QUIZ: loops

Write a program that prints the integers from -7 to 15 (inclusive) using:

- **for** loop
- **while** loop
- **do...while** loop
for (i = -7; i < 16; i++)
    printf("....");

i = -7;
while (i < 16) {
    printf("....");
    i++;
}
i = -7;

do {
    printf(....);
    i++;
} while (i < 16);
QUIZ: