* `14 % 4` *is equal to* `2`

```
5 % 2 → 1 ;   5 % 5 → 0 ;      5 % 10 = 5
```

- Most commonly used with integer values.
- It has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:

```
(cost + tax) * discount
 cost + (tax * discount)
```

- Without parentheses, an expressions is evaluated according to the *rules of precedence.*

Order of precedence

Highest (evaluated first)	
`()`	`inside-out`
`*, /, %`	`left-to-right`
`+, -`	`left-to-right`
Lowest (evaluated last)	

Sample expression

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	`rate * rate + delta`	`(rate * rate) + delta`
$2(salary + bonus)$	`2 * (salary + bonus)`	`2 * (salary + bonus)`
$\frac{1}{time + 3mass}$	`1 / (time + 3 * mass)`	`1 / (time + (3 * mass))`
$\frac{a - 7}{t + 9v}$	`(a - 7) / (t + 9 * v)`	`(a - 7) / (t + (9 * v))`

Assignment operator

- Syntax

```
leftSide = rightSide
```

assigns the value on the right side to the variable appearing on the left side

- Example:

```
i = 1;
start = i;
avg = (one + two + three)/3;
```

The right side of the assignment operator

- The right side may be either
  - Literal

```
i = 1;
```

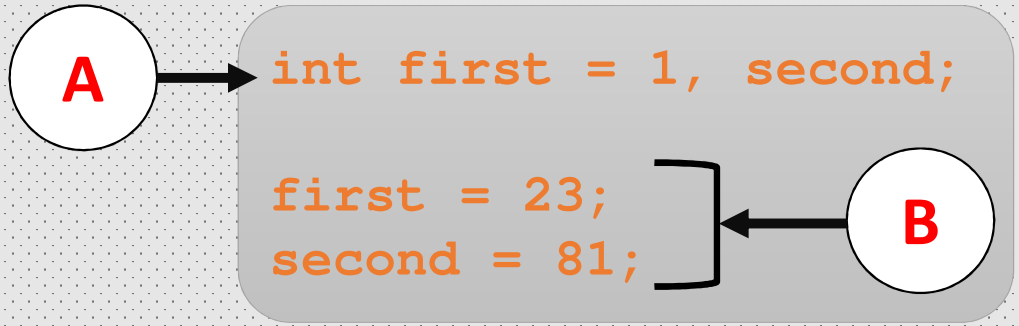
  - Variable identifier

```
start = i;
```

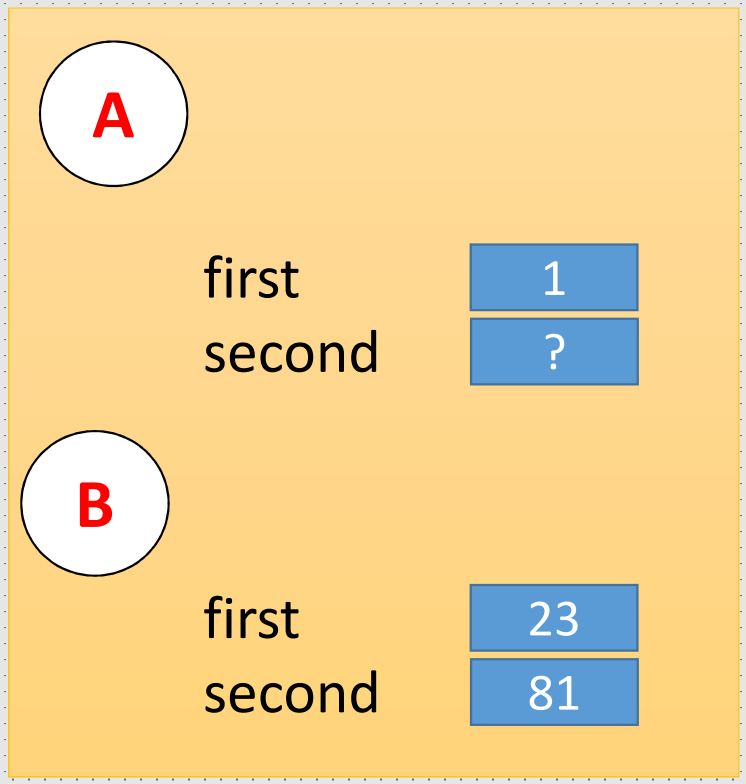
  - Expression

```
sum = first + second;
```

