Programming Practice

2018-09-20

Week 3

Notice

Assignment Submission Deadline & Late Submission

Final Exam, Report of Absence

Login Account

NOTICE : Assignment Submission

- Submission Deadline
  - Until the following lecture day 14:00 (not lab session)
  - Ex) This assignment - 27th September 14:00
- Late submission
  - Submit by email to pp20182ta@gmail.com
  - Delay penalty
    - ~ 24 hours : 20% deduction
    - ~ 48 hours : 50% deduction
    - After 48 hours : no score

NOTICE

- Final Term
  - **15th December 13:00 ~ 18:00**
  - If you have any questions, ask Professor or email to pp20182ta@gmail.com

- Report of absence
  - Submit to TA

Do not use martini.snucse.org when

- Using Linux Environment
  - Lab computer
  - Virtual box/Vmware
  - …
- Using Mac(optional)
  - Final term will be done in Linux environment.
  - You may need to install GCC.

→ Use your own terminal!

Login Account

- pp** account
  - Only for http://pp2018f.snucse.org:8888/ website

- snucse account
  - For others (Lab computer, martini.snucse.org, snucse.org, id.snucse.org …)

Practice Lecture

Data Type & Type Conversion

ASCII Code

I/O Redirection

EOF

Nested Loop

Data Type

- C language has some primitive types such as

- Character: char, signed char, unsigned char
- Integer: (unsigned or signed) short, int, long, and long long
- Floating-point number: float, double, long double

Data Type – Character or Integer

Type	Explanation and Range	Format specifier
`char`	can be either signed or unsigned.	`%c`
`signed char`	[-128 ~ 127]	`%c, %hhi`
`unsigned char`	[0 ~ 255]	`%c, %hhu`
`(signed) short (int)`	[-32,768 ~ 32,767]	`%hi`
`unsigned short (int)`	[0 ~ 65,535]	`%hu`
`(signed) int`	[–2,147,483,648 ~ 2,147,483,647]	`%i, %d`
`unsigned int`	[0 ~ 4,294,967,295]	`%u`
`(signed) long (int)`	[–2,147,483,648 ~ 2,147,483,647]	`%li, %ld`
`unsigned long (int)`	[0 ~ 4,294,967,295]	`%lu`
`(signed) long long (int)`	[–9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807]	`%lli, %lld`
`unsigned long long (int)`	[0 ~ 18,446,744,073,709,551,615]	`%llu`

Data Type – Floating-Point

Type	Explanation and Range	Format specifier

`float`	single precision floating-point type.
	precision of 6 significant figures.
	[-3.4E+38 ~ +3.4E+38]
		`%f, %F,`
		`%e, %E`

`double`	double precision floating-point type.
	precision of 15 significant figures.
	[-1.7E+308 ~ +1.7E+308]
		`%lf, %lF,`
		`%le, %lE`

`long double`	double precision floating-point type.
	precision of 15 significant figures.
	[-1.7E+308 ~ +1.7E+308]
		`%Lf, %LF,`
		`%Le, %LE`

Data Type – Type Conversion

- For binary operations with operands of different types, the “lower” type is promoted to the “higher” type before operation proceeds.
- For assignment operations, the value of the right side is converted to the type of the left, which is the type of the result.
  - Please read lecture slides or books if you want to learn more.
- Type Casting : Explicit Conversion

