CSCI 2132
Software Development

Lecture 9:

Formatted Input and Output in C

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Extended regular expressions (ERE)
- Grep variations
- Introduction to C
  - writing a simple program
  - compilation process
  - C basics, similarities and differences with Java
  - printing output and reading input
  - a simple HST program

Formatted Input and Output

- Necessary to use `#include <stdio.h>`
- Main functions: `printf` (output), and `scanf` (input)
- `printf` syntax:

```
printf(format_string, exp1, exp2, ...);
```

- `format_string` is a string that may contain *conversion specifications* (starting with '`%`')
- Example: `printf("x = %d\n", x);`

How `printf` works

- `printf` reads format the format string and prints it to the output:
  `printf(format_string, exp1, exp2, ...);`
- Each time it comes across a conversion specification (starting with `%`, it interprets the next parameter as the specified type, and prints it to the output
- Warning: If we make an error in type specification, it will be undetected!

`printf` Example

```
int i = 7;
double x = 2.71;
char c = 'A';
printf("Printing vars: i = %d, "
       "x = %.2f, c = %c\n",
       i, x, c);
```

would produce:

```
Printing vars: i = 7, x = 2.71, c = A
```

Structure of printf Conversion Specification

- Starts with `%` and ends with *conversion specifier*
- Conversion Specifiers:
  - `d` for an integer
  - `f` for a float or double
  - `c` for a char
  - `s` for a string
  - `%` for a literal % (no conversion)

Conversion Specification Details in printf

- Full format: `%`*`(flags)(minw)`**`.`**`(prec)(lenm)spec`
- Optional items between `%` and conversion specifier:
  - *flags*: e.g., `+` (mandatory sign), `-` (left justification)
  - *minimal width*: e.g., `%10d`
  - **`.`***precision*: e.g., `%.2f`, or `%.10s` (precision for floating-point numbers, minimal number of digits for integers, and maximal length for strings)
  - *length modifier*: e.g., `%lf`
- Read `man 3 printf` for more information

Formatted Input: `scanf`

- Works similarly to `printf`; syntax:
  `scanf(format_string, addr1, addr2, ...);`
- `scanf` reads the format string and the standard input, matching them
- On each conversion specification, input is interpreted and stored into the given memory address
- If matching fails, `scanf` returns without finishing reading the format string, and it stops reading input
- Return value: number of converted values (a special value `EOF` is returned in some cases)
- Read `man 3 scanf` for more info

A `scanf` Example

```
int i, j;
double x, y;
scanf("%d%d%lf%lf", &i, &j, &x, &y);
```

Consider input:

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

- Why do we need `&` with variable names?
- In C: function arguments are passed by value.

`scanf` Conversion Specifications

- White-space characters are matched with white-space characters or nothing
- Characters other than `%` are matched with themselves in input
- `scanf` conversion specifications start with % and end with a conversion specifier, which can be:
  - `%` for no conversion, just matching `%`
  - `d` matches decimal integers and converts it to `int`
  - `f` matches floating-point number, and stores it as float (or double)

`scanf` Conversion Specifications (cont.)

- `s` matches non-white-space characters and stores them into a string
- `c` matches character (or characters if length is given) and stores them without terminating `\0`; does not skip white-space
- `[` matches a non-empty sequence of characters and stores them into a string. Similar to regular expressions: e.g., `%[abc]`, `%[^abc]`, `%[a-zA-Z]`
- `n` no matching, but stores number of characters consumed so far into an `int` argument
- Numeric conversions skip any leading white-space

`scanf` Conversion Modifiers

- The main ones:
  - `*` — discard value (no storing)
  - `l` — used with `long int` (`%ld`), and `double` (`%lf`)
- You can read more using `man 3 scanf`

scanf Example

- Consider the following code:

```
int i, j;
double x, y;
scanf("%d%d%lf%lf", &i, &j, &x, &y);
```

- and the following input:

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

- or consider input: `1-20.3-4.0e3`