Laboratory 4: Standard Input/Output: *printf(), scanf(), arrays, and for loops*

Lecture notes: (Note: start comment with // is C++ syntax that gcc compiles o.k. without -ansi option)

1. printf()
   a. Syntax: *printf("format string", arg_0, arg_1, …, arg_n);*
   b. Format sting:
      i. text will be printed out as is. E.g. Hello, world!
      ii. % is the format indicator, which tells the computer that the value (can be char or string, not necessarily numbers only) of a variable will be printed out. The parameter(s) after % indicates the variable type, precision, output position and format, etc.
      iii. The number of %'s in " " should match the number of arguments (variable names) at the end.
      iv. The type of the argument should also match the corresponding type indicated by the parameter after %.
   c. Variable type:

      int: %d, %i

      char: %c

      float, double: %f, %e, %E, %g, %G

      string: %s
   d. Field width:
      i. default:

         int: just enough space to print out the value.

         printf("%d", 123); 123

         float/double: 6 digits after the decimal point; just enough space for the integer part.

         printf("%f", -1.2); -1.200000

         for scientific notation:

         printf("%E", 1234.5); 1.234500E+03

         character/string: depends on the variable value. \t, \n, …

         printf("%c%c", 'A', 66); AB

      ii. %kx: use k spaces to print out value of type x. right aligned, unused spaces are left blank.

         printf("%10f %4d", 1.23, 45); _ _1.230000_ _45

         printf("%14e", -2.0); _ -2.000000E+00

iii. %0kx: use k spaces to print out value of type x and fill the extra spaces by 0.

printf("%04d", 45); 0045

iv. %*x: variable field width, the number of spaces is determined by the value that * corresponds to; namely, the next integer argument in the list.

printf("%*d", 4, 45);

prinf("%04d",45);

v. Field width vs. correctness

printf("%4d", 12345);

e. Precision of floating point numbers:

i. Default: 6 digits after the decimal point

ii. %.pf: p digits after the decimal point

printf("%.3f", 1.23); 1.230

iii. %k.pf: use k spaces including p digits after decimal point for the value of the real number.

printf("%5.2f", 3.1416); _ 3.14

iv. %*.*f: variable field width and variable precision

printf("*.*f", 5, 2, 3.1416);

f. Wildcard character * and . for int and char:

i. printf("%5.2d", 1); _ _ _ 01

ii. printf("%10.5s", "Hello, world."); _ _ _ _ _ Hello

iii. * is the same as before

g. Format (alignment and sign bit):

i. %-x: left aligned. If not specified, it is right aligned by default.

printf("%12f", 3.1416);

printf("%-12f", 3.1416);

ii. %+x: enforce the (+ or -) sign bit. By default, + is not printed for positive numbers.

% x: (there is one empty space between % and x) omit + for positive numbers, but leave one empty space for sign.

printf("%f\n%+f\n% f\n", 3.1416, 3.1416, 3.1416);

h. All the above features can be combined. However, they need to follow the following order: format, width, . (period), precision, type. Note that one or both of width and precision can be replaced by the wildcard character *.

%+10.*f // pay attention to the order of these features

2. scanf()
    a. syntax: *scanf("format string", &arg$_0$, &arg$_1$, …, &arg$_n$);*

       read in the value of the expected type (as indicated in the format string) and assign it to the specified variable.

       scanf("%d %f", &a, &b);

    b. Variable type

       Same as in printf() except that & needs to be put in front of the variable.

       However,

       scanf("%s", string_name); // note, no & before string name.

       Question: what if the input is longer than the size of the declared array?

    c. Multiple data in one scanf() statement
        i. Use empty space or tab to separate the multiple %'s in the format string.
        ii. If characters other than %, space, and tab are included in the format string, these characters are expected to be read in from input in their position.
        iii. In input, empty space or tab or new line can be used to separate multiple input numerical data (either integer or real number) and string.
        iv. For %c, white spaces will be treated as input data.

       Question: what if the variable number of type do not match?

    d. Matching characters in *scanf("%s", string_name);*
        i. %[abc]: read in only letters a, b, and c, in any order and any times. Stop at the first character that is not a, b, or c.
        ii. %[a-zA-Z]: read in only English alphabet.
        iii. %[0-9]: read in only numbers (as a text string, not number)
        iv. %[a-zA-Z]: read in English letters and space. Note that there is an extra space at the end.

    e. Excluding characters in *scanf("%s", string_name);*
        i. %[^a-z]: skip all the lower case letters
        ii. %*: skip a string

            *scanf("%*s %d %*s %c", &a, &c);*

3. carefully examine codes: array.c, string.c, printf.c, printf2.c, printf3.c, scanf.c, scanf_printf.c
4. Reading : textbook section 6.1; also, suggested is Gottfried (Schaum's) Chapt. 4.

Name: _____ Section: 010__ Date: _____

Lab Report

1. Write a simple C code to test on GLUE UNIX, what is the rule to print out a real number when the number's precision is higher than the output requirement. For example, what will be printed out on statements *printf("%5.2f", 3.1416);* and *printf("%5.3f", 3.1416);* ? Write down the rule.

2. Write down all the different ways that you can find to print out special characters such as % and \

3. Use loops to print out the followings:

   1) 28 *'s in four lines, where there is no leading space on the first line

```
*******
 *******
  *******
   *******
```

   2) 30 %'s in five lines, where there is no leading space on the first line

```
%%%%%%%%%%
 %%%%%%%%
  %%%%%%
   %%%%
    %%
```

4. What is implicit cast and what is explicit cast? For the following expressions, identify when they happen.

```
int a = 2, b = 3;
float f = 2.5;
double d = -1.2;
int int_result;
float real_result;

int_result = a * f;
real_result = a * f;
real_result = (float) a * b;
d = a + b / a * f;
d = f * b / a + a;
```