Computer Programming I

Binnur Kurt, PhD

BAU
Bahçeşehir University

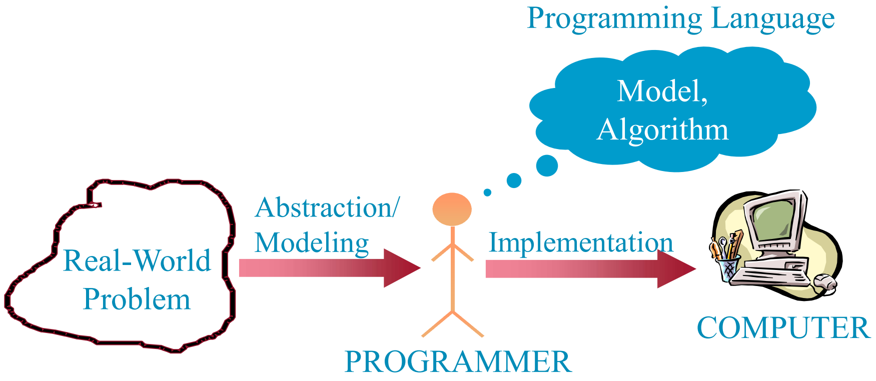
binnur.kurt@rc.bau.edu.tr

MODULE 2

INTRODUCTION TO PROGRAMMING

What is Programming?

- After abstraction, you work only on the model, you leave the problem
- So, you must make sure that the model captures all the required details of the real problem
- You solve the model, not the problem



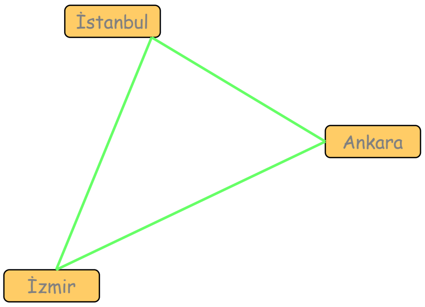
Computer Programs

- how to represent/model the problem?
  - computers work on numbers
  - program about the highways in Turkey
  - entities: cities and roads
  - representing a city: name, latitude, longitude
  - Data Structures Course deals with different modeling tools to represent the problem in computers
  - But we will use simple tools such as single variables, (static/dynamic) arrays, ADT.
- how to express the solution?
  - We will use simple ones: Flowchart, Pseudo Code

MODELING

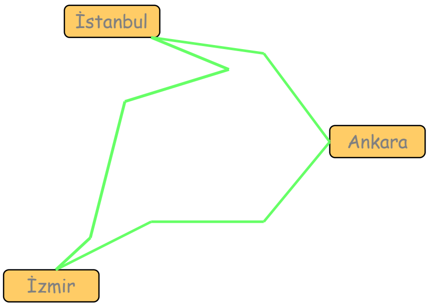
Representing the Problem

- Representing a road: line
  - assume the road is straight
  - start and end cities
- Very easy
- Very inaccurate



Representing the Problem

- Representing a road: consecutive lines
- Very hard
- More accurate → More complicated



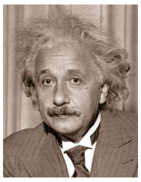
Representation: Model

- FIRST STEP: build a correct/accurate (and feasible) model
- what you are solving is the model, not the problem itself
  - incorrect model → incorrect solution
  - inaccurate model → meaningless solution
  - infeasible model → expensive implementation

Representing the Problem

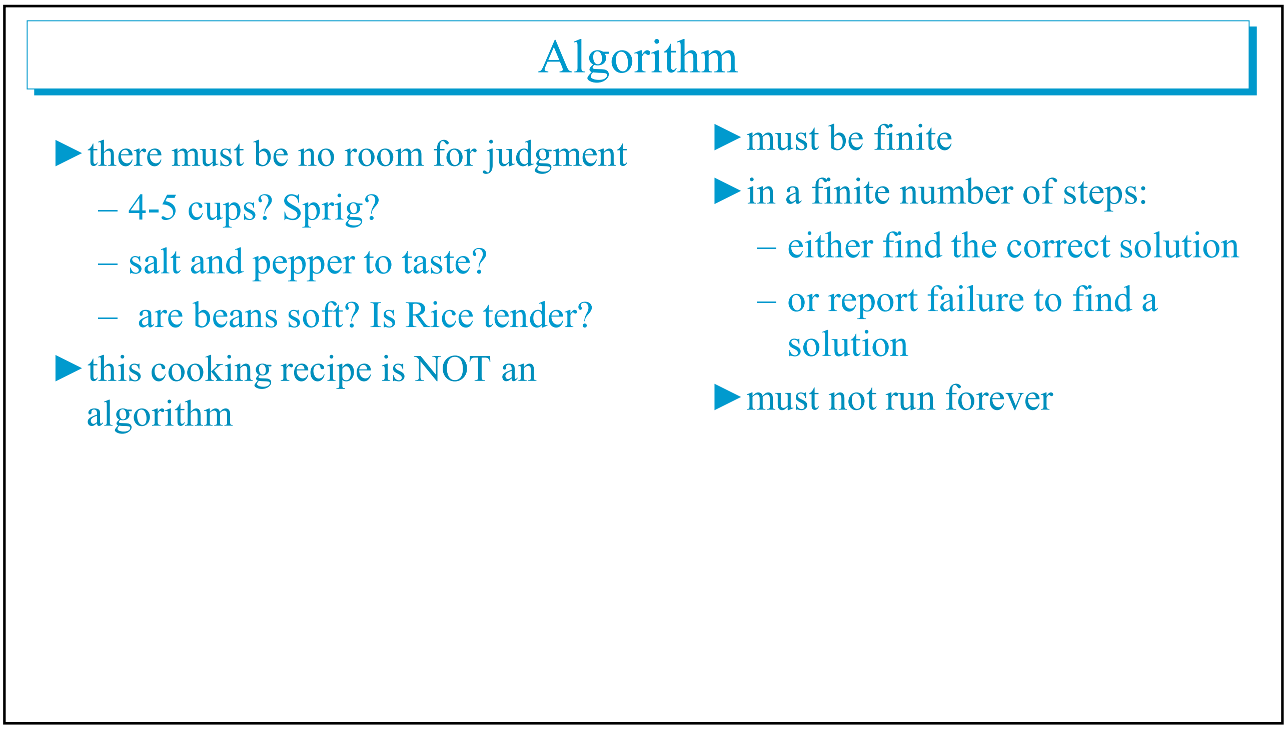
- Representing highways:
  - if you are only interested in total distances, you can use lines
  - if you will talk about "the 274th km of the İstanbul-Ankara highway", you should use consecutive lines

"Everything should be made as simple as possible, but not simpler."



