COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr

CHAPTER 5

Algorithms

In the introductory chapter we learned that the central theme of computer science is the study of algorithms. It is time now for us to focus on this core topic. Our goal is to explore enough of this foundational material so that we can truly understand and appreciate the science of computing.

Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

The Abstract Nature of Algorithms

An algorithm is abstract and distinct from its representation.

A single algorithm can be represented in many ways.

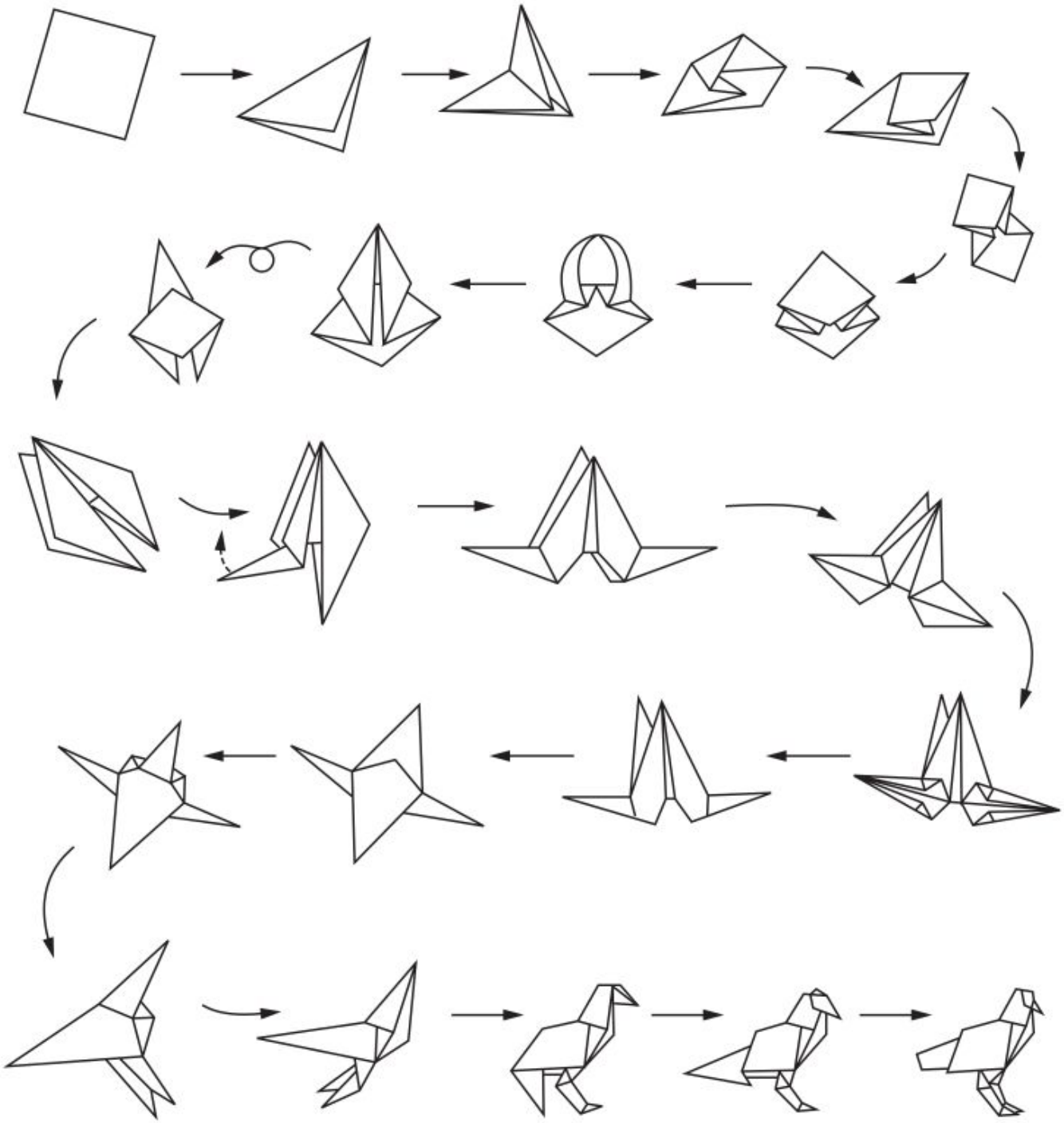
Converting temperature readings from Celsius to Fahrenheit is traditionally represented as the algebraic formula

"F = (⅘)C + 32"

But it could be represented by the instruction

"Multiply the temperature reading in Celsius by ⅘ and then add 32 to the product"

or even in the form of an electronic circuit.



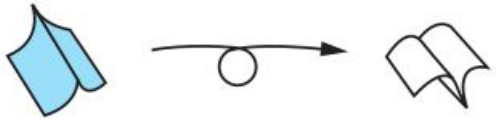
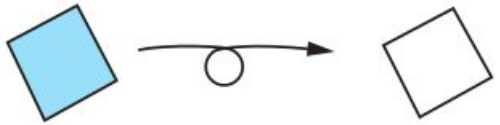
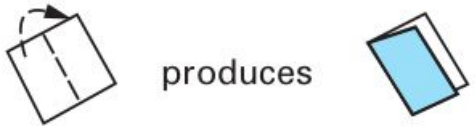
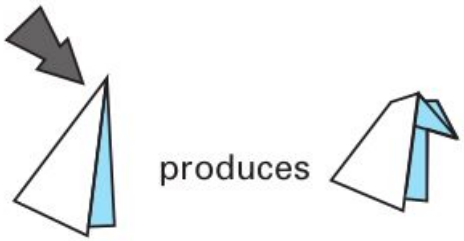
Primitives

Building blocks of an algorithm.

A collection of primitives along with a collection of rules stating how the primitives can be combined to represent more complex ideas constitutes a **programming language**.

Each primitive has its own syntax and semantics.

- Syntax refers to the primitive's symbolic representation;
- semantics refers to the meaning of the primitive.

Syntax	Semantics
	Turn paper over as in 
Shade one side of paper	Distinguishes between different sides of paper as in 
	Represents a valley fold so that  represents
	Represents a mountain fold so that  represents
	Fold over so that  produces
	Push in so that  produces

Pseudocode

A pseudocode is a notational system in which ideas can be expressed informally during the algorithm development process.

A pseudocode must have a consistent, concise notation for representing recurring semantic structures.

There are many such pseudocode variants, because there are many programming languages in existence.

- Algo1
- Pascal
- Java
- C
- Python

From Pseudocode to Python

Our pseudocode in this chapter closely mirrors actual Python syntax for the `if` and `while` structures, as well as function definition and calling syntax.

```
if (condition):
  activity
else:
  activity
```

Our pseudocode `if` and `while` structures can be converted to Python simply by being more precise with the **`condition`** and **`activity`** portions. For example, rather than the English phrase, "sales have decreased" as a condition, we would need a proper Python comparison expression, such as

```
if (sales_current < sales_previous):
```

where the sales variables had already been assigned in previous lines of the script. Similarly, informal English sentences or phrases used as the **`activity`** in our pseudocode would need to be replaced with Python statements and expressions such as those we have already seen in earlier chapters.

How can you tell the difference between pseudocode and actual Python code in this book? As a rule, real Python code uses operators (like "<", "=", or "+") to string together multiple named variables into more complex expressions, or commas to separate a list of parameters being sent to a function. Articles like "the" and "a," or prepositions like "from" appear in pseudocode. We use periods at the end of sentences in pseudocode; Python statements do not have punctuation at the end.

The Art of Problem Solving

Basic principles by G. Polya in 1945

Phase 1. Understand the problem.

Phase 2. Devise a plan for solving the problem.

Phase 3. Carry out the plan.

Phase 4. Evaluate the solution for accuracy and for its potential as a tool for solving other problems.

Translated into the context of program development, these phases become

Phase 1. Understand the problem.

Phase 2. Get an idea of how an algorithmic function might solve the problem.

Phase 3. Formulate the algorithm and represent it as a program.

Phase 4. Evaluate the program for accuracy and for its potential as a tool for solving other problems.

The Art of Problem Solving

One could argue, however, that a true understanding of a problem is not obtained until a solution has been found.

The mere fact that a problem is unsolved implies a lack of understanding.

To insist on a complete understanding of the problem before proposing any solutions is therefore somewhat idealistic.

The Art of Problem Solving

Person A is charged with the task of determining the ages of person B's three children. B tells A that the product of the children's ages is 36. After considering this clue, A replies that another clue is required, so B tells A the sum of the children's ages. Again, A replies that another clue is needed, so B tells A that the oldest child plays the piano. After hearing this clue, A tells B the ages of the three children.

How old are the three children?

The Art of Problem Solving

Person A is charged with the task of determining the ages of person B's three children. B tells A that the product of the children's ages is 36. After considering this clue, A replies that another clue is required, so B tells A the sum of the children's ages. Again, A replies that another clue is needed, so B tells A that the oldest child plays the piano. After hearing this clue, A tells B the ages of the three children.
How old are the three children?

At first glance the last clue seems to be totally unrelated to the problem, yet it is apparently this clue that allows A to finally determine the ages of the children.

How can this be?

The Art of Problem Solving

Person A is charged with the task of determining the ages of person B's three children. B tells A that the product of the children's ages is 36. After considering this clue, A replies that another clue is required, so B tells A the sum of the children's ages. Again, A replies that another clue is needed, so B tells A that the oldest child plays the piano. After hearing this clue, A tells B the ages of the three children.

How old are the three children?

a. Triples whose product is 36

(1,1,36)	(1,6,6)
(1,2,18)	(2,2,9)
(1,3,12)	(2,3,6)
(1,4,9)	(3,3,4)

b. Sums of triples from part (a)

1 + 1 + 36 = 38	1 + 6 + 6 = 13
1 + 2 + 18 = 21	2 + 2 + 9 = 13
1 + 3 + 12 = 16	2 + 3 + 6 = 11
1 + 4 + 9 = 14	3 + 3 + 4 = 10

Getting a Foot in the Door

Before A, B, C, and D ran a race they made the following predictions:

- A predicted that B would win.
- B predicted that D would be last.
- C predicted that A would be third.
- D predicted that A's prediction would be correct.

Only one of these predictions was true, and this was the prediction made by the winner.

In what order did A, B, C, and D finish the race?

Algorithm Structures

Iterative structure

- A collection of instructions is repeated in a looping manner

Recursive structure

- Repeating the set of instructions as a subtask of itself

Iterative Structure

The Sequential Search Algorithm

Consider the problem of searching within a list for the occurrence of a particular target value.

We want to develop an algorithm that determines whether that value is in the list.

- If the value is in the list, we consider the search a success;
- otherwise we consider it a failure.

We assume that the list is sorted according to some rule for ordering its entries.

