Standard I/O Lesson Outline

1. Standard I/O Lesson Outline
2. Output via printf
3. Placeholders
4. Placeholders for Various Data Types
5. Mixing Literal Text and Variables’ Values #1
6. Mixing Literal Text and Variables’ Values #2
7. Placeholder & Variable in Same Statement
8. Placeholder/Variable Same Statement: Example
9. Input via scanf
10. Input via scanf: Ampersand Before Variable
11. Input via scanf Example
12. Input via scanf Example’s Flowchart
13. Reading Multiple Variables with a Single scanf
14. Multiple Variables per scanf Example #1
15. Multiple Variables per scanf Example #2
16. printf vs scanf
17. Programming Exercise
Output via `printf`

In C, we output to standard output using a `printf` statement:

```c
printf("This will be output to stdout.\n");
```

A `printf` statement can output a string literal, but it can also output the value of a variable, a literal constant or a named constant:

```c
printf("%d", number_of_students);
```

The statement above outputs to `stdout` (the terminal screen) the value of a variable named `number_of_students` of type `int` (presumably declared previously in the program that contains this `printf` statement).

The string literal in a `printf` statement is known as a `format string`. 
Placeholders

\texttt{printf("%d", \text{number\_of\_students});}

The statement above:
- outputs to standard output (stdout)
- the value of the variable named \texttt{number\_of\_students}
- which is of type \texttt{int}
- (declared previously in the program that contains this \texttt{printf} statement).

The \texttt{%d} is known as a \textit{placeholder}: it holds the place of the value of the variable that we actually want to output.

Another name for a placeholder is a \textit{format specifier}, but we’ll typically say placeholder in CS1313.
Placeholders for Various Data Types

- **int:** \( %d \)
  - printf("%d", number_of_students);
- **float:** \( %f \)
  - printf("%f", pi);
- **char:** \( %c \)
  - printf("%c", middle_initial);
Mixing Literal Text and Variables’ Values #1

We now know that we can output a string literal:

```c
printf("This will be output to stdout.\n");
```

We also know that we can output the value of a variable:

```c
printf("%d", number_of_students);
```

Not surprisingly, we can **mix and match** the two:

```c
printf(" on your %d income.\n", tax_year);
```

We can even mix and match while outputting the values of multiple variables of various data types:

```c
printf("The %d federal income tax on $%f\n", tax_year, income);
```
Mixing Literal Text and Variables’ Values #2

In a `printf` statement’s **format specifier**, we can mix and match literal text and variables’ values while outputting the values of multiple variables of various data types:

```c
printf("The %d federal income tax on $%f\n",
    tax_year, income);
```

This statement means:
- Output to `stdout` (the terminal screen)
- the literal text "The ", and then
- the value of the `int` variable named `tax_year`, and then
- the literal text " federal income tax on ", and then
- the value of the `float` variable named `income`, and then
- a newline.
Placeholder & Variable in Same Statement

When you use a placeholder inside the string literal of a printf statement, the variable whose place is being held by the placeholder MUST MUST MUST MUST be in the same printf statement as the placeholder.

Putting the placeholder in one printf statement and the variable in a different printf statement is BAD BAD BAD!

/* These printfs are GOOD GOOD GOOD! */
printf("f1=%f, ", f1);
printf("i1=%d, GOOD!\n", i1);

/* These printfs are BAD  BAD BAD BAD! */
printf("BAD! f2=%f, i2=%d, ");
printf("BAD!\n", f2, i2);

NOTE: The same rule applies to scanf statements (coming up).
Placeholder/Variable Same Statement: Example

% cat placeholder.c
#include <stdio.h>

int main ()
{ /* main */
   float f1, f2;
   int   i1, i2;

   f1 = 3.75;
   f2 = 5.25;
   i1 = 6;
   i2 = 8;
   /* These printfs are GOOD GOOD GOOD! */
   printf("f1=%f, ", f1);
   printf("i1=%d, GOOD!\n", i1);
   /* These printfs are BAD  BAD BAD!  */
   printf("BAD! f2=%f, i2=%d, ");
   printf("BAD!\n", f2, i2);
   /* This printf is GOOD GOOD GOOD! */
   printf("f2=%f, i2=%d, GOOD!\n", f2, i2);
} /* main */
%
% gcc -o placeholder placeholder.c
% placeholder
f1=3.750000, i1=6, GOOD!
BAD! f2=3.750000, i2=134513662, BAD!
f2=5.250000, i2=8, GOOD!
Input via scanf

The `printf` statement **outputs** to `stdout` (the terminal screen).
Likewise, the `scanf` statement **inputs** from `stdin` (a user typing at the keyboard).
The `scanf` statement has a somewhat strange syntax:

```c
scanf("%d", &height_in_cm);
```

This statement says:
- input from `stdin` (a user typing at the keyboard)
- an `int` value
- and place that `int` value into the memory location associated with the `int` variable named `height_in_cm`. 
Input via `scanf`: Ampersand Before Variable

The `scanf` statement has a somewhat strange syntax:

```c
scanf("%d", &height_in_cm);
```

Notice the **ampersand** `&` before the name of the variable that you’re inputting into.

For now, you must simply **ACCEPT THIS ON FAITH**.

Time permitting, toward the end of the semester we’ll learn about what the ampersand means.
Input via scanf Example

% cat read_variable.c
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;

    printf("What's my height in centimeters?\n");
    scanf("%d", &height_in_cm);
    printf("My height is %d cm.\n", height_in_cm);
} /* main */

% gcc -o read_variable read_variable.c
% read_variable
What's my height in centimeters?
160
My height is 160 cm.
Input via `scanf` Example’s Flowchart

```c
printf("What's my height in centimeters?\n");
scanf("%d", &height_in_cm);
printf("My height is %d cm.\n", height_in_cm);
```

Start

Prompt for height in cm.

Input height in cm.

Output height in cm.

End
Reading Multiple Variables with a Single `scanf`

C allows inputting multiple variables per `scanf` statement.

At runtime, when the user types in the input values, they can separate the individual input values

- by blank spaces, and/or
- by tabs, and/or
- by carriage returns (newlines).

Blank spaces, tabs and carriage returns, as a group, are known as `white space`. 
# Multiple Variables per `scanf` Example #1

```c
#include <stdio.h>

int main ()
{ /* main */
    float CS1313_average_height_in_m;
    int number_of_silly_people, number_of_nonsilly_people;
    char Henrys_middle_initial;

    printf("In CS1313, how many silly people are there, \n");
    printf(" and how many non-silly people are there? \n");
    scanf("%d %d",
         &number_of_silly_people,
         &number_of_nonsilly_people);
    printf("What is the average height in m in CS1313, \n");
    printf(" and what is Henry's middle initial? \n");
    scanf("%f %c",
         &CS1313_average_height_in_m, &Henrys_middle_initial);
    printf("In CS1313, there are %d silly people\n",
           number_of_silly_people);
    printf(" and %d non-silly people. \n",
           number_of_nonsilly_people);
    printf("In CS1313, the average height is %f m. \n",
           CS1313_average_height_in_m);
    printf("Henry's middle initial is %c. \n",
           Henrys_middle_initial);
} /* main */
```
In CS1313, how many silly people are there, and how many non-silly people are there?

20 120

What is the average height in m in CS1313, and what is Henry's middle initial?

1.75

J

In CS1313, there are 20 silly people and 120 non-silly people. In CS1313, the average height is 1.750000 m. Henry's middle initial is J.
printf vs scanf

- **printf**
  - outputs
  - to stdout
  - the string literal **CAN** (and typically does) contain literal text as well as placeholders
  - the string literal typically **DOES** end with a newline (but that’s **NOT** required)
  - variable names after the string literal **CANNOT** be preceded by &

- **scanf**
  - inputs
  - from stdin
  - the string literal **CANNOT** contain literal text – **EXCEPT**, if there are multiple placeholders, then between each adjacent pair of placeholders there **MUST** be a **SINGLE BLANK SPACE (REQUIRED)**
  - the string literal **CANNOT** contain a newline
  - variable names after the string literal **MUST** be preceded by &
Programming Exercise

Create a program that:

1. Greets the user.
2. Prompts the user for their age in years.
3. Inputs the user’s age in years.
4. Outputs the user’s age in years.

Begin by drawing a flowchart, and then write the program.

The program does not have to have comments.
The data type for the age variable must be appropriate.