Assigning literals

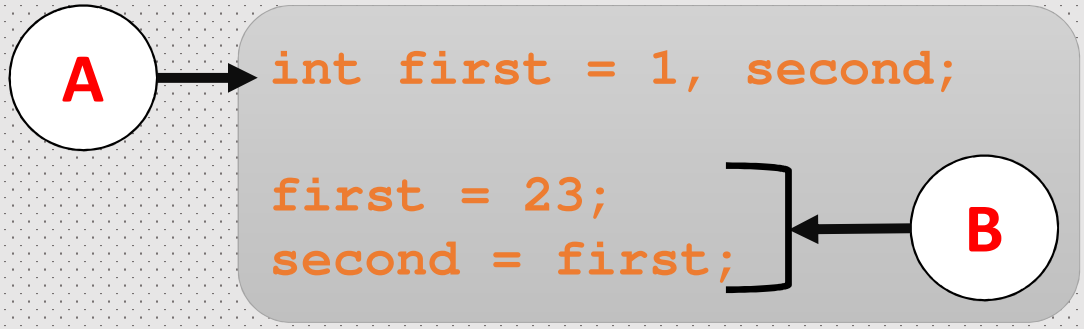


Code

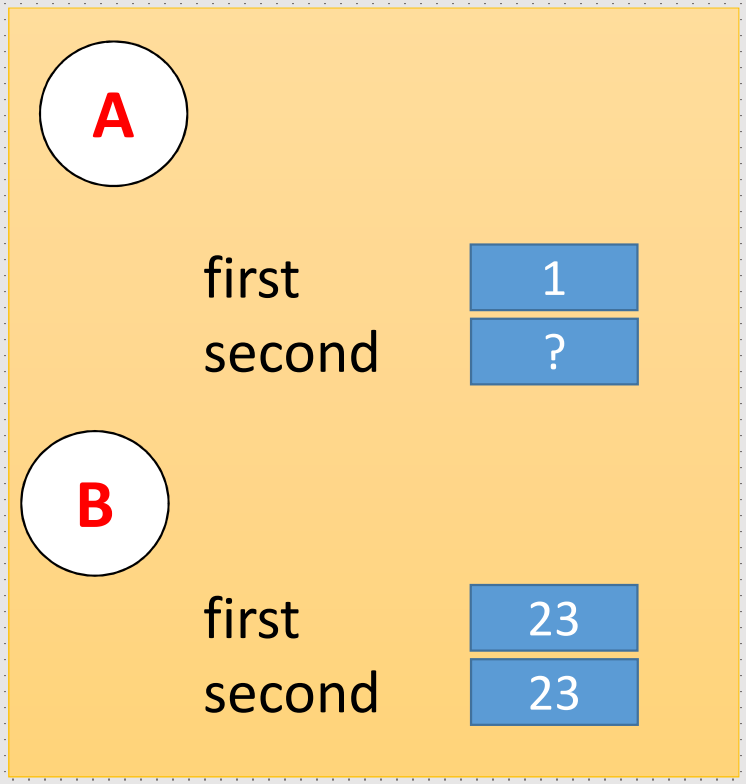


State of memory

Assigning variables

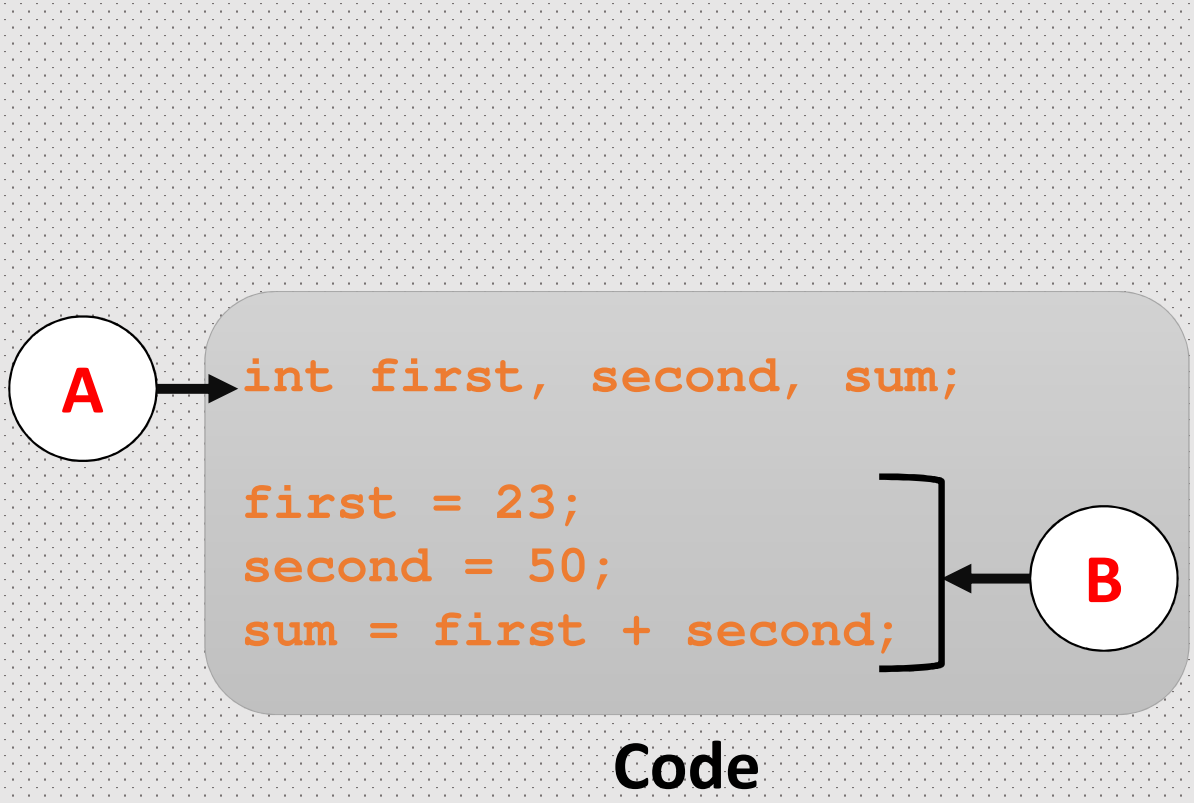


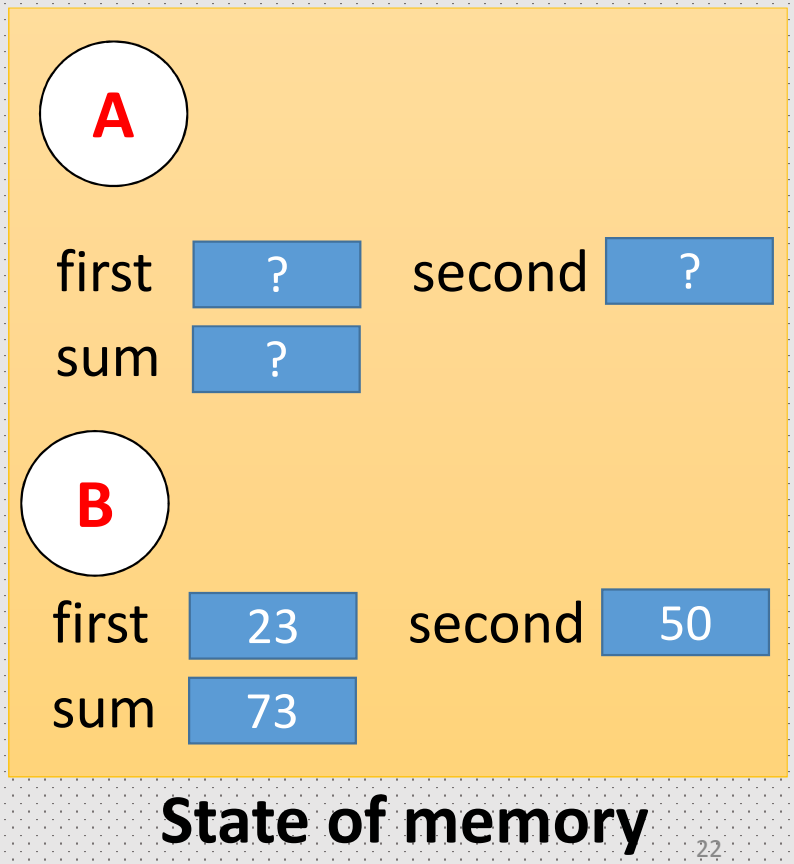
Code



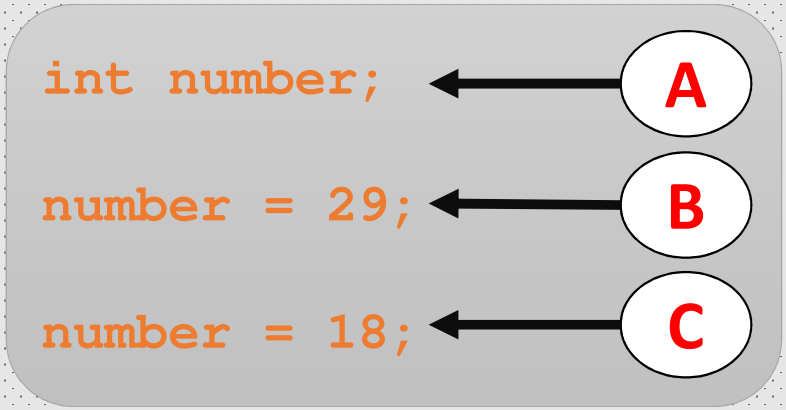
State of memory

Assigning expressions

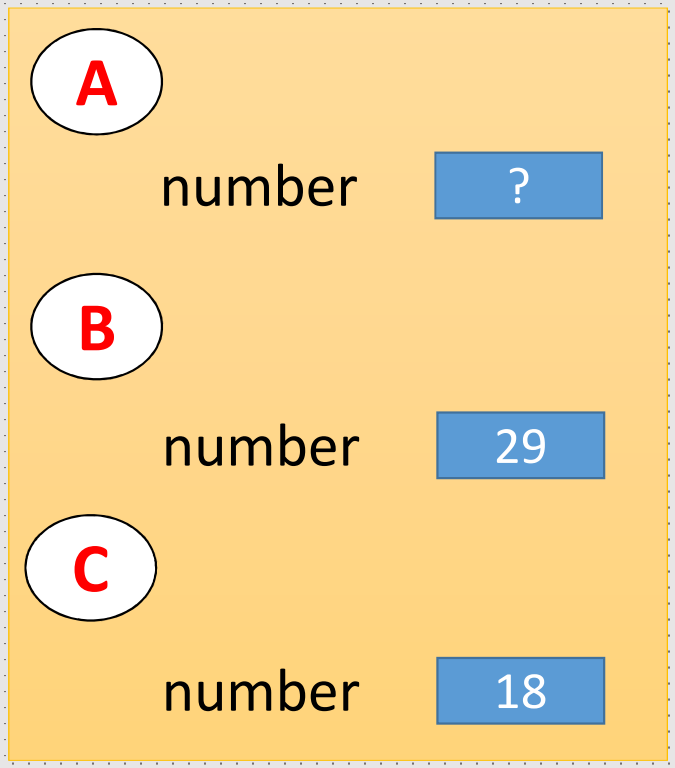




Updating data



Code



State of memory

Arithmetic & assignment operators

- Java allows combining arithmetic and assignment operators into a single operator:
  - Addition & assignment `+=`
  - Subtraction & assignment `-=`
  - Multiplication & assignment `*=`
  - Division & assignment `/=`
  - Remainder & assignment `%=`

Syntax

```
leftSide  Op= rightSide;
```

- `leftSide` : *always a variable identifier*
- `rightSide` : *either a literal, variable identifier or an expression*
- `Op` : *arithmetic operator*

