Learn Programming Basics (C Language)

LESSON #005 (Lab)
printf/scanf, precedence table, lvalue, rvalue

Purpose

- printf prints characters on console
- scanf scan characters on console and convert into type

Objective

- learn some basic programming typing starting with printf and scanf
- learn the basic of input and output
- learn how printf works
- learn how scanf works
- learn about some operator

printf - Intro

Remember this snippet of code?

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      printf("peko peko");
6      return 0;
7  }
8
```

Remember this snippet of code?

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      printf("peko peko");
6      return 0;
7  }
8
```

<- include <stdio.h> library

Remember this snippet of code?

```c
#include <stdio.h>

int main(void)
{
    printf("peko peko");
    return 0;
}

```

<- main point of entry of a program

Remember this snippet of code?

```c
#include <stdio.h>

int main(void)
{
    printf("peko peko");
    return 0;
}

```

- the function “printf” is from the library <stdio.h>

Remember this snippet of code?

```
1
2
3   int main(void)
4   {
5       printf("peko peko");
6       return 0;
7   }
8
```

- the function “printf” came from the library <stdio.h>
- if without the #include<stdio.h>, it causes a linker error during the linking stage,
- The error will be compainling that it could not find the function’s declaration, More details in next future lessons

Remember this snippet of code?

```c
#include <stdio.h>

int main(void)
{
    printf("peko peko");
    return 0;
}

```

- return 0; is to return a value to end a function execution / returning resources. More details in next future lessons

What is printf?

- printf is a function that allows to print out variables' value or data in string format

- it's a output function

document:
http://www.cplusplus.com/reference/cstdio/printf/

printf - Format (Specifier)

Example 1: printing out a string literal

```c
#include <stdio.h>

int main(void)
{
    printf("peko peko");
    return 0;
}
```

<- print the string “peko peko”

Compile with this command line, follow by the file “a.c”

```
$ gcc -Werror -Wall -Wextra -ansi -pedantic a.c
```

```
$ ./a
peko peko
```

Compile and run, it should print “peko peko”

Example 2: printing out an int

```
int main(void)
{
    int i = 6;
    printf("%d", i);
    return 0;
}
```

Example 3: printing out a float

```
int main(void)
{
    printf("%f", 10.30);
    return 0;
}
```

Reference: http://www.cplusplus.com/reference/cstdio/printf/

- specifier is used to specify which type to print

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

printf - Format (Width, precision)

Reference: http://www.cplusplus.com/reference/cstdio/printf/

- %[flags][width][.precision][length]specifier
- Focus on width, precision and specifier which is mostly used

width

- the minimum number of characters that will be printed for the current specifier
- extra spaces is added if the width is shorter than the minimum number

precision

- The amount of floating point to display

Lets analyze:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%5d:				1	2										
%5.4d:		0	0	3	4										
%5.6d:	0	0	0	0	5	6									
%f	1	2	.	3	4	0	0	0	0						
%8.4f		1	2	.	3	4	0	0							

a = 37, a represent the number of characters printed successfully

Let’s analyze 1:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%5d:				1	2										

- minimum width of 5 characters to be printed
- empty spaces is added to the front to “fill up” at least 5 characters

Let's analyze 2:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%5.4d:		0	0	3	4											

- minimum width of 5 characters to be printed

- a precision of 4
- zeros is added instead of empty spaces for the case of integer

Let’s analyze 3:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%5.6d:	0	0	0	0	5	6									

- minimum width of 5 characters to be printed

- a precision of 6
- zeros is added

Let’s analyze 4:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%f	1	2	.	3	4	0	0	0	0						

- be default, floating point is printed with a precision of 6 floating point

Let’s analyze 5:

```
int main(void)
{
    int a = printf("%5d\n%5.4d\n%5.6d\n%f\n%5.4f\n", 12, 34, 56, 12.34, 12.34);
    printf("%d\n", a);
    return 0;
}
```

%8.4f		1	2	.	3	4	0	0							

- minimum width of 8 characters to be printed

- a precision of 4

Try it yourself

scanf

What is scanf?

- scanf is a function that takes in user's keys input, and store the data into the memory for usage

- it's a input function

http://www.cplusplus.com/reference/cstdio/scanf/

- %[*][width][length]specifier
- Focus only on specifier which is mostly used

Lets analyze 1:

```
int main(void)
{
    /*
    int a = printf("%5d\n%5.4d\n%5.6d\
    printf("%d\n", a);
    */
    int i = 1;
    float j = 1.0f;
    printf("try input:");
    int b = scanf(" %d %f", &i, &j);
    printf("i=%d,j=%f\n", i, j);
    printf("b=%d\n", b);
    return 0;
}
```

- b represent the number of characters successfully, “scanned”
- compile and run the program
- try the following input:
  a. 22 33.33
  b. -33.33
  c. .33
  d. a11.11

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- firstly, scanf scan the first specification, in this case it’s a space,
- scanf will either ignore whitespaces or follow through if the input buffer exist

input buffer:	2	2		3	3	.	3	3

- ~~scanf finds no “whitespaces”~~
- scanf finds no “whitespaces”

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- next, scanf points to the next buffer and it finds “%”,
- it will trigger and continue to find a specifier which is “d”, an integer
- scanf will now stop, and read from the input buffer again

input buffer:	2	2		3	3	.	3	3

- scanf finds the input buffer that is an integer,

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

input buffer:	2	2		3	3	.	3	3

- scanf continues to read the input buffer and finds a “space” which is not an integer, and stops reading

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- scanf read “22” and convert into integer and store it back to “i”

input buffer:	2	2		3	3	.	3	3

- then stores whatever it have read and store convert into integer and store it

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- now scanf continues to read and it and finds “whitespaces”, and ignore

input buffer:	2	2		3	3	.	3	3

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;
scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- likewise scanf now finds “%f” and stop
- scanf now read from the input buffer and finds anything that represent floating number

input buffer:	2	2		3	3	.	3	3

- “whitespaces” at the front are ignored
- reads the entire buffer that can represent floating number
- it stop reading when reaching characters that can’t be represented as floating number or it stops when it reached the end of buffer

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- scanf now convert “33.33” to float and store it in j

input buffer:	2	2		3	3	.	3	3

Lets analyze 1a step by step:

```
int i = 1;
float j = 1.0f;
scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- reach the end of buffer and stop

input buffer:	2	2		3	3	.	3	3

Lets analyze 1b (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f		“),&i,&j

- ignore “whitespaces”
- ignore “whitespaces”

input buffer:	-	3	3	.	3	3		

- i = -33, j = 0.33

Lets analyze 1c (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- ignore “whitespaces”

- scanf stops all operation when it does not match

input buffer:	#	3	3					

Lets analyze 1c (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- scanf stops all operation when it does not match

input buffer:	#	3	3					

Lets analyze 1d (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf(" %d %f", &i, &j);
```

scanf(“		%d		%f	“),&i,&j

- ignore “whitespaces”

- scanf stops all operation when it does not match

input buffer:	a	1	1	.	1	1		

Let's analyze 2:

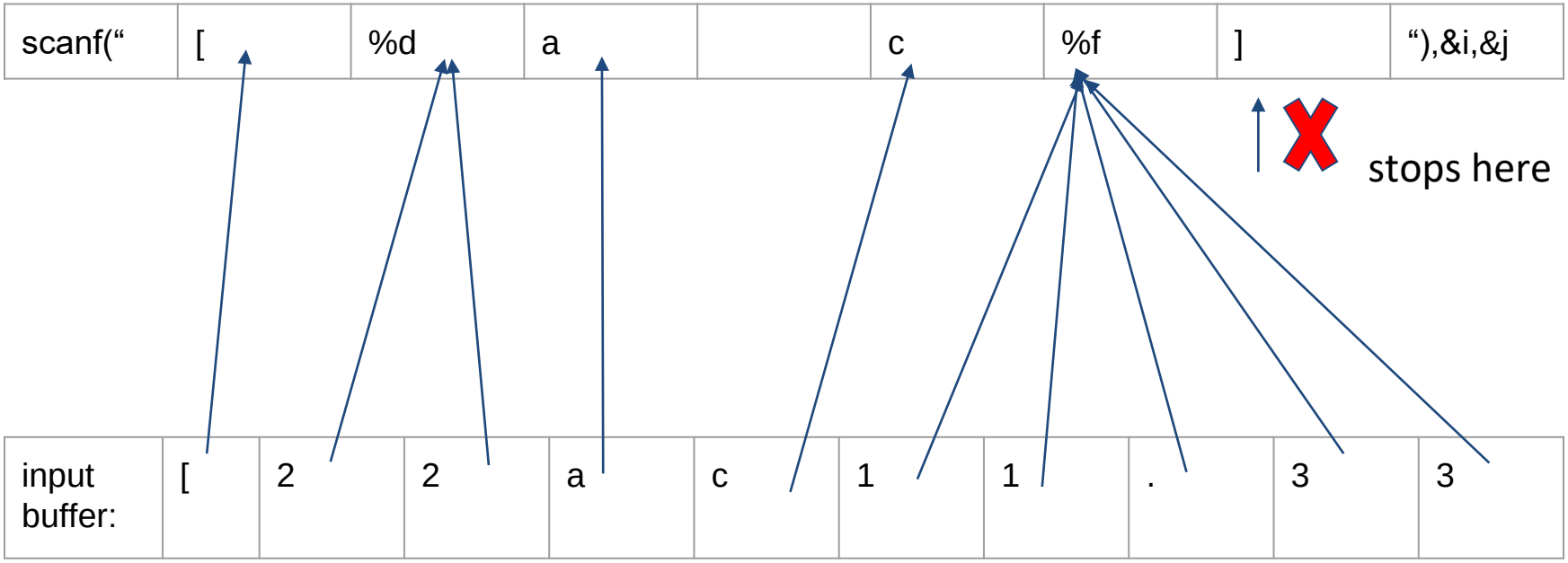
```
int main(void)
{
    /*
    int a = printf("%5d\n%5.4d\n%5.6d\n%
    printf("%d\n", a);
    */
    int i = 1;
    float j = 1.0f;
    printf("try input:");
    int b = scanf("[%da c%f]", &i, &j);
    printf("i=%d,j=%f\n", i, j);
    printf("b=%d\n", b);
    return 0;
}
```

- b represent the number of characters successfully "scanned"
- compile and run the program
- try the following input:
  a. [22ac11.33
  b. [22a c 11. 33
  c. [22a c..33]
  d. [0a c.11.333

Lets analyze 2a (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf("[%da c%f]", &i, &j);
```



- i = 22, j = 11.33

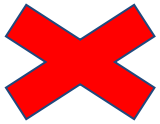
Lets analyze 2b (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf("[%da c%f]", &i, &j);
```

scanf(“	[	%d	a		c	%f	]	“),&i,&j

stops here



input buffer:	[	2	2	a		c		1	1	.		3	3

- i = 22, j = 11.000000

Lets analyze 2c (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf("[%da c%f]", &i, &j);
```

scanf(“	[	%d	a		c	%f	]	“),&i,&j

stops here

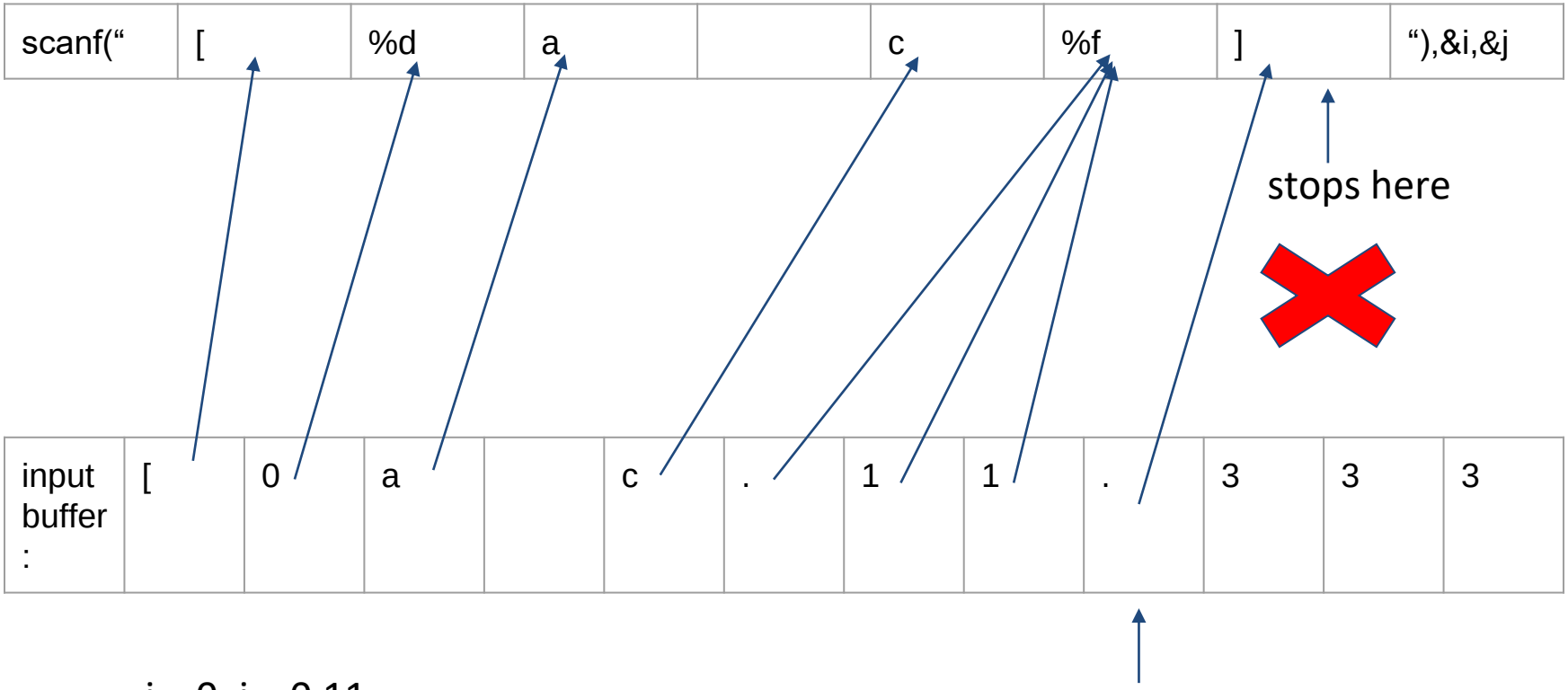
input buffer:	[	2	2	a		c	.	.	3	3	

- i = 22, j = 1.0

Lets analyze 2d (tips and tricks):

```
int i = 1;
float j = 1.0f;

scanf("[%da c%f]", &i, &j);
```



- i = 0, j = 0.11

precedence table

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right

2	`++ --`
	`()`
	`[]`
	`.`
	`->`
		Suffix/postfix increment and decrement
		Function call
		Array subscripting
		Element selection by reference
		Element selection through pointer	

3	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`new, new[]`
	`delete, delete[]`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		Type cast
		Indirection (dereference)
		Address-of
		Size-of
		Dynamic memory allocation
		Dynamic memory deallocation
			Right-to-left

4	`.* ->*`	Pointer to member	Left-to-right
5	`* / %`	Multiplication, division, and remainder	
6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	

15	`?:`
	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Ternary conditional
		Direct assignment (provided by default for C++ classes)
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR
			Right-to-left

16	`throw`	Throw operator (for exceptions)	
17	`,`	Comma	Left-to-right

what is precedence?

- setting the priority of operation
  - example: 1 + 3 * 5 - 2 / 4
  - what operation to be done first?

postfix increment/decrement

- i++ , j--
  - Evaluate to the original value of i
  - and then Increment/decrement by one to itself.
- int i = 1
  - printf(“i=%d”,i++); putput: i=1
  - printf(“i=%d”,i); output: i=2
- int j = 1
  - printf(“j=%d, j=%d”, j--, j); putput: j=1, j=0

prefix increment/decrement

- ++i , --j
  - Increment/decrement by one to itself firstly
- int i = 1
  - printf(“i=%d, i=%d”,++i, i); putput: i=2, i=2
  - printf(“i=%d”,i); output: i=2
- int j = 1
  - printf(“j=%d, j=%d”, --j, j); putput: j=0, j=0

L-value, R-value

Lvalue

- Objects that have a memory location

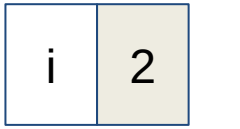
Rvalue

- Objects that does not have a memory location
- Objects that will be discarded if is not assigned to another object
- a.k.a temporary value
- Every C operator operation evaluate to a Rvalue

Example:

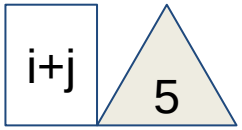
- int i = 2, j = 3, k = 0;
- k = i + j
- right not, i, j, k is a Lvalue

Evaluation:

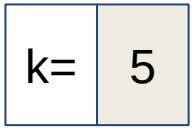


Lvalue

Lvalue



Rvalue



The evaluated value has becomes Lvalue

Exercise:

- int i = 10, j = 15, k = 20;
  - i++ - ++j + --k
  - ++i - j++ + k--
  - ++i + i++ + i++