Uvod v računalništvo

Aleksander Sadikov

2015/2016

Clear, precise, unambiguous…

FIGURE 2.1

Initially, set the value of the variable *carry* to 0 and the value of the variable *i* to 0. When these initializations have been completed, begin looping as long as the value of the variable *i* is less than or equal to $(m - 1)$. First, add together the values of the two digits a_i and b_i and the current value of the carry digit to get the result called c_i. Now check the value of c_i to see whether it is greater than or equal to 10. If c_i is greater than or equal to 10, then reset the value of *carry* to 1 and reduce the value of c_i by 10; otherwise, set the value of *carry* to 0. When you are finished with that operation, add 1 to *i* and begin the loop all over again. When the loop has completed execution, set the leftmost digit of the result c_m to the value of *carry* and print out the final result, which consists of the digits $c_m\ c_{m-1} \ldots c_0$. After printing the result, the algorithm is finished, and it terminates.

The addition algorithm of Figure 1.2 expressed in natural language

FIGURE 2.1

Initially, set the value of the variable *carry* to 0 and the value of the variable *i* to 0. When these initializations have been completed, begin looping as long as the value of the variable *i* is less than or equal to $(m - 1)$. First, add together the values of the two digits a_i and b_i and the current value of the carry digit to get the result called c_i. Now check the value of c_i to see whether it is greater than or equal to 10. If c_i is greater than or equal to 10, then reset the value of c_i by subtracting 10 from it and set the value of *carry* to 1. Otherwise, set the value of *carry* to 0. In either case, add 1 to *i* and begin the loop all over again. When the loop has completed execution, set the leftmost digit of the result c_m to the value of *carry* and print out the final result, which consists of the digits $c_m\ c_{m-1} \ldots c_0$. After printing the result, the algorithm is finished, and it terminates.

Rambling, unstructured, and hard-to-follow.
(Imagine 5, 10 or 100 pages of this?)

The addition algorithm of Figure 1.2 expressed in natural language

```
Scanner inp = new Scanner(System.in);
int i, m, carry;
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];
m = inp.nextInt();
for (int j = 0;j <= m-1;j++) {
        a[j] = inp.nextInt();
        b[j] = inp.nextInt();
}
carry = 0;
i = 0;
while (i < m) {
    c[i] = a[i] + b[i] + carry;
    if (c[i] >= 10)
                .
                .
                .
```

When creating algorithms, a programmer should no more worry about semicolons and capitalization than a novelist should worry about typography and cover design when writing the first draft.

Pseudocode

A set of English language costructs designed to resemble programming language statements. Simple, highly readable, virtually no grammatical rules.

Sometimes called: “a programming language without details”.

Given: $m \geq 1$ and two positive numbers each containing m digits, $a_{m-1}\, a_{m-2} \ldots a_0$ and $b_{m-1}\, b_{m-2} \ldots b_0$

Wanted: $c_m\, c_{m-1}\, c_{m-2} \ldots c_0$, where $c_m c_{m-1}\, c_{m-2} \ldots c_0 = (a_{m-1}\, a_{m-2} \ldots a_0) + (b_{m-1}\, b_{m-2} \ldots b_0)$

Algorithm:

Step 1 Set the value of *carry* to 0

Step 2 Set the value of i to 0

Step 3 While the value of i is less than or equal to $m - 1$, repeat the instructions in Steps 4 through 6

Step 4 Add the two digits a_i and b_i to the current value of *carry* to get c_i

Step 5 If $c_i \geq 10$, then reset c_i to $(c_i - 10)$ and reset the value of *carry* to 1; otherwise, set the new value of *carry* to 0

Step 6 Add 1 to i, effectively moving one column to the left

Step 7 Set c_m to the value of *carry*

Step 8 Print out the final answer, $c_m\, c_{m-1}\, c_{m-2} \ldots c_0$

Step 9 Stop

Pseudocode is **not** a formal language
with rigidly defined standards…

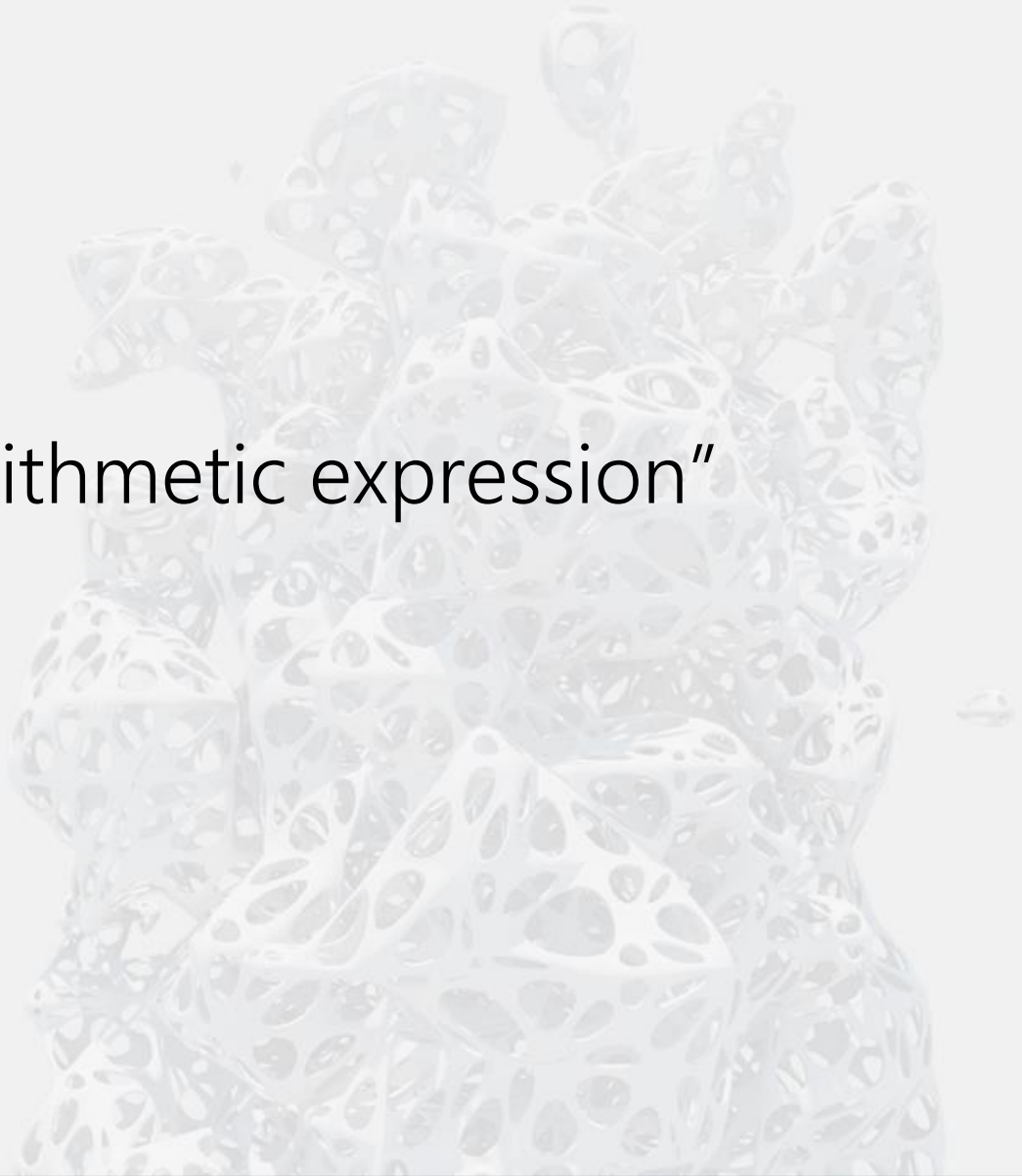
If you do not like the constructs we use, feel free to modify them or select others that are more helpful to you

