CS2211b

Software Tools and Systems Programming

Week 8a

Input & Output

Midterm

- Saturday March 3rd @ 9:30 AM
- Location: WSC 55
- Length: 2 hours
- Content: Everything up to (but not including) C Programming (week 6b)
- Format: Mixed (True/False, Multiple Choice, Short Answer, Long Answer)
- Study questions posted on OWL
- One page handwritten notes: one side only, letter size.
- No electronics or calculators

Midterm

Cover Page

Exam ID

Western UNIVERSITY · CANADA

Midterm Examination

CS 2211b Software Tools and Systems Programming

9:30am, March 3rd, 2018

Time for Exam: 120 minutes

This exam has a maximum possible score of 100 points.
One single-sided letter-size page of hand-written notes is allowed.

Name: _____

Student Number: _____

Instructions (PLEASE READ):

- Fill in your name and student number above immediately.
- Have your student card out and on the desk.
- For multiple choice questions, please circle the correct response on this exam paper.
- For short-answer and coding questions, provide your answer in the space provided.
- Write your name on every page in the space provided.
- If you finish within 15 minutes of the end of the exam, you must wait until the exam ends before leaving so as not to distract those who are still working.
- Sheets for rough work may be provided upon request. All paper must be returned with the exam.
- No electronic devices are allowed. **Please turn off your cell phone.**
- **DO NOT TURN THIS PAGE UNTIL DIRECTED TO DO SO**
- The table below is for grading, do not fill in.

Part	Out of	Mark
1. True or False	10	
2. Multiple Choice	30	
3. Short Answers	30	
4. Long form questions	30	
Total	100	

Midterm

Break Down

- 20 True/False Questions: worth 0.5 marks each.
- 30 Multiple Choice Questions: worth 1 mark each.
- 15 Short Answer Questions: worth 2 marks each.
- 3 Long Answer Questions: 2 worth 7.5 marks, 1 worth 15 marks

- Long answer questions will involve writing shell scripts.
- Other question types may cover content up until (but not including) C programming.
- 20 pages long including cover, appendix and blank page.

Midterm

Possible Content

- UNIX/Linux History
- UNIX/Linux Commands
- Operating Systems (kernel, shell, etc.)
- UNIX File System
- UNIX Permissions (symbolic and numeric)
- Filters
- Quoting/escaping
- Processes
- Shell Wildcards
- Regular Expressions
- ASCII
- Binary and Hexadecimal
- Pipes
- Redirection
- Inodes
- Shell Variables
- Text Files vs. Binary Files
- Shell Scripting
  - Loops
  - Arithmetic
  - Conditional Statements
  - Arguments
  - Etc.
- **Anything covered in a lecture.**
- **Anything covered in a lab.**

Midterm

Appendix

You will be given this page.

Do not have to have it on cheat sheet.

Appendix *(anything on this page will not be marked)*

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[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	[NEGATIVE 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	[ENG OF 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	[SUBSTITUTE]	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	[DEL]

Decimal	Binary	Hexadecimal
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Quiz #2 Review

8. The while loop: `while read a; do echo $a; done`

a)Reads only the first word of standard input and prints it until EOF is input/hit.

b)Prints each argument given to the script.

c)Reads a whole line from standard input and prints it until EOF is input/hit.

d)Prints the first argument to the script over and over.

Quiz #2 Review

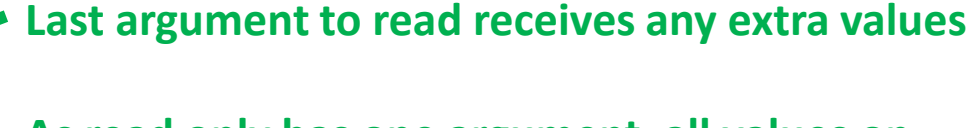
8. The while loop: `while read a; do echo $a; done`

a)Reads only the first word of standard input and prints it until EOF is input/hit.

b)Prints each argument given to the script.

c)Reads a whole line from standard input and prints it until EOF is input/hit.

d)Prints the first argument to the script over and over.




```
while read a; do
   echo $a
done
```

Quiz #2 Review

8. The while loop: `while read a; do echo $a; done`

a)Reads only the first word of standard input and prints it until EOF is input/hit.

b)Prints each argument given to the script.

c)Reads a whole line from standard input and prints it until EOF is input/hit.

d)Prints the first argument to the script over and over.

```
while read a; do
   echo $a
done
```

Output:

```
[dservos5@cs2211b week8]$ quizex1.sh
this is a test
this is a test
hello world 1 2 3
hello world 1 2 3
```

Input via stdin

Output via stdout

Quiz #2 Review

8. The while loop: `while read a; do echo $a; done`

a)Reads only the first word of standard input and prints it until EOF is input/hit.

b)Prints each argument given to the script.

c)Reads a whole line from standard input and prints it until EOF is input/hit.

d)Prints the first argument to the script over and over.

A different example:

```
while read a b; do
   echo $a
done
```

Output:

```
[dservos5@cs2211b week8]$ quizex2.sh
this is a test
this
hello world 1 2 3
hello
```

Input via stdin

Output via stdout

Input & Output

The `printf` Function

- We have seen that the printf function can be used for printing strings:

```
printf("hello world!\n");
```

- but it can also be used for printing the value of variables:

```
int x = 5;
float a = 3.4f;
printf("My vars are equal to: %d and %f\n", x, a);
```

- **Output:**

```
My vars are equal to: 5 and 3.4
```

The printf Function

- We have seen that the printf function can be used for printing strings:

```
printf("hello world!\n");
```

- but it can also be used for printing the value of variables:

```
int x = 5;
float a = 3.4f;
printf("My vars are equal to: %d and %f\n", x, a);
```

- **Syntax:**

```
printf("format string", vars …)
```

The `printf` Function

Format specifiers:

%c	Character
%s	String
%u	Unsigned Integer
%d	Integer
%i	Same as %d (Integer)
%f	Floating-point number (Double)
%%	The % sign
%e	Floating-point number in exponential notation
%g	Floating-point number in exponential or normal notation depending on the numbers size.
%p	Pointer (memory address)
%l<c>	Long format (Where <c> is another format specifier.)

The `printf` Function

Escape Sequences:

\n	**Newline**
\b	**Backspace**
\t	**Tab**
\r	**Carriage Return**
\a	**System Bell**
****	**The Literal \ Character**
\"	**The Literal " Character**
\'	**The Literal ' Character**

The `printf` Function

Escape Sequences:

\n	Newline
\b	Backspace
\t	Tab
\r	Cartridge Return
\a	System Bell
\\	The Literal \ Chara
\"	The Literal " Chara
\'	The Literal ' Chara

Newline vs. Cartridge Return

The cartridge return (also called carriage return) character (\r) is a concept that dates back to typewriters. The cartridge return on typewriters referred to a lever that would cause the carriage to return to the far left after typing a line of text.

For computers, it is an ASCII character that tells the cursor to return to the far left of a line of text. Was use in early printers and to overwrite the current line of text.

The `printf` Function

Escape Sequences:

\n	Newline
\b	Backspace
\t	Tab
\r	Cartridge Return
\a	System Bell
\\	The Literal \ Chara
\"	The Literal " Chara
\'	The Literal ' Chara

Newline vs. Cartridge Return

Lead to a few different standards of line breaks:

UNIX and UNIX-Like systems just use \n to denote a newline (for example in text files).

Windows, DOS and a few others use \r followed by \n to denote a line break (i.e. \r\n).

Some older systems (8bit commodore machines) just used \r.

Other obscure systems (RISCO OS, BBC Micro) used \n\r

As we are using UNIX, just use \n for line break.

The `printf` Function

Escape Sequences:

\n	Newline
\b	Backspace
\t	Tab
\r	Cartridge Return
\a	System Bell
\\	The Literal \ Character
\"	The Literal " Character
\'	The Literal ' Character

Newline vs. Cartridge Return

If you need to convert a UNIX text file to use Windows style Line Breaks (or vice versa) you can use the commands: `unix2dos` and `dos2unix`

The `printf` Function

Escape Sequences:

\n	**Newline**
\b	**Backspace**
\t	**Tab**
\r	**Cartridge Return**
\a	**System Bell**
****	**The Literal \ Chara**
\"	**The Literal " Chara**
\'	**The Literal ' Character**

Bell Character

Used to ring a small electromechanical bell on teleprinters and teletypewriters.

May still be supported today for backwards compatibility. Often plays the system warning sound.

Example: /cs2211/week8/bell.c
May or may not play something based on your terminal and settings.

The printf Function

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

The printf Function

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf =
        printf("i = %d\tf =
        printf("f = %f\tf =

        return 0;
}
```

Variable	Value
i	42
f	3.1415
c	'a'

Declare and initialize variables.

Note that we have to use single quotes (') when giving character constants to tell C that this is the character a and not the variable a.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Print the literal text: i =

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Print an integer value.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Print a tab character.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);
}
```

Print the literal text: f =

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\t

        return 0;
}
```

Print a floating point value.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Print a tab character.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

What does this section do?

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Print a line break/new line.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

First argument to printf is used as first value in format string.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Second argument to printf is used as second value in format string.

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

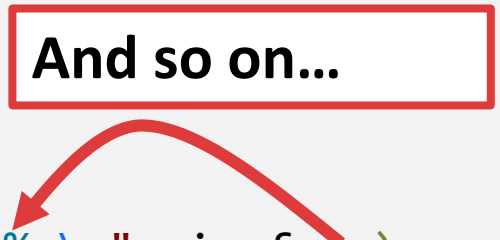
/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```



The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

What will this line output?

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Output of first printf:

```
i = 42   f = 3.141500   c = a
```

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        re
}
```

Arguments to printf don't have to be variables. Can be constants or expressions too.

Adding x to character moves it up by x on the ASCII table. For example 'a' + 1 = 'b' and 'X' + 5 = ']'

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        re
}
```

Output of this line:

```
i = 420  f = 0.314150  c = b
```

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Floats and doubles can be printed with %f, %e or %g format specifiers (described on slide 14).

The printf Function

Variable	Value
i	42
f	3.1415
c	'a'

Example 1:

/cs2211/week8/ex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        printf("i = %d\tf = %f\tc = %c\n", i, f, c);
        printf("i = %d\tf = %f\tc = %c\n", i*10, f/10.0, c+1);
        printf("f = %f\tf = %e\tf = %g\n", f, f, f);

        return 0;
}
```

Output of this line:

```
f = 3.141500  f = 3.141500e+00 f = 3.1415
```

The `printf` Function

Important Considerations

- `printf` does not check that the number of arguments match the number of format specifiers.
  - Giving too many or too few can lead to implementation specific or undefined behavior.

- `printf` does not check that type of arguments given match the type of the format specifiers.
  - Mismatched types may produce undesirable results.

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```
#include <stdio.h>

int main() {
        int
        floa
        char

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Has more format specifiers than arguments.

Values printed for %f and %c will be “unpredictable”.

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'A';

        //Too few arguments:
        printf("i = %d\n\n", i);
        //Too many arguments:
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Has more arguments than format specifiers

Values of variables **f** and **c** will not be printed.

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```
#include <std

int main() {
        int i
        float
        char

        //Too
        print
        //Too
        print

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Using the wrong specifiers for the given variable types.

In some cases the output will be “unpredictable”, in other cases C will try to interpret the variable as that type (e.g. character variable c will be printed as decimal ASCII value of 'a').

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        prin

        retu
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

Should be printing value of i not f.

Undefined behavior can produce strange and hard to understand results.

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

The printf Function

Bad Example 1:

/cs2211/week8/badex1.c

```c
#include <stdio.h>

int main() {
        int i = 42;
        float f = 3.1415f;
        char c = 'a';

        //Too few arguments.
        printf("i = %d\tf = %f\tc = %c\n", i);
        //Too many arguments.
        printf("i = %d\n\n", i, f, c);

        //Wrong specifiers
        printf("i = %f\n", i);
        printf("f = %d\n", f);
        printf("c = %d\n", c);

        return 0;
}
```

Output:

```
i = 42   f = 0.000000     c = X
i = 42

i = 3.141500
f = -409333760
c = 97
```

97 is 'a' on ASCII table.

The `printf` Function

Formatting Output

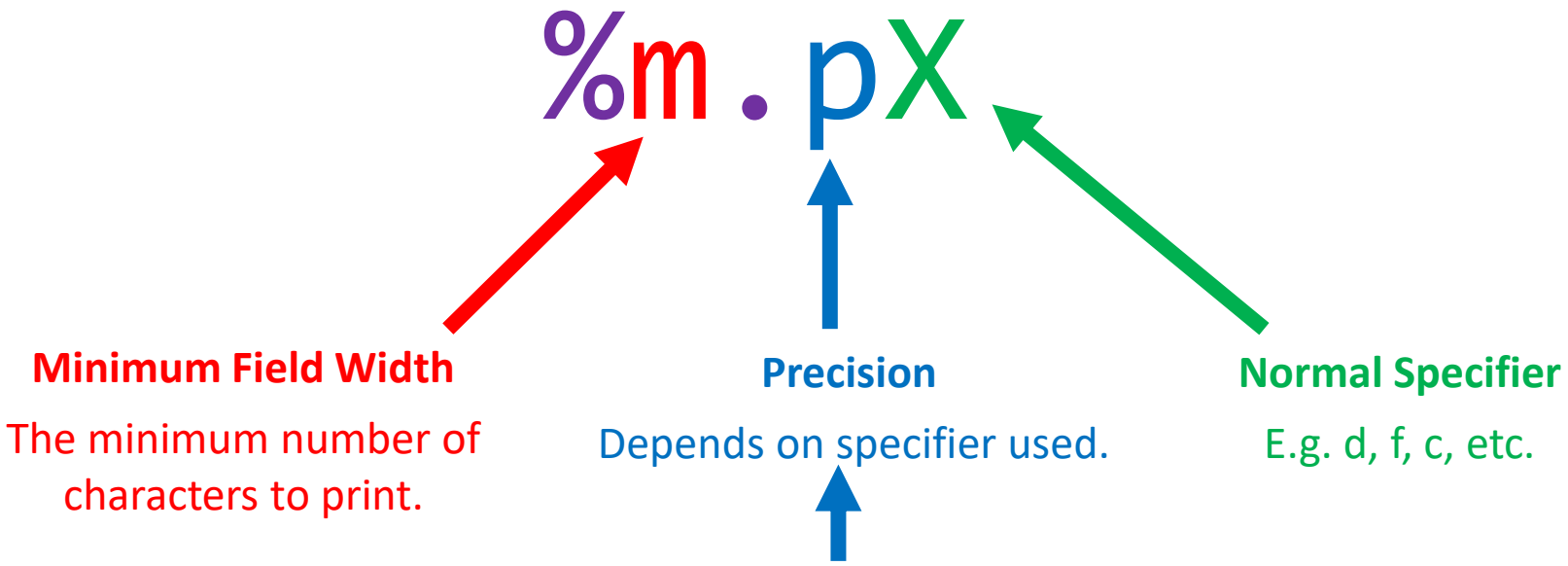
- `printf` also supports specifiers to control the size and precision of the output.
- **Specifier Syntax:**

```
%m.pX
```

The `printf` Function

Formatting Output

- `printf` also supports specifiers to control the size and precision of the output.
- **Specifier Syntax:**



The printf Function

Formatting Output

Example 2:

/cs2211/week8/ex2.c

```c
#include <stdio.h>

int main() {
        float f = 3.1415f;
        int i = 42;

        printf("|%.5d|%.2f|\n", i, f);

        printf("|%4d|%10f|\n", i, f);

        printf("|%4.3d|%10.3f|\n", i, f);

        return 0;
}
```

The pri

Formatting

Example 2:

/cs2211/week8/ex

```
#include <stdi

int main() {
        float
        int i

        printf("|%.5d|%.2f|\n", i, f);

        printf("|%4d|%10f|\n", i, f);

        printf("|%4.3d|%10.3f|\n", i, f);

        return 0;
}
```

Precision of integer specifier set to 5, precision of float specifier set to 2.

Integer must have at least 5 digits (will pad with zeros).

Float can have at most 2 decimal places (will be rounded).

| is a literal pipe character.

The pri

Formatting

Example 2:

/cs2211/week8/ex

```
#include <stdi

int main() {
        float
        int i

        printf("|%.5d|%.2f|\n", i, f);

        printf

        printf

        return 0;
}
```

Precision of integer specifier set to 5, precision of float specifier set to 2.

Integer must have at least **5 digits** (will pad with zeros).

Float can have at most **2 decimal places** (will be rounded).

| is a literal pipe character.

Output of this line:

|00042|3.14|

The printf Function

Formatting

Example 2:

/cs2211/week8/ex2

```
#include <stdi

int main() {
        float
        int i

        printf

        printf("|%4d|%10f|\n", i, f);

        printf("|%4.3d|%10.3f|\n", i, f);

        return 0;
}
```

Minimum width of integer specifier set to 4, minimum width of float specifier set to 10.

Integer must take up at least 4 spaces (whitespace will be padded to front of output).

Float must take up at least 10 spaces (whitespace will be padded to front of output).

| is a literal pipe character.

The printf Function

Formatting

Example 2:

/cs2211/week8/ex2

```
#include <stdi

int main() {
        float
        int i

        printf

        printf("|%4d|%10f|\n", i, f);

        printf(

        return
}
```

Minimum width of integer specifier set to 4, minimum width of float specifier set to 10.

Integer must take up at least 4 spaces (whitespace will be padded to front of output).

Float must take up at least 10 spaces (whitespace will be padded to front of output).

| is a literal pipe character.

Output of this line:

```
|  42|  3.141500|
```

Two blank spaces

Two blank spaces (decimal point takes up a space)

The `printf` Function

Formatting Output

Example 2:

/cs2211/week8/

```
#include <st

int main() {
        floa
        int

        prin

        prin

        printf("|%4.3d|%10.3f|\n", i, f);

        return 0;
}
```

Setting both minimum width and precision of both specifiers.

Integer must show at least 3 digits and take up at least 4 spaces.

Float must show at most 3 decimals and take up at least 10 spaces.

The printf Function

Formatting Output

Example 2:

/cs2211/week8/

```
#include <st

int main() {
        floa
        int

        prin

        prin

        printf("|%4.3d|%10.3f|\n", i, f);

        retu
}
```

Setting both minimum width and precision of both specifiers.

Integer must show at least 3 digits and take up at least 4 spaces.

Float must show at most 3 decimals and take up at least 10 spaces.

Output of this line:

```
|  042|     3.141|
```

The printf Function

Formatting Output

Example 2:

/cs2211/week8/

```
#include <st

int main() {
        floa
        int

        prin

        prin

        printf("|%4.3d|%10.3f|\n", i, f);

        retu
}
```

Setting both minimum width and precision of both specifiers.

Integer must show at least 3 digits and take up at least 4 spaces.

Float must show at most 3 decimals and take up at least 10 spaces.

Output of this line:

Five blank spaces

One blank space → | 042| 3.141|

The `scanf` Function

- The `scanf` function allows us to read into input from the user via the **standard input stream**.
- Uses a similar syntax and format string as `printf` but is quite different in a number of important ways.

- **Syntax:**

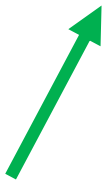
```
scanf("format string", vars …)
```

The `scanf` Function

- The `scanf` function allows us to read into input from the user via the **standard input stream**.
- Uses a similar syntax and format string as `printf` but is quite different in a number of important ways.

- **Syntax:**

```
scanf("format string", vars …)
```



Format String

Specifies how/what values will be read in.

Address of (pointers to) variables to read values into.

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Variables x, y, a and b declared but not initialized.

The scanf Function

Exampl

/cs2211/v

```
#includ

int mai
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

scanf format string

Specifies that the input will be an integer followed by an integer, followed by a float, followed by a float.

Specifiers are same as in `printf` (%d for int, %f for float).

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

}
```

Read first value into variable x

The & here means that we are passing scanf the address of the variable x and not the value of the variable x (pass by reference and not pass by value).

This will become more clear when we get to functions and pointers. For now, know that we need to use an & in front of each variable when using scanf (at least in most cases).

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Read second value into variable y

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Read third value into variable a and so on…

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Output the values of the variables.

The scanf Function

Example 3:

/cs2211/week8/ex3.c

```c
#include <stdio.h>

        float a, b;

        scanf("%d%d%f%f", &

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Input via keyboard (stdin)

Example 3 Input/Output 1:

```
[dservos5@cs2211b week8]$ ex3
5 42 0.123 10.34
x=5 y=42
a=0.123000 b=10.340000
```

Output (stdout)

The scanf Function

scanf ignores whitespace between numbers.

```c
int main() {
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 1:

```
[dservos5@cs2211b week8]$ ex3
5 42 0.123 10.34
x=5 y=42
a=0.123000 b=10.340000
```

The scanf Function

scanf ignores whitespace between numbers.

Includes line breaks.

```
        float a, b;

        scanf("%d%d%f%f", &

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 2:

```
[dservos5@cs2211b week8]$ ex3
1 2
3.4 5.6
x=1 y=2
a=3.400000 b=5.600000
```

nction

scanf supports negative numbers and different formats for floats without having to use %e or %g.

```
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 3:

```
[dservos5@cs2211b week8]$ ex3
-10 -30 .123 4.5e3
x=-10 y=-30
a=0.123000 b=4500.000000
```

nction

scanf keeps reading characters until they can no longer be part of the given format specifier. It then moves on to the next format specifier.

```
        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 4:

```
[dservos5@cs2211b week8]$ ex3
5-30.5-4.0
x=5 y=-30
a=0.500000 b=-4.000000
```

The scanf Function

- (minus sign) can only appear at start of number so scanf knows this is the end of the first value and the start of the second.

```
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 4:

```
[dservos5@cs2211b week8]$ ex3
5-30.5-4.0
x=5 y=-30
a=0.500000 b=-4.000000
```

The scanf Function

Integers (%d) can not have decimal places so scanf knows this is the end of the second value and the start of the third value (a float).

```
        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 4:

```
[dservos5@cs2211b week8]$ ex3
5-30.5-4.0
x=5 y=-30
a=0.500000 b=-4.000000
```

The scanf Function

Just like before, the minus sign indicates the end of the third value and the start of the fourth.

```
        int x, y;
        float a, b;

        scanf("%d%d%f%f", &x, &y, &a, &b);

        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 4:

```
[dservos5@cs2211b week8]$ ex3
5-30.5-4.0
x=5 y=-30
a=0.500000 b=-4.000000
```

The scanf Function

If scanf reaches input that can not possibly be valid, it will abort and leave the remaining variables uninitialized.

```
        scanf("%d%d%f%f", &

        printf("x=%d y=%d\r
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 5:

```
[dservos5@cs2211b week8]$ ex3
5 bad input
x=5 y=0
a=0.000000
b=3589968825719207484823961 6.0
00000
```

The scanf Function

If scanf reaches input that can not possibly be valid, it will abort and leave the remaining variables uninitialized.

Example 3

Only x is given a value as it was read before "bad input"

```
[dservos5@cs2211b week8]$ ex3
5 bad input
x=5 y=0
a=0.000000
b=3589968825719207484823961 6.0
00000
```

```
        scanf("%d%d%f%f", &

        printf("x=%d y=%d\r
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

The scanf Function

If too many values are given, they are ignored an left in the input buffer.

If scanf is called again it will start by reading these, not new input.

```
        printf("x=%d y=%d\n", x, y);
        printf("a=%f b=%f\n", a, b);

        return 0;
}
```

Example 3 Input/Output 6:

```
[dservos5@cs2211b week8]$ ex3
1 2 3.3 4.4 5 6 7 8 9 10
x=1 y=2
a=3.300000 b=4.400000
```

The scanf Function

White Space & Other Characters

- In addition to format specifiers, we can also use white space and other characters in scanf's format string.
- If non format specifier characters are used, scanf will read these literal characters from the **buffer** but not use them as part of a variable's value.
- If extra whitespace is placed in the format string, scanf is instructed to read and ignore **zero or more** white space characters from the **buffer**.

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

The scanf Function

Example 4:

/cs2211/week8/

```c
#include <s

int main()
        int

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

Literal characters + and - must be matched between integer values.

These characters will be ignored if matched. Will not become part of value (does not make the third value negative).

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

Example 4 Input/Output 1:

```
[dservos5@cs2211b week8]$ ex4
5+7-3
5 + 7 - 3 = 9
```

Arithmetic and printing the values, +, - and = is done by `printf` line.

`scanf` just read 5 into x, 7 into y and 3 into z.

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

Example 4 Input/Output 2:

```
[dservos5@cs2211b week8]$ ex4
5 7 3
5 + 0 - 32765 = -32760
```

If we fail to include the + or – in our input we will have issues.

First value read ok, but scanf aborts after failing to find a +.

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

Example 4 Input/Output 2:

```
[dservos5@cs2211b week8]$ ex4
5 7 3
5 + 0 - 32765 = -32760
```

If we fail to include the + or – in our input we will have issues.

First value read ok, but scanf aborts after failing to find a +.

Variables y and z are not initialized.

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

        return 0;
}
```

Example 4 Input/Output 3:

```
[dservos5@cs2211b week8]$ ex4
5+ 7- 3
5 + 7 - 3 = 9
```

Having spaces after the + and - work ok, as scanf ignores blank spaces when trying to read in a number (%d or %f).

But….

The scanf Function

Example 4:

/cs2211/week8/ex4.c

```c
#include <stdio.h>

int main() {
        int x, y, z;

        scanf("%d+%d-%d", &x, &y, &z);

        printf("%d + %d - %d = %d\n", x, y, z, x + y - z);

}
```

Example 4 Input/Output 4:

```
[dservos5@cs2211b week8]$ ex4
5 +7 -3
5 + 0 - 32767 = -32762
```

Having them before will cause an issue as `scanf` is looking for a + and not a space.

We can fix this by adding a space in the format string before the literal characters: `"%d +%d -%d"`

This will work as a space specifies that `scanf` should read zero or more spaces. *Try /cs2211/week8/ex4b.c*

The scanf Function

Minimum Width

- Like printf, scanf supports a minimum width in its specifiers:

%mX

- **Example:**

```
scanf("%3d%3d%4d", &x, &y, &z);
printf("(%d) %d-%d\n", x, y, z);
```

```
[dservos5@cs2211b week8]$ exminwidth
5199145555
(519) 914-5555
```