Lesson 7

Reading and Writing
a.k.a.
Input and Output
## Escape sequences for `printf` strings

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Hex value in ASCII</th>
<th>Character represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>07</td>
<td>Alarm (Beep, Bell)</td>
</tr>
<tr>
<td>\b</td>
<td>08</td>
<td>Backspace</td>
</tr>
<tr>
<td>\f</td>
<td>0C</td>
<td>Formfeed</td>
</tr>
<tr>
<td>\n</td>
<td>0A</td>
<td>Newline (Line Feed); see notes below</td>
</tr>
<tr>
<td>\r</td>
<td>0D</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>\t</td>
<td>09</td>
<td>Horizontal Tab</td>
</tr>
<tr>
<td>\v</td>
<td>0B</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>\ \</td>
<td>5C</td>
<td>Backslash</td>
</tr>
<tr>
<td>\ '</td>
<td>27</td>
<td>Single quotation mark</td>
</tr>
<tr>
<td>\ &quot;</td>
<td>22</td>
<td>Double quotation mark</td>
</tr>
<tr>
<td>\ ?</td>
<td>3F</td>
<td>Question mark</td>
</tr>
<tr>
<td>\ nnn</td>
<td>any</td>
<td>The character whose numerical value is given by <code>nnn</code> interpreted as an octal number</td>
</tr>
<tr>
<td>\xhhh</td>
<td>any</td>
<td>The character whose numerical value is given by <code>hh</code> interpreted as a hexadecimal number</td>
</tr>
</tbody>
</table>

Escape sequences for `printf` strings

Why do we need escape sequences?
Can we not use the ASCII characters directly?
E.g., instead of `\n`, we could use ASCII `10` (Newline).

```c
#include <stdio.h>

void main(){
    printf("Hello\%c World!\n\n", 10);
}
```

```
Hello
World!
```
Conversion specifiers for `printf` a.k.a. placeholders

**TABLE 7.2** The Most Commonly Needed Conversion Specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Meaning</th>
<th>Types Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%c</code></td>
<td>Single character</td>
<td>char</td>
</tr>
<tr>
<td><code>%d</code></td>
<td>Signed decimal integer</td>
<td>int, short</td>
</tr>
<tr>
<td><code>%ld</code></td>
<td>Signed long decimal integer</td>
<td>long</td>
</tr>
<tr>
<td><code>%f</code></td>
<td>Decimal floating-point number</td>
<td>float, double</td>
</tr>
<tr>
<td><code>%s</code></td>
<td>Character string</td>
<td>char arrays</td>
</tr>
<tr>
<td><code>%u</code></td>
<td>Unsigned decimal integer</td>
<td>unsigned int, unsigned short</td>
</tr>
<tr>
<td><code>%lu</code></td>
<td>Unsigned long decimal integer</td>
<td>unsigned_long</td>
</tr>
<tr>
<td><code>%%lld</code></td>
<td>Signed long long integer</td>
<td>long long</td>
</tr>
<tr>
<td><code>%%llu</code></td>
<td><strong>Unsigned</strong> long long integer</td>
<td>unsigned long long</td>
</tr>
<tr>
<td><code>%%Lf</code></td>
<td>Note the uppercase L!</td>
<td>long double</td>
</tr>
</tbody>
</table>
Any float argument passed to \texttt{printf} is automatically converted to double \texttt{before printf} receives it!

This means that, technically, we’re never printing doubles with \texttt{printf} – only floats. This does \textbf{not} mean that there’s no difference between printing doubles and printing floats!

- Doubles have more precision than floats, as seen in this example:

```c
void main(void)
{
    float a = 1.0f/3.0f;
    double b = 1.0/3.0;

    printf("%.2f\n%.2lf\n", a, b);
}
```

```
0.3333333334326744080000
0.33333333333333331000
```
For this reason, before C99 there was only one specifier for both floats and doubles: \%f.

In C99 (and C11):
• \%lf was defined to represent doubles
• \%Lf was defined to represent long doubles

In this class, it is **required** to use \%f for float, \%lf for double, and \%Lf for long double. This is also my recommendation for your future programming career, since type-related errors are very common (and hard to spot).
Example 2 Input

printf("This prints a character, %c\na number, %d\na floating point, %f", 'z', 123, 456.789 );

Example 2 Output

This prints a character, z
a number, 123
a floating point, 456.789
Example not in text: printing a string

```c
#include <stdio.h>

void main(){
    printf("This is the string: \%s \n\n", "Random string!");
}
```

```
C:\Windows\system32\cmd.exe
This is the string: Random string!
```
Listing 7.2  nums.c: Using printf() to Display Numerical Values

1: /* Demonstration using printf() to display numerical values. */
2: 
3: #include <stdio.h>
4: 
5: int a = 2, b = 10, c = 50;
6: float f = 1.05, g = 25.5, h = -0.1;
7: 
8: int main( void )
9: {
10:     printf("\nDecimal values without tabs: %d %d %d", a, b, c);
11:     printf("\nDecimal values with tabs: \t%d \t%d \t%d", a, b, c);
12: 
13:     printf("\nThree floats on 1 line: \t%f\t%f\t%f", f, g, h);
14:     printf("\nThree floats on 3 lines: \n\t%f\n\t%f\n\t%f", f, g, h);
15: 
16:     printf("\nThe rate is %.2f", f);
17:     printf("\nThe rate to 2 decimal places is %.2f", f);
18:     printf("\nThe rate to 1 decimal place is %.1f", f);
19:     printf("\nThe result of 1/f = %f\n", g, f, g / f);
20: 
21:     return 0;
22: }
Listing 7.1

16:     while (choice != QUIT)
17:         {
18:             choice = get_menu_choice();
19:
20:             if (choice == 1)
21:                 printf("\nBeeping the computer\a\a\a" );
22:             else
23:                 {
24:                 if (choice == 2)
25:                     print_report();
26:                 }
27:             }
28:         printf("You chose to quit!\n");
What is the most appropriate loop to use in this case?

```c
#define QUIT 3

int choice = 0;

while (choice != QUIT) {
    choice = get_menu_choice();

    if (choice == 1)
        printf("\nBeeping the computer\a\a\a"");
    else
        {
            if (choice == 2)
                print_report();
        }

    printf("You chose to quit!\n");
```
Another output function: `puts()`

It’s specialized for strings only.

We can use escape sequences, as in `printf()`.

Includes a newline (`\n`) at the end automatically.
Strictly FYI: Digraphs and Trigraphs

They were introduced to allow programming in C on systems with character sets not including all the punctuation required in C, like ...

Which of the trigraph symbols are missing here?

Source: http://de.wikipedia.org/wiki/DIN_66003
Strictly FYI: Digraphs and Trigraphs

Although in most of today’s compilers digraphs and trigraphs are not implemented or turned off by default, refrain from using double question marks (both inside and outside strings!)

**Example 1**

```c
int x, y, z;
scanf("%d %d %d", &x, &y, &z);
```

**Example 2**

```c
#include <stdio.h>
int main( void )
{
    float y;
    int x;
    puts( "Enter a float, then an int" );
    scanf("%f %d", &y, &x);
    printf( "\nYou entered %f and %d ", y, x );
    return 0;
}```
while (choice != QUIT)
{
    choice = get_menu_choice();

    if (choice == 1)
    {
        puts("\nEnter a signed decimal integer (i.e. -123")");
        scanf("%d", &int_var);
    }
    if (choice == 2)
    {
        puts("\nEnter a decimal floating-point number\n        (e.g. 1.23")");
        scanf("%f", &float_var);
    }
    if (choice == 3)
    {
        puts("\nEnter an unsigned decimal integer \n        (e.g. 123")");
        scanf("%u", &unsigned_var);
    }
}
7. **BUG BUSTER:** Find the error(s) in the following code fragment:

```c
printf( "Jack said, "Peter Piper picked a peck of pickled peppers." ");
```

8. **BUG BUSTER:** Find the error(s) in the following program:

```c
int get_1_or_2( void )
{
    int answer = 0;
    while (answer < 1 || answer > 2)
    {
        printf(Enter 1 for Yes, 2 for No);
        scanf("%f", answer);
    }
    return answer;
}
```
To do for next time:
• Read Q&A
• Solve Quizzes 1 – 5 in notebook
• Exercises 1, 2, 3
Lesson 8

Arrays
An array is a group of consecutive memory locations.

All the elements of an array have:

- the same name and data type
- An individual index, starting with 0
Is this an array?
Is this an array?

No, b/c the elements are not all of the same data type!
How Arrays are Stored in Memory

Example:

```c
int a[6];
```

This 1D array `a` has six elements.
The data type of each array element is int.
Array `a` occupies 24 consecutive bytes in memory.
The elements of an array can be referred as

```
name[position_number]
```

First element at position 0. In our example, the elements of array `a` can be referred to as

```
a[0], a[1], a[2], a[3], a[4], a[5].
```
type name[expr];

• **type** is a data type, e.g. int, char, float
• **name** is a valid identifier (cannot start w/digit!)
• **expr** is an expression that evaluates to an integer type
  • Cannot involve variables!
  • Can involve calculations of constants, e.g. 4x10
  • The value of **expr** is the number of elements in the array.

For example,

```
int a[6];
```

declares a one-dimensional array named `a`. The data type of each array element is `int`. Array `a` has 6 elements from `a[0]` to `a[5].`
How arrays are declared and used in a program

```c
float expenses[100];
int a[10];
/* additional statements go here */
extenses[i] = 100;       // i is an integer variable
expenses[2 + 3] = 100;   // equivalent to expenses[5]
extenses[a[2]] = 100;    // a[] is an integer array
```
QUIZ  Declare the following:

• Array of 100 unsigned integers
• Array of 50 long long integers
• Array of 1000 doubles

• In each case, add up the first and last elements!
How arrays are declared and used in a program

Explain what’s happening to the array ar:

```c
int i, ar[10];
.
.
.
.
.
.
.
ar[0] = 10;
ar[5] = 2*ar[0];
i = ar[0] + ar[5];
ar[10] = 42;
```
Arrays and **for** loops are made for each other!

(Listing 8.1)
Two-dimensional (2D) arrays

type name[expr1][expr2];

• `expr1` and `expr2` evaluate to an integral type
• The value of `expr1` is the # of rows and the value of `expr2` is the # of columns of the array

Example:

```c
int a[2][3];
```
declares a two-dimensional array. Array `a` has 2 rows and 3 columns, for a total of 6 elements.
Example:

```c
int b[2][3];
```

Array `b` has six elements (2x3) with 2 horizontal rows and 3 vertical columns. From a programming point of view, it is a block of memory with each row of the array lying in a contiguous block of memory as shown below. The elements of a two-dimensional array can be referred to like this:

```
arrayname[row_index][column_index]
```
How 2D arrays are stored in memory

```c
int b[2][3];
```

Abstract matrix view

Storage “by rows”, or in “row-major order”
void printMat(float mat[3][3], short rows, short cols){
    short i, j;
    for (i=0; i<rows; i++){
        for (j=0; j<cols; j++)
            printf("%4.1f ", mat[i][j]);
        printf("\n");
    }
}

void addMat(float A[3][3], float B[3][3], float C[3][3],
            short rows, short cols){
    short i, j;
    for (i=0; i<rows; i++)
        for (j=0; j<cols; j++)
            C[i][j] = A[i][j] + B[i][j];
}
Should end lecture here, with a quiz on initializing 1D and 2D arrays!!!
Initializing 1D-arrays (of numbers)

1. int a[5] = {1, 2, 3, 4, 5};
   • If not enough initializers, rightmost elements become 0
   • If too many, syntax error

2. int a[5] = {0};
   • All elements 0

3. int a[] = {1, 2, 3, 4, 5};
   • The size is omitted, initializers determine it
   • 5 initializers, therefore, five-element array
   • The size of array a is 20 bytes.

4. char str1[] = "Hello";
   • null character '\0' terminates strings
   • str1 actually has 6 elements

5. char str1[] = { 'H', 'e', 'l', 'l', 'o', '\0' };

Arrays of characters will be covered in Lesson 10
What does this program do?

```c
#include <stdio.h>

void main()
{
    int a[10] = {2,3,5,7,11,13,17,19,23,29};
    int i;

    for (i=0;i<10;i++)
        printf("a[%2d] = %2d\n", i, a[i]);
}
```
What does this program do?

```c
#include <stdio.h>

void main(){
    int a[10] = {2,3,5,7,11,13,17,19};
    int i;

    for (i=0;i<10;i++)
        printf("a[\%2d] = \%2d\n", i, a[i]);
}
```
What does this program do?

```
#include <stdio.h>

void main()
{
    int i;

    for (i=0; i<10; i++)
        printf("a[%2d] = %2d\n", i, a[i]);
}
```
QUIZ: Write C code that creates this array:

1 3 5 7 9 11
QUIZ: Write C code that creates this array:

1 3 5 7 9 11

int a[10] = {1, 3, 5, 7, 9, 11};
QUIZ: Write C code that creates this array:

1 3 5 7 .... 101

Hint: Use a `for` loop.
QUIZ: Write C code that creates this array:

1 3 5 7 .... 101

int a[10];
int i;
for (i=0; i<51; i++)
a[i] = 2*i + 1;

Make sure the last value is correct!
Initializing 2D-arrays

1. `int a[2][2] = {{ 1, 2 },
                  { 3, 4 } };`
   - Initializers grouped by row in braces
   - `a[0][0] = 1, a[0][1] = 2, a[1][0] = 3, a[1][1] = 4`

2. `int b[2][2] = {{ 1 },
                  { 3, 4 } };`
   - If not enough, unspecified elements set to zero
   - `b[0][0] = 1, b[0][1] = 0, b[1][0] = 3, b[1][1] = 4`

3. `int c[2][2] = {1, 2, 3, 4};`
   - `c[0][0] = 1, c[0][1] = 2, c[1][0] = 3, c[1][1] = 4`
Your turn! Draw the 2-D arrays (a.k.a. matrices) created by these declarations:

```c
double mat1[3][3] = {
    {2.0, 3.5, 4.1},
    {5.33},
    {-10.0, -100.5}
};

double mat2[4][2] = {
    {1.0, 2.5},
    {3.2, 4.2},
    {-1.1}
};
```
Your turn! Draw the 2-D arrays (a.k.a. matrices) created by these declarations:

double mat1[3][3] = {
    {2.0, 3.5, 4.1},
    {5.33},
    {-10.0, -100.5}
};

double mat2[3][4] = {
    {1.0, 2.5},
    {3.2, 4.2},
    {-1.1}
};
Your turn! Draw the 2-D arrays (a.k.a. matrices) created by this declaration:

```c
float mat3[4][2] = { 1.0, 2.5, 3.2, 4.2, 0.1 };  
```

Remember that C stores 2D arrays in memory in row-major order!
Your turn! Draw the 2-D arrays (a.k.a. matrices) created by these declarations:

```c
float mat3[4][2] = { 1.0, 2.5, 3.2, 4.2, 0.1 };```

```
1.0  2.5
3.2  4.2
0.1  0.0
0.0  0.0
```
QUIZ: Write a program that creates this array:

1  2  3  4
5  6  7  8
Extra-credit
To do for next time:
• Read C/C++ memory concepts - arrays, pointers and history on our webpage
• Read Q&A
• Solve Quizzes 1 – 4 in notebook
• Solve Exercises 1, 2
Homework for Chs. 7 & 8:

Ch. 7: 10, 11, 12
Ch. 8: 6, 7, 9

Due Fri, Oct. 14