

Introduction to C, C++, and Unix/ Linux

CS 60

Lecture 4: Printf/Scanf

Today

→ printf/scanf

- Reading for next class: K&R ch. 1-3 & 7.1-7.4

Note: Lexical elements of C

- **Keywords**
 - Reserved words that may not be used for anything else
 - **Identifiers**
 - Variable names, function names...
 - **Constants**
 - E.g., the number 5
 - **String constants**
 - E.g. “Hello, world\n”
 - **Operators**
 - E.g., +, -, =, ++
 - **Punctuators**
 - E.g., { } () ; ,
- These are the basic tokens that the compiler cares about

Formatted console output: `printf()`

```
int printf(char *format, arg1, arg2, ...)
```

- `printf` converts, formats, and prints its arguments on the standard output
 - It returns the number of characters printed (including carriage returns, etc.)
- The `format` string controls the formatting
 - Text string including conversion specifications
 - `%d, %c, %f, %s, %3.2f, %-010d, ...`

printf conversion specification: %_c

- Each conversion specification starts with % and ends with a conversion character
- In between, there may be
 - A minus sign (left adjustment)
 - A number (minimum field width)
 - A period (separates field width from precision)
 - A number (the precision)
 - h (short) or l (long)

TABLE 7-1. BASIC PRINTF CONVERSIONS

CHARACTER	ARGUMENT TYPE; PRINTED AS
d, i	int; decimal number.
o	int; unsigned octal number (without a leading zero).
x, X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.
u	int; unsigned decimal number.
c	int; single character.
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]m.ddddd, where the number of d's is given by the precision (default 6).
e, E	double; [-]m.ddddd e±xx or [-]m.ddddd E±xx, where the number of d's is given by the precision (default 6).
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %.

Minimum field specs for printf

- Leave 5 spaces to print the value:

```
int num;
```

```
Ans: 42 days
```

```
printf ("Ans:%5d days", num);
```

- Pad the left blanks with zeros:

```
Ans:00042 days
```

```
printf ("Ans:%05d days", num);
```

- Left justify the value:

```
Ans:42 days
```

```
printf ("Ans:%-5d days", num);
```

Printing floats

```
double pi = 3.1415927;
```

```
printf ("PI is %f", pi);
```

```
PI is 3.141593
```

```
printf ("PI is %.2f", pi);
```

```
PI is 3.14
```

```
printf ("PI is %6.1f", pi);
```

```
PI is      3.1
```

Common printf mistake

```
double pi = 3.1415927;  
printf ("PI is %d", pi);
```

prints out:

```
PI is 1518260631
```

%f



Other formatting characters

- `\n` – newline
- `\t` – tab
- `\b` – backspace
- `\r` – carriage return
- `\a` – audible alert
- `\\` – backslash (`\`)
- `\?` – question mark (`?`)
- `\`` – single quote (```)
- `\"` – double quote (`"`)

```
printf ("%d\t%d\n", fahr, celc);
```

Several times ...

10 -12

15 -9

20 -6

25 -3

30 -1

35 1

40 4

```
printf ("%5d%5d\n", fahr, celc);
```

Several times ...

10 -12

15 -9

20 -6

25 -3

30 -1

35 1

40 4

Other printf() magic

- Precision specification

```
printf ( "%.4f\n", 123.45678 );  
/* prints out 123.4568 */
```

- Range of strings

```
printf ( "%15.10s\n", str );  
/* at most 10 chars from str in field  
of 15 spaces */
```

- Variable field width

```
printf ( "%*d\n", i, num );  
/* field width of i characters */
```

```
% ramp
1
02
003
0004
00005
000006
0000007
00000008
000000009
000000009
00000008
0000007
000006
00005
0004
003
02
1
```

Text formatting is easy and powerful in C!

```
#include <stdio.h>
int main (void)
{
    int i;
    for (i=1; i < 10; i++)
        printf ("%0*d\n", i, i);
    while (i-- > 1)
        printf ("%0*d\n", i, i);
}
```

Scanf()

- **Scanf** works much like **printf**, but for user input
– typing to the console

```
int x, y;  
printf("Input (x, y) values:");  
scanf("%d%d", &x, &y);  
printf("Okay, x is %d and y is %d\n",  
      x, y);
```

Pointers to variables

Most common `scanf` mistake

```
int x, y;  
printf("Input (x, y) values:");  
scanf("%d%d", x, y);
```

must be

`&x, &y`