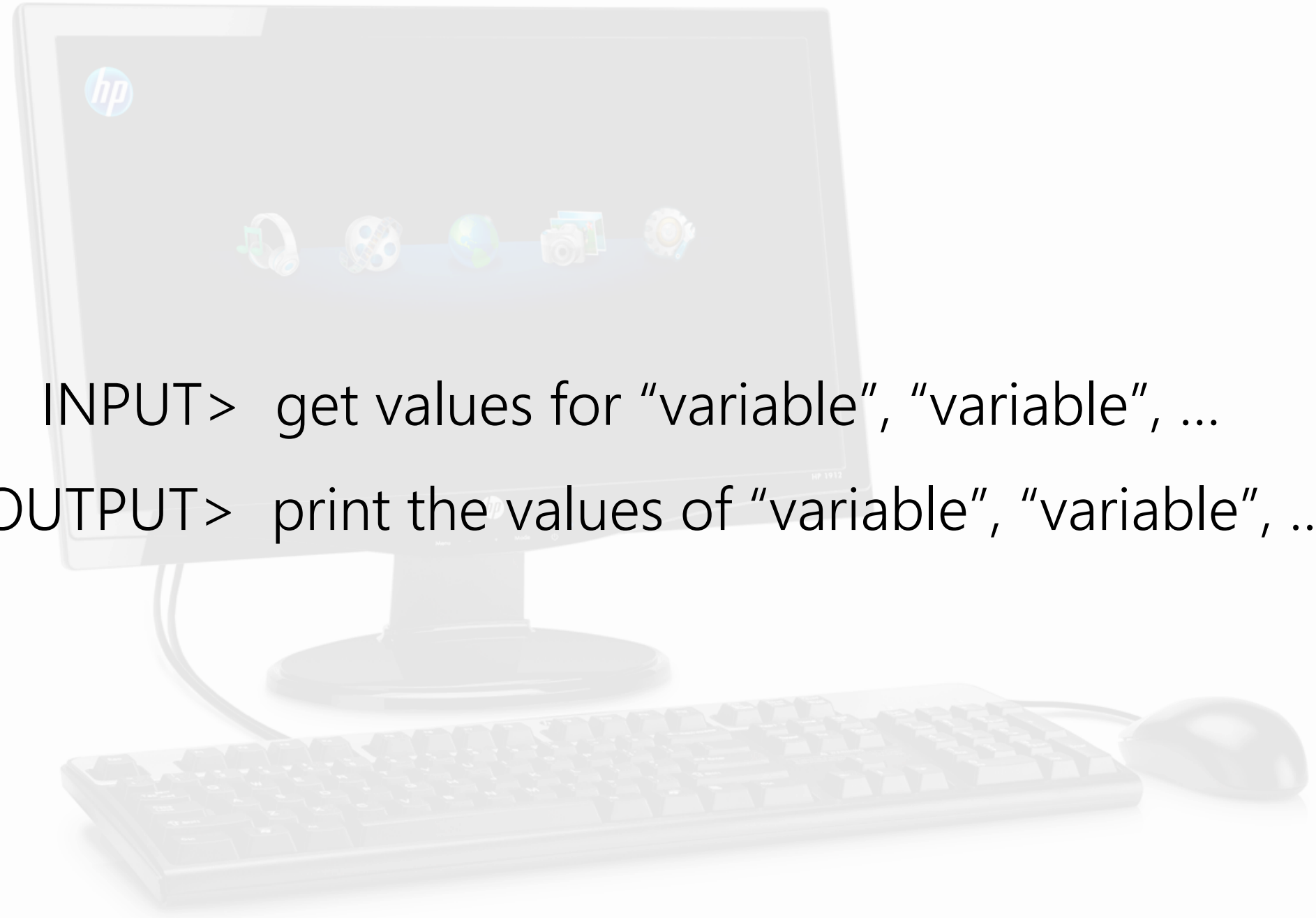
3

- Sequential operations
- Conditional operations
- Iterative operations

Sequential operations need to take care of three basic things: computation, input, and output.