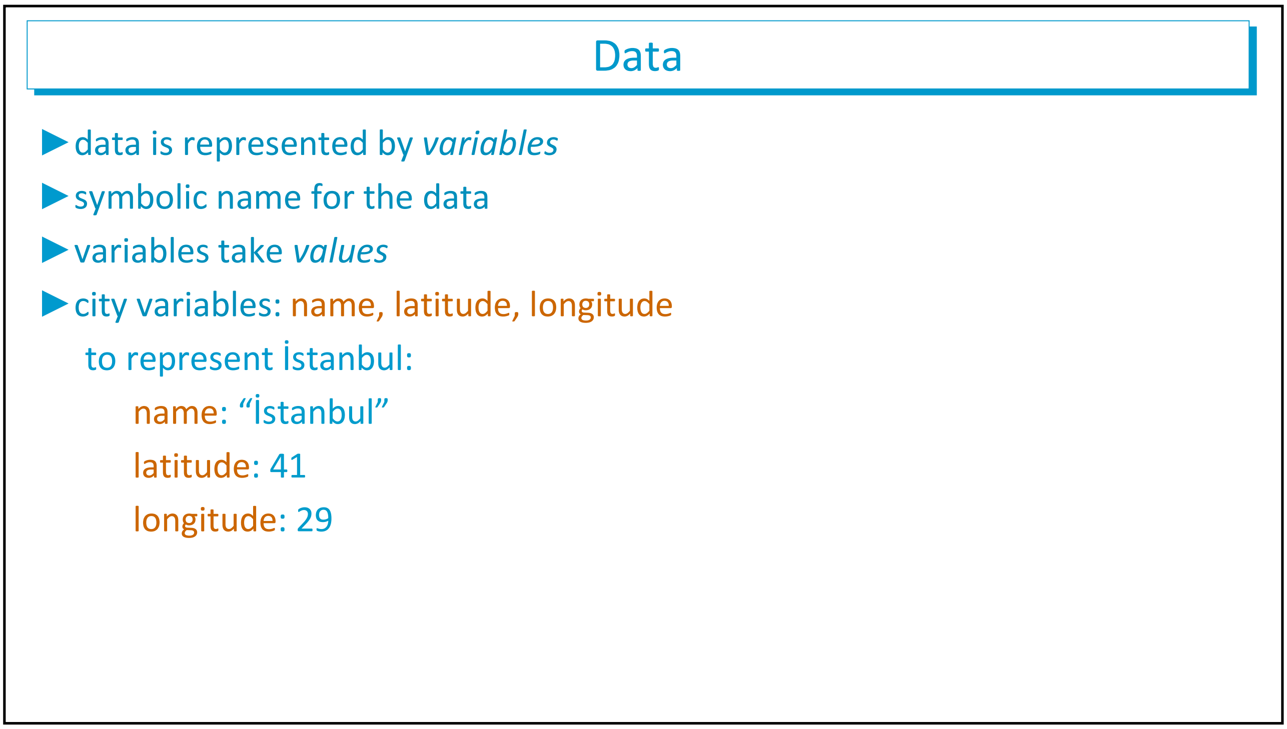
Expressing the Solution

- step-by-step guide to the solution: *algorithm*
- recipe for Jamaican rice and peas:
  - put 1 1/2 cans of beans in 4-5 cups of water
  - add 1/4 can of coconut milk, one sprig of thyme, and salt and pepper to taste
- cook until beans are soft
- smash the bottom of the green onion and add it to the pot along with 2 cups of rice and 1/4 can of coconut milk, and two sprigs of thyme
- remove any access water over 2cm above the rice
- bring to a boil for 5 min
- continue to cook covered until rice is tender



Algorithm

- there must be no room for judgment
  - 4-5 cups? Sprig?
  - salt and pepper to taste?
  - are beans soft? Is Rice tender?
- this cooking recipe is NOT an algorithm
- must be finite
- in a finite number of steps:
  - either find the correct solution
  - or report failure to find a solution
- must not run forever



Data

- data is represented by *variables*
- symbolic name for the data
- variables take *values*
- city variables: name, latitude, longitude

  to represent İstanbul:

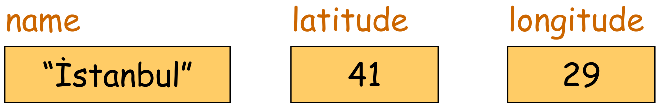
  name: “İstanbul”

  latitude: 41

  longitude: 29

Variables

- ▶ variables are kept in memory
  - – variable is the name of the memory cell
  - – value is the content of the memory cell



name	latitude	longitude
“İstanbul”	41	29

Assignment

- ▶ block structured programs proceed by assigning values to variables
- ▶ notation: latitude ← 41
  - – “store the value 41 in the memory cell named latitude.”
- ▶ left-hand side is a variable
- ▶ right-hand side is an *expression*
  - – a computation that yields a value

Expressions

- can be a single value or variable:
  - 41
  - latitude
- can be combinations of values and variables connected via *operators*:
  - 4 * longitude multiplication operator
  - latitude + longitude addition operator

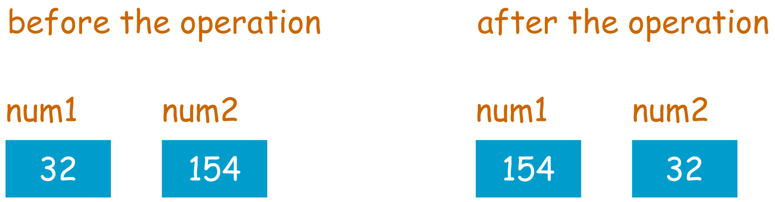
Assignment

- ASSIGNMENT IS NOT EQUALITY!!!
- 41 ← latitude doesn’t make sense
- i ← i + 1 means: increment the value of i by 1
  - if i was 5 before the operation, it will become 6 after the operation
- Mathematically, it would be incorrect:

  0 = 1

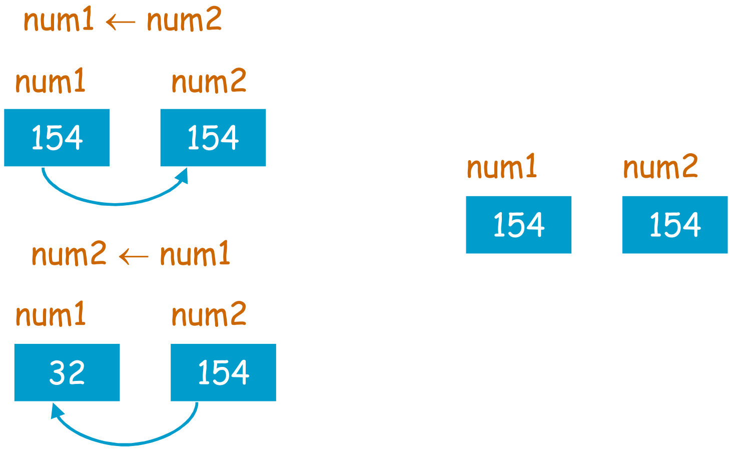
Swap

- swap the values of two variables:



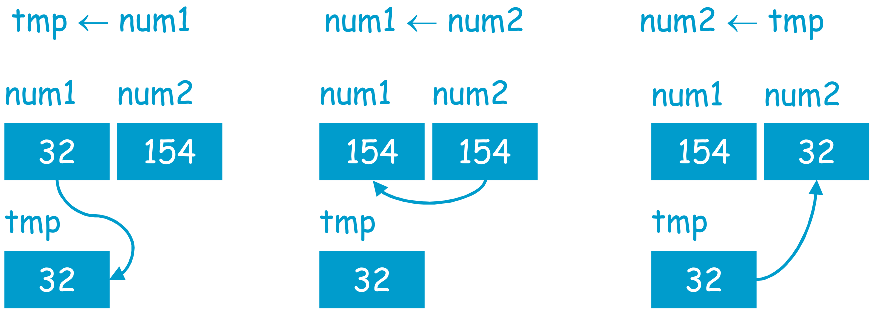
Swap: Incorrect

- num1 ← num2
- num2 ← num1



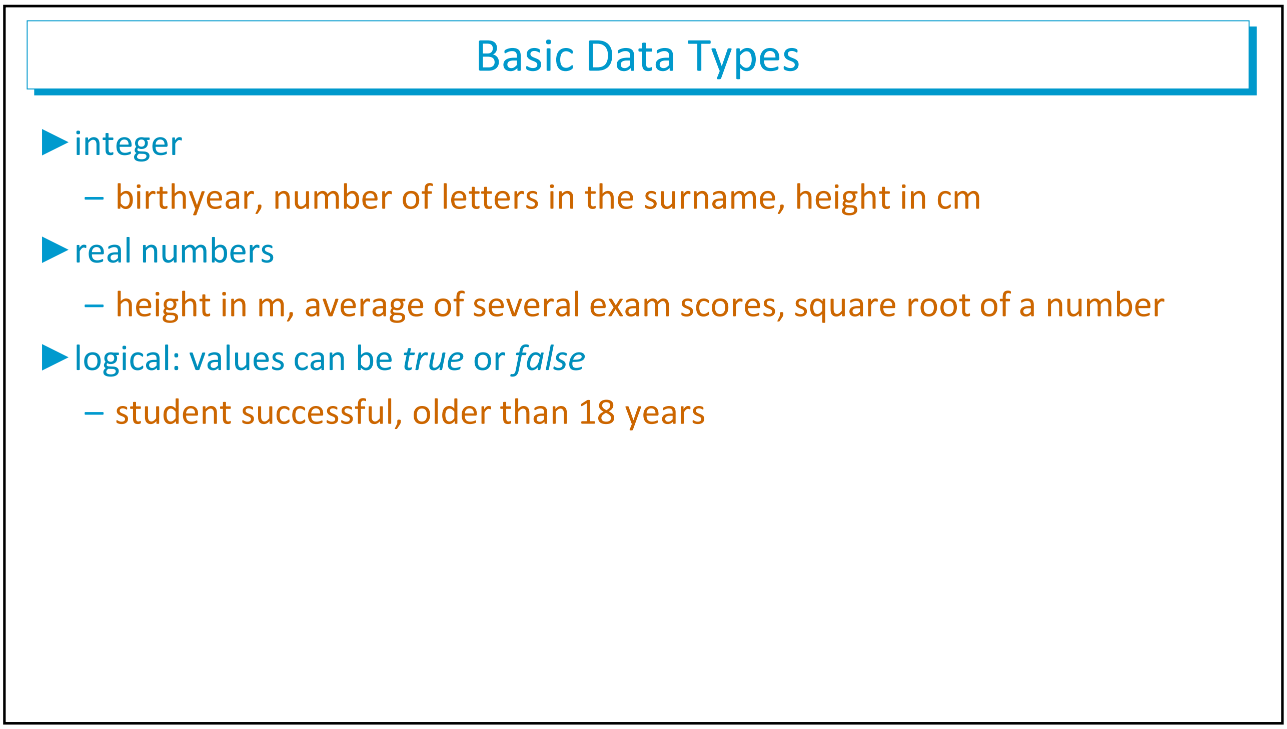
Swap

- tmp ← num1
- num1 ← num2
- num2 ← tmp



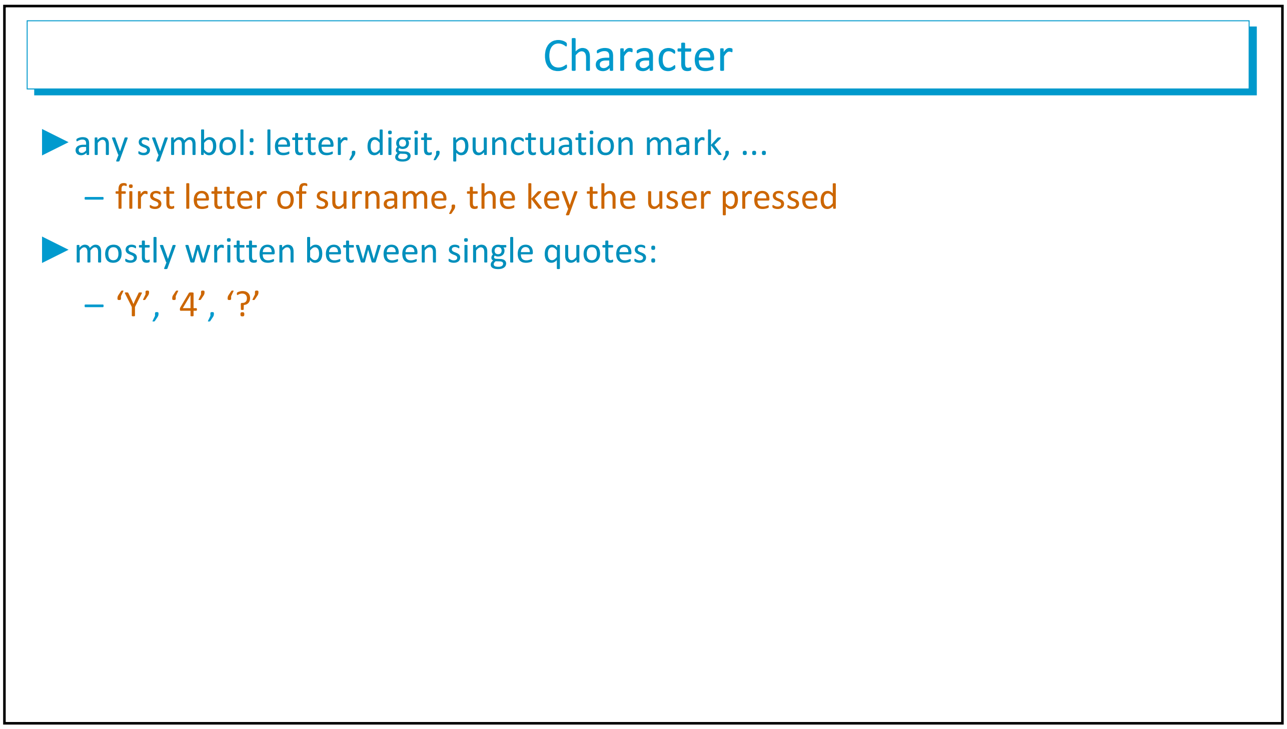
Data Types

- basic data types:
  - integer
  - real number
  - logical
  - character
  - string
- composite data types: record
- vector data types: array



Basic Data Types

- integer
  - birthyear, number of letters in the surname, height in cm
- real numbers
  - height in m, average of several exam scores, square root of a number
- logical: values can be *true* or *false*
  - student successful, older than 18 years

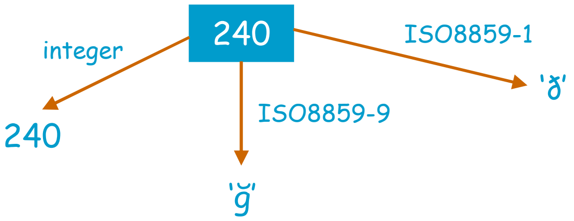


Character

- any symbol: letter, digit, punctuation mark, ...
  - first letter of surname, the key the user pressed
- mostly written between single quotes:
  - ‘Y’, ‘4’, ‘?’

Encoding

- numbers correspond to symbols
- ASCII, ISO8859-X, Unicode



240

integer

240

ISO8859-9

'ğ'

ISO8859-1

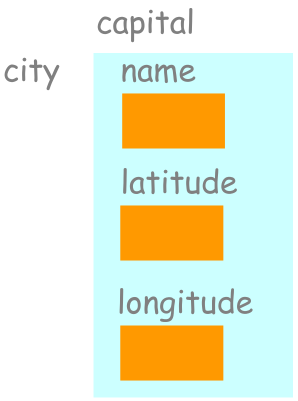
'ð'

Strings

- name, word, sentence, ISBN number, ...
- mostly written between double quotes:
  - “Dennis Ritchie”, “ISBN 0-13-110362-8”
- use numbers if you plan to make arithmetic operations on it:
  - student numbers at ITU: 9-digit numbers
  - will you add/multiply/... student numbers?
  - no sense in using integers, use strings

Composite Data Types

- grouping types to form a new type

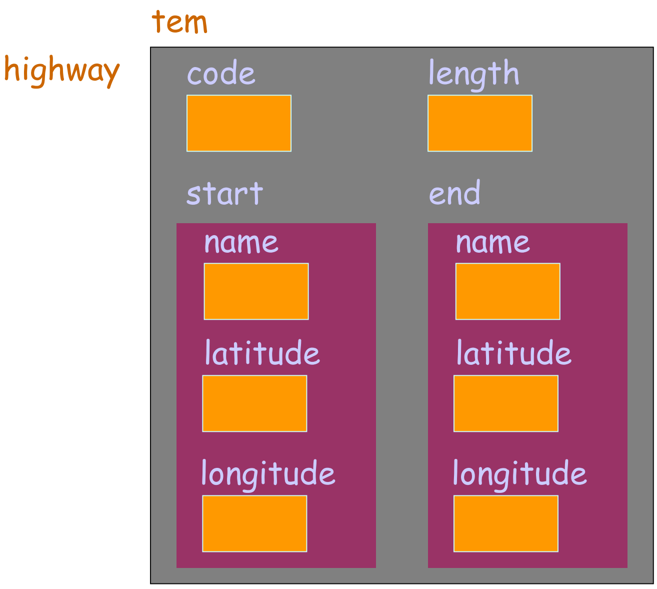


- access:
  - capital.name ← “Ankara”
  - NOT city.name ← “Ankara”

Composite Data Types

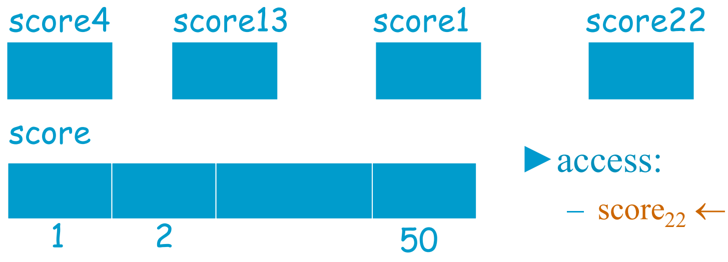
- composite types can be nested

- access:
  - tem.code ← “E-6”
  - tem.end.name ← “Ankara”



Vector Data Types

- grouping elements of the same type
  - exam scores for a 50 student class:
    - 50 integer variables: score1, score2, ..., score50
    - an integer array with 50 elements: score



- access:
  - $score_{22} \leftarrow 95$

Strings

- strings are usually arrays of characters

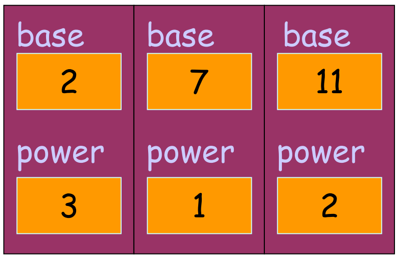
fullname

'D'	'e'	'n'	'n'	'i'	's'	' '	'R'	'i'	...
1	2	3	4	5	6	7	8	9	10

Arrays of Records

- represent the factors of a number:
  - an array where each member is a factor
  - a factor is a record of a base and a power

factors

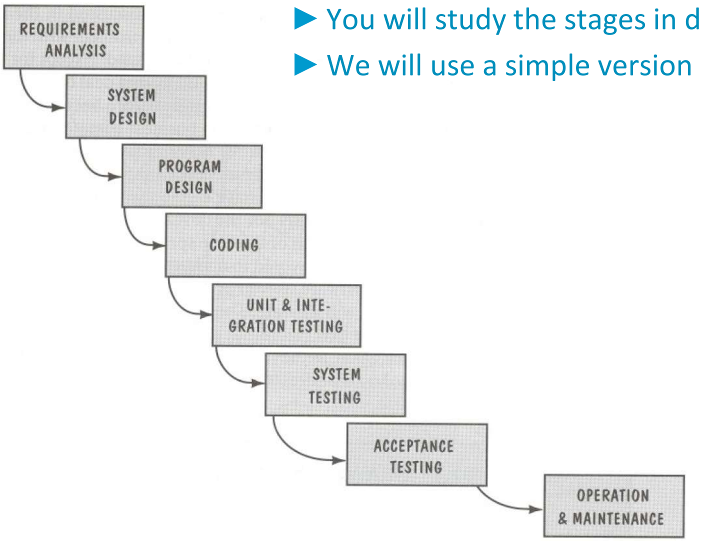


base	base	base
2	7	11
power	power	power
3	1	2

- access:
  - $factors_2.base \leftarrow 7$

PROGRAM DEVELOPMENT

Development Stages



- ▶ You will study the stages in detail: Software Engineering Course
- ▶ We will use a simple version

Development Stages

- ▶ design: on “paper”
  - – model, algorithm
  - – which programming language?
  - – software engineering
- ▶ coding: writing the program
  - – source code
  - – editor
- ▶ testing: does the program work as expected?
  - – scenarios
- ▶ debugging: finding and correcting the errors
  - – debugger

Phase 1: Define the Problem

- In this phase, create a program specification.
- Specification defines the input data, processing that occurs, and output data.
- This phase requires cooperative work with the programmer and the problem owner.
- Other phases are only for the programmer.

Phase 2: Design the Program

- Program design is done from the specification
- Identify the main components of a program and how they work together
  - Identify the main goal of a program
  - Then, break it down into sub-goals
  - Keep refining it until the program is designed
- Algorithms are created during design. An algorithm can be in one of the following forms:
  - Flow Chart
  - Pseudo Code

Phase 3: Coding the Program

- After designing the program, it is coded in the language chosen (C++ language)
- Coding is also known as implementation
- A compiler software must be used to compile the program source code.
- Compiler:
  - Compiler checks your source code and finds all syntax errors, such as mistyping
  - It translates the code all at once and produces “object code.”
  - Then the Linker uses the “object code” to create the “executable code.”
  - Executable code is a binary file which means you can just Double-click it and it runs

Phase 4: Testing and Debugging the Program

- You should run your program and see how it works
- Testing is the final phase before releasing the program
  - Various input data values should be used
  - Observed output values should be compared to the expected output values
- In this phase, you should find any logic errors such as wrong calculation, or using wrong input data
- Errors are called “bugs” informally. The process of finding bugs is called “debugging”

ALGORITHM: EXPRESSING THE SOLUTION

Algorithms

- Algorithms are created for program design.
- An algorithm is a set of steps for carrying out a task.
- It is a step-by-step solution to the given problem.
- An algorithm can be represented with one of the following methods:
- Flow Chart
  - Is composed of standard shapes and symbols
  - It gives the solution steps to the problem
- Pseudo Code
  - Is an alternative to flow chart
  - Constrained form of English to produce steps involved in a problem solution

Algorithms

- Pseudo code
  - is not a programming language, so not prepared for compilation
  - contains both some structural features of a programming language and a speaking language such as English
  - is easily understandable by non-programmers
  - aims at defining a solution to a problem
- Rules for writing a pseudo-code:
  - Use clear and proper English (or Turkish)
  - Use simple and short sentences (avoid compound sentences)
  - Each sentence is to indicate an action

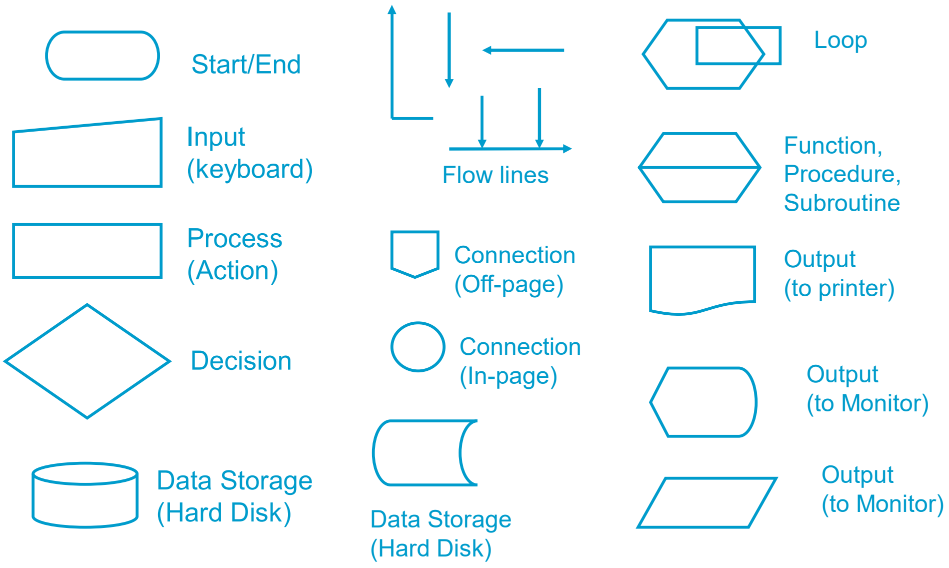
Flow Chart

- A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to get the solution to a problem.
- Flowcharts are drawn in the first stages of program development.
- Flowcharts facilitate communication between programmers and non-programmer people.
- Flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems.
- Once the flowchart is drawn, it becomes easy to write the program in any high-level language such as Python.

Flowcharts

- ▶ Describe algorithms
- ▶ Elements:
  - – box: an operation
  - – arrow: flow direction
  - – diamond: decision point
  - – parallelogram: input/output operation

Flow Chart Symbols



Start/End

Input
(keyboard)

Process
(Action)

Decision

Data Storage
(Hard Disk)

Flow lines

Connection
(Off-page)

Connection
(In-page)

Data Storage
(Hard Disk)

Loop

Function,
Procedure,
Subroutine

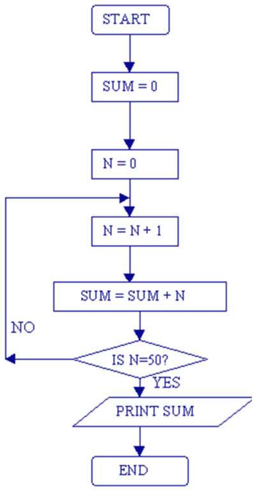
Output
(to printer)

Output
(to Monitor)

Output
(to Monitor)

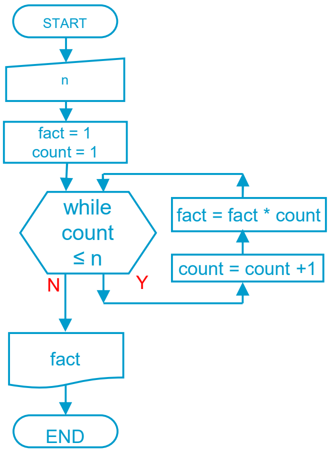
Flowchart Example

- ▶ This flowchart finds the sum of the first 50 natural numbers.
- ▶ Example: 1+2+3+4+…+50



Flowchart Example

- ▶ This flowchart calculates the factorial of a given number n.
- ▶ (n! = 1*2*3*4*.....*n)



Flowcharts

- find the maximum score in an exam with 50 students
  - represent exam scores by a 50-element integer array (variable: score)
  - represent maximum score by an integer (variable: max)

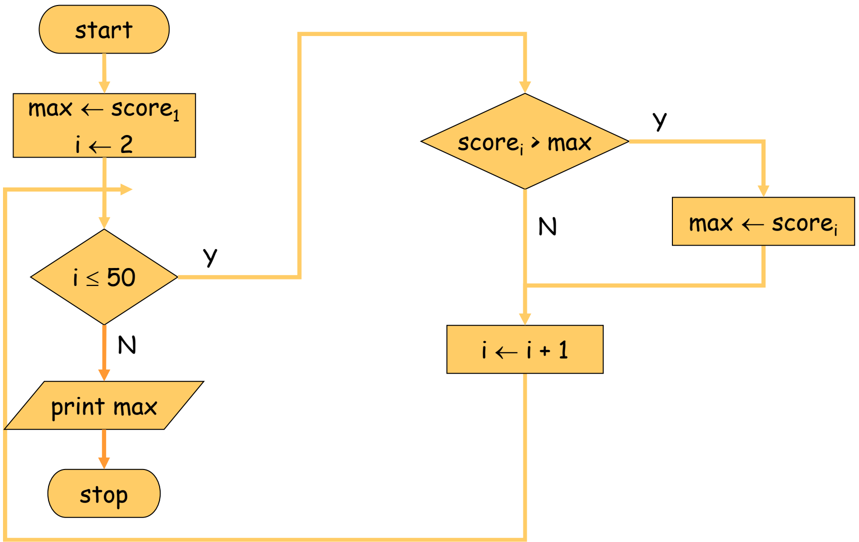
1. choose the first score as maximum
2. if there are more students, go to step 3, else go to step 5
3. if the next score is higher than the maximum, choose it as maximum
4. proceed to the next student
5. print the maximum

Flowcharts

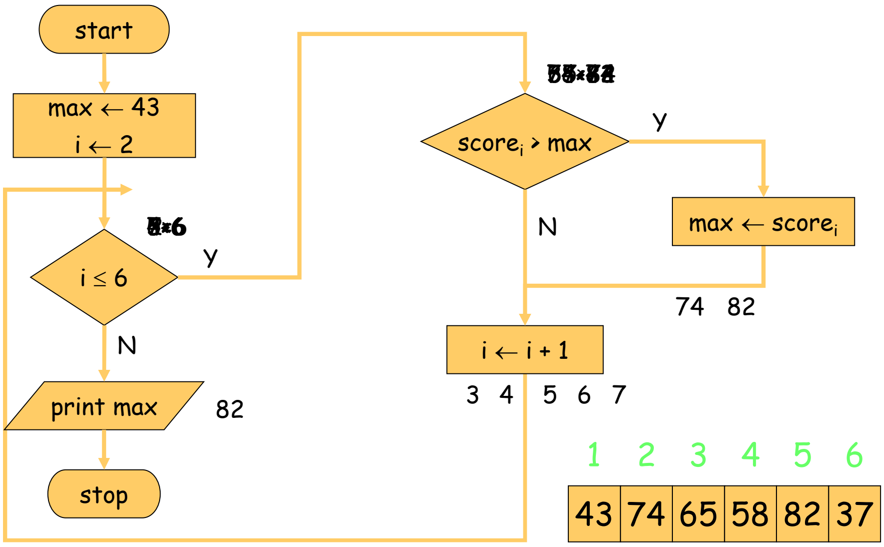
- representation problem:
  - more students? Next score?
  - counter variable: i

```
1. $max \leftarrow score_1$, $i \leftarrow 2$
2. if $i \leq 50$ go to step 3 else go to step 5
3. if $score_i > max$
   then $max \leftarrow score_i$
4. $i \leftarrow i + 1$ and go to step 2
5. print max
```

Flowcharts



Flowcharts



1	2	3	4	5	6
43	74	65	58	82	37

Flowcharts

- using tables to represent flow:

max	i	$i \leq 6$	$score_i > max$
43	2	T (2 < 6)	T (74 > 43)
74	3	T (3 < 6)	F (65 < 74)
	4	T (4 < 6)	F (58 < 74)
	5	T (5 < 6)	T (82 > 74)
82	6	T (6 = 6)	F (37 < 82)
	7	F (7 > 6)	

Flowcharts

- number guessing game:
  - one player picks a number (target) between lower and upper bounds
  - the other player makes a guess:
    - if the guess is bigger than the target, the picker says “smaller.”
    - if the guess is smaller than the target, the picker says “bigger.”
    - game ends when guess = target

Flowcharts

- Algorithm I:
  - start with lower, increment by 1 until found

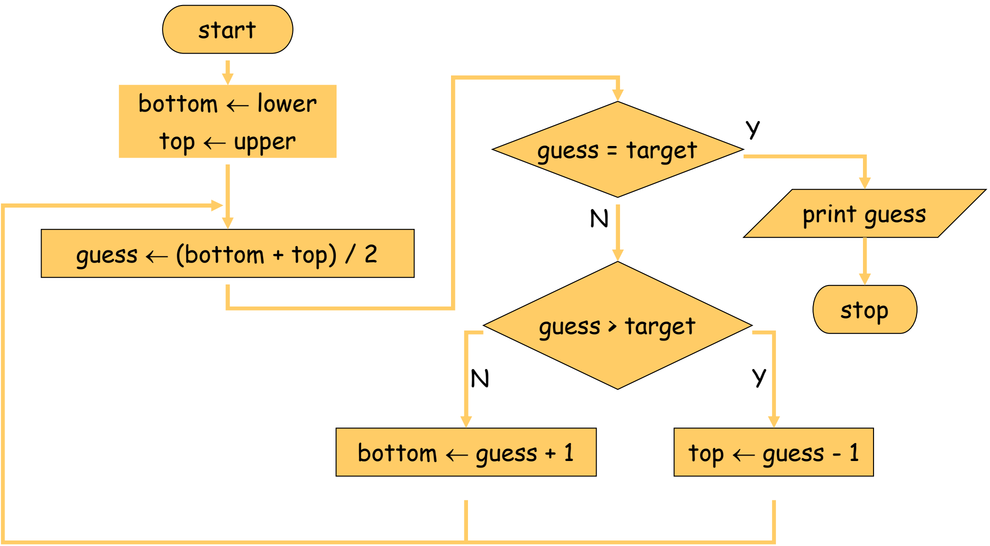
```
1. guess ← lower
2. if guess = target stop
3. guess ← guess + 1 and go to step 2
```

- Algorithm II:
  - try the number in the middle
  - if “smaller,” narrow the search to the bottom half
  - if “bigger,” narrow the search to the top half

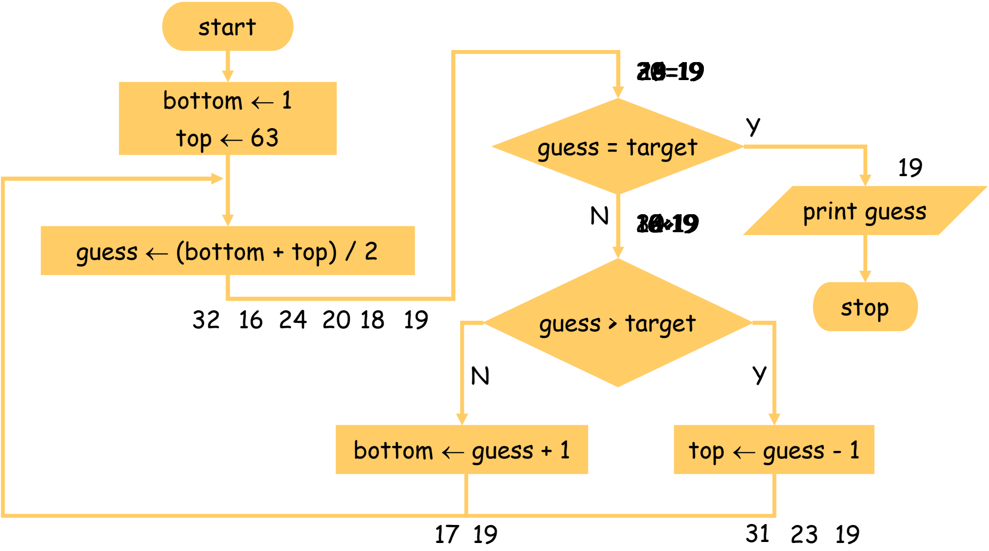
Flowcharts

```
1. bottom ← lower, top ← upper
2. guess ← (top + bottom) /2
3. if guess = target stop
4. if guess > target then top ← guess - 1
   otherwise bottom ← guess + 1
   and go to step 2
```

Flowcharts



Flowcharts



Flowcharts

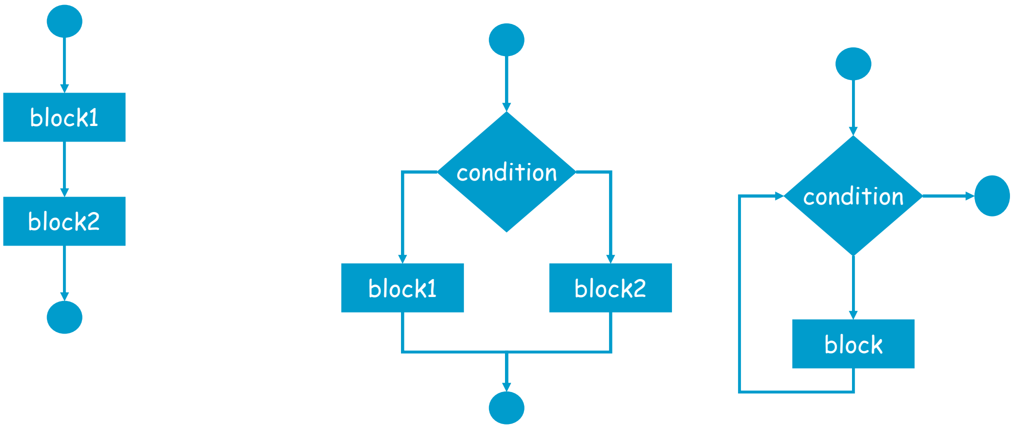
bottom	top	guess	guess=target
1	63	32	
	31	16	F (32 > 19)
17		24	F (16 < 19)
	23	20	F (24 > 19)
	19	18	F (20 > 19)
19		19	F (18 < 19)
			T (19 = 19)

Comparing Algorithms

- number guessing: which algorithm is better?
- speed:
  - worst case: first one 63, second one 6
  - average case: first one 32, second one ~5
- size: the second one requires two more variables

Block Structures

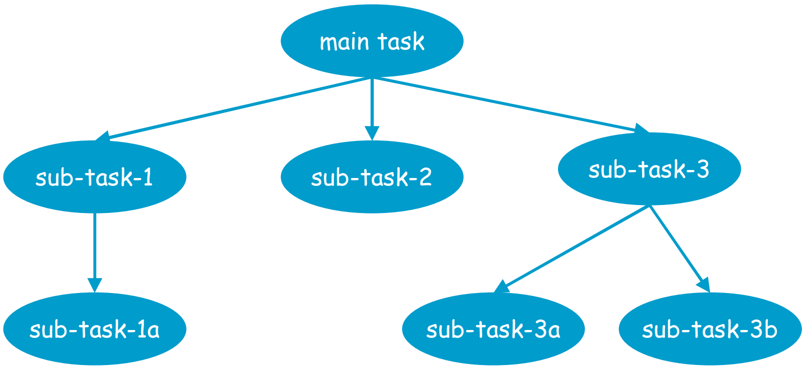
- ▶ sequence
- ▶ repetition
- ▶ selection



Abstraction

- ▶ divide the main task to sub-tasks
- ▶ consider each sub-task as the main task and divide it into sub-sub-tasks, ...
  - – divide-and-conquer
- ▶ top-down design
- ▶ each task is implemented by a procedure (in a Python function)

Abstraction



Abstraction

- procedures are only interested in WHAT sub-procedures are doing, not HOW they are doing it
- smaller units are easier to manage
- maintaining is easier
  - if the HOW of the sub-procedure changes, the super-procedure is not affected

Abstraction

- procedures should be general:
  - instead of “find the maximum score in the final exam of AIN1001”
  - do “find the maximum of any array.”
  - you can use this to find the “maximum score in the final exam of AIN1001”
  - and also to find the “maximum shoe size of the LA Lakers players.”

Parameters

- which data will the procedure work on?
  - input parameter:
    - the scores in the final exam of BIL105E
    - the shoe sizes of the LA Lakers players
- what value will the procedure produce?
  - output parameter:
    - maximum score
    - maximum shoe size

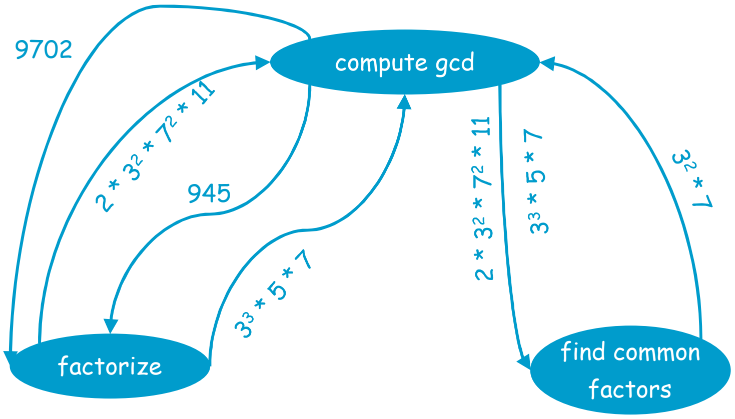
Abstraction

- find the greatest common divisor (gcd) of two numbers:
  1. decompose the first number to its prime factors
  2. decompose the second number to its prime factors
  3. find the common factors of both numbers
  4. compute the gcd from the common factors

Abstraction

- sample numbers: 9702 and 945
  1. $9702 = 2 * 3^2 * 7^2 * 11$
  2. $945 = 3^3 * 5 * 7$
  3. $3^2 * 7$
  4. 63

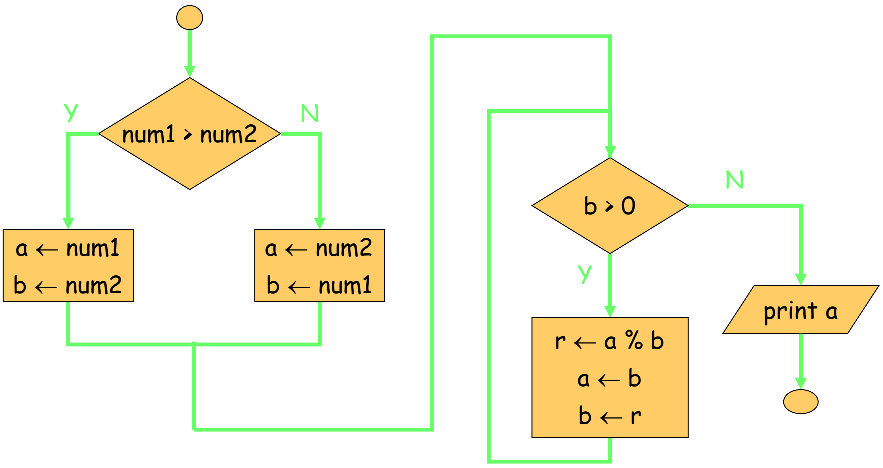
Abstraction



Euclid's Algorithm

- finding the greatest common divisor (gcd) of two numbers: Euclides algorithm
  - let a be the bigger number and b the smaller number
  - the gcd of a and b is the same as the gcd of b and a % b

Euclid's Algorithm



Euclid's Algorithm

a	b	r
9702	945	252
945	252	189
252	189	63
189	63	0

Comparing Algorithms

- Algorithm I:
  - hard to factorize numbers
  - easier to compute the gcd/lcm of more than two numbers?
- Algorithm II (Euclides):
  - very fast
  - very easy to implement

Input

- most programs read the data from outside:
  - ask the user: get from the keyboard
  - read from a file
  - read from the environment: get the temperature of the room via a sensor
- input commands transfer the value read from outside to a variable

Output

- what to do with the produced results?
  - tell the user: print it on the screen
  - write it to a file or printer
  - send to the environment: control the valve of a gas pipe
- output commands send results to output units
- error messages to error unit

Program Types

- console / command line / text mode programs:
  - read inputs
  - process data and produce results
  - show outputs

- graphical programs are event-driven:
  - prepare the environment (windows, buttons, ...)
  - wait for events (mouse click, key press, ...)
  - respond to events

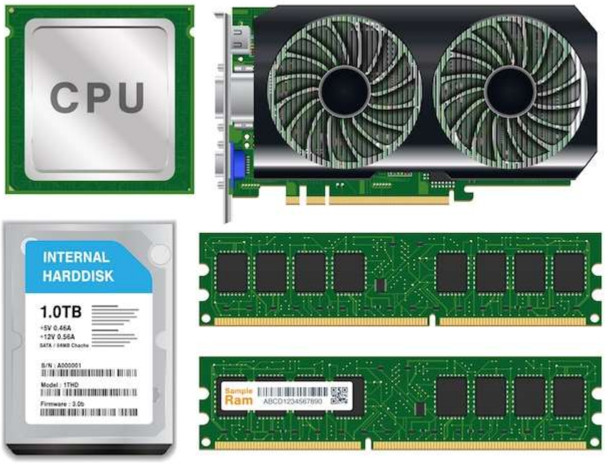
Errors

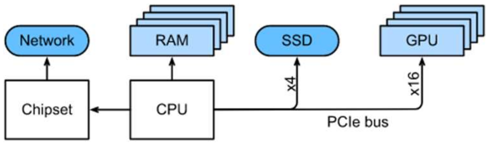
- syntax errors
  - not conforming to the rules of the language
- logical errors
  - division by zero

Evaluating Programs

- efficiency
  - speed
  - hardware requirements
- portability
  - can it run on another platform without much change?
  - source code portability
- understandability
  - can others (or you) understand your code?
- ease of maintenance
  - can new features be added?
- robustness
  - can it tolerate user errors?

Computer Hardware



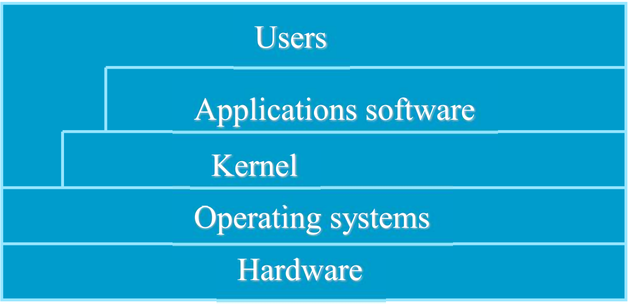


What is an Operating System?

- An elephant is a mouse with an operating system.

What is an Operating System?

- ►A piece of software that provides a convenient, efficient environment for the execution of user programs.



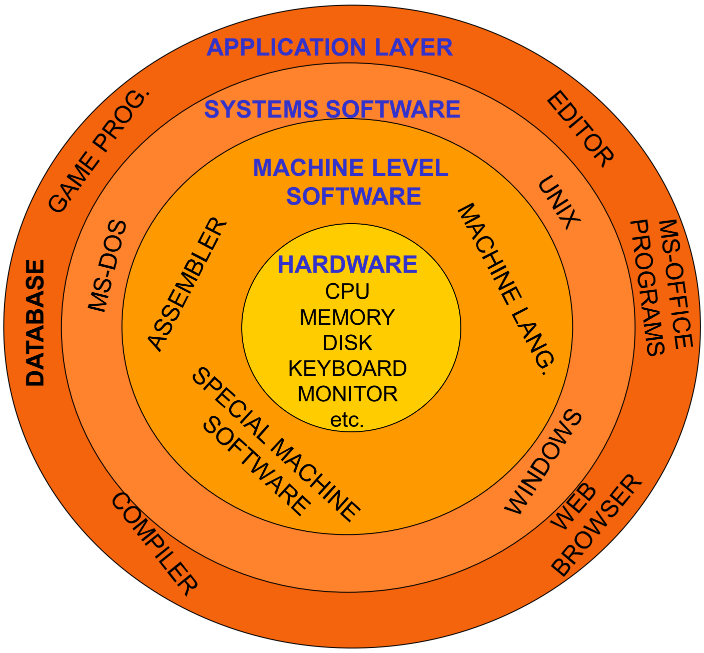
Users

Applications software

Kernel

Operating systems

Hardware



Machine Code

- executable files have to have a computer-understandable format
- executable format differs between
  - hardware
  - operating systems
- programmers cannot write directly in machine code
- write source code in a high-level language
- use tools to convert source code to machine code

Conversion

- *interpreted*
  - read a command from source code
  - convert
  - execute
  - repeat for next command
  - if error report (only first error) and abort
- *compiled*
  - read the whole source code
  - convert and build an executable file
  - if an error report (all errors) and no executable
  - conversion is done once ⇒ much faster

EXAMPLE PROBLEM:
CALCULATE THE GRADE OF A STUDENT

Phase 1: Define the Problem

- PURPOSE: A student’s passing grade will be calculated and printed on the screen.
- Grade should be calculated with the following weights:
  - %30 of the first midterm exam
  - %30 of the second midterm exam
  - %40 of the final exam.
- INPUTS: Inputs are the numeric values of the first midterm exam, second midterm exam, and final exam.
- OUTPUT: Output is the numeric value of Grade.

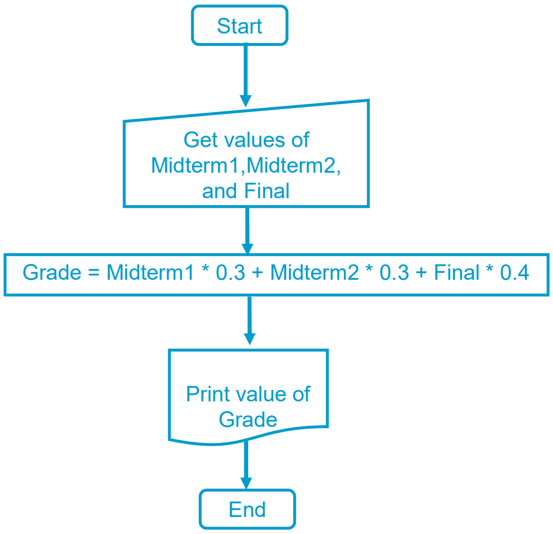
Phase 2: Design the Program

- (Pseudo Code)

```
1. Get values of Midterm1, Midterm2, and Final from the user
2. Grade ← Midterm1 * 0.3 + Midterm2 * 0.3 + Final * 0.4
3. Print value of Grade on screen
4. End
```

Phase 2: Design the Program

- (Flow Chart)



Start

Get values of Midterm1,Midterm2, and Final

Grade = Midterm1 * 0.3 + Midterm2 * 0.3 + Final * 0.4

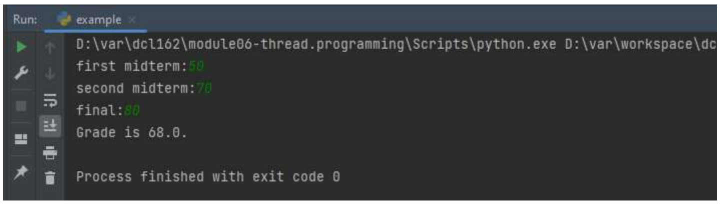
Print value of Grade

End

Program Listing

```python
midterm1 = float(input("first midterm:"))
midterm2 = float(input("second midterm:"))
final = float(input("final:"))
grade = 0.3 * (midterm1 + midterm2) + 0.4 * final
print(f"Grade is {grade}.")
```

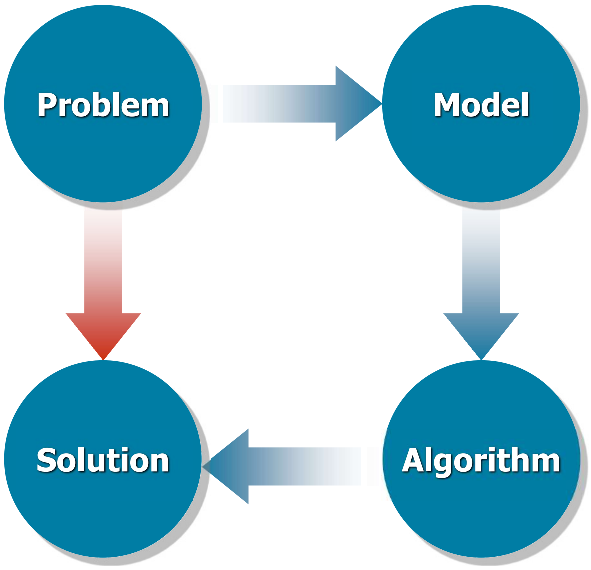
Phase 4: Testing the Program



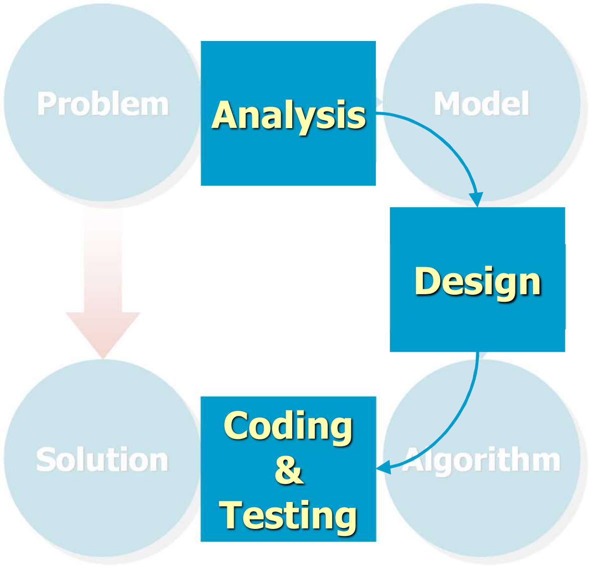
```
D:\var\dcl162\module06-thread.programming\Scripts\python.exe D:\var\workspace\dc
first midterm:50
second midterm:70
final:80
Grade is 68.0.

Process finished with exit code 0
```

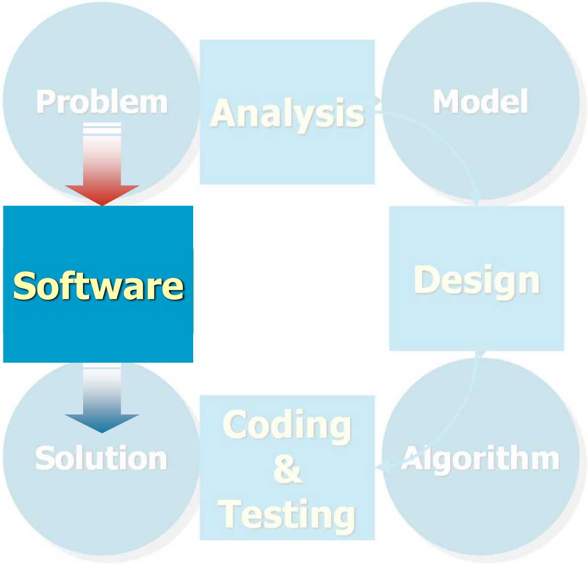
Summary



Summary



Summary


Problem
Analysis
Model
Software
Design
Solution
Coding
&
Testing
Algorithm