- This is equivalent to:

```
leftSide  = leftSide Op rightSide;
```

Examples

```
count -= 1;          →   count = count - 1;

amount /= count;     →   amount = amount / count;

z *= x + y * w;      →   z = z * (x + y * w);

x %= 5;              →   x = x % 5;
```

Increment & decrement operators

- Only use `++` or `--` when a variable is being *incremented/decremented by 1* as a statement by it itself
- The increment operator

```
count++;    or   ++count;
```

- The decrement operator

```
count--;    or   --count;
```

Increment & decrement operators

- equivalent operations:

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

- equivalent operations:

```
count--;
--count;
count = count - 1;
count -= 1;
```

Increment & decrement operators in expressions

- **Post-Increment (`count++`)** : value is first used for computing the result and then **incremented**.

- **Pre-Increment (`++count`)**: value is **incremented** first and then result is computed.

The same applies to the decrement operator (`count--` and `--count`)

Increment & decrement operators in expressions

```
int x = 3;
int result = 5 * (++x);
```

After execution

x	4
result	20

```
int x = 3;
int result = 5 * (x++);
```

After execution

x	4
result	15

Increment & decrement operators in expressions

- This expression:

```
int sum = 5 + a++;
```

Is equivalent to:

```
int sum = 5 + a;
a = a + 1;
```

- This expression:

```
int sum = 5 + ++a;
```

Is equivalent to:

```
a = a + 1;
int sum = 5 + a;
```

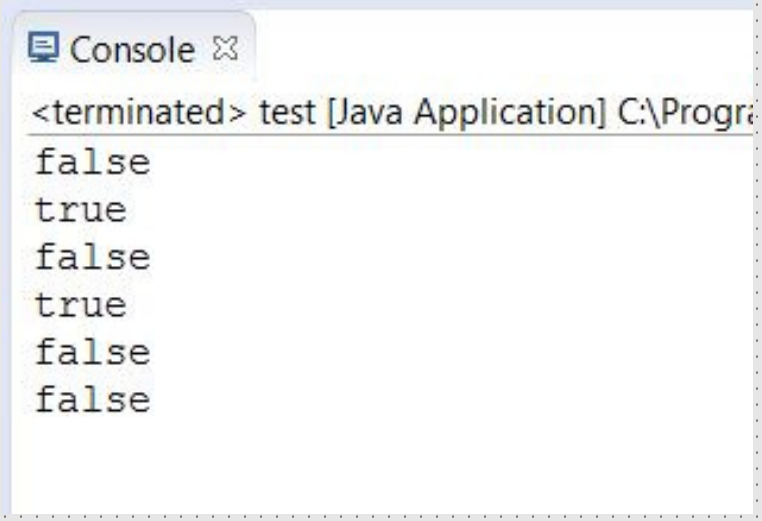
Relational operators

- Relational operators compare two values
- They produce a boolean value (`true` or `false`) depending on the relationship

Operator	Name	Example	meaning
==	Equal to	x == y	`true` if x equals y, otherwise `false`
!=	Not equal to	x != y	`true` if x is not equal to y, otherwise `false`
>	Greater than	x > y	`true` if x is greater than y, otherwise `false`
<	Less than	x < y	`true` if x is less than y, otherwise `false`
>=	Greater than or equal to	x >= y	`true` if x is greater than or equal to y, otherwise `false`
<=	Less than or equal to	x <= y	`true` if x is less than or equal to y, otherwise `false`

Examples

```
int a = 3, b = 5, c = 10, d = 3;
boolean f1;

f1 = a != d;
System.out.println(f1);
System.out.println(a <= b);
System.out.println(d > 3);
System.out.println(1 + d > 3);
System.out.println(c == d);
System.out.println(a < d);
```



Logical operators

- Logical operators are used to determine the logic between variables or values

Logical Operator	Java operator
AND	&&
OR	||
NOT	!

x	y	x && y	x || y	!x
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Examples

```
boolean x = true;
boolean y = false;
boolean result;
result = (x && y);                  // result is assigned false
result = ((x || y) && x);           // (x || y) is true
                                    // (true && x) is true
                                    // result is now assigned true
System.out.println(result);
```

Examples

```
int x = 5, y = 10, z = -2;
boolean result, output;
result = (x != y) && (z < 0);   // x!=y is false
                                // result is false
result = (x != y) || (z < 0);   // x!=y is false
                                // z < 0 is true
                                // result is true
output = !result;               // output is now false
                                // result IS STILL true
```

Operators precedence

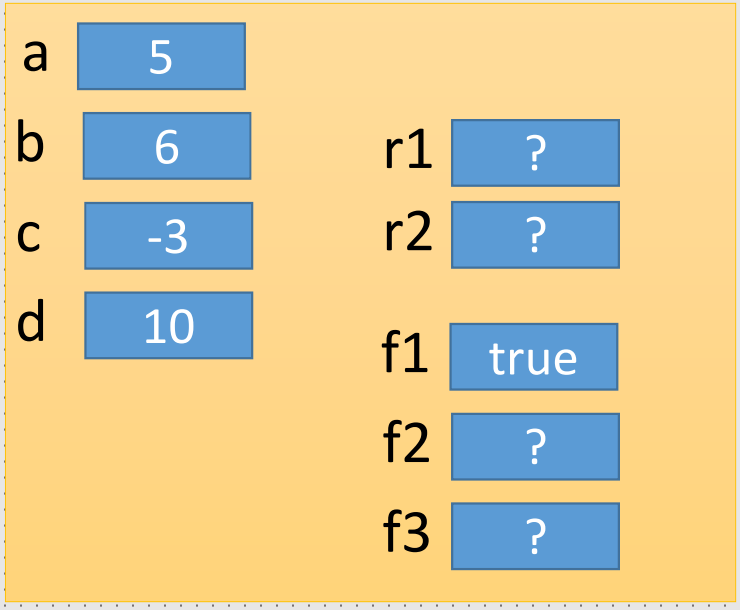
highest

lowest

Parentheses	**()**	inside-out , left to right
unary operator	**++, --, +, -, !**	right to left
Multiplicative	***, /, %**	left to right
Additive & string concatenation	**+ , -** **+**	left to right
Relational	**<, >, <=, >=**	left to right
Equality	**==, !=**	left to right
Logical AND	**&&**	left to right
Logical OR	**||**	left to right
Assignment	**=, +=, -=, *=, /=, %=**	right to left

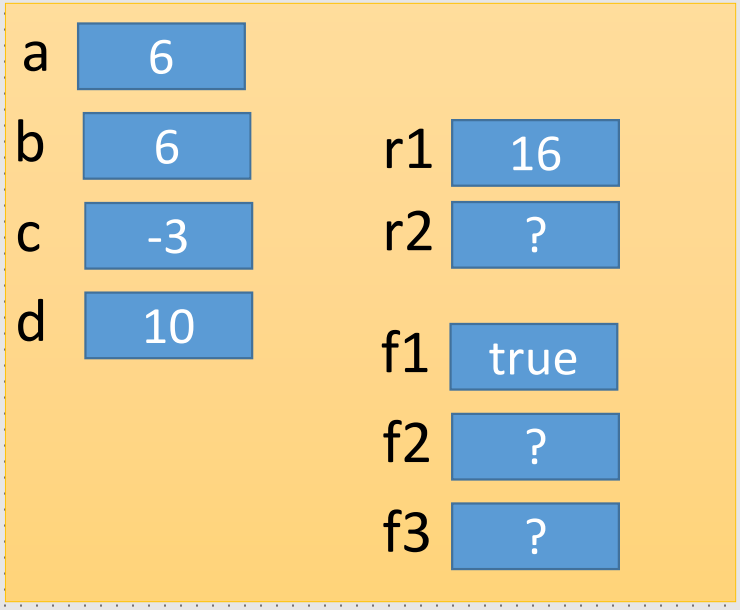
Operators precedence: Example

```
int a = 5, b = 6 , c = -3, d = 10;
int r1, r2;
boolean f1 = true, f2, f3;
```