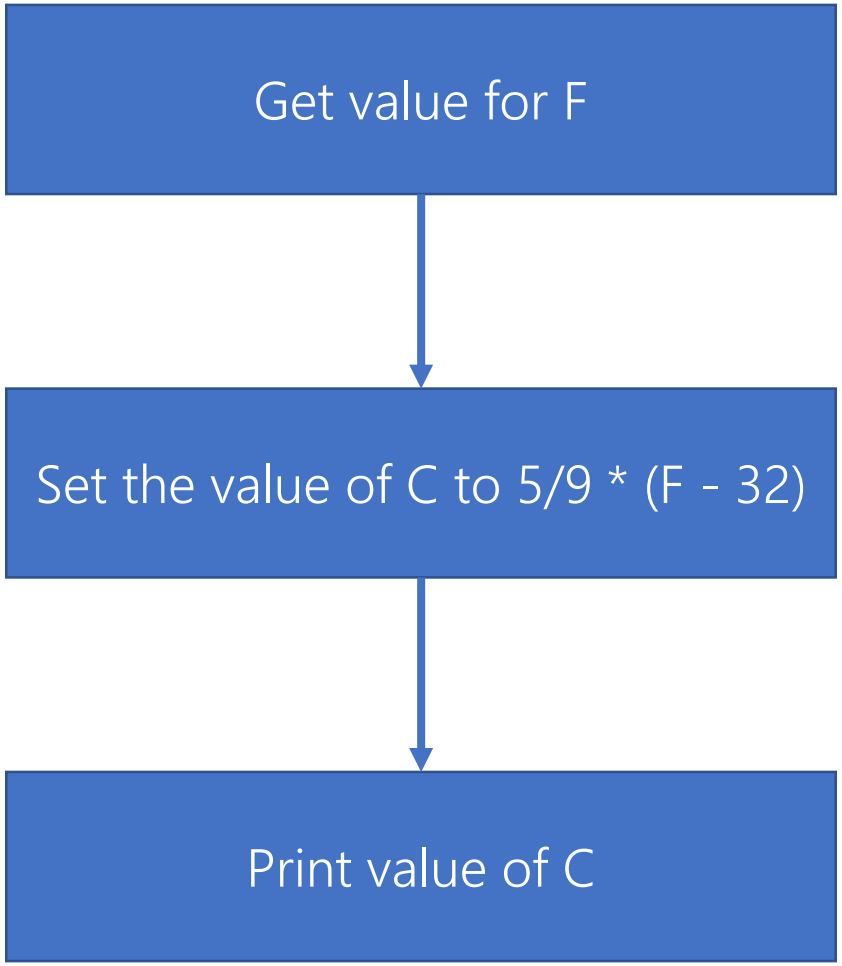
Set the value of “variable” to “arithmetic expression”





INPUT> get values for “variable”, “variable”, …

OUTPUT> print the values of “variable”, “variable”, …


Get value for F
Set the value of C to 5/9 * (F - 32)
Print value of C

Control flow operations:
conditional and iterative operations.

These two types of operations allow us to alter
the normal sequential flow of control in an algorithm.

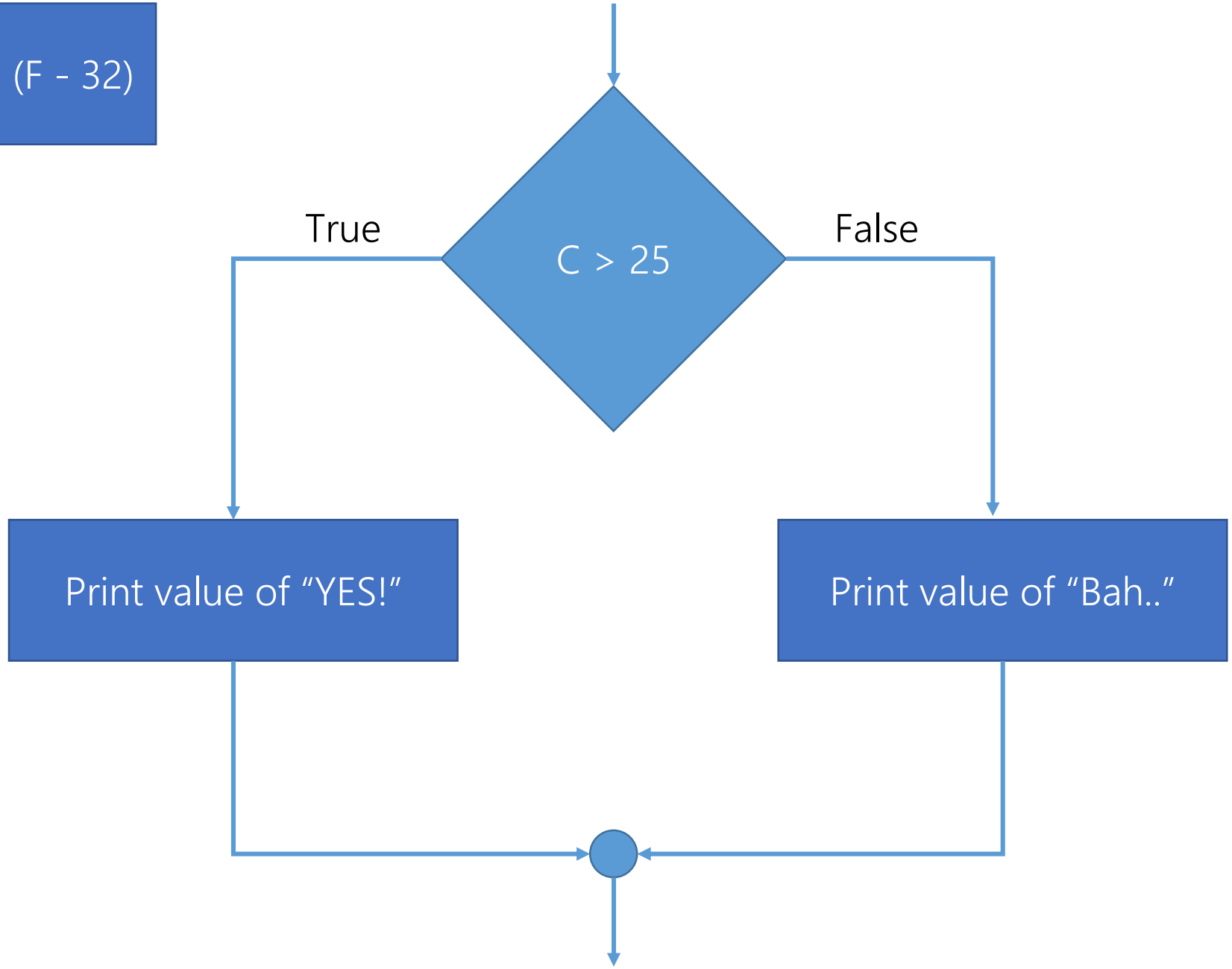
If “a true/false condition” is true then