ex)

```
int a = 3;
printf("%f\n", (float) a / 2);
```

ASCII Code

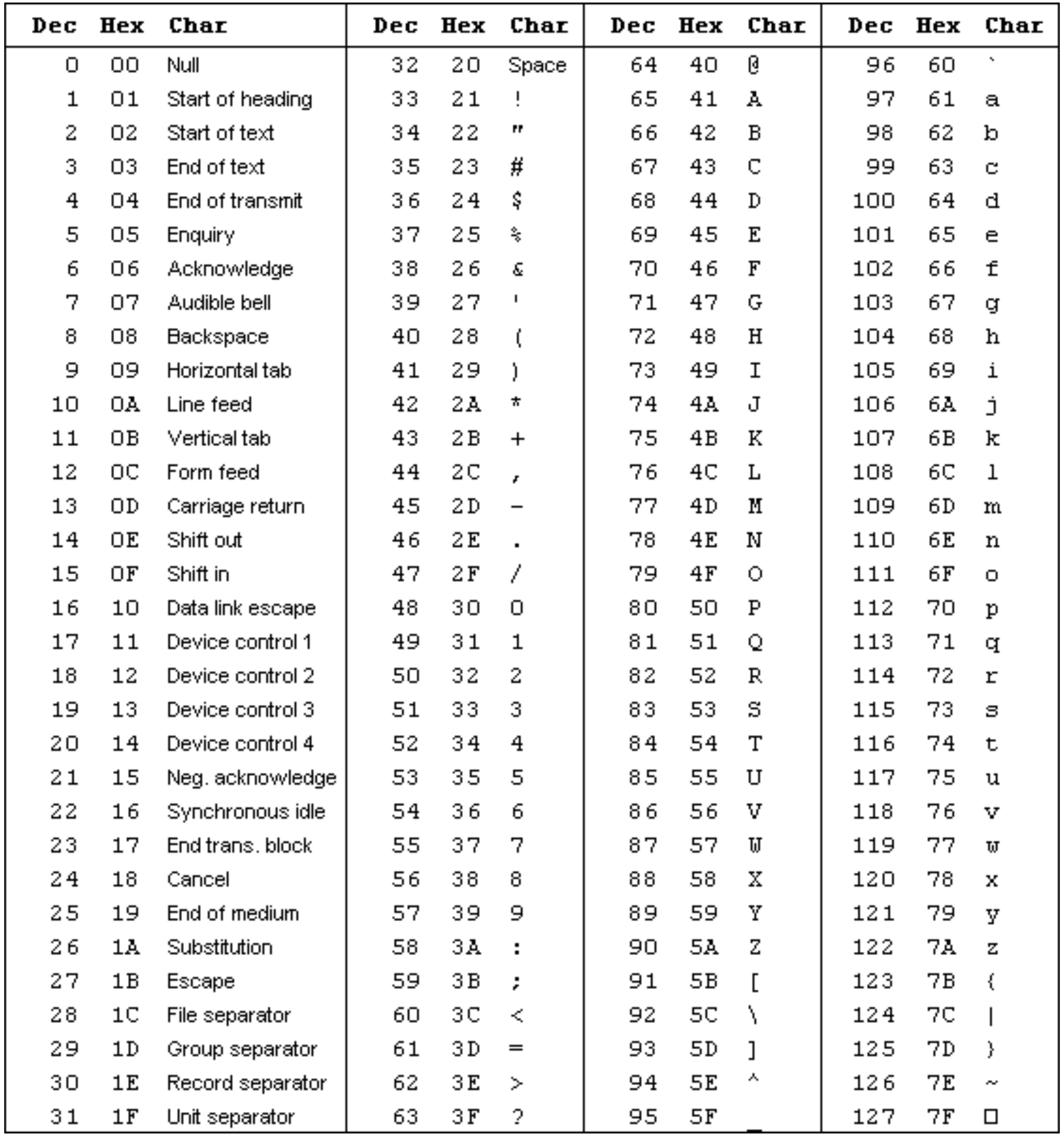
- A character encoding-scheme
- Each character constant has its corresponding integer value.
- No particular relationship between the value of the character constant representing a digit and the digit's intrinsic integer value.

```
'2' != 2
```

and

```
'2' == 50
```

ASCII Code



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

I/O Redirection

- I/O = Input/Output

- To redirect standard input/output/error to a File

- You can use file content as the input for a program
  or save the output of a program as file content

- You can reuse input and output

I/O Redirection

- I/O = Input/Output
- To redirect standard input/output/error to a File
- Usage

```
$ ./program < input
$ ./program > output
$ ./program < input > output
```

I/O Redirection

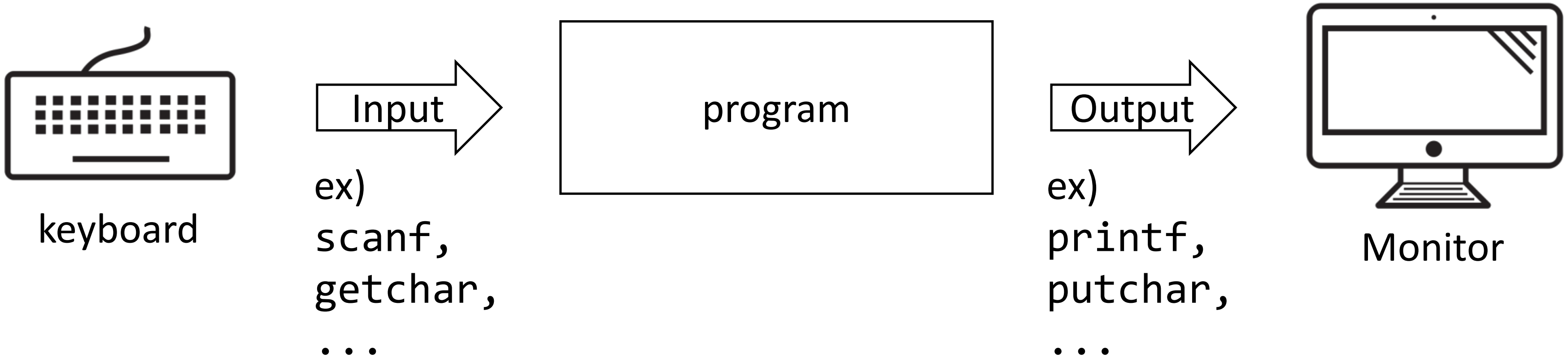
- Usage Example

- If the program, input file, and output file names are 'printer', 'data.txt‘, and 'result.txt‘, respectively.

```
$ ./printer < data.txt
$ ./printer > result.txt
$ ./printer < data.txt > result.txt
```

I/O Redirection – Input Redirection

```
$ ./program
```



keyboard

Input

ex)
`scanf,`
`getchar,`
`...`

program

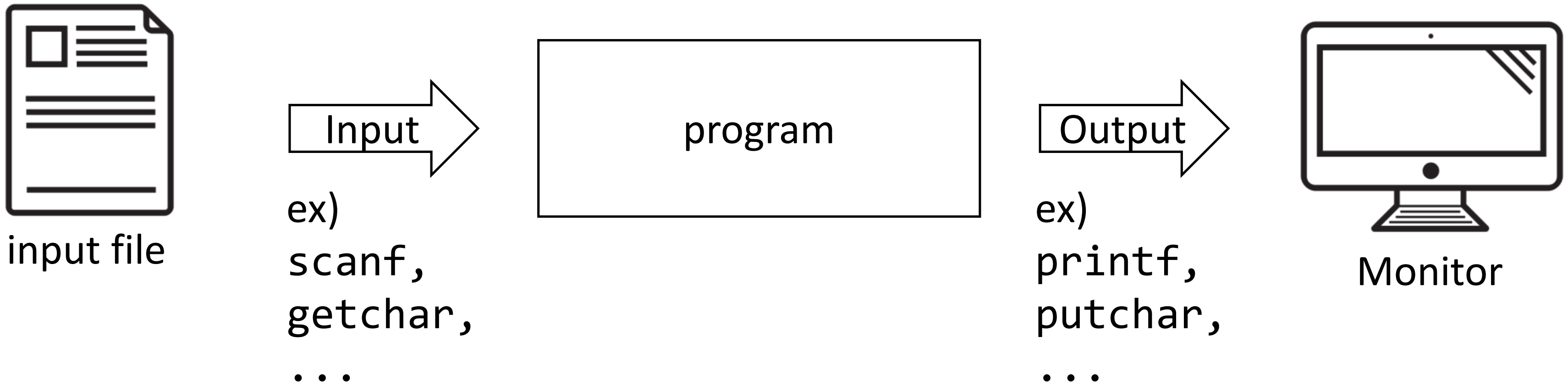
Output

ex)
`printf,`
`putchar,`
`...`

Monitor

I/O Redirection – Input Redirection

```
$ ./program < input
```



input file

Input

ex)
`scanf,`
`getchar,`
`...`

program

Output

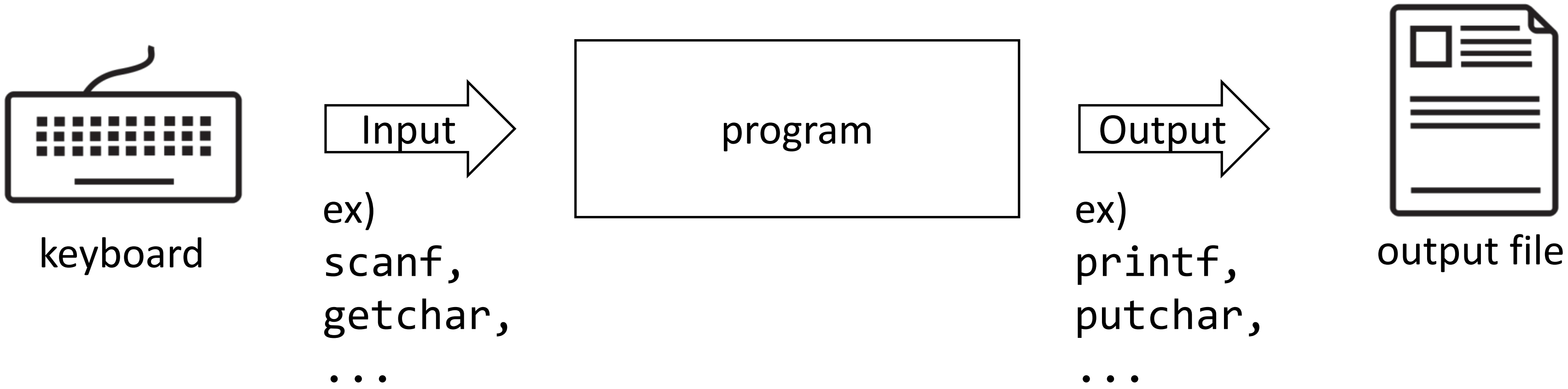
ex)
`printf,`
`putchar,`
`...`

Monitor

I/O Redirection – Output Redirection

```
$ ./program > output
```

(output file will be created or overwritten.)

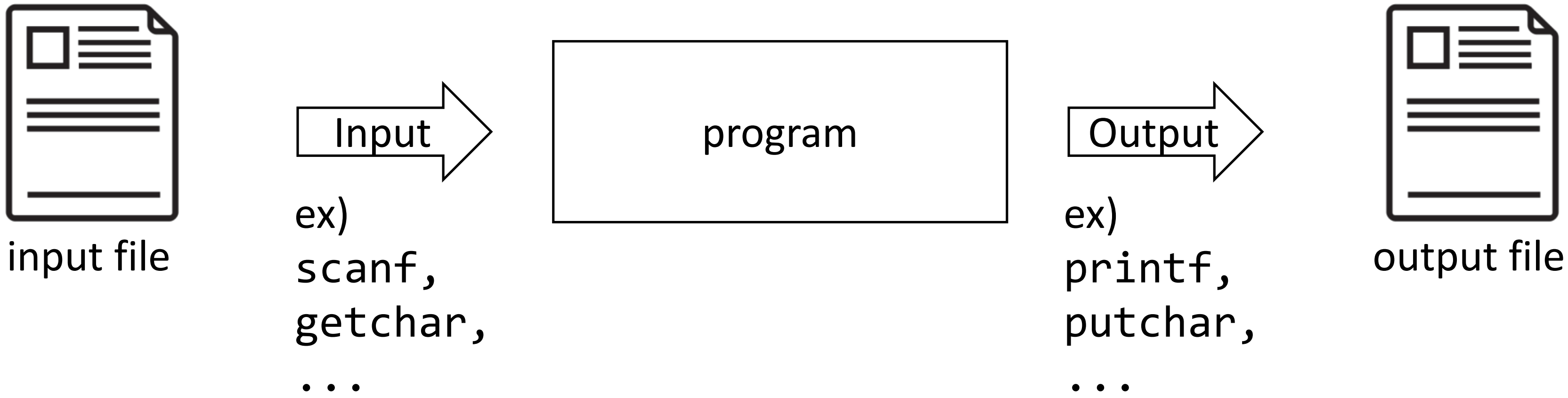


ex)
`scanf,`
`getchar,`
`...`

ex)
`printf,`
`putchar,`
`...`

I/O Redirection

```
$ ./program < input > output
```



I/O Redirection Practice

shell (terminal)

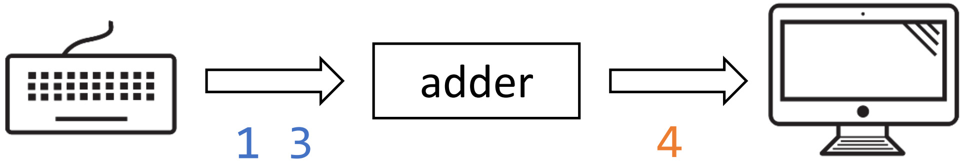
```
$ gcc adder.c -o adder
$ ./adder
1 3
4
$
```

adder.c

```
#include <stdio.h>

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", a + b);
    return 0;
}
```

- 1 3 : Type by keyboard
- 4 : Printed by program



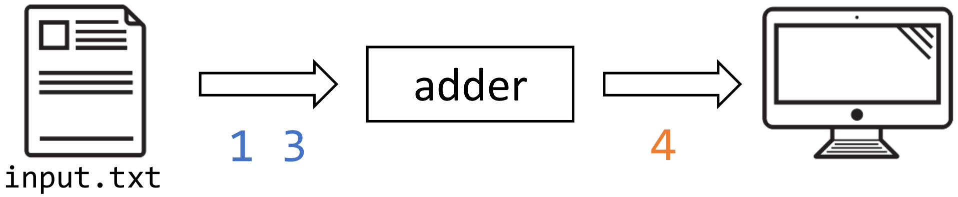
I/O Redirection Practice

shell (terminal)

```
$ ./adder < input.txt
4
$
```

input.txt

```
1 3
```



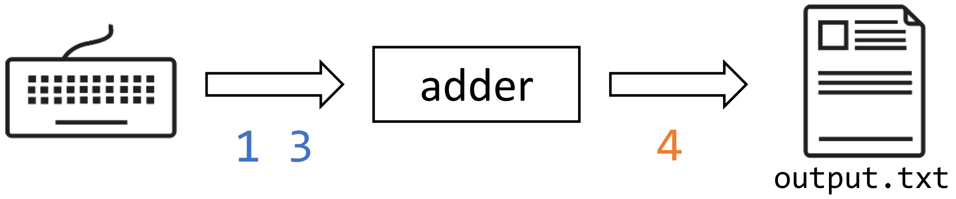
I/O Redirection Practice

shell (terminal)

```
$ ./adder > output.txt
1 3
$ vim output.txt
```

output.txt

```
4
```



I/O Redirection Practice

shell (terminal)

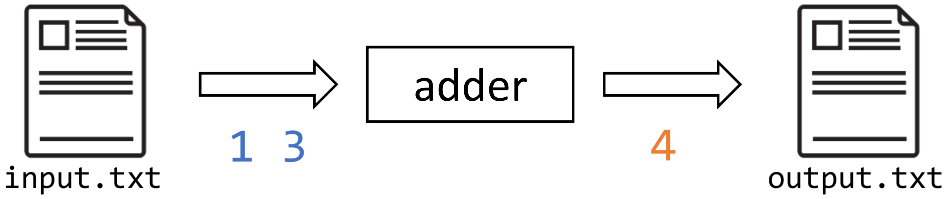
```
$ ./adder < input.txt > output.txt
$ vim output.txt
```

input.txt

```
1 3
```

output.txt

```
4
```



EOF

- EOF is abbreviation of End-of-File
  - EOF is a symbolic constant that stands for End of File
  - EOF is a condition where no more data can be read from a data source

(* data source : file or stream)

- In terminal, we can enter <control + d> as EOF

EOF

- EOF is abbreviation of End-of-File
    - EOF is a symbolic constant that stands for End of File
    - EOF is a condition where no more data can be read from a data source

(* data source : file or stream)

- In C, we can check whether EOF comes

```
while( scanf("%d", &a) != EOF )
```

or

```
while( (c = getchar()) != EOF )
```

EOF Practice

shell (terminal)

```
$ gcc adder_eof.c -o adder_eof
$ ./adder_eof
3 10 2 16 8
[ctrl + d]
39
$
```

adder_eof.c

```
#include <stdio.h>

int main() {
    int a, s = 0;
    while (scanf("%d", &a) != EOF) {
        s += a;
    }
    printf("%d\n", a);
    return 0;
}
```

EOF Practice

shell (terminal)

```
$ ./adder < input.txt
39
$
```

input.txt

```
3 10 2 16 8
```

Nested Loop

- We can use `while/for` loop in other `while/for` loop.

nested_loop.c

```c
#include <stdio.h>

int main() {
    int i, j;
    for (i = 0; i < 10; ++i) { // outer loop
        for (j = 0; j <= i; ++j) { // inner loop
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

Nested Loop

```
for (i = 0; i < 10; ++i) {
    for (j = 0; j <= i; ++j) {
        printf("*");
    }
    printf("\n");
}
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Nested Loop

i = 0

```
for (i = 0; i < 10; ++i) {
    for (j = 0; j <= 0; ++j) {
        printf("*");
    }
    printf("\n");
}
```

```
*
```

Nested Loop

i = 1

```
for (i = 0; i < 10; ++i) {
    for (j = 0; j <= 1; ++j) {
        printf("*");
    }
    printf("\n");
}
```

```
*
**
```

Nested Loop

i = 2

```
for (i = 0; i < 10; ++i) {
    for (j = 0; j <= 2; ++j) {
        printf("*");
    }
    printf("\n");
}
```

```
*
**
***
```

Nested Loop

i = 9

```
for (i = 0; i < 10; ++i) {
    for (j = 0; j <= 9; ++j) {
        printf("*");
    }
    printf("\n");
}
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

References

- https://en.wikipedia.org/wiki/C_data_types
- https://ascii.cl/
- https://en.wikipedia.org/wiki/End-of-file
- http://www.glue.umd.edu/afs/glue.umd.edu/system/info/olh/Programming/C_Programming_on_Glue/The_Third_C_Program_Character_Data/eof_stdio

Homework Problems

1. Data-types Practice
2. Average
3. Conversion
4. Multiplication Table

Problem. **1**

Data-types Practice

Description

Let’s see how data types work in the C language!

Copy and submit the following exact program that prints results of some arithmetic operations. Use the following exact code.

Please make sure you understand why the code prints this output.

Input

None

Output

Results of following operations.

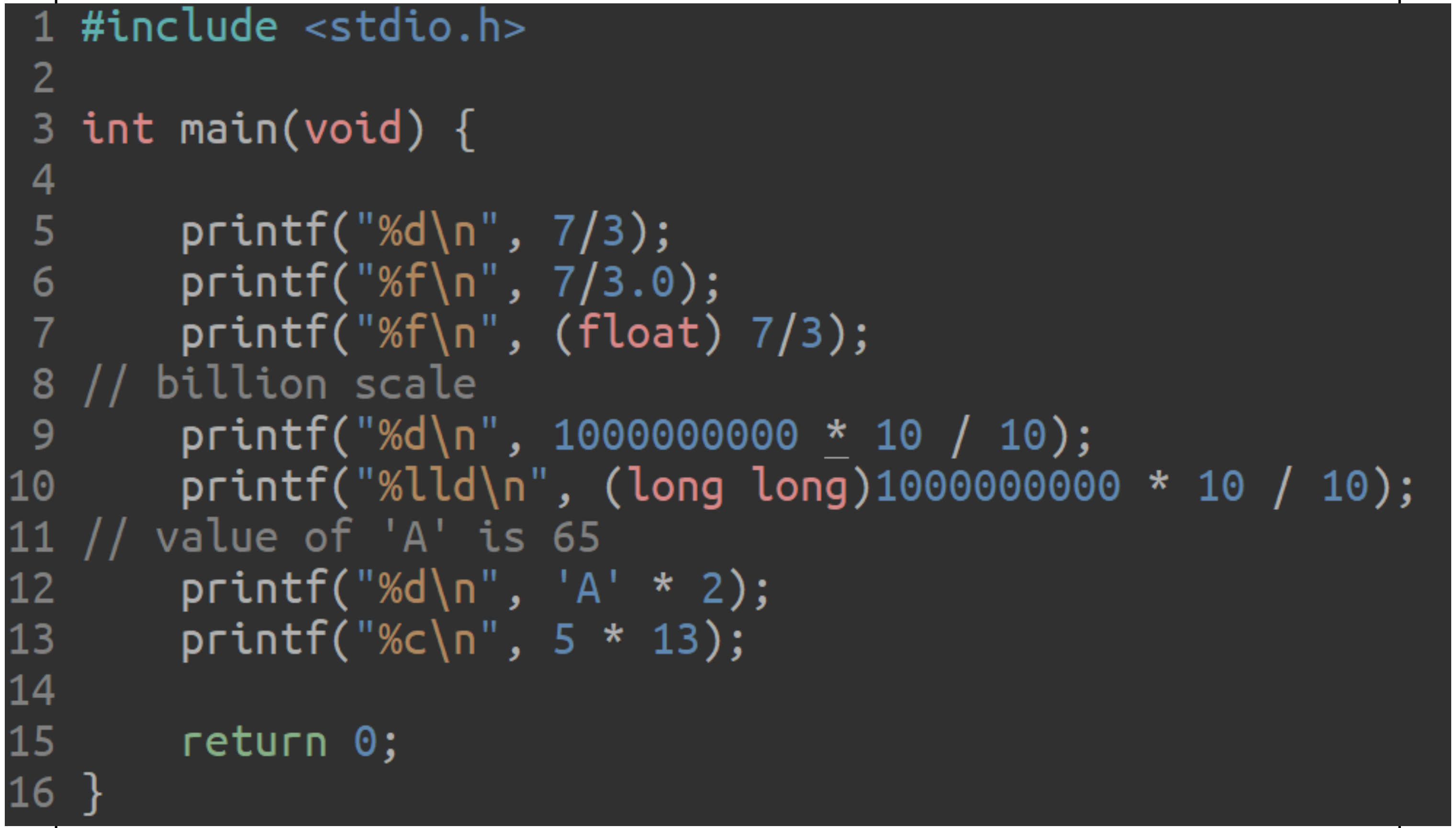
Sample

```
[output]
2
2.333333
2.333333
141006540
1000000000
130
A
```

Problem. **1**

Data-types Practice

Description



```c
#include <stdio.h>

int main(void) {

    printf("%d\n", 7/3);
    printf("%f\n", 7/3.0);
    printf("%f\n", (float) 7/3);
// billion scale
    printf("%d\n", 1000000000 * 10 / 10);
    printf("%lld\n", (long long)1000000000 * 10 / 10);
// value of 'A' is 65
    printf("%d\n", 'A' * 2);
    printf("%c\n", 5 * 13);

    return 0;
}
```

Results of following operations.

Sample

```
[output]
2
2.333333
2.333333
141006540
1000000000
130
A
```

Problem. **2**

Average

Description

Write a program that calculates the average of given integers.

Any number of integers may be given as input – you must continuously get input until EOF.

It is guaranteed that at most 1000 integers will be given, and the absolute value of each given integer will be less than or equal to 100000.

Input

A single line with any number of integers.

Output

Print the average of give integers. Absolute error is allowed up to 10^-6

Sample

```
[input]
1 2 3 4


[output]
2.500000
```

Problem. **3**

Conversion

Description

Write a program that gets any number of characters as input, and convert any occurrence of upper-case letters to lower-case letters and any occurrence of lower-case letters to upper-case letters.

Don’t do anything for non-alphabet characters.

For instance, there may be whitespaces included in the input – just leave them as whitespaces.

Input

A single line with any number of characters. It is guaranteed that every character is of ASCII table.

Output

Print the same line as input, except convert all upper-case letters to lower-case and all lower-case letters to upper-case.

Sample

```
[input]
abcd123 ABCDEF


[output]
ABCD123 abcdef
```

Problem. **4**

Multiplication Table

Description

Print a multiplication table(1~9), one equation at each line.

Don’t contain any spaces.

Input

None

Output

Multiplication table.

Sample

```
[output]
1*1=1
1*2=2
…
1*9=9
2*1=2
2*2=4
…
9*9=81
```