Iterative Structure

The Sequential Search Algorithm

```
def Search(List, TargetValue):
  if (List is empty):
    Declare search a failure.
  else:
    Select the first entry in List to be TestEntry.
    while (TargetValue > TestEntry and
        there remain entries to be considered):
      Select the next entry in List as TestEntry.
    if (TargetValue == TestEntry):
      Declare search a success.
    else:
      Declare search a failure.
```

Iterative Structure

LOOP Control

One method of implementing repetition

- in which a collection of instructions, called the body of the loop, is executed in a repetitive fashion
  - under the direction of some control process.

```
while (condition):
   Body
```

```
check the condition.
execute the body.
check the condition.
execute the body.
   .
   .
   .
check the condition.
```

until the condition fails.

Iterative Structure

LOOP Control

To execute the statement

Add a drop of sulfuric acid.

three times, we could write:

Add a drop of sulfuric acid.

Add a drop of sulfuric acid.

Add a drop of sulfuric acid.

A similar sequence that is equivalent to the loop structure

while (the pH level is greater than 4):

add a drop of sulfuric acid.

Iterative Structure

LOOP Control

To execute the statement

Add a drop of sulfuric acid.

three times, we could write:

Add a drop of sulfuric acid.

Add a drop of sulfuric acid.

Add a drop of sulfuric acid.

A similar sequence that is equivalent to the loop structure

while (the pH level is greater than 4):

add a drop of sulfuric acid.

TERMINATION CONTROL

Iterative Structure

Initialize: Establish an initial state that will be modified toward the termination condition

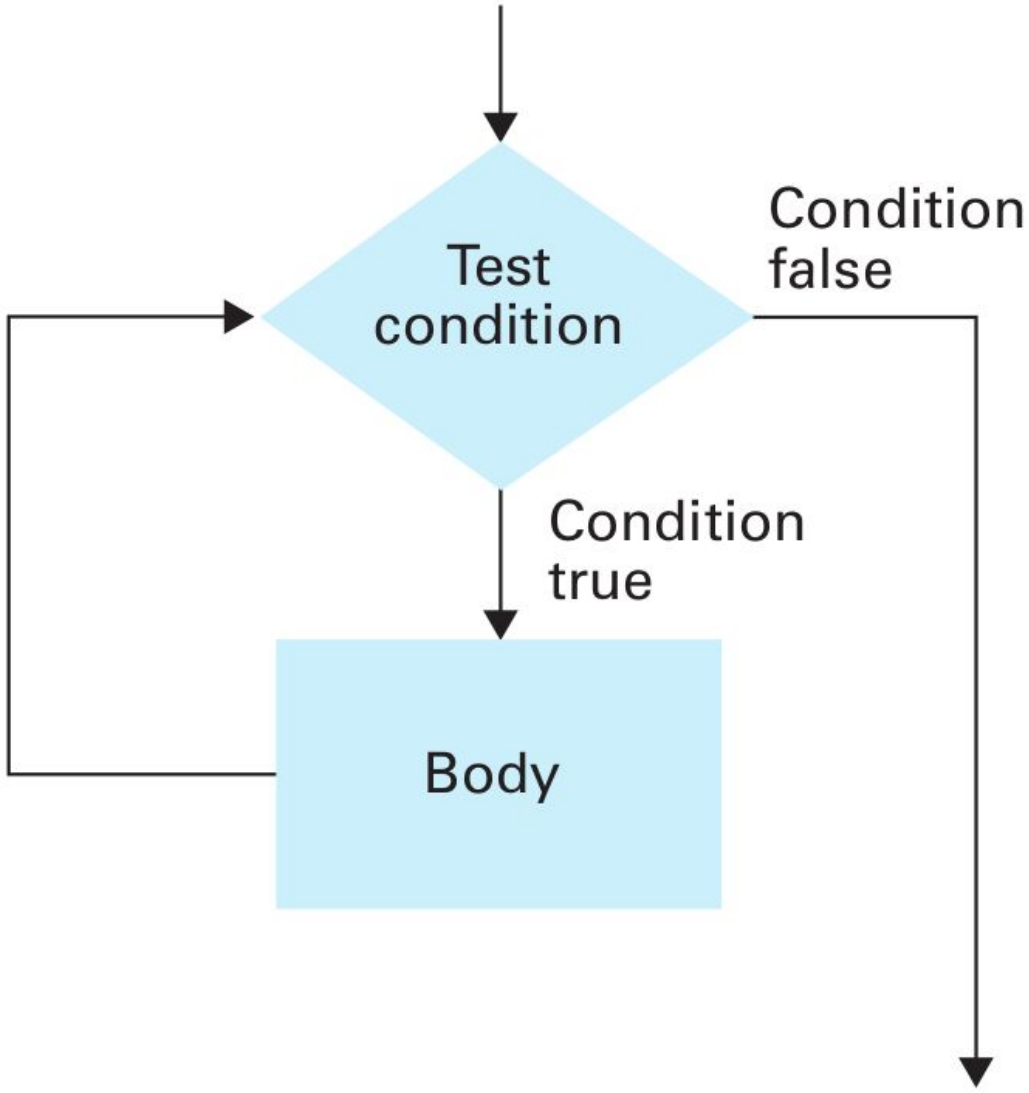
Test: Compare the current state to the termination condition and terminate the repetition if equal

Modify: Change the state in such a way that it moves toward the termination condition

```
Number = 1
while (Number != 6):
   Number = Number + 2
```