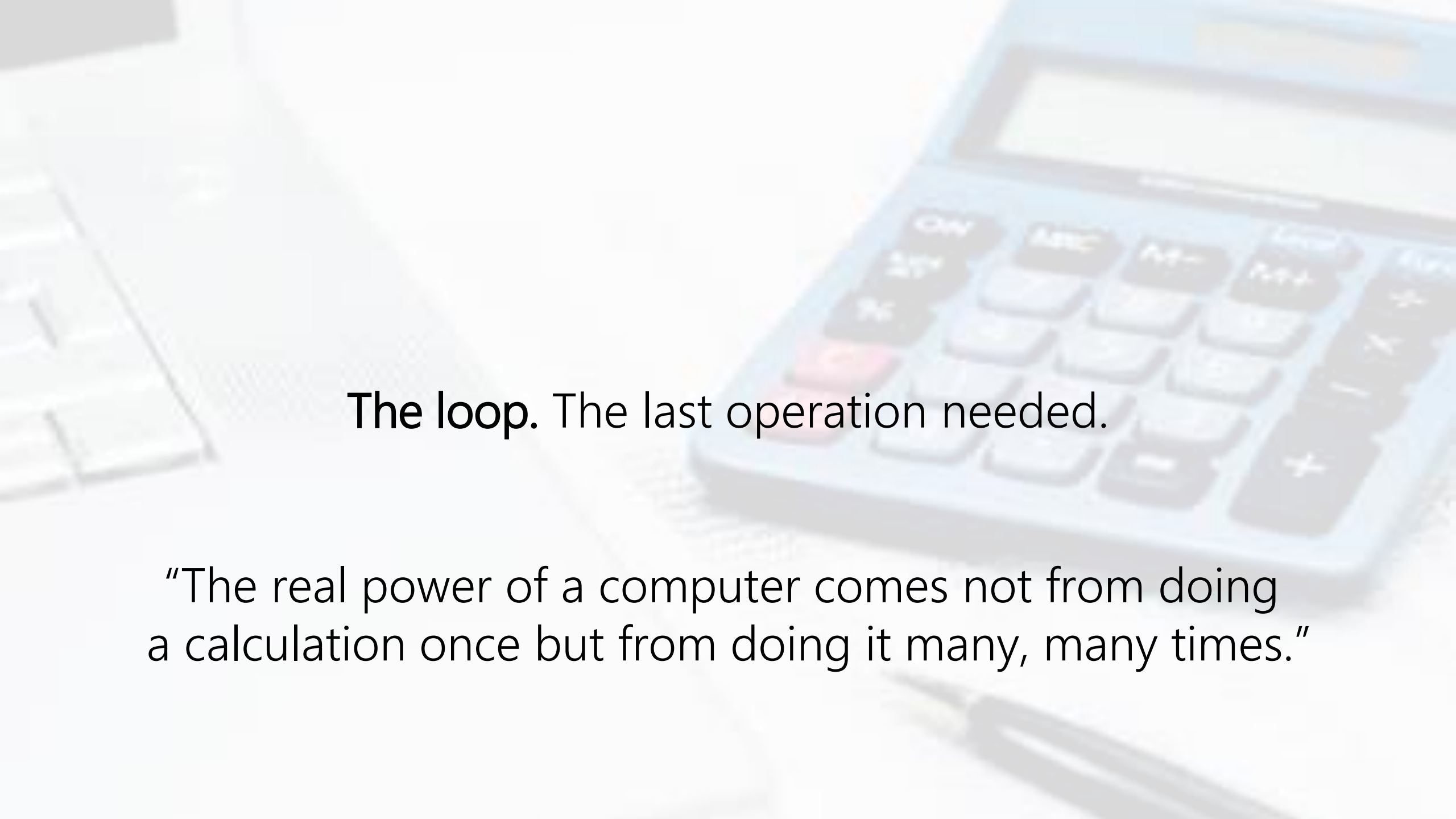
“first set of operations”

Else

“second set of operations”

Set the value of C to 5/9 * (F - 32)





The loop. The last operation needed.

"The real power of a computer comes not from doing a calculation once but from doing it many, many times."

