Let’s describe printf function

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf</td>
</tr>
<tr>
<td>1) “Takes a set of data values”</td>
</tr>
<tr>
<td>2) “Converts them to a text stream using formatting instructions contained in the format control string”</td>
</tr>
<tr>
<td>3) “Send the resulting text string to monitor (output).”</td>
</tr>
</tbody>
</table>

Example:

1) 123 – Integer
2) printf(..) – Takes step 1 as input
3) Creates a text stream

4) Text stream is printed on the monitor

Conversion Specification

<table>
<thead>
<tr>
<th>%</th>
<th>Flag</th>
<th>Minimum Width</th>
<th>Precision</th>
<th>Size</th>
<th>Code</th>
</tr>
</thead>
</table>

The options in red are compulsory. So, you have to specify them. The other specifiers are optional.

Example: Print values using only the compulsory specifiers. For other’s, the default value is taken by the system.

<table>
<thead>
<tr>
<th>Type</th>
<th>Specifier</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>%c</td>
<td>char letter_grade = 'A'; printf(&quot;letter_grade is %c&quot;, letter_grade);</td>
</tr>
<tr>
<td>short int</td>
<td>%hd</td>
<td>short int assignment_1_score = 100; printf(&quot;assignment_1_score is %hd&quot;, assignment_1_score);</td>
</tr>
<tr>
<td>int</td>
<td>%d</td>
<td>int assignment_1_score = 100; printf(&quot;assignment_1_score is %d&quot;, assignment_1_score);</td>
</tr>
<tr>
<td>long int</td>
<td>%ld</td>
<td>long int assignment_1_score = 100; printf(&quot;assignment_1_score is %ld&quot;, assignment_1_score);</td>
</tr>
<tr>
<td>long long int</td>
<td>%lld</td>
<td>long long int assignment_1_score = 100; printf(&quot;assignment_1_score is %lld&quot;, assignment_1_score);</td>
</tr>
<tr>
<td>float</td>
<td>%f</td>
<td>float assignment_1_score = 95.5; printf(&quot;assignment_1_score is %f&quot;, assignment_1_score);</td>
</tr>
<tr>
<td>double</td>
<td>%f</td>
<td>double assignment_1_score = 95.5; printf(&quot;assignment_1_score is %f&quot;, assignment_1_score);</td>
</tr>
</tbody>
</table>
The options in red specify the size.
Let’s consider the value of pi. The value of pi is 3.1415926536…

Suppose we want to print the value of pi as:
1) 3.1
2) 3.14
3) 3.142
4) 3.1416

How do we print the value of pi up to a certain number of decimal places?
Answer: We can use the precision identifier.

<table>
<thead>
<tr>
<th>Value of pi</th>
<th>Example</th>
</tr>
</thead>
</table>
| 3.1         | double pi = 3.1415926536;
              | printf("\n Value of pi is %.1f", pi); |
| 3.14        | double pi = 3.1415926536;
              | printf("\n Value of pi is %.2f", pi); |
| 3.142       | double pi = 3.1415926536;
              | printf("\n Value of pi is %.3f", pi); |
| 3.1416      | double pi = 3.1415926536;
              | printf("\n Value of pi is %.4f", pi); |

For printing a floating point number up to m decimal places, the format would .m

**Minimum width**
Gives the minimum number of positions in the output.

**Integer**

1) printf("\n Example: %d", 3);
2) printf(“\n Example: %2d”, 3);
   Exa mple: 3

3) printf(“\n Example: %3d”, 3);
   Exa mple: 3

4) printf(“\n Example: %4d”, 3);
   Exa mple: 3

5) printf(“\n Example: %5d”, 3);
   Exa mple: 3

Character

1) printf(“\n Example: %c”, ‘a’);
   Exa mple: a

2) printf(“\n Example: %2c”, ‘a’);
   Exa mple: a

3) printf(“\n Example: %3c”, ‘a’);
   Exa mple: a

4) printf(“\n Example: %4c”, ‘a’);
   Exa mple: a

5) printf(“\n Example: %5c”, ‘a’);
**Float**

1) `printf("\n Example: %f", 3.1);`

```
Example: 3.100000
```

2) `printf("\n Example: %3.1f", 3.1);`

```
Example: 3.1
```

3) `printf("\n Example: %4.1f", 3.1);`

```
Example: 3.1
```

4) `printf("\n Example: %5.1f", 'a');`

```
Example: 3.1
```

5) `printf("\n Example: %6.1f", 'a');`

```
Example: 3.1
```

**Width and precision**

1) `printf("\n Example: %.2f", 3.1416);`

```
Example: 3.14
```

2) `printf("\n Example: %4.2f", 3.1416);`

```
Example: 3.14
```

3) `printf("\n Example: %5.2f", 3.1416);`
4) `printf("\n Example: %6.2f", 3.1416);`

5) `printf("\n Example: %7.2f", 3.1416);`

**Justification**

<table>
<thead>
<tr>
<th>Flag Code</th>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>right justified</td>
</tr>
<tr>
<td>-</td>
<td>left justified</td>
</tr>
</tbody>
</table>

1) `printf("\n Example: %7.2f *", 3.1416);`

2) `printf("\n Example: %-7.2f *", 3.1416);`

**Padding**

<table>
<thead>
<tr>
<th>Flag Code</th>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>space padding</td>
</tr>
<tr>
<td>0</td>
<td>zero padding</td>
</tr>
</tbody>
</table>
“zero flag is ignored if it is used with left justification because adding zeros after a number changes its value”

1) printf("\n Example: %07d *");

```
Example: 0 0 0 0 0 0 3 *
```

2) printf("\n Example: %0-7d *");

```
Example: 3 *
```

### Sign

<table>
<thead>
<tr>
<th>Flag Code</th>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>positive value: no sign</td>
</tr>
<tr>
<td></td>
<td>negative value: -</td>
</tr>
<tr>
<td>+</td>
<td>positive value: +</td>
</tr>
<tr>
<td></td>
<td>negative value: -</td>
</tr>
</tbody>
</table>

1) printf("\n Example: %+7d *");

```
Example: + 3 *
```

2) printf("\n Example: %7d *");

```
Example: 3 *
```
3) printf("\n Example: %7d *", -3);

4) printf("\n Example: %+7d *", -3);

Examples with \n \r \’ “\"

<table>
<thead>
<tr>
<th>Control char</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\’</td>
<td>Single quote. Has to be escaped with .</td>
</tr>
<tr>
<td>\”</td>
<td>Double quote. Has to be escaped with .</td>
</tr>
<tr>
<td>\</td>
<td>Backslash. Has to be escaped with .</td>
</tr>
<tr>
<td>\0</td>
<td>Null character. End of string. Nothing beyond it gets printed.</td>
</tr>
<tr>
<td>\r</td>
<td>“repositions the output at the beginning of the current line without advancing the line”.</td>
</tr>
<tr>
<td>\n</td>
<td>Introduces a newline.</td>
</tr>
</tbody>
</table>

1) printf("\n Example: \’");
Let’s describe scanf function

**Description**
scanf

1) “Takes a text stream from the keyboard”

2) “Extracts and formats data from the stream according to a format control string”

3) “Stores the data in specified program variables.”

Example:
1) From the keyboard 123 is typed
2) scan(“%d”,&a) – Takes step 1 as input
3) Formats 123 as an integer and stores it in variable a

Conversion Specification (Figure 2.21 from textbook)

<table>
<thead>
<tr>
<th>%</th>
<th>Flag</th>
<th>Maximum Width</th>
<th>Size</th>
<th>Code</th>
</tr>
</thead>
</table>

The options in red are compulsory. So, you have to specify them. The other specifiers are optional.

Examples with integer input

<table>
<thead>
<tr>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
</tr>
</tbody>
</table>

1) INPUT
scanf("%d", &assignment_1_score);
printf("%d", assignment_1_score);

OUTPUT

1 0 0

2) INPUT

1 0 0 9 5 9 0

scanf("%d %d %d", &assignment_1_score, &assignment_2_score, &assignment_3_score);
printf("%d %d %d", assignment_1_score, assignment_2_score, assignment_3_score);

OUTPUT

1 0 0 9 5 9 0

3) INPUT

1 0 0 9 5 9 0

scanf("%d %d %d", &assignment_1_score, &assignment_2_score, &assignment_3_score);
printf("%d %d %d", assignment_1_score, assignment_2_score, assignment_3_score);

OUTPUT
4) INPUT (Taken from textbook)

scanf("%d/%d %d/%d", &num1, &den1, &num2, &den2);
printf("%d/%d %d/%d", num1,den1,num2,den2);

5) INPUT (Taken from textbook)

scanf("%2d-%2d-%2d", &a,&b,&c);
printf("%2d-%2d-%2d", a,b,c);

Examples with real number input

Formatting

%f
1) INPUT

\[
\text{scanf} \("\%f\", \&\text{assignment\_1\_score}\); \\
\text{printf} \("\%.2f\", \text{assignment\_1\_score}\);
\]

OUTPUT

\[
9 \ 5 \ . \ 5 \ 0
\]

2) INPUT

\[
\text{scanf} \("\%f\%f\%f\", \&\text{assignment\_1\_score}, \\
\&\text{assignment\_2\_score}, \&\text{assignment\_3\_score}\); \\
\text{printf} \("\%.2f \ %.2f \ %.2f\", \text{assignment\_1\_score}, \\
\text{assignment\_2\_score}, \text{assignment\_3\_score}\);
\]

OUTPUT

\[
9 \ 5 \ . \ 5 \ 0 \ 9 \ 7 \ . \ 5 \ 1 \ 0 \ 0 \ . \ 0 \ 0
\]

3) INPUT

\[
\text{scanf} \("\%f \%f \%f\", \&\text{assignment\_1\_score}, \\
\&\text{assignment\_2\_score}, \&\text{assignment\_3\_score}\); \\
\text{printf} \("\%.2f \ %.2f \ %.2f\", \text{assignment\_1\_score}, \\
\text{assignment\_2\_score}, \text{assignment\_3\_score}\);
\]

OUTPUT
Examples with character input

<table>
<thead>
<tr>
<th>Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
</tr>
</tbody>
</table>

Please put this line in your code

```c
#define FLUSH while ( getchar( ) != ‘\n’);
```

1) INPUT

```
printf("Please enter the letter grade");
FLUSH;
scanf(" %c", &letter_grade);
printf("%c", letter_grade);
```

OUTPUT

```
A
```

2) INPUT

```
printf("Please enter the letter grade");
FLUSH;
scanf(" %c", &letter_grade);
printf("%c", letter_grade);
```
A

3) INPUT

A  B  A

printf("Please enter the letter grades of first three students");
FLUSH;
scanf(" %c %c %c", &student1_grade, &student2_grade, &student3_grade);
printf("\n student1_grade: %c", student1_grade);
printf("\n student2_grade: %c", student2_grade);
printf("\n student3_grade: %c", student3_grade);

OUTPUT

 student1_grade: A
 student2_grade: B
 student3_grade: A

Summarizing rules for scanf (Table 2-12)
"1. The conversion operation processes until:
   a. End of file is reached.
   b. The maximum number of characters has been processed.
   c. A whitespace character is found after a digit in a numeric specification.
   d. An error is detected.
2. There must be a conversion specification for each variable to be read.
3. There must be a variable address of the proper type for each conversion specification.
4. Any character in the format string other than whitespace or conversion specification must be exactly matched by the user during input. If the input stream does not match the character specified, an error is signaled and scanf stops.
5. It is a fatal error to end the format string with a whitespace character. Your program will not run correctly if you do.”