# CSE 220 – C Programming

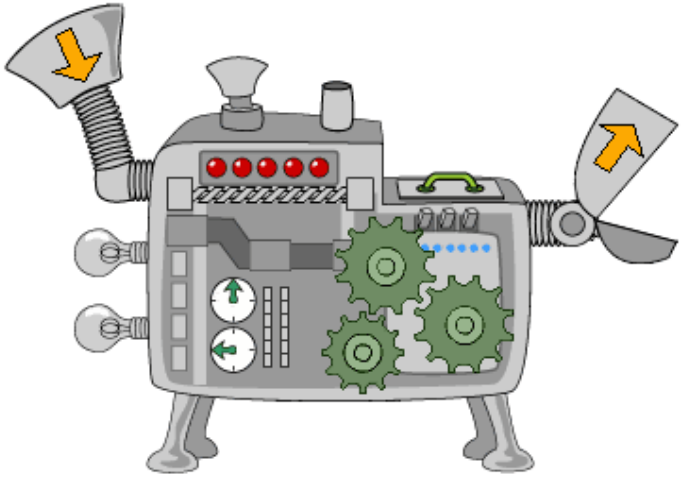
Formatted Input and Output

# Lab Zoom Meeting

- 3 breakout rooms
  - 10-12 people for each room
  - Free discussion in each room
- TA will access each room every 15 mins
- TA will post some common Q&A on Piazza
- The last 30 mins are reserved for credit checking

# Summary

- **Printf General Syntax**
- Scanf General Syntax
- Common Mistakes



http://grade5eisnor.weebly.com/uploads/2/5/1/5/25151059/8233546_orig.gif

# What do you think the following code outputs?

```
int a = 220;
float b = 3.5f;
printf("Your grade in %d is %f\n",
      a, b);
```

Your grade in %d is %f\n

Your grade in 220 is 3.5

Your grade in a is b

Your grade in 220 is 4.0

# Printing output

- printf: used to print output to screen
- Defined in stdio.h
- Usage:

```
printf(format_string, expr1, expr2, expr3, …);
```

- No limit on the number of expressions

# Expressions

```
printf(format_string, expr1, expr2, expr3, …);
```

- expr1, expr2, …: constants, variables, complicated expressions

```
printf(“The value of %d multiplied by %f is %f\n”,
       2, PI, (2*PI) );
```

# Format String

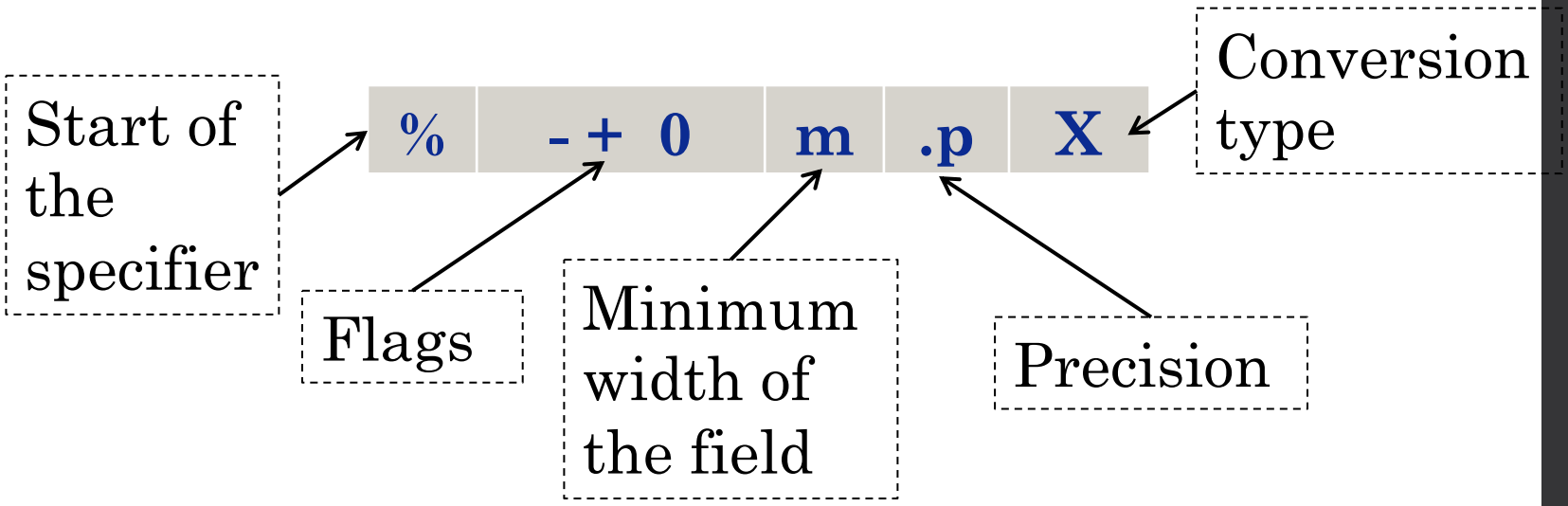
```
printf(format_string, expr1,
 expr2, expr3, …);
```

- Contains: ordinary characters and conversion characters:
- Conversion characters:
  - placeholder for a value to be filled
  - specifies how to convert the value into printed form

```
printf(“The value of %d multiplied by %f is %f\n”, 2, PI, (2*PI) );
```

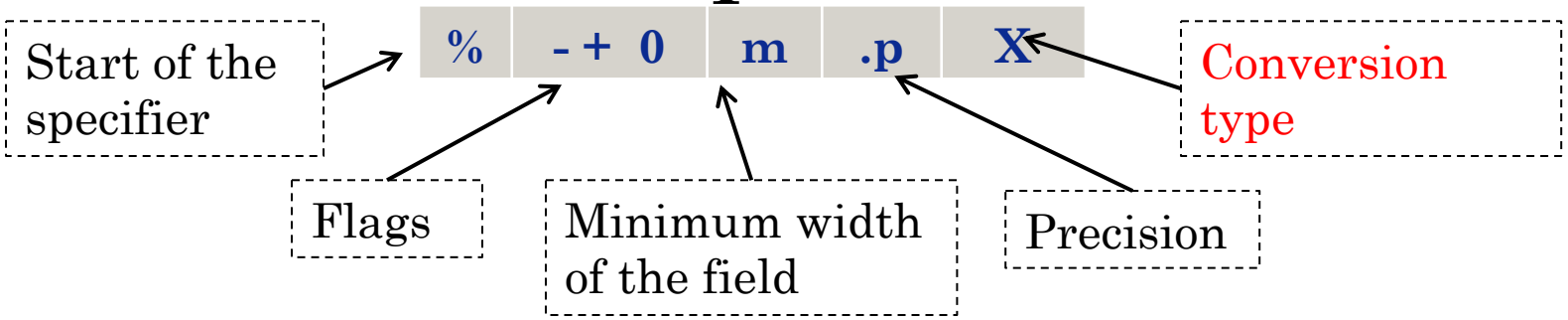
# Conversion Specification

General format:




```
printf("Your score is: %-4d", x);
printf("%-6.3d", x);
```

# Conversion Specification




- Conversion type:
  - c: a single character
  - s: string
  - d: integer
  - f: floating point notation
  - E,e: scientific notation
  - u: unsigned integer
  - X,x: hexadecimal number

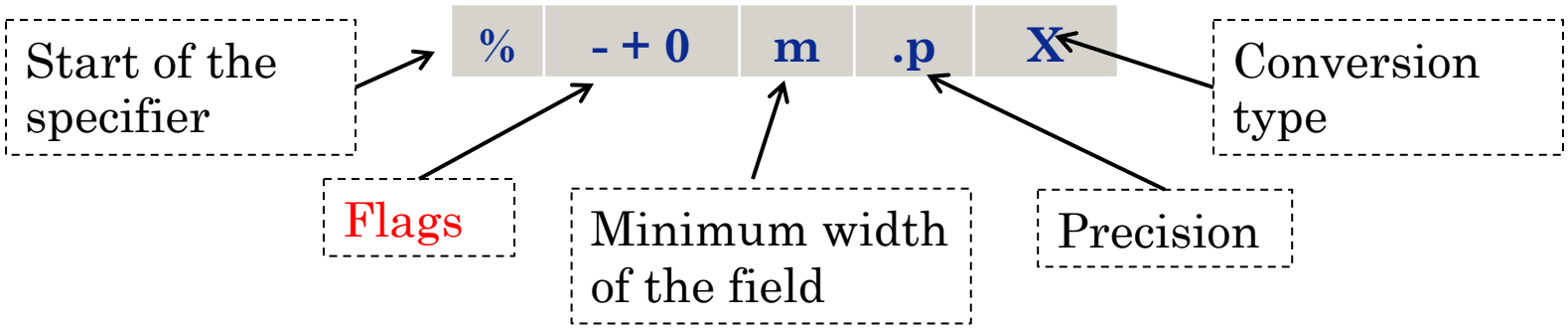
# Conversion Specification

```
int x = 20;
float y = 74.0231f;
char z = 'd';
printf("%d %f %c\n", x, y, z);   20 74.023100 d
printf("%e\n", y);               7.402310e+01
printf("%E\n", y);               7.402310E+01
printf("%d %c\n", x);            20 ?
printf("%d\n", x, z);            20
```

2nd value is unpredictable

*z is not printed since no placeholder for it*

# Conversion Specification



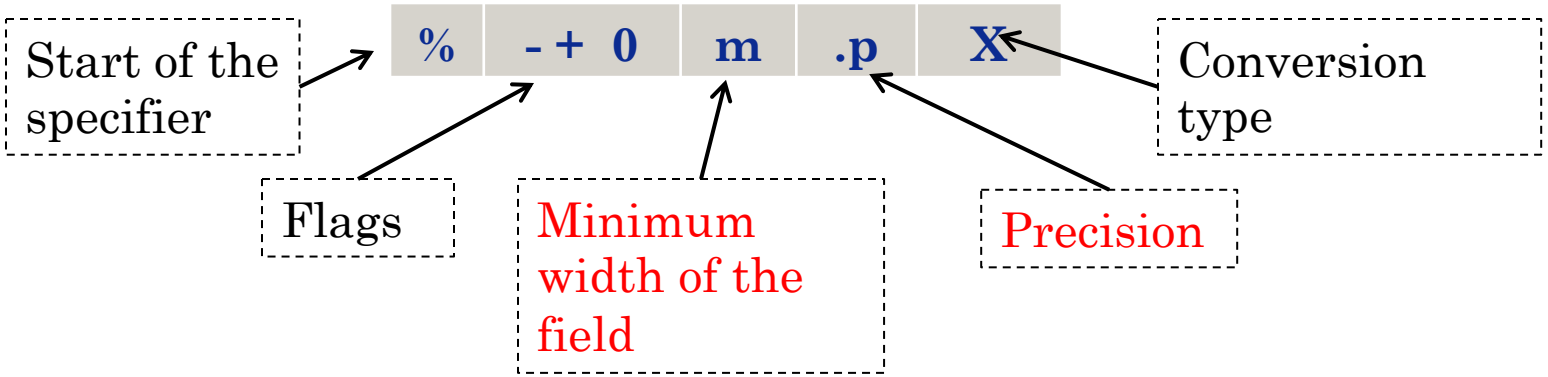

Flags:

- : Left justify
+: always print sign (+/-)
0: pad with leading zeros instead of spaces

Can multiple flags in one specifier

```
int x = 20, y = -20;
printf("%d %d\n", x, y);
20 -20
printf("%+d %+d\n", x, y);
+20 -20
```

# Conversion Specification




- Minimum width:
  - The minimum characters to print
  - Pads with spaces if not enough characters
- Precision:
  - depends on the conversion specifier
  - with e and f: number of decimal digits
  - with d: minimum number of digits

# Example

```
float x = 5.123456f;

printf(“%f\n”, x);

5.123456

printf(“%+.3f\n”, x);

+5.123

printf(“%+10.3f\n”, x);

     +5.123

printf(“%-10.3f is my lucky number!\n”, x);

5.123      is my lucky number!
```

4 leading spaces to make total count 10

# Escape Sequence

\a: alert (bell sound)

\n: new line

\t: horizontal tab

\b: backspace

\": quotation mark

Want: printf(""Hello"");

Interpreted: printf(""Hello"");

Use: printf("\"Hello\"");

\\: single \ character

printf("\\Hello\\");

Will print: \Hello\

# What do you think the following code outputs?

```
int a = 220;
float b = 3.5f;
printf("\"grade\" in %d \\\t %f",
     a, b);
```

"grade" in 220 \\t 3.5

\grade\ in 220 \  3.5

"grade" in 220 \  3.5

None of the above

# Error Checking

- C compilers are not required to check that the number of conversion specifications matches the number of output items:

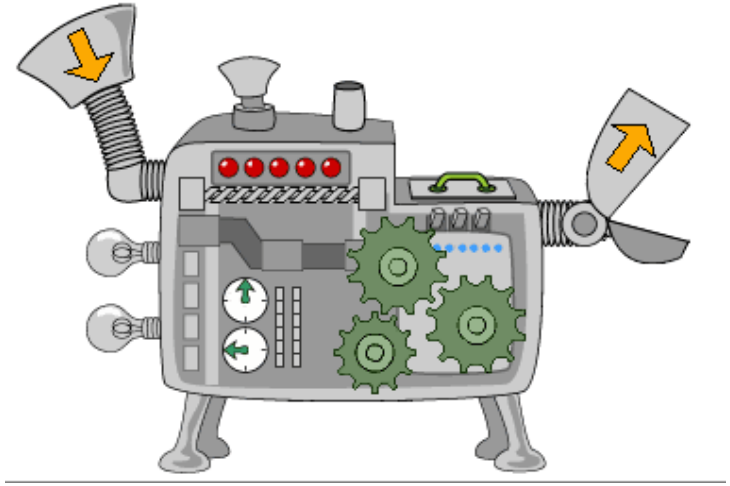
```
printf("%d %d", x, y, z);
printf("%d %d", x);
```

- C compilers are not required to check that the type of conversion specification is appropriate

```
int myInt;
float myFloat;
printf("%f %d", myInt, myFloat);
```

# Summary

- Printf General Syntax
- **Scanf General Syntax**
- Common Mistakes



http://grade5eisnor.weebly.com/uploads/2/5/1/5/25151059/8233546_orig.gif

# Reading input

- scanf: used to read input according to given format
- Defined in stdio.h
- Usage:

```
scanf(format_string, var1, var2, var3, ….);
```

- No limit on the number of variables

# Format String

- Contains: ordinary characters and conversion characters:
- Conversion characters:  same as printf

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

  - Convert first value to an integer
  - Convert second value to a float
  - %e, %f: are interchangeable for scanf

# Maximum-Length

- If you don't want to consume an entire number from input, you can specify a maximum length for a conversion.
- Example:
  - I want to only store the first two digits of the input 8492 (i.e. 84)
  - `scanf("%2d", &var);`
  - The next scanf starts at the 9 (the unconsumed input).

If the input is "480274", what does the following code output?

```
int a, b, c;
scanf("%1d", &a);
scanf("%2d%d", &b, &c);
printf("%d %d %d", c, b, a);
```

274 80 4

3 2 1

4 80 274

None of the above

# Error Checking

- C compilers are not required to check that the number of conversion specifications matches the number of output items
- C compilers are not required to check that the type of conversion specification is appropriate
- C compilers are not required to not check for the  (usually) required & in scanf
  - program crash, value not read, warning

# How scanf works

- Reads input data from left
- Skips blanks
- Reads the item until it reads a character that cannot belong to the item according to the conversion specification

```
%d:     ----10.4---------5
```

  - In this case, the %d matches the 10 (integers don't have decimal points).
- If successful: continues processing the format string
- If not: returns immediately
- If more input, belongs to next scanf call

# Ordinary Characters

- Pattern matching
- If white space in format string:
  - keeps reading, matching with whitespaces in input
  - One white space character in format string matches any number of white spaces in input
- If other character:
  - If matching: discards input, continue processing
  - Otherwise: aborts

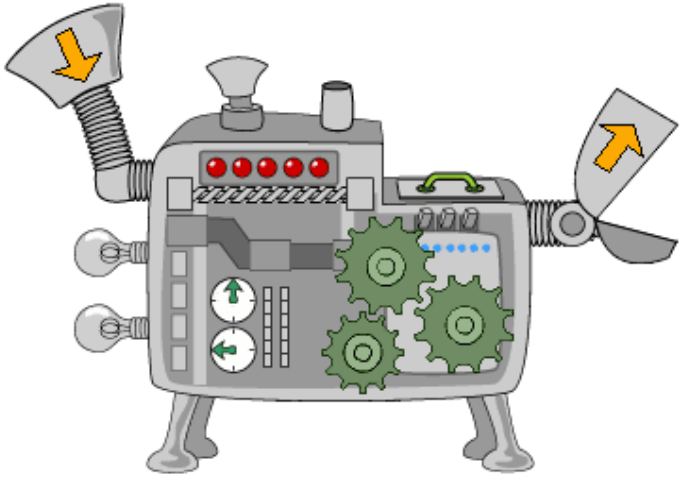
# Example

~ : *one space*

- Format String: `%d/%d`
- Input: `~10/~35`
  - Skip white space, match %d with 10, match / with /, skip white space, match %d with 35
- Input: `~10~/~35`
  - Skip white space, match %d with 10, fail to match ~ with /, abort
- How to allow whitespaces around /?
- Format string: `%d%f` Input: `20.3~5.0`

# Summary

- Printf General Syntax
- Scanf General Syntax
- **Common Mistakes**



http://grade5eisnor.weebly.com/uploads/2/5/1/5/25151059/8233546_orig.gif

# Common Mistakes

- Using & in printf
- Forgetting & in scanf
- Using format string in scanf similar to printf
- Adding \n to scanf