While (“a true/false condition”) do

operation

operation

…

End of the loop

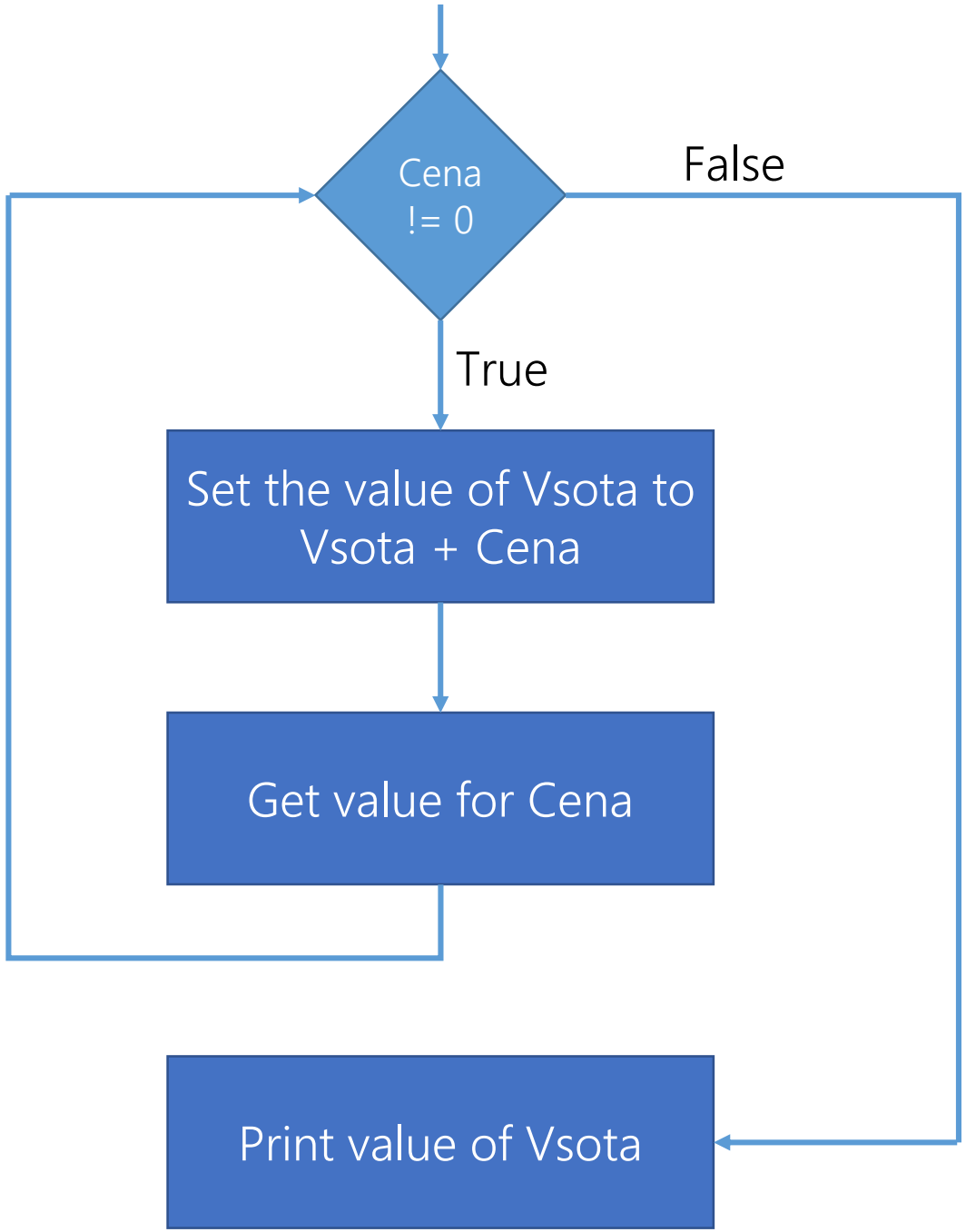
Do

operation

operation

…

While (“a true/false condition”)


Cena
!= 0
False
True
Set the value of Vsota to Vsota + Cena
Get value for Cena
Print value of Vsota

Properties of algorithms.

Ok, algorithms solve problems.
But, are some algorithms better than others?
Why?

Correctness.

Maintenance.

Efficiency.

