Lab 7 – InLab

CSC 3320 SYSTEM LEVEL PROGRAMMING

Objective

- To learn how to print output using **printf**
- To learn how to get input using **scanf**

printf function

- Print output: e.g. printf("%d", x);
- *printf* function must be supplied with a ***format string*** as 1st argument.
- The format string may contain both **ordinary characters** and **Conversion specification-s.**
  - E.g. "x has a value %d, y has a value %f"
    - **%d** and **%f** are type conversion code
    - Other characters are ordinary characters

print Function

`printf(formatString, expr1, expr2, ...);`

- **Ordinary characters** in a format string are printed as they appear in the string
- **Conversion specifications** are replaced by values of expressions in a sequential manner.

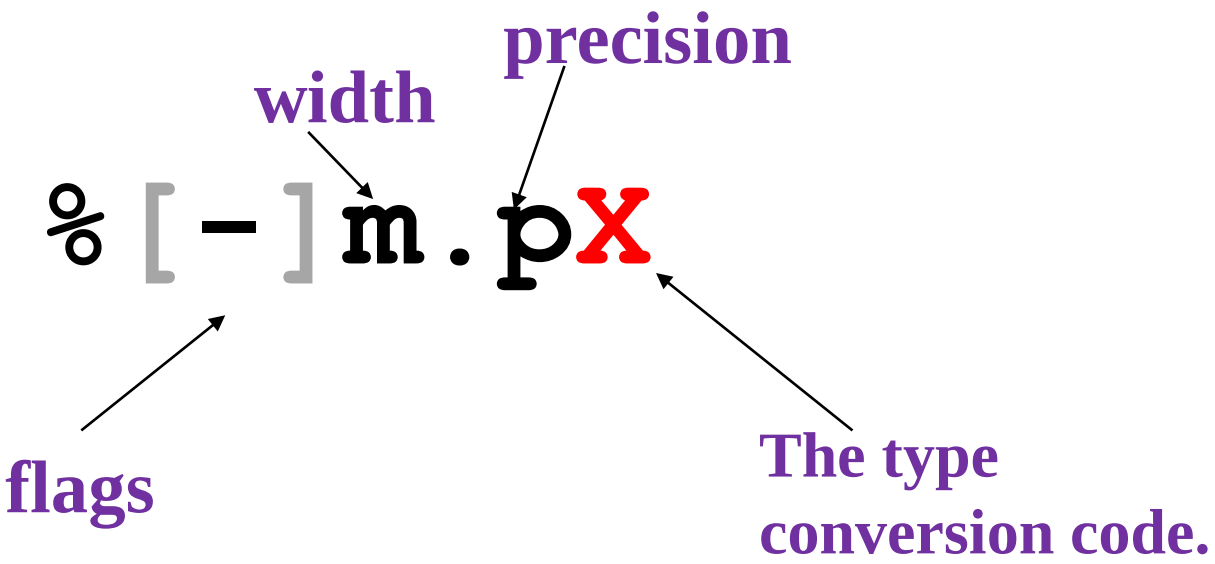
Example:

```
int i, j;
float x, y;

i = 10;
j = 20;
x = 43.2892f;
y = 5527.0f;

printf("i = %d, j = %d, x = %f, y = %f\n", i, j, x, y+1);
```

Output:

i = 10, j = 20, x = 43.289200, y = 5528.000000

Conversion Specifications



Type Conversion Code

Type	Type Conversion Code	Type & Format
Integer	**%d**	(signed) int
	%o	int in octal
	%x	int in hexdecimal
	%l	long
	%h	short
Floating-point	**%f**	float
	%lf	double
Character	**%c**	char
String	**%s**	string stored in char []

Conversion Specifications - Width

%[-]m.pX

- **m**: Specifies the minimum number of characters to print.
  - If the value has a width less than m, the field will be padded with spaces
  - Else, the value will not be truncated.

- E.g. `%10d` will display an `int` right justified in a ten character space.

Conversion Specifications - Flags

%[-]m.pX

- **-** : Putting a minus sign in front of ***m*** causes left justification; otherwise, only right justification

- E.g. `%-10d` will display an `int` left justified in a ten character space.

Conversion Specifications

%[-]m.pX

- . : Matches a decimal point.
- **p**: Precision.
  - For floating-point number, it represents the number of characters used after the decimal.

- E.g. %-10.3f will display a float using ten characters, with **three** digits after the decimal point.

More practice

- Write down the output for the following statements.

(a) `printf("%6d,%4d", 25, 1234);`

□□□□**25,1234**

(b) `printf("%.1f", 3.1415926);`

3.1

(c) `printf("%8.1f", 20.253);`

□□□□**20.3**

(d) `printf("%-10.5f", 20.253);`

20.25300□□

scanf Function

& must be used for int, char, double and float variables

- Get the input: e.g. scanf("%d", &x);
- In many cases, a `scanf` format string will contain only conversion specifications.

Example: `scanf("%d%d%f%f", &i, &j, &x, &y);`

- The input as below

```
   1
-20   .3
   -4.0e3
```

What value will be assigned to each variable?

How **scanf** work?

- `scanf` tries to **match groups of input characters** with conversion specifications in the format string.
- For each conversion specification, `scanf` tries to locate an item of the appropriate type in the input data, **skipping blank space before it if necessary**.
- `scanf` then reads the item, **stopping** when it reaches a character that can't belong to the item.
- If the item was read successfully, `scanf` continues processing the rest of the format string.

scanf Function

Example: `scanf("%d%d%f%f", &i, &j, &x, &y);`

- The input is as below:

```
   1
-20   .3
   -4.0e3
```

- `scanf` sees a stream of characters (¤ represents new-line, □ represents a space):

```
□□1¤-20□□□.3¤□□□-4.0e3¤
ssrsrrrssrrssssrrrrrr
```

(`s` = skipped; `r` = read)

- `scanf` "peeks" at the final new-line without reading it.

scanf Function

- A `scanf` format string can also contain literal strings in it.

- Then the input must match the literal string.

Example

%d/%d will match □5/□96, but not □5□/□96

%d□/%d will match □5□/□96