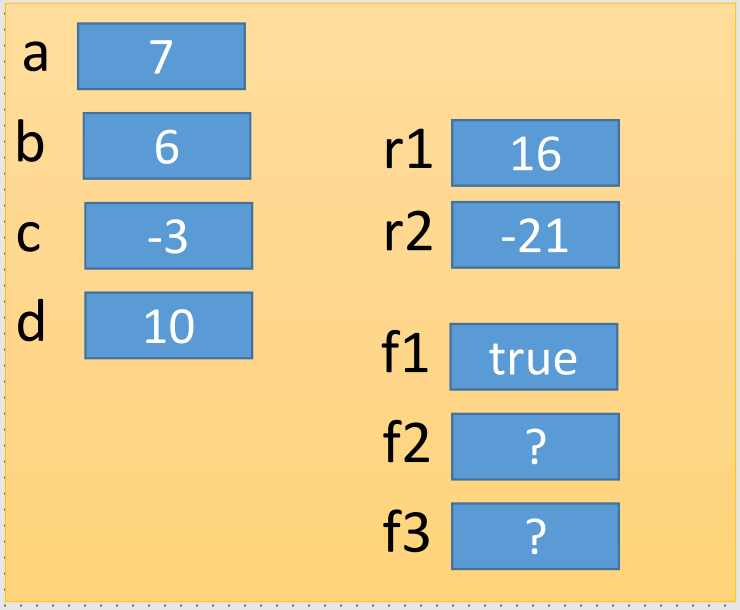
Operators precedence: Example

```
int a = 5, b = 6 , c = -3, d = 10;
int r1, r2;
boolean f1 = true, f2, f3;

r1 = ++a * -c + b / c;
```



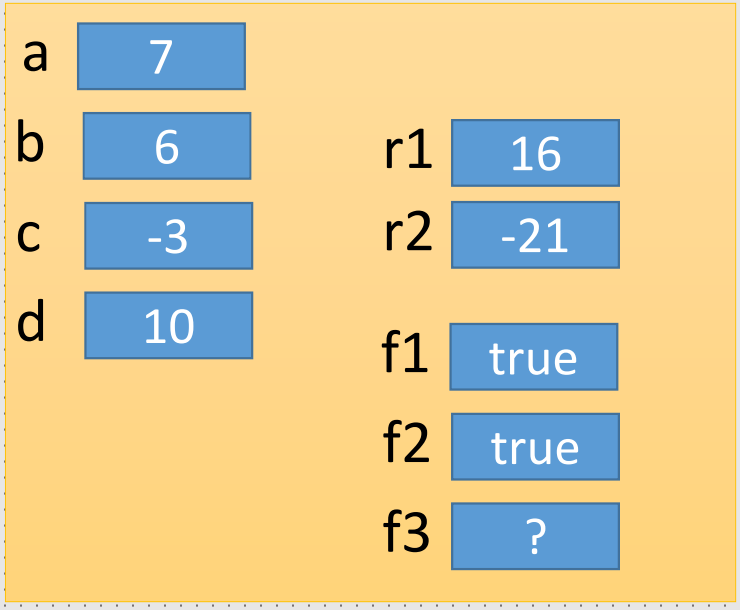
Operators precedence: Example

```
int a = 5, b = 6 , c = -3, d = 10;
int r1, r2;
boolean f1 = true, f2, f3;

r1 = ++a * -c + b / c;
r2 = ++a * (-c + b) / c;
```



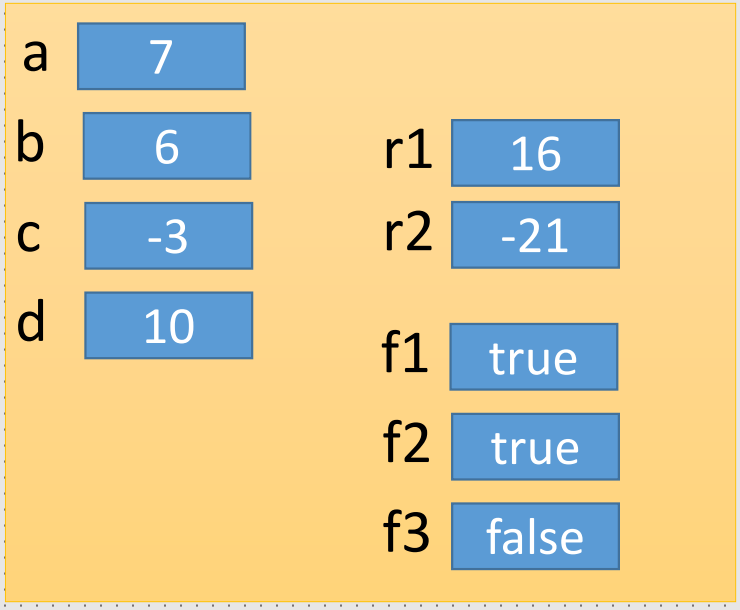
Operators precedence: Example

```
int a = 5, b = 6 , c = -3, d = 10;
int r1, r2;
boolean f1 = true, f2, f3;

r1 = ++a * -c + b / c;
r2 = ++a * (-c + b) / c;
f2 = c < 0 || d != 10 && !f1;
```