We’ve seen rapid advancement of technology, but…
and it’s a very big BUT ;)

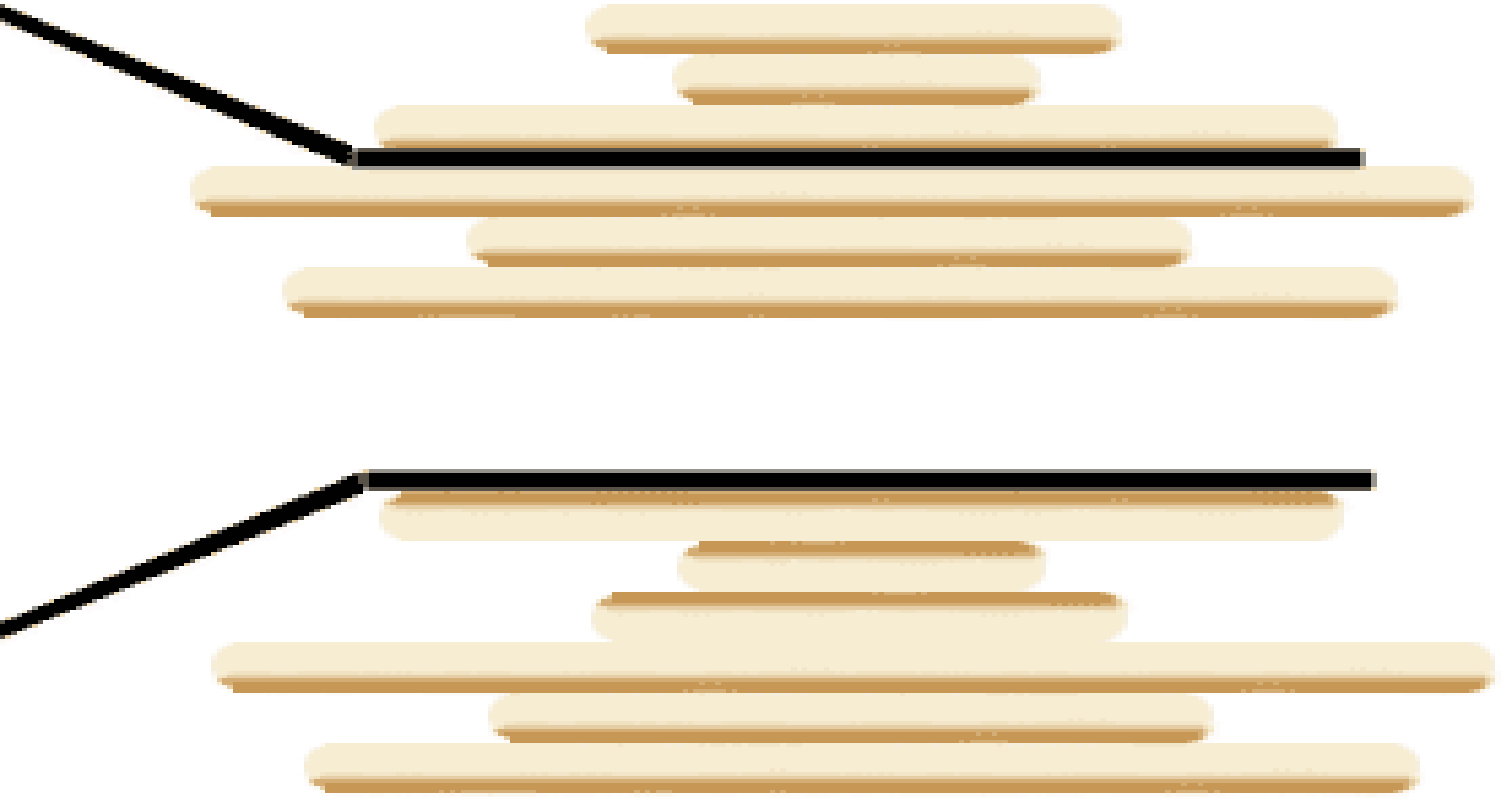
How do we measure efficiency? In what terms?

But how about time?

Simply timing the running of an algorithm is more likely to reflect machine speed or variations in input data than the efficiency of the algorithm.

The study of efficiency of algorithms
is called the analysis of algorithms,
and it is an important part of computer science.

… fundamental units of work…



Take away lesson #P3

Technology and the world changes rapidly these days.

But even so, efficiency of algorithms is important
as the computers will never be fast enough
or their memory big enough.
Remember: not all algorithms are made equal.