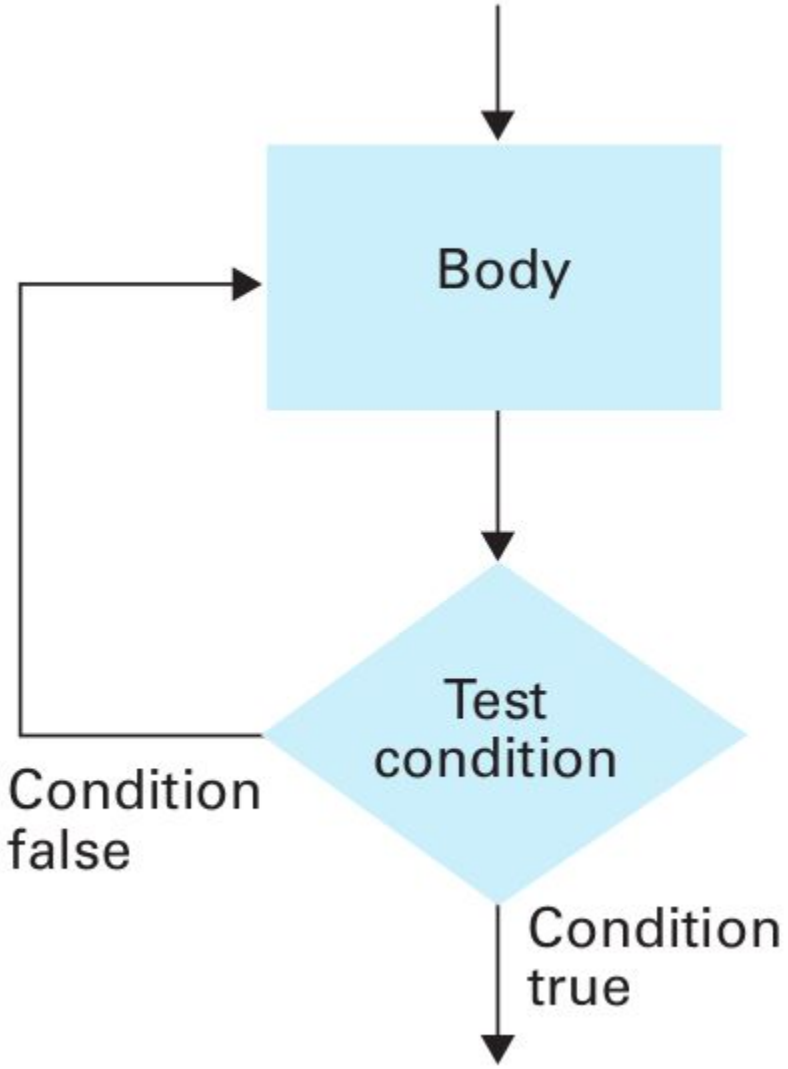
Iterative Structure

While loop structure : Pretest loop



Iterative Structure

Repeat loop structure : Posttest loop



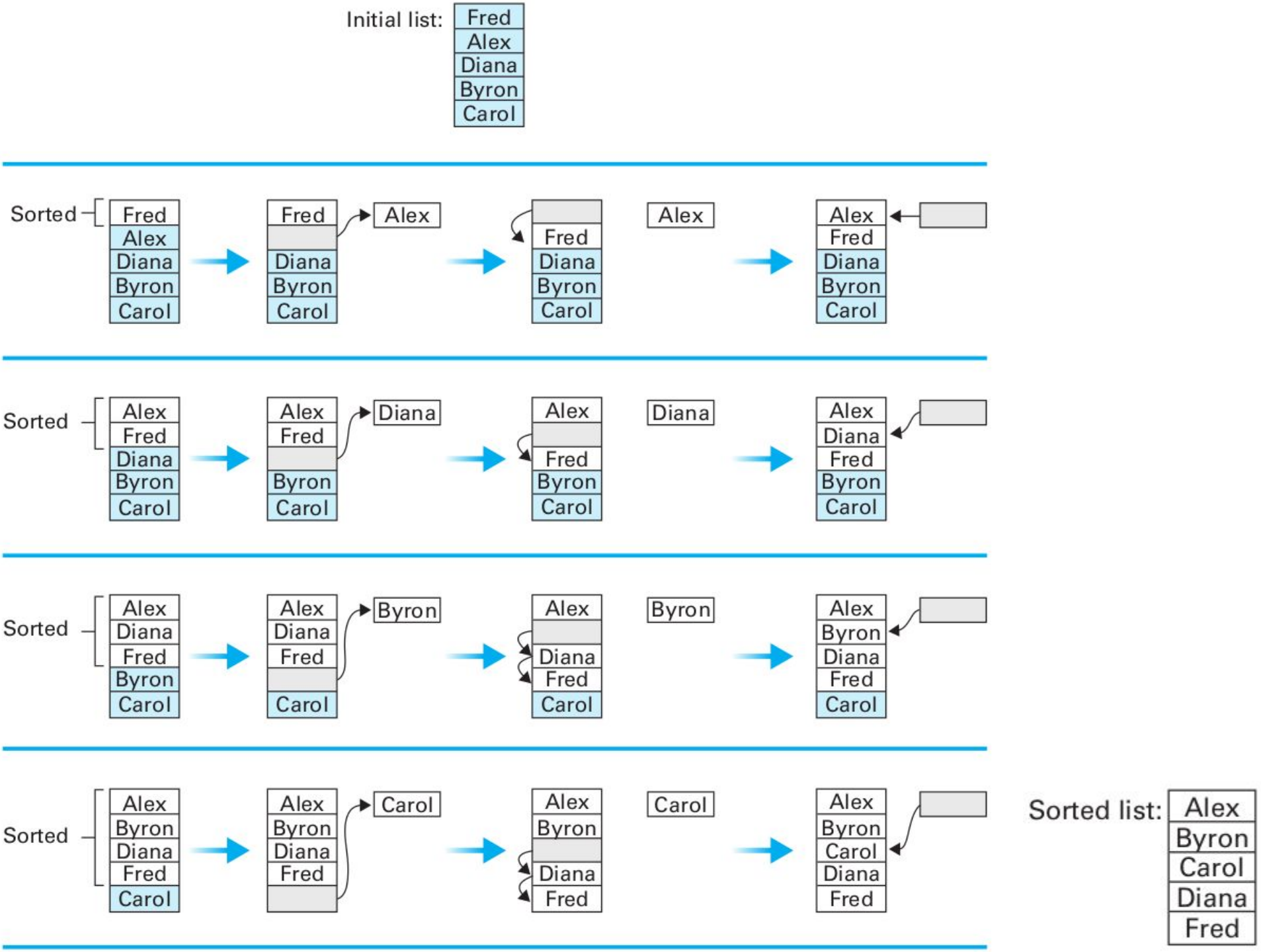
Iterative Structure

Insertion Sort

- to sort the list “within itself.”
  - In order to use the storage space available in an efficient manner

Consider the list of names

- Fred
- Alex
- Diana
- Byron
- Carol


Initial list:
Fred
Alex
Diana
Byron
Carol
Sorted
Alex
Diana
Byron
Carol
Sorted list:

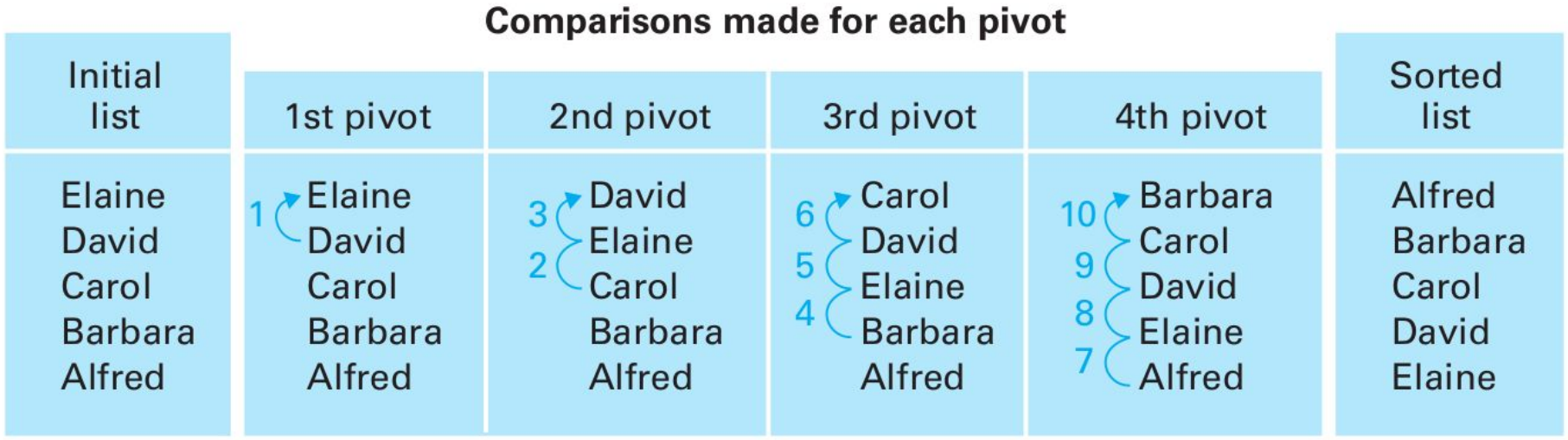
Iterative Structure

Insertion Sort

```
def Sort (List):
  N = 2
  while (the value of N does not exceed the length of List):
    Select the Nth entry in List as the pivot entry.
    Move the pivot entry to a temporary location leaving
      a hole in List.
    while (there is a name above the hole and that name
        is greater than the pivot):
      Move the name above the hole down into the hole
        leaving a hole above the name.
    Move the pivot entry into the hole in List.
    N = N + 1
```

Iterative Structure

Insertion Sort



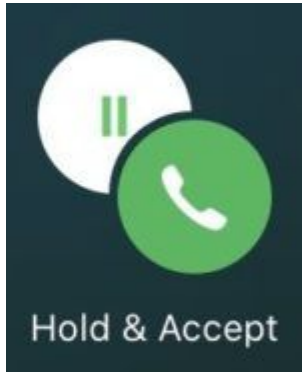
Recursive Structure

Recursive structures provide an alternative to the loop paradigm for implementing the repetition of activities.

- Whereas a loop involves repeating a set of instructions in a manner in which the set is completed and then repeated,
- recursion involves repeating the set of instructions as a subtask of itself.

As an analogy, consider the process of conducting telephone conversations with the call waiting feature.

- An incomplete telephone conversation is set aside
  - while another incoming call is processed.
- Two conversations take place.
- Not performed one-after-the-other
  - (as in a loop structure)
  - but instead one is performed within the other.