Operators precedence: Example

```
int a = 5, b = 6 , c = -3, d = 10;
int r1, r2;
boolean f1 = true, f2, f3;

r1 = ++a * -c + b / c;
r2 = ++a * (-c + b) / c;
f2 = c < 0 || d != 10 && !f1;
f3 = (c < 0 || d != 10) && !f1;
```



Case study:

Vending Machine Change Example

Vending machine change

- Requirements:
  - The user enters an amount of riyals
  - The program determines and displays a combination of 500's, 100's, 50's, 10's, 5's and 1's paper currencies.

  - For example, 762 riyals can be 1 (500), 2 (100), 1 (50), 1 (10) and 2 (1).

Vending machine change

- Algorithm:
  1. Read the amount.
  2. Find the maximum number of 500's in the amount.
  3. Subtract the value of the 500's from the amount.
  4. Repeat the last two steps for 100's, 50's, 10's, 5's and 1's.
  5. Print the quantities of each paper currency.

Vending machine change

- How do we determine the number of 500's (or 100's, etc.) in an amount?
- There are two 500's in 1350, but there are also 2 500's in 1050
- That's because

```
1350 / 500 = 2 and 1050 / 500 = 2
```

Vending machine change

- How do we determine remaining amount?
- The remaining amount can be determined using the `%` operator

```
1350 % 500 = 350 and 1050 % 500 = 50
```

- Similarly for 100's, 50's, 10's, 5's and 1's paper currencies.

Vending machine change - program

- Variables needed?

```
int amount= 0,
    n500 = 0,
    n100 = 0,
    n50 = 0,
    n10 = 0,
    n5 = 0;
```

```
import java.util.Scanner;

public class VendingMachineChange {
          public static void main(String[] args) {

                int amount = 0, n500 = 0 , n100 = 0,
                n50 = 0, n10 = 0, n5 = 0;
                Scanner input = new Scanner(System.in);

                System.out.println("Enter the amount to be changed: ");
                amount = input.nextInt();

                n500 = amount / 500;
                amount = amount % 500;

                n100 = amount / 100;
                amount = amount % 100;

                n50 = amount / 50;
                amount = amount % 50;

                n10 = amount / 10;
                amount = amount % 10;

                n5 = amount / 5;
                amount = amount % 5;

                System.out.println("The number of 500's is: " + n500);
                System.out.println("The number of 100's is: " + n100);
                System.out.println("The number of 50's is: " + n50);
                System.out.println("The number of 10's is: " + n10);
                System.out.println("The number of 5's is: " + n5);
                System.out.println("The number of 1's is: " + amount);

                }
}
```

```
import java.util.Scanner;
public class VendingMachineChange1 {

        public static void main(String[] args) {

            int amount = 0;
            Scanner input = new Scanner(System.in);

            System.out.println("Enter the amount to be changed: ");
            amount = input.nextInt();

            System.out.println("The number of 500's is: " + amount/500);

            amount %= 500;
            System.out.println("The number of 100's is: " + amount/100);

            amount %= 100;
            System.out.println("The number of 50's is: " + amount/50);

            amount %= 50;
            System.out.println("The number of 10's is: " + amount/10);

            amount %= 10;
            System.out.println("The number of 5's is: " + amount/5);

            amount %= 5;
            System.out.println("The number of 1's is: " + amount);
        }
}
```