CSC 101/CSC 111 – INTRODUCTION TO COMPUTER SCIENCE (3 UNITS) LECTURE 4

BALOGUN JEREMIAH ADEMOLA

ASSISTANT LECTURER,

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS

MOUNTAIN TOP UNIVERSITY, OGUN STATE, NIGERIA

COURSE OUTLINE

- Definition of Computer
- History and Overview of Computing and Computers
- Evolution of Ideas and Machines from Mechanical Computer to Multimedia Computer
- Introduction to computing system
  - Basic elements of a computer system hardware
  - Block diagram, data/instruction flow, control flow
- Software types, packages and applications
- Characteristics of computer
- Problem solving using flowcharts and algorithms
- Data representation in computer system
- Communications and networks
  - World wide web, network access, network architectures, data communications. Safety and security
- File management in Windows and basic word processors, spreadsheets, presentation, graphics and other applications
- Introduction to programming:
  - Statements, symbolic names, arrays, expressions and control statements

PROBLEM SOLVING TECHNIQUES

PROBLEM SOLVING TECHNIQUES

- Problem Solving is the sequential process of analysing information related to a given solution and generating appropriate response options.

- Examples of tasks include:
  - If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something.
  - A student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable.
  - If someone asks to you, what is time now?
  - Some students in a class plan to go on picnic and decide to share the expenses among them.

- There are six (6) steps that should be followed in problem solving:
  - Understand the problem
  - Formulate the model
  - Develop an algorithm
  - Write a program
  - Test the program
  - Evaluate the solution

PROBLEM SOLVING TECHNIQUES

- Understanding the Problem
  - The first step to solving any problem is to make sure that you understand the problem that you are trying to solve.
  - You need to know:
    - What input data/information is available ?
    - What does it represent ?
    - What format is it in ?
    - Is anything missing ?
    - Do I have everything that I need ?
    - What output information am I trying to produce ?
    - What do I want the result to look like … text, a picture, a graph … ?
    - What am I going to have to compute ?

PROBLEM SOLVING TECHNIQUES…

- Formulate a Model
  - Now we need to understand the processing part of the problem.
  - Many problems break down into smaller problems that require some kind of simple mathematical computations in order to process the data.
  - If there is no such "formula", we need to develop one.
  - Often, however, the problem breaks down into simple computations that we well understand.
  - Sometimes, we can look up certain formulas in a book or online if we get stuck.
  - In order to come up with a model, we need to fully understand the information available to us.

PROBLEM SOLVING TECHNIQUES…

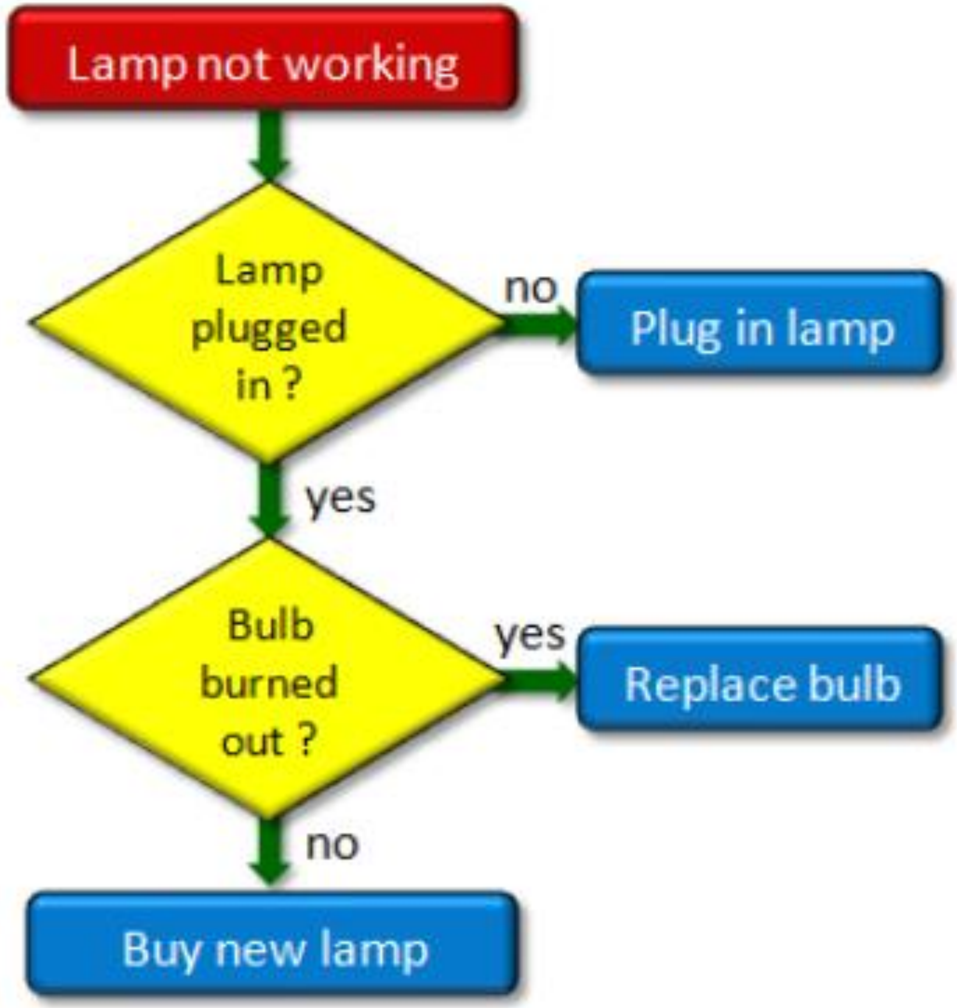
- Develop and Algorithm
  - An algorithm is a precise sequence of instructions for solving a problem.
  - To develop an algorithm, we need to represent the instructions in some way that is understandable to a person who is trying to figure out the steps involved.
  - Two commonly used representations for an algorithm is by using:
    - pseudo code; or
    - flow charts.

  - Pseudocode is a simple and concise sequence of English-like instructions to solve the problem unlike Flowcharts which are graphical.
    - Pseudocode is often used as a way of describing a computer program to someone who doesn't understand how to program a computer.
    - When learning to program, it is important to write pseudocode because it helps you clearly understand the problem that you are trying to solve.

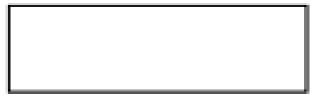
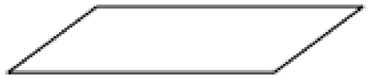
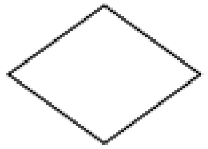
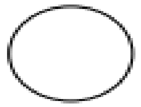
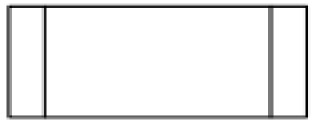
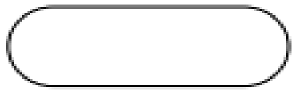
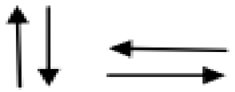
PSEUDOCODE AND FLOWCHARTS

Pseudo Code

```
1. IF lamp works, go to step 7.
2. Check if lamp is plugged in.
3. IF not plugged in, plug in lamp.
4. Check if bulb is burnt out.
5. IF blub is burnt, replace bulb.
6. IF lamp doesn't work buy new lamp.
7. Quit ... problem is solved.
```



PSEUDOCODE AND FLOWCHARTS…

Symbol	Name	Function
	**Process**	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

PSEUDOCODE

- Consider the pseudo-code of the broken lamp shown earlier.
- Various parts of the pseudo-code could be explained as follows:
  - **Sequence** – listing instructions step by step in order

```
1. Make sure switch is turned on
2. Check if lamp is plugged in
3. Check if bulb is burned out
4. ...
```

  - **Condition** – making decision and doing one thing or something else depending on the outcome of the decision

```
if lamp is not plugged in
   then plug it in
```

```
if bulb is burned out
  then replace bulb
otherwise buy new lamp
```

PSEUDOCODE…

- **Repetition** – repeating something a fixed number of times or until some conditions occurs

```
repeat
    get a new light bulb
    put it in the lamp
until lamp works or no more bulbs left
```

```
repeat 3 times
    unplug lamp
    plug into different socket
…
```

- **Storage** – storing information for use in instructions further down the list

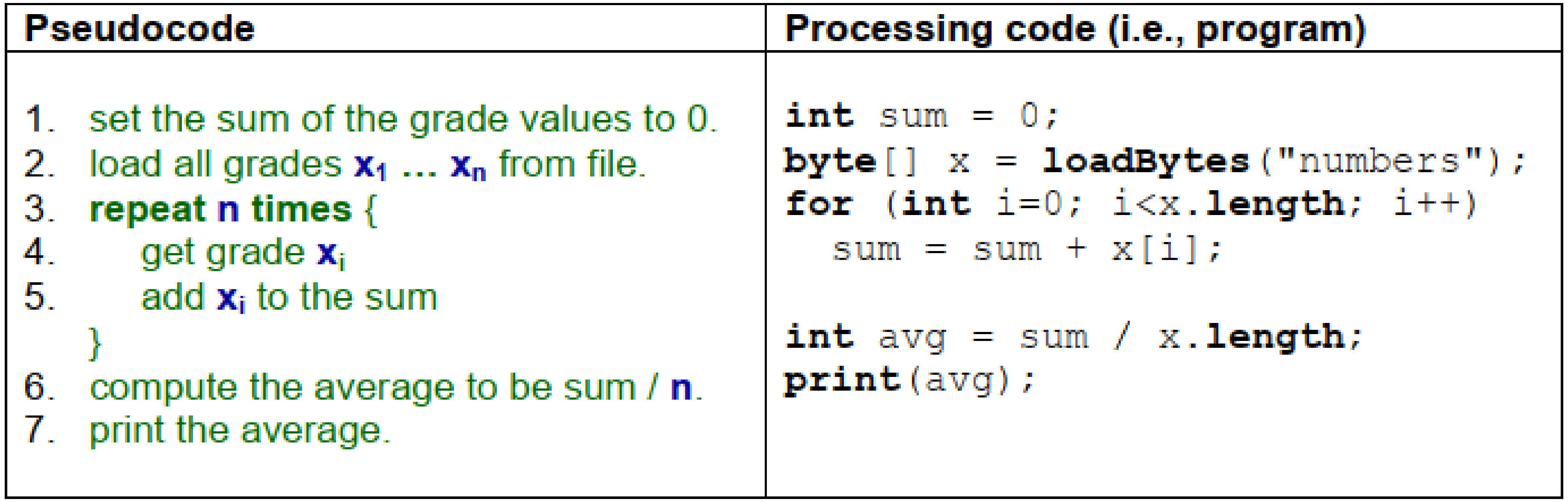
```
x ← a new bulb
count ← 8
```

- **Jumping** – being able to jump to a specific step when needed

```
if bulb works
    then goto step 7
```

PROBLEM SOLVING TECHNIQUES…

- Write the Program
  - Writing a program is often called writing code or implementing an algorithm.
  - The code (or source code) is actually the program itself.
  - In the figure below, the code looks quite similar in structure to the pseudocode, however, the processing code is less readable and seems somewhat more mathematical.



Pseudocode

```
1.  set the sum of the grade values to 0.
2.  load all grades $x_1$ … $x_n$ from file.
3.  repeat n times {
4.      get grade $x_i$
5.      add $x_i$ to the sum
    }
6.  compute the average to be sum / n.
7.  print the average.
```

Processing code (i.e., program)

```
int sum = 0;
byte[] x = loadBytes("numbers");
for (int i=0; i<x.length; i++)
   sum = sum + x[i];

int avg = sum / x.length;
print(avg);
```

  - At this stage, the program is compiled by the computer.
  - Compilation is the process of converting a program into instructions that can be understood by the computer.

PROBLEM SOLVING TECHNIQUES…

- Test the Program
  - **Running** a program is the process of telling the computer to evaluate the compiled instructions.
    - When you run your program, if all is well, you should see the correct output.
      - It is possible however, that your program works correctly for some set of data input but not for all.
    - If the output of your program is incorrect, it is possible that you did not convert your algorithm properly into a proper program.
    - It is also possible that you did not produce a proper algorithm back in step 3 that handles all situations that could arise.
    - Maybe you performed some instructions out of sequence.
      - Whatever happened, such problems with your program are known as bugs.

  - **Bugs** are problems/errors within a program that causes it to stop working or produce incorrect or undesirable results.
    - As a result of this, the reasons for these bugs must be found and corrected.

PROBLEM SOLVING TECHNIQUES…

- Evaluate the Solution
  - Once your program produces a result that seems correct, you need to re-consider the original problem and make sure that the answer is formatted into a proper solution to the problem.
    - It is often the case that you realize that your program solution does not solve the problem the way that you wanted it to.
    - You may realize that more steps are involved.
  - It is also possible that when you examine your results, you realize that you need additional data to fully solve the problem.
    - Or, perhaps you need to adjust the results to solve the problem more efficiently.
  - It is important to remember that the computer will only do what you told it to do.
    - It is up to you to interpret the results in a meaningful way and determine whether or not it solves the original problem.
    - It may be necessary to re-do some of the steps again, perhaps going as far back as step 1 again, if data was missing

DATA REPRESENTATION

DATA REPRESENTATION

- Data representation is a major part of the software-hardware interface.

- As a result of this, data need to be represented in a convenient way that simplifies
  - The common operations, such as: addition, subtraction, comparison etc.
  - Hardware implementation thus making them faster cheaper, and more reliable.

- The Bit is short for Binary digit – the smallest amount of meaningful information stored on a computer.
  - The advantage includes:
    - Too little information per bit (each bit stores 2 possible states)
    - In order to store more information, one needs more bits.

  - The disadvantage includes:
    - It is easy to do computation with bits
    - They are more reliable
    - Used to construct simpler digital circuits.

DATA REPRESENTATION-POSITIONAL NUMBERING

- All number are represented using a polynomial standard representation which is a sum of the product of each digit with the power of its base representation.
  - Consider an *n* digit number with the most significant digit on the left and the least significant digit on the right
  - Given a base representation *b* for each digit a_i, then:

$$(a_n \times b^{n-1}) + (a_{n-1} \times b^{n-2}) + \cdots + (a_2 \times b^1)+(a_1 \times b^0)$$

  - *where* $0 \leq i \leq n-1$
- Decimal Representations
  - If a number is in base 10, then the number 56432 can be represented as:

$$(5\times 10^4) + (6 \times 10^3) + (4 \times 10^2) + (3 \times 10^1)+(2 \times 10^0)$$

- Binary Representations
  - If a number is in base 2, then the number 10101_2 can be represented as:

$$(1\times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^2)+(1 \times 2^0)$$

DATA REPRESENTATION-POSITIONAL NUMBERING

- The general idea behind positional numbering systems is that a numeric value is represented through increasing powers of a radix (or base number).

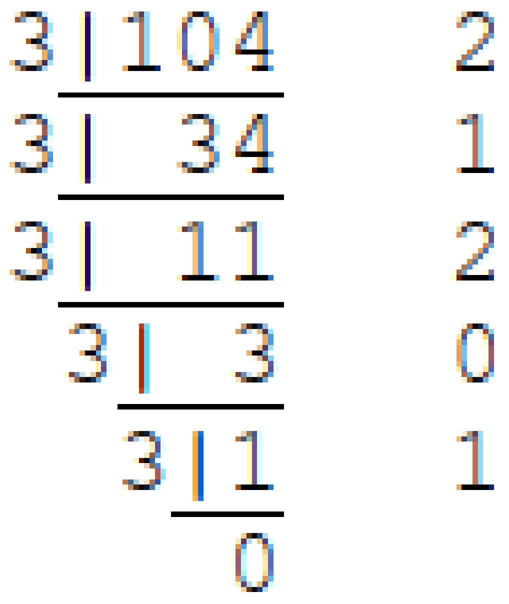
- The popular number systems to be considered are presented as follows:

Number System	Radix	Allowable Digits
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- There are two important groups of number base conversions:
  - Converting decimal numbers to base-r numbers
  - Converting base-r numbers to decimal numbers

CONVERTING BASE 10 INTEGERS TO OTHER BASES

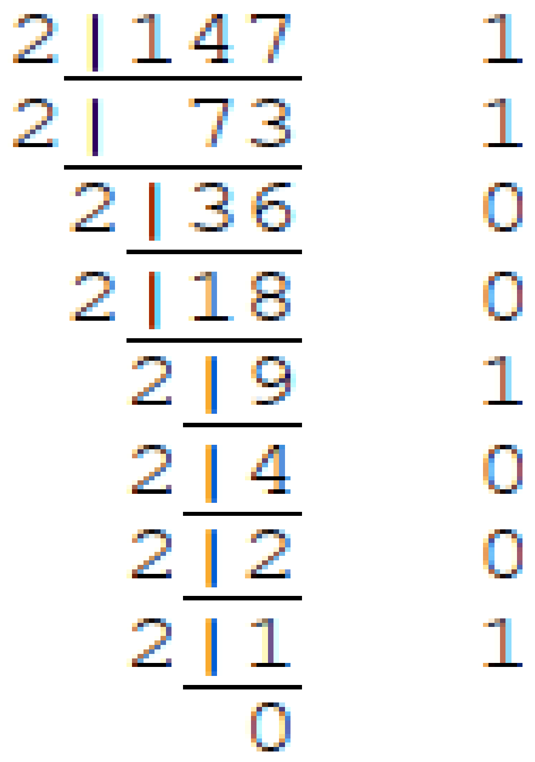
- Convert 104_{10} to base 3 using the division-remainder method



```
3|104    2
 -----
3| 34    1
 -----
3| 11    2
 -----
 3| 3    0
  ----
  3|1    1
   ---
    0
```

- $104_{10} = 10212_3$

- Convert 147_{10} to base 2 using the division-remainder method



```
2|147    1
 -----
2| 73    1
 -----
 2|36    0
  ----
 2|18    0
  ----
  2|9    1
   ---
  2|4    0
   ---
  2|2    0
   ---
  2|1    1
   ---
    0
```

- $147_{10} = 10010011_2$

CONVERTING OTHER INTEGER BASES TO BASE 10

- Other bases can be converted to base 10 using the standard polynomial expressions shown earlier.

- Given a base representation *b* for each digit a_i, then:

$$(a_n \times b^{n-1}) + (a_{n-1} \times b^{n-2}) + \cdots + (a_2 \times b^1)+(a_1 \times b^0)$$

- *where* $0 \leq i \leq n-1$

- For example,

- $\mathbf{10011_2} = (1\times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^2)+(1 \times 2^0)$ = 16+0+0+4+1

- $\mathbf{2634_8} = (2\times 8^3) + (6 \times 8^2) + (3 \times 8^1) + (4 \times 8^0)$ = 512+384+24+4

- $\boldsymbol{AE63C_{16}} = (10\times 16^4) + (15 \times 16^3) + (6 \times 16^2) + (3 \times 16^2)+(\mathrm{C} \times 16^0)$

CONVERTING OTHER FRACTIONAL BASES TO BASE 10

- Other fractional bases can be converted to base 10 using the standard polynomial expressions.

- Consider, a set of digits a_i occurring after the decimal point of a fraction expressed in a given base, *b;* the fraction can be converted to decimal fraction according to:

$$(a_1 \times b^{-1}) + (a_2 \times b^{-2}) + \cdots + (a_{n-1} \times b^{-(n-1)}) + (a_n \times b^{-n})$$

- *Where n* is the number of digits on the right hand side of the decimal point

- For example,
- $\mathbf{0.1101_2} = (1\times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-1})$ = 0.5+0.25+0+0.0625

- $\mathbf{0.425_8} = (4\times 8^{-1}) + (2 \times 8^{-2}) + (5 \times 8^{-3})$ = 0.5+0.03125+0.009765625

- $\mathbf{0.}\boldsymbol{E2AC}_{\mathbf{16}} = (15 \times 16^{-1}) + (2 \times 16^{-2}) + (10 \times 16^{-3}) + (C \times 16^{-4})$

PUTTING IT ALL TOGETHER – MIXED FRACTIONS

- Putting the conversion of the whole part and the fractional part together, it is now possible to convert any mixed fraction of a given base into a base 10 number.

- For example:

- $\mathbf{1011.01_2} = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) = 8 + 0 + 2 + 1 + 0 + 0.25 = 11.25$

- $\mathbf{653.23_8} = (6 \times 8^2) + (5 \times 8^1) + (3 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2}) = 384 + 40 + 0 + 0.25 + 0.046875 = 424.296875$

- $\mathbf{19.\mathit{E}4_{16}} = (1 \times 16^2) + (9 \times 16^1) + (15 \times 16^{-1}) + (4 \times 16^{-2}) = 256 + 144 + 0.9375 + 0.015625 = 400.953125$

CONVERTING BASE 10 FRACTIONS TO OTHER BASES

- In order to convert decimal (base 10) fractions into other bases
  a. The fractional part is multiplied by the required base, *b*
  b. The integer part of the result is recorded and removed from the result
  c. Repeat steps (a) and (b)
  d. The conversion of the decimal fraction to other bases is a collection of the recorded integer part arranged in order of record.

- For example,
  - Convert $(0.513)_{10}$ to octal (base 8)
    - 0.513 x 8 = 4.104
    - 0.104 x 8 = 0.832
    - 0.832 x 8 = 6.656
    - 0.656 x 8 = 5.248
    - 0.248 x 8 = 1.984
    - 0.984 x 8 = 7.872

$$\therefore\ (\mathbf{0.513})_{10} = (\mathbf{0.406517}\ \ldots)_8$$

DATA REPRESENTATION - STRINGS

- Strings unlike numbers are sequences of characters (alphabets and special symbols).
  - e.g. player, #saveLara, McLaren, steven123, etc..

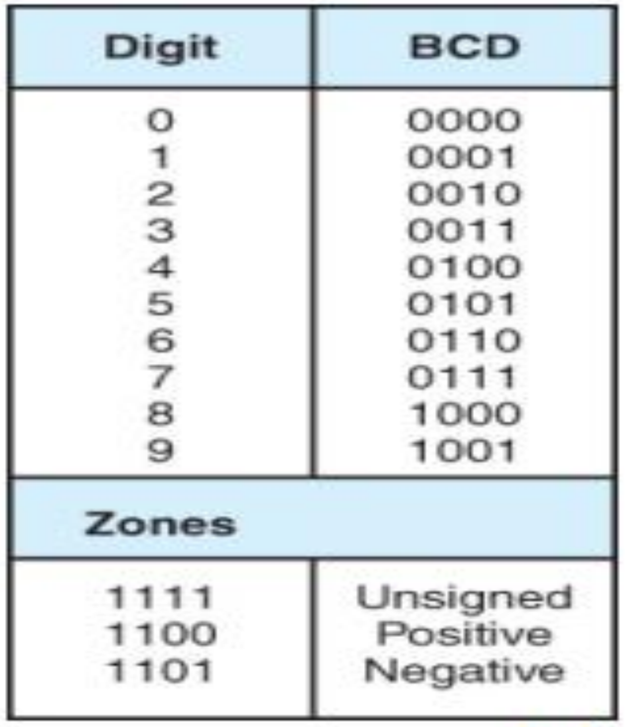
- Strings are human-readable characters and must be converted to computer-understandable bit patterns.

- The conversion is achieved using some form of character encoding scheme.

- Examples of character encoding scheme include:
  - Binary Coded Decimal (BCD)
  - Extended Binary-Coded Decimal Interchange Code (EBCDIC)
  - American Standard Code for Information Interchange (ASCII)
  - Unicode

REPRESENTING STRINGS - BCD

- Binary-Coded Decimal (BCD) is a numeric coding system used primarily in IBM mainframe and midrange systems in the 1950s and 1960s.
- BCD is very common in electronics, particularly those that display numerical data, such as alarm clocks and calculators.
- BCD encodes each digit of a decimal number into a 4-bit binary form.
- When stored in an 8-bit byte, the upper nibble is called zone and the lower part is called the digit.



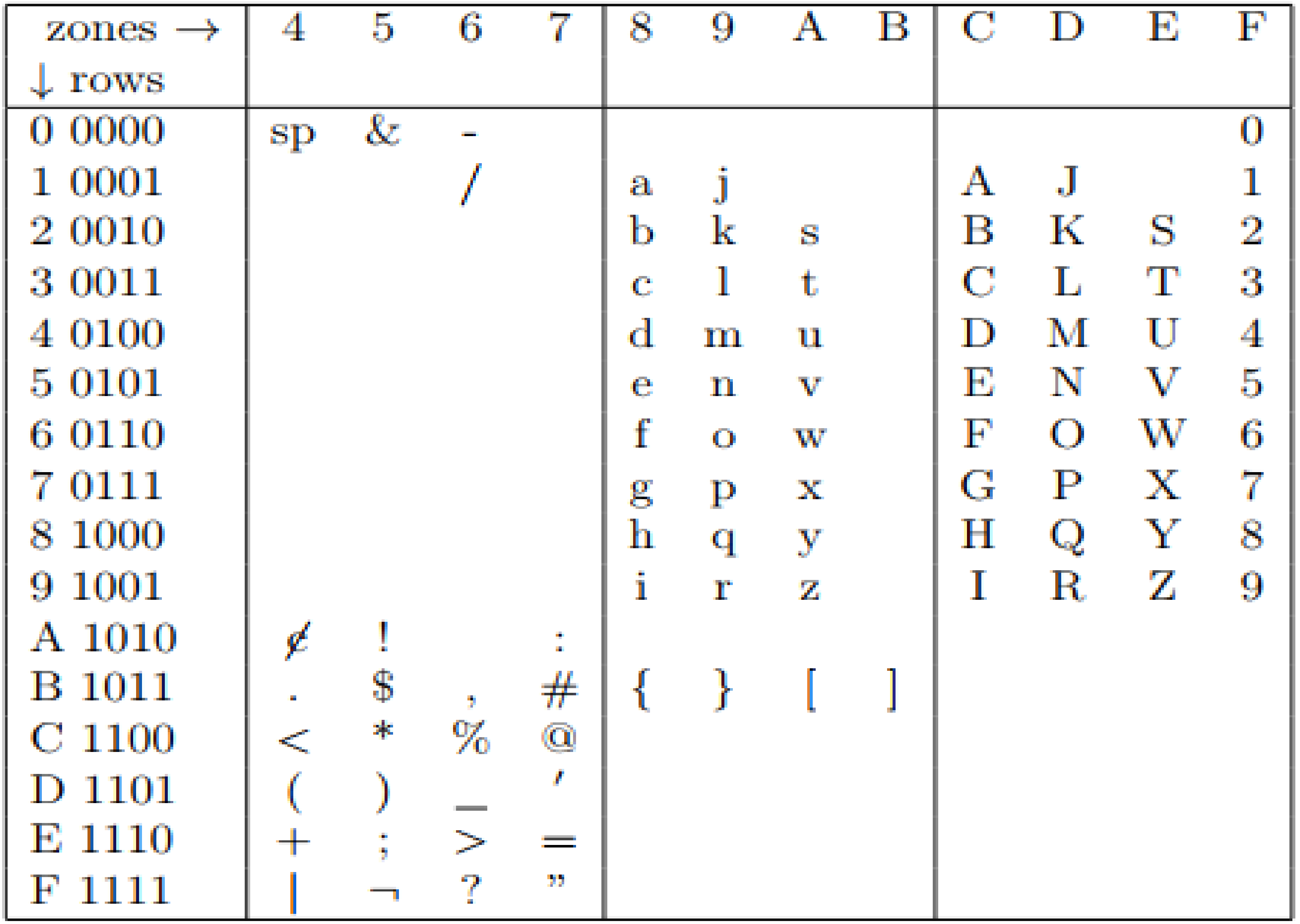
Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Zones	
1111	Unsigned
1100	Positive
1101	Negative

REPRESENTING STRINGS…

- **Extended Binary-Coded Decimal Interchange Code (EBCIDIC)**
  - In 1964, BCD was extended to an 8-bit code, Extended Binary-Coded Decimal
  - Interchange Code (EBCDIC).
  - EBCDIC was one of the first widely-used computer codes that supported upper and lowercase alphabetic characters, in addition to special characters, such as punctuation and control characters.
  - EBCDIC and BCD are still in use by IBM mainframes today.

- **American Standard Code for Information Interchange (ASCII)**
  - ASCII is based on a 7-bit representation scheme
  - It can represent the characters A – Z, a – z, 0 – 9, and control characters.
  - 128 unique characters are represented using the ASCII representation
  - ASCII is basically only for Roman, unaccented characters, although many people have created their own variations of ASCII with different characters.

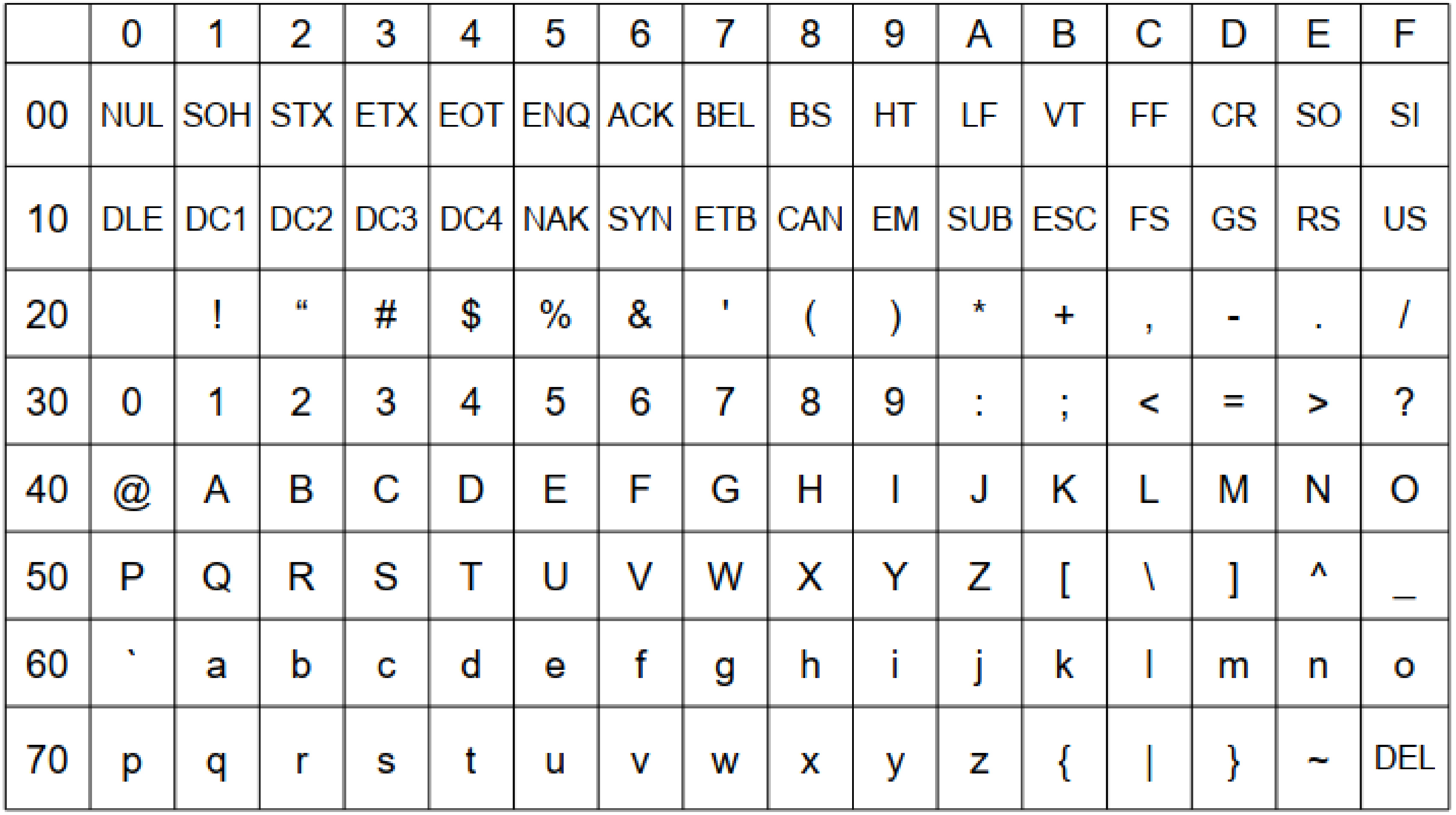
REPRESENTING STRINGS – EBCDIC REPRESENTATION



zones → ↓ rows	4	5	6	7	8	9	A	B	C	D	E	F
0 0000	sp	&	-									0
1 0001			/		a	j			A	J		1
2 0010					b	k	s		B	K	S	2
3 0011					c	l	t		C	L	T	3
4 0100					d	m	u		D	M	U	4
5 0101					e	n	v		E	N	V	5
6 0110					f	o	w		F	O	W	6
7 0111					g	p	x		G	P	X	7
8 1000					h	q	y		H	Q	Y	8
9 1001					i	r	z		I	R	Z	9
A 1010	¢	!		:								
B 1011	.	$	,	#	{	}	[	]				
C 1100	<	*	%	@								
D 1101	(	)	_	'								
E 1110	+	;	>	=								
F 1111	|	¬	?	"								

REPRESENTING STRINGS – ASCII REPRESENTATION

ASCII to Hex conversion: e.g. A is hex 41, C is hex 43, S is hex 53

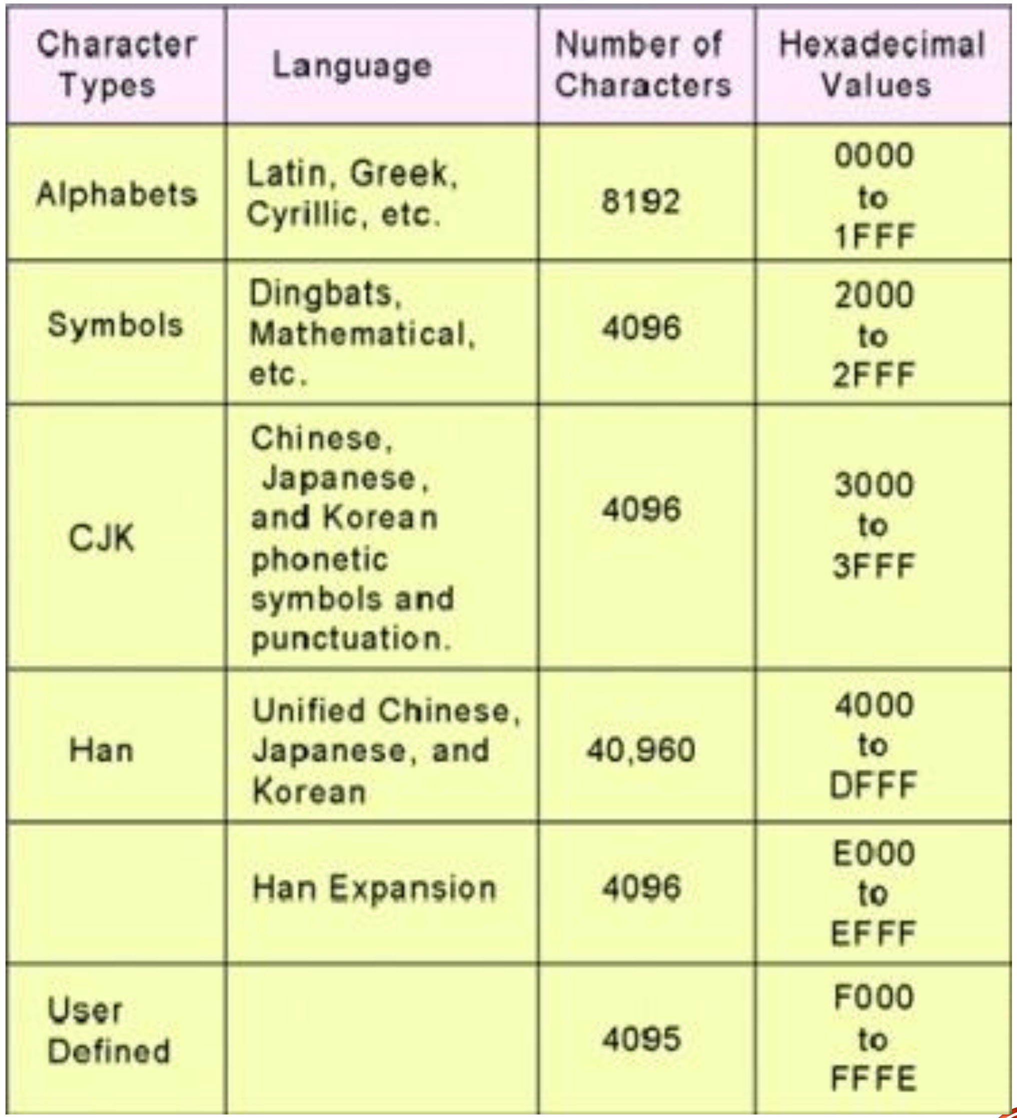


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	“	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

REPRESENTING STRINGS - UNICODE

- Both EBCDIC and ASCII were built around the Latin alphabet.
  - In 1991, a new international information exchange code called Unicode was created.
  - Unicode is a 16-bit alphabet that is downward compatible with ASCII and Latin-1 character set.
- Because the base coding of Unicode is 16 bits, it has the capacity to encode the majority of characters used in every language of the world.
- Unicode is currently the default character set of the Java programming language.

- The Unicode code-space is divided into six parts.
  - The first part is for Western alphabet codes, including English, Greek, and Russian.
  - The lowest-numbered Unicode characters comprise the ASCII code.
  - The highest-numbered Unicode characters provide for user-defined codes.



Character Types	Language	Number of Characters	Hexadecimal Values
Alphabets	Latin, Greek, Cyrillic, etc.	8192	0000 to 1FFF
Symbols	Dingbats, Mathematical, etc.	4096	2000 to 2FFF
CJK	Chinese, Japanese, and Korean phonetic symbols and punctuation.	4096	3000 to 3FFF
Han	Unified Chinese, Japanese, and Korean	40,960	4000 to DFFF
	Han Expansion	4096	E000 to EFFF
User Defined		4095	F000 to FFFE