Recursive Structure

The Binary Search Algorithm

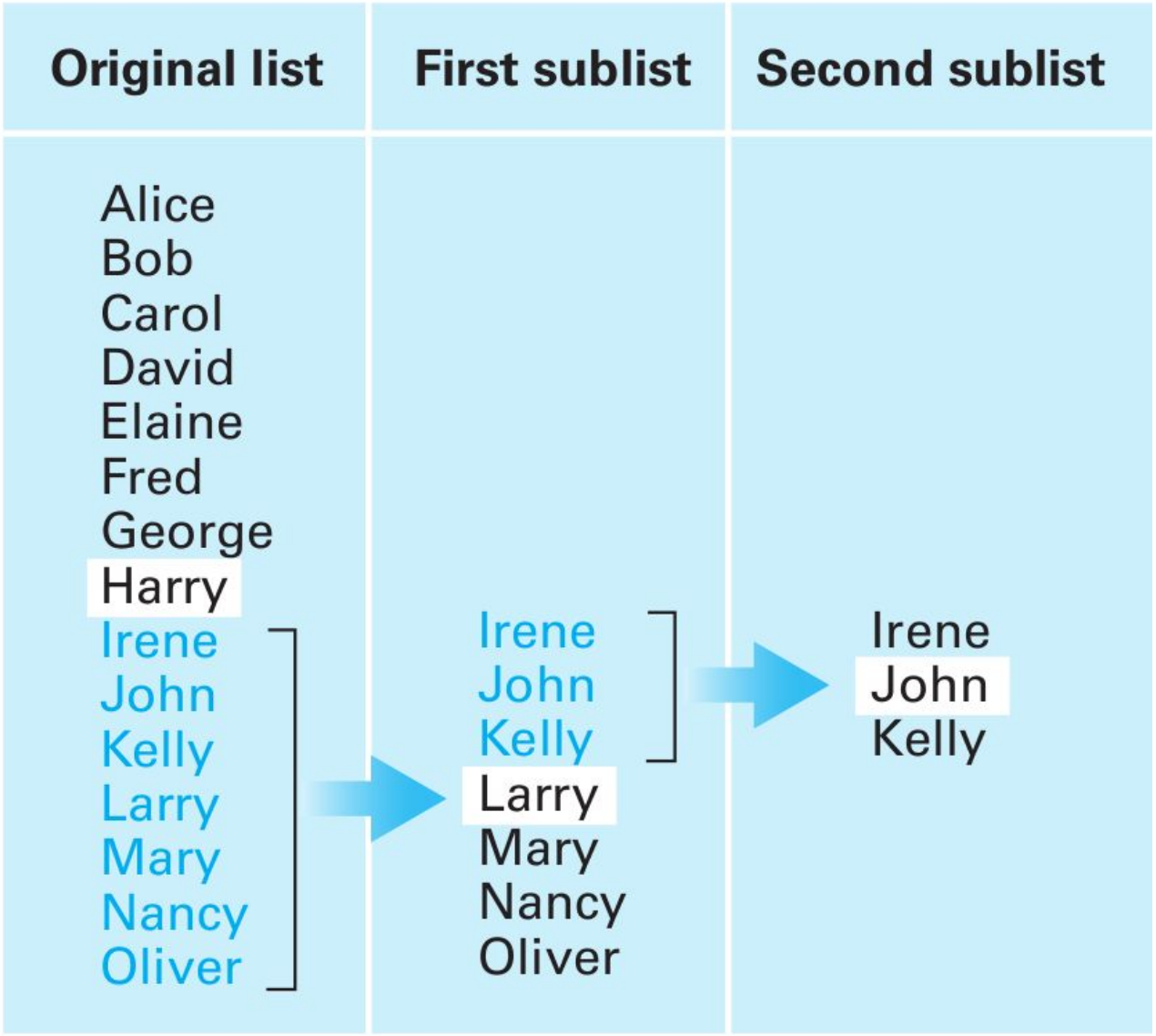
Let us again tackle the problem of searching to see whether a particular entry is in a sorted list,

but this time: searching a dictionary

In this case we do not perform a sequential entry-by-entry or even a page-by-page procedure.

- Rather, we begin by opening the directory to a page in the area where we believe the target entry is located.
  - If we are lucky, we will find the target value there; otherwise, we must continue searching.
  - We will narrow our search considerably.

Recursive Structure



Recursive Structure

```
if (List is empty):
  Report that the search failed.
else:
  TestEntry = the "middle" entry in the List
  if (TargetValue == TestEntry):
    Report that the search succeeded.
  if (TargetValue < TestEntry):
    Search() the portion of List preceding TestEntry for TargetValue,
      and report the result of that search.
  if (TargetValue > TestEntry):
    Search() the portion of List following TestEntry for TargetValue,
      and report the result of that search.
```

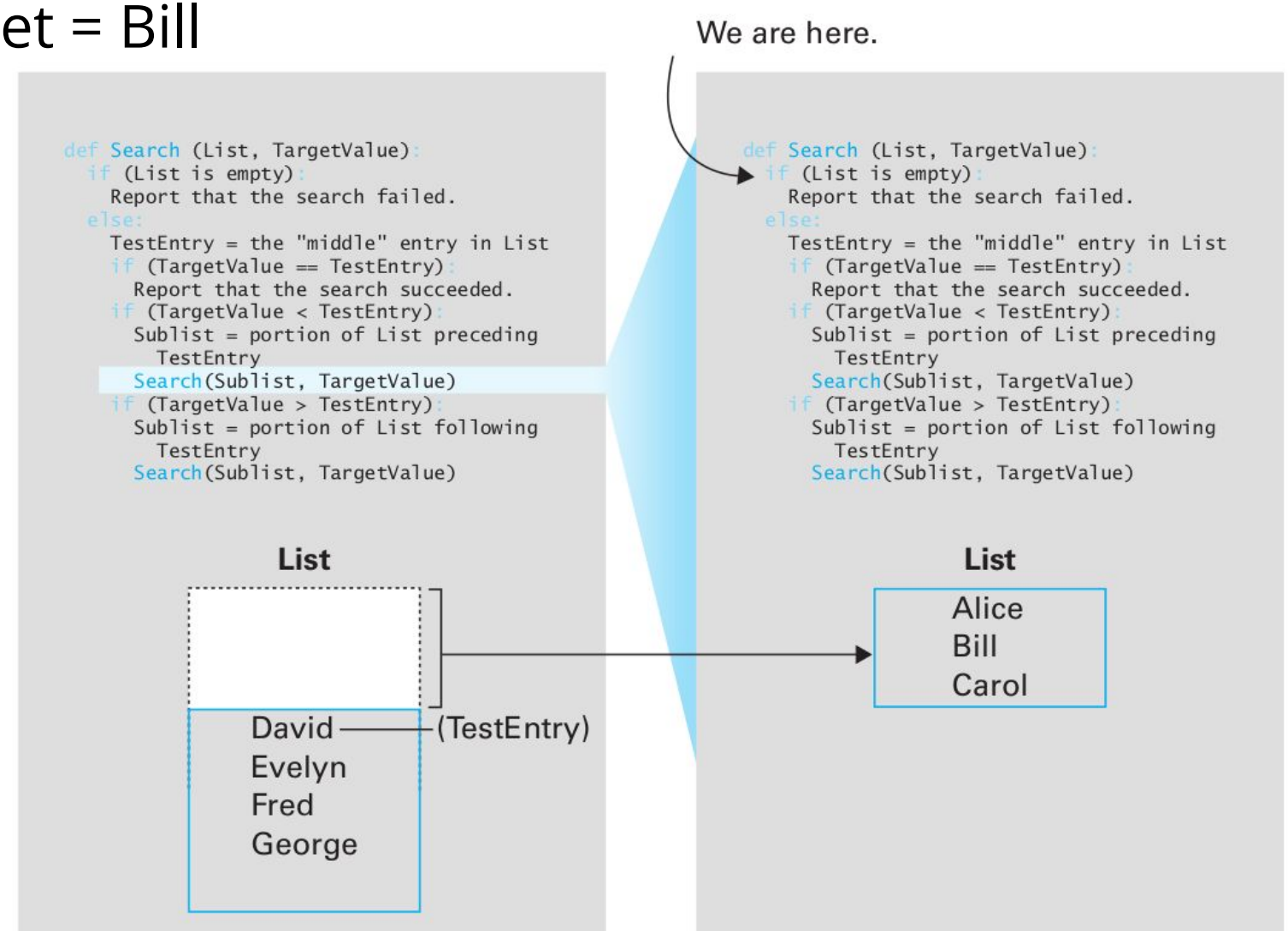
Recursive Structure

```
def Search(List, TargetValue):
  if (List is empty):
    Report that the search failed.
  else:
    TestEntry = the "middle" entry in List
    if (TargetValue == TestEntry):
      Report that the search succeeded.
    if (TargetValue < TestEntry):
      Sublist = portion of List preceding
        TestEntry
      Search(Sublist, TargetValue)
    if (TargetValue > TestEntry):
      Sublist = portion of List following
        TestEntry
      Search(Sublist, TargetValue)
```

Recursive Structure

List = [Alice, Bill, Carol, David, Evelyn, Fred, George]

