COMP1002
Computational Thinking and Problem Solving

Lecture 5
Computation III

Lecture 5

- Computation in Computers
  - Addition
  - Multiplication
- Numeric computation vs Symbolic computation
- Top-down approach vs Bottom-up approach
- Understanding the limitations of Computers

Computation

- Teaching the computer to add numbers
- How do we add numbers?
  - 2+7?
  - 3+8?
  - 11+12?
  - 17+18?
  - 51+56?
  - 67+89?
  - 1357+8642?

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

Computation

- How do computers add numbers?
- 2+7?
  - 10+111
- 11+12?
  - 1011+1100
- 51+56?
  - 110011+111000
- 1357+8642?
  - 10101001101+10000111000010

	0	1
0	00	01
1	01	10

  1 0
1 1 1

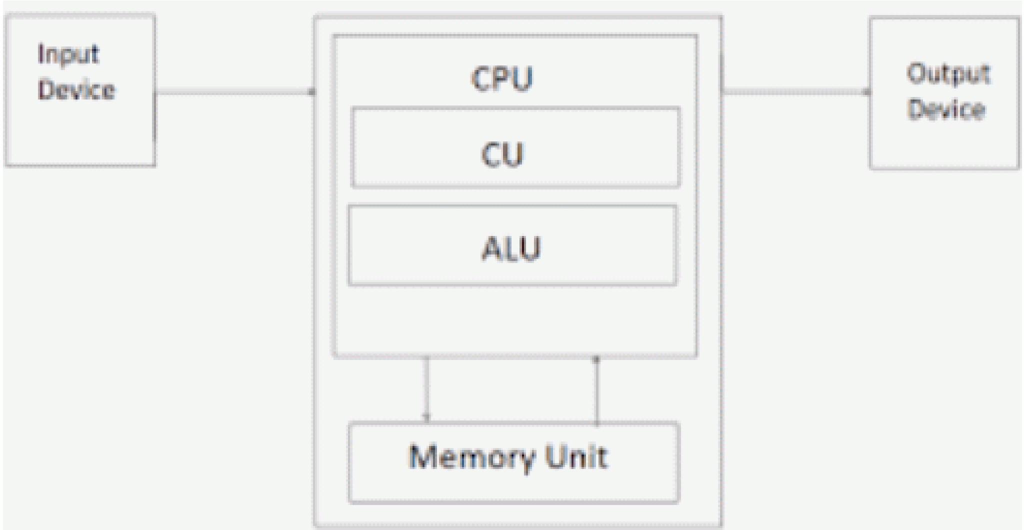
1 0 1 1
1 1 0 0

Computation

- Teaching the computer to add numbers in human way!
  - 11+12
    - Add unit: 1+2 = 3
    - Add ten's: 1+1 = 2
    - Answer 23
  - 17+18
    - Add unit: 7+8 = 15, carry a one to ten's
    - Add ten's: 1+1 = 2, add carry 2+1 = 3
    - Answer 35

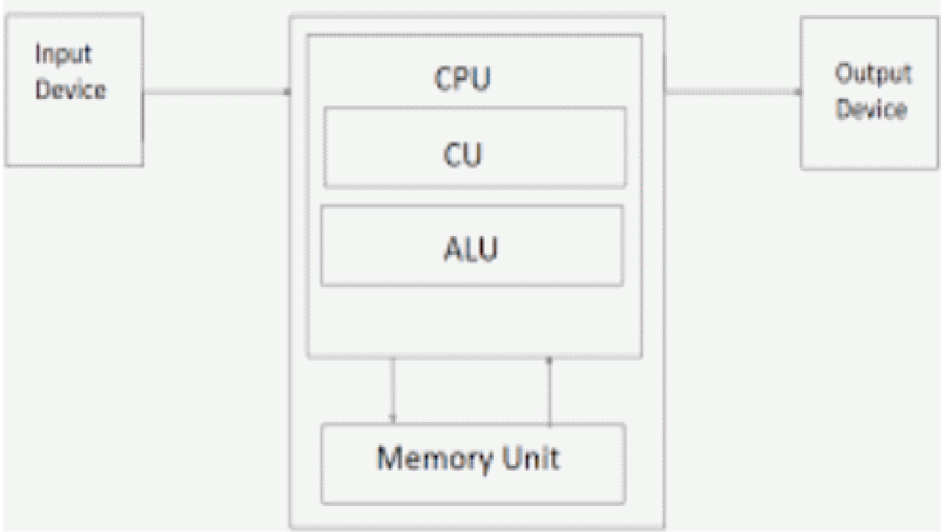
Computation in Computer

› A computer
  – Input
    › Keyboard
  – Output
    › Screen
  – Memory
    › Store data and program
  – CPU (central processing unit)
    › Perform calculation and run program
    › ALU (arithmetic logic unit): perform calculation
    › CU (control unit): control signals inside CPU and arrange for program statement execution using ALU



Computation in Computer

- Input 3+5
- Store 3 as 00000011 in binary (assuming 8 bits)
- Store 5 as 00000101 in binary
- Compute 00000011+00000101 inside ALU
- Prepare result 00001000 for output
- Convert it into decimal as 8 to be output



Computation in Computer

- A side note on why computers prefer two's complement for negative numbers
  - Consider -3 + 5
    - In sign-and-magnitude with 8 bits
      - 10000011 + 00000101 = 10001000 (i.e., -8)
    - In two's complement with 8 bits
      - 11111101 + 00000101 = 00000010 (i.e., 2)
    - Try other additions
      - -5 + 3 and -5 + (-3)
      - Note that -5 = 10000101 / 11111011 in the the above representations respectively

Addition

- Teaching the computer to add numbers in our way!
  - We human can add two numbers of arbitrary size
  - Can we add many numbers?
    - Example: 37+23+33+17+37+40
  - Two approaches:
    - Add all unit digits first, then tens, hundreds
    - Add two numbers first, then add a third one
  - Which is a better approach?
    - For human and computer?
  - How to express your approach in pseudo-code?

Addition

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

- Pseudo-code for adding 2 single-digit numbers:
  - Input: two single-digit numbers x and y
  - Output: the two digits of the sum s_{10}, s_1

```
look up addition table for row x and column y
let s be the entry
set s10 to be the first (left) digit of s
set s1 to be the second (right) digit of s
return the pair s10, s1
```

- Give a name to this "function", called `add2small(x, y)`

Addition

- Pseudo-code for adding 3 single-digit numbers (to handle a potential carry):
  - Input: three single-digit numbers x, y and c
  - Output: the two digits of the sum s_{10}, s_1

a_{10}, a_1 = add2small(x, y)
b_{10}, b_1 = add2small(a_1, c)
d_{10}, d_1 = add2small(a_{10}, b_{10})
set s_1 to be b_1
set s_{10} to be d_1
return the pair s_{10}, s_1

- Give a name to this “function”, called `add3small(x, y, c)`

Addition

› Pseudo-code for adding 2 multi-digit numbers:
  – Input: two multi-digit numbers $X = (x_m x_{m-1} \ldots x_1)$ and $Y = (y_n y_{n-1} \ldots y_1)$, where X and Y have m and n digits respectively
  – Output: all digits of the sum $S = (s_a s_{a-1} \ldots s_1)$

```
set p = maximum of m and n (at least longer of X and Y)
set c_1 = 0
for each digit i running from 1 to p
    a_10, a_1 = add3small(x_i, y_i, c_i)
    set c_i+1 = a_10
    set s_i = a_1
return the number c_p+1 s_p s_p-1 ... s_1
```

  – Give a name to this "function", called `add2large(X, Y)`

Addition

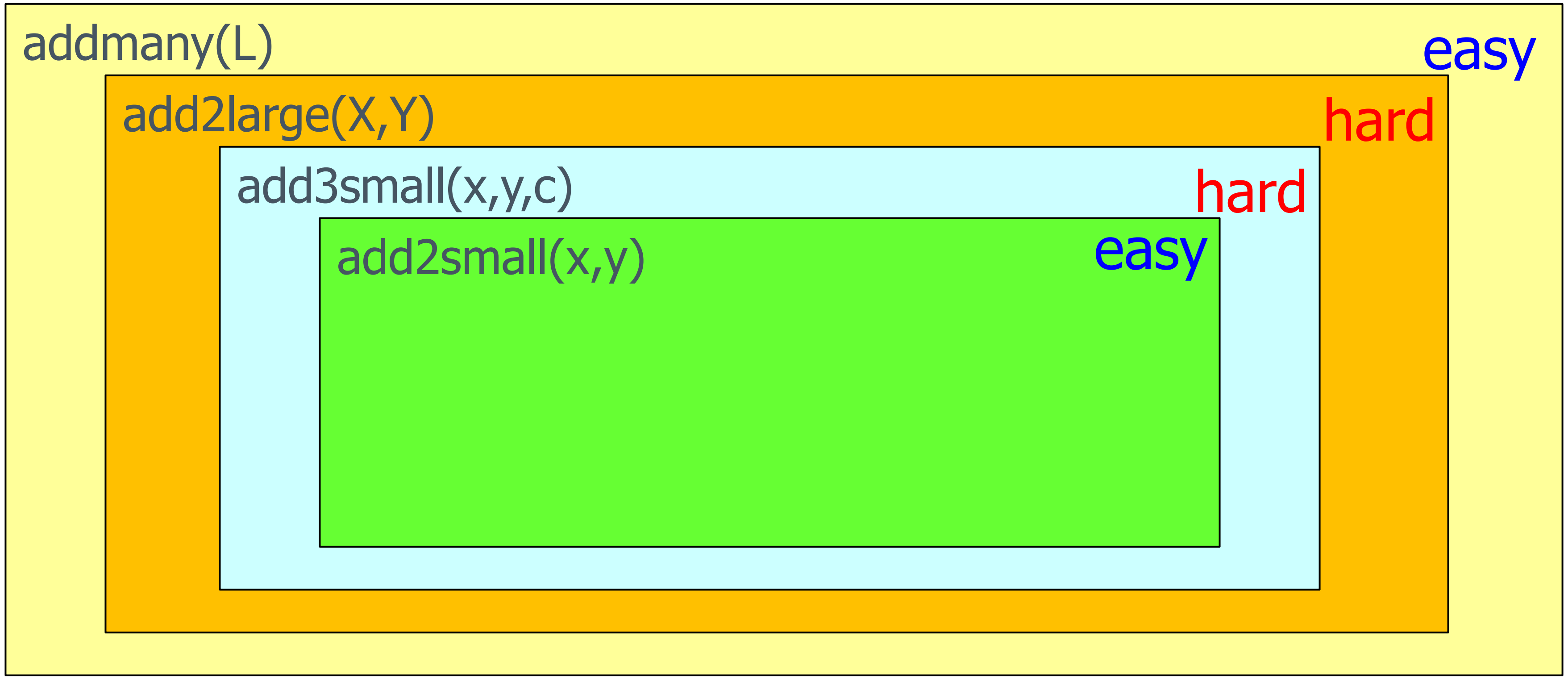
- We complete the final piece of the puzzle
- Pseudo-code for adding multiple numbers
  - Input: a list of numbers, L
  - Output: the sum S

```
set S to 0
for each number n in L
    set S = add2large(S, n)
return S
```

- Give a name to this "function", called `addmany(L)`

Addition

› Now, we can add a list of many numbers, each of arbitrary length!



Addition

- Three types of computational statements are sufficient in our four-step solution
  - Conditional is even not necessary here!
  - Usage of functions will help to reduce 3 copies of add2small(x, y) inside add3small(x, y, c)
- One simple addition-table lookup is sufficient
  - As long as we can add two single-digit numbers, we can add two numbers of arbitrary sizes and then multiple numbers
  - Complex solutions could be built upon simpler ones

Computation

› We can refine our solution from high level gradually to low level
  - Top-down approach
    › Example: in sorting programs, we assume lower level things can be done without doing them in our design, e.g., finding the smallest number, inserting an item in proper position (Lecture 4)

› We can build up our solution from simple blocks that we know to work
  - Bottom-up approach
    › Example: in human way of addition program, we build up from adding 2 small numbers, to 3 small numbers, to 2 big numbers, to many big numbers gradually

Computation

- Teaching the computer to add numbers in our way
  - Advantage: no limit on the size of the numbers
  - Disadvantage: slower than binary addition
- Great abstraction
  - Based on very simple table lookup (or definition)
  - It may not be the add operator, but could be multiply, or even any mysterious operator
  - It may not be numbers, but symbols
  - Can be generalized into non-numerical (symbolic) computation

Computation

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

x	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Computation

-- PLAINTEXT --

KEY

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V

Computation

- To produce meaningful answers, you do not have to understand what the symbols stand for or why the manipulations are correct (Hector Levesque)
  - Computers are dumb but will work for you following instructions
- The “trick” of computation (Levesque)
  - Computers can perform a wide variety of impressive activities precisely because those activities can be described as a type of symbol processing that can be carried out purely mechanically
  - Computation is the process of taking symbolic structures, breaking them apart, comparing them, and reassembling them according to a precise recipe called a procedure (or algorithm)
- This is called symbolic computation

Computation

- Symbolic computation
  - Computation involving symbols to represent data, in exact value/form
- Numeric computation
  - Computation involving numbers to represent data, sometimes in approximated value/form
    - Numbers carry real meaning
    - Equation solving
    - Average, summary and statistics of data
    - Sorting and searching of data
    - Big data processing, e.g., data mining, clustering
  - Majority of common programs

Multiplication

- Consider the following:
- 23 x 14
  - 4 by 3
  - 4 by 2
  - Get 92 for 4 by 23
  - 1 by 3
  - 1 by 2
  - Get 23 for 1 by 23
  - Add them up

$$\begin{array}{r} 23 \\ \underline{14} \\ 12 \end{array} \qquad \begin{array}{r} 23 \\ \underline{14} \\ 12 \\ 8 \end{array} \qquad \begin{array}{r} 23 \\ \underline{14} \\ 92 \end{array}$$

$$\begin{array}{r} 23 \\ \underline{14} \\ 92 \\ 3 \end{array} \qquad \begin{array}{r} 23 \\ \underline{14} \\ 92 \\ 3 \\ 2 \end{array} \qquad \begin{array}{r} 23 \\ \underline{14} \\ 92 \\ \\ 23 \end{array}$$

$$\begin{array}{r} 23 \\ \underline{14} \\ 92 \\ \underline{23} \\ 322 \end{array}$$

Multiplication in Computers

- 23 x 14
  - 10111 x 1110
  - 0 by 10111 = 0
  - 1 by 10111 = 10111 next
  - Add
  - 1 by 10111 = 10111 next
  - Add
  - 1 by 10111 = 10111 next
  - Add
  - Answer is $101000010_2 = 322_{10}$
- The operation is simple
  - For every digit in the multiplier (starting from the right to left), add the multiplicand for “1”, and no add for “0”
  - Shift one digit to the left for the multiplicand after each add/no add

```
 10111
  1110
 -----
 00000
```

```
 10111
  1110
------
 00000
101110
```

```
 10111
  1110
------
 00000
101110
------
101110
```

```
  10111
   1110
  -----
  00000
 101110
 ------
 101110
1011100
```

```
   10111
    1110
   -----
   00000
  101110
  ------
  101110
 1011100
--------
10001010
```

```
   10111
    1110
   -----
   00000
  101110
  ------
  101110
 1011100
--------
10001010
10111000
```

```
    10111
     1110
    -----
    00000
   101110
   ------
   101110
  1011100
 --------
 10001010
 10111000
 --------
101000010
```

Multiplying Large Numbers

- Our previous solution, `addmany(L)`, to add a list of large numbers is built based on simple table lookup
  - There is no real mathematics done!
- Now we are able to use real mathematics to help
  - Just generate pseudo-code (then program) that follows the human ways of multiplying decimal numbers together
  - Home Exercise
    - Write down the pseudocode for multiplying a list of integers

Computation

- Are computers incredibly dumb?
  - Basically yes, but …
  - Recent machine learning techniques enable computers to deduce "knowledge" from large collection of data for artificial intelligence
  - Computers can now generate some "new" things!
- Are computers incredibly accurate?
  - Basically yes, but …
  - Not all computers can calculate 123456789 * 987654321 accurately
  - Try this in Python
    - 12345678.9 * 98765432.1
    - How can we make it accurate?

```
>>> 12345678.9 * 98765432.1
1219326311126352.8
```

Computation

- We need to be careful and always be aware of limitations and pitfalls when exercising computational thinking to solve problems
- We should design the program (algorithm/pseudo-code) in a more general manner
- We should normally not assume input data to be correct and should perform proper error checking (called input validation)
  - You cannot add together inputs that are not numbers
  - You cannot divide a number by zero
  - You cannot compare an integer with a string to see which one is larger
  - You may not allow punctuation marks in the name of person

Summary

- Computation in Computers
  - Addition
  - Multiplication
- Numeric computation vs Symbolic computation
- Top-down approach vs Bottom-up approach
- Understanding the limitations of Computers