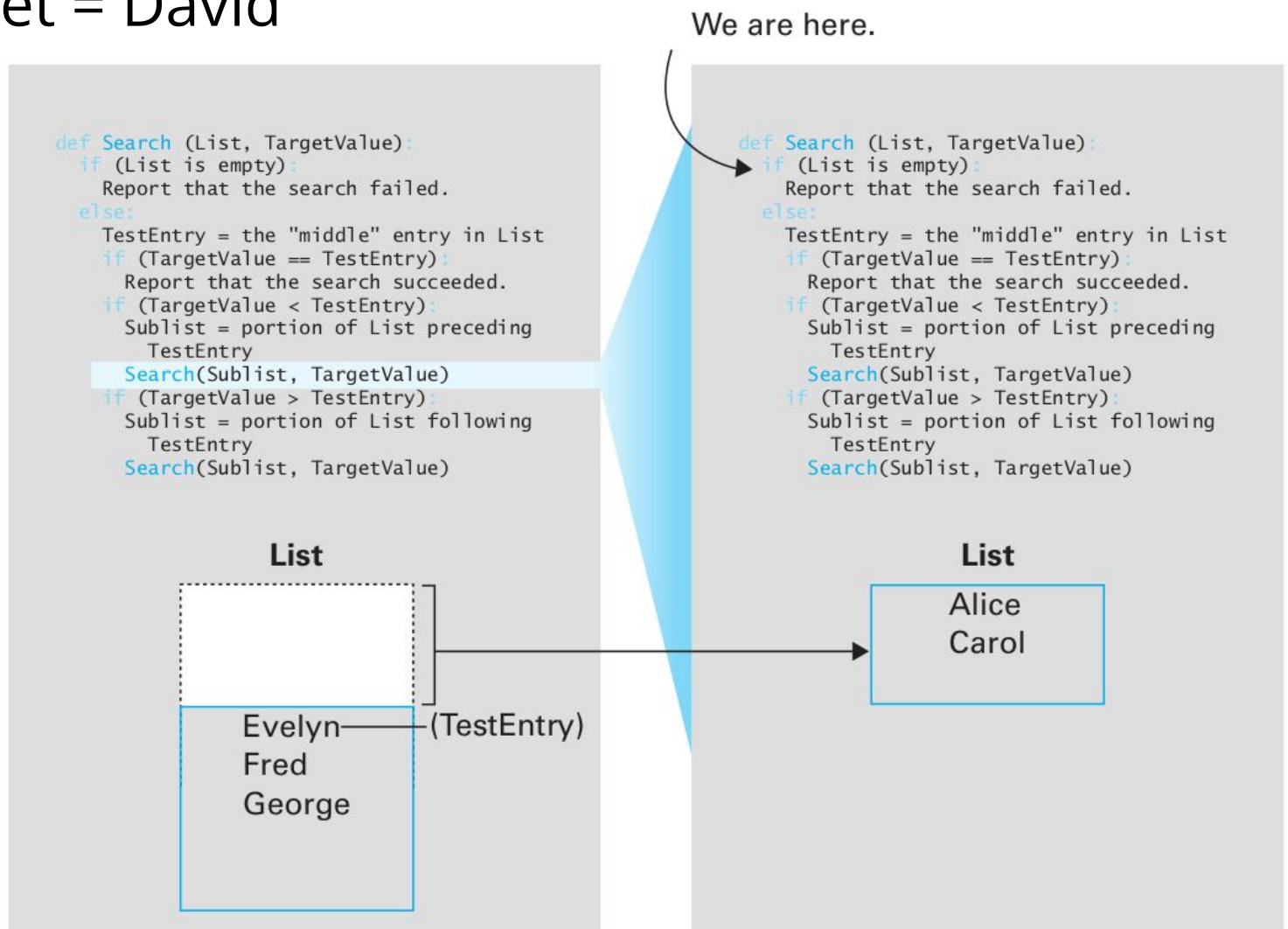
Target = Bill



Recursive Structure

List = [Alice,, Carol, Evelyn, Fred, George]

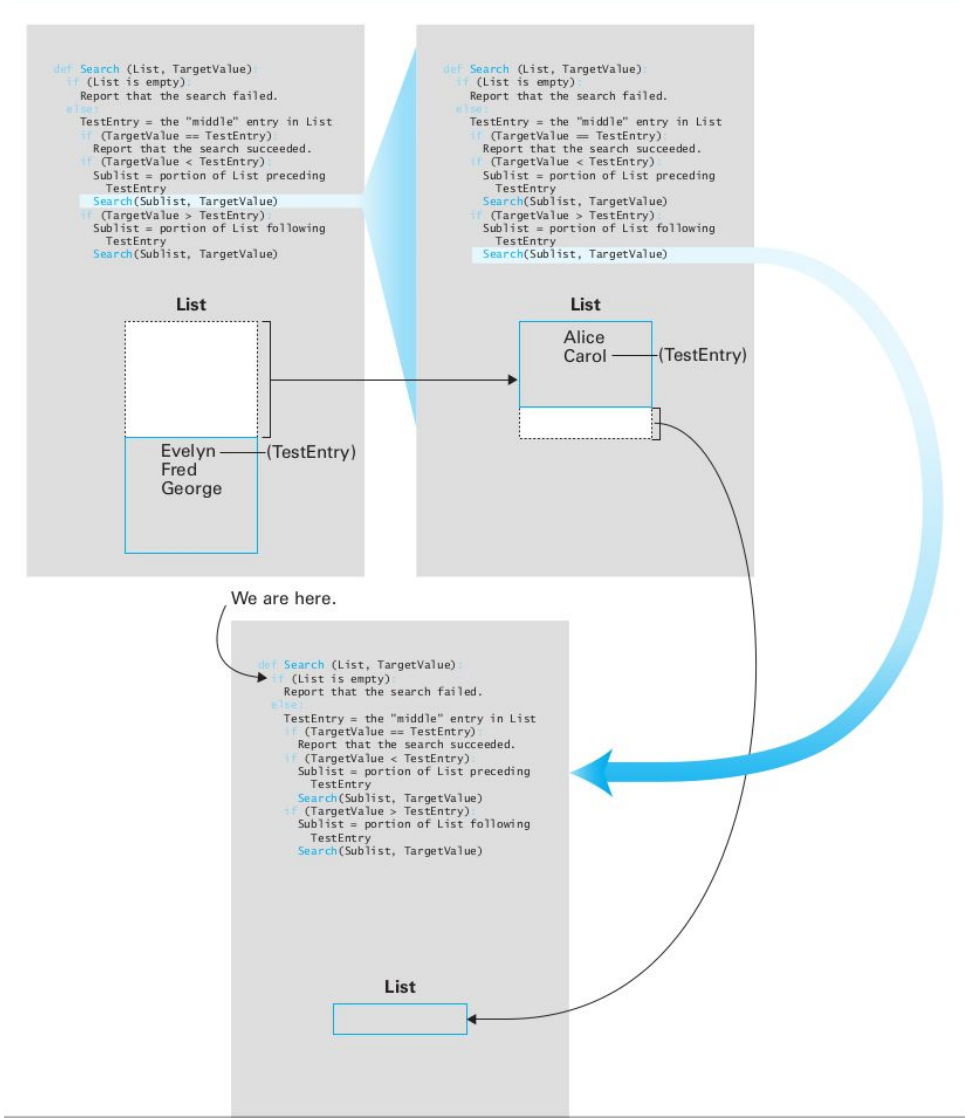
Target = David



Recursive Structure

List = [Alice,, Carol, Evelyn, Fred, George]

Target = David



Extra Reading

Brookshear JG, Smith D, Brylow D.

“Computer science: an overview”.

Read the following section in the textbook.

Chapter: 5.6 Efficiency and Correctness

Summarize the section in 2 page (no figure should be included.)

Template for summarization assignments:

https://docs.google.com/document/d/1i1tU3AdIvPgRxqzGmXrsnVVCRXws5V3caDTCPMbPJ_0/edit?usp=sharing

Research Themes

1. Algorithm efficiency
2. Search algorithms
3. Hashing algorithms
4. Sorting algorithms
5. Dynamic programming
6. NP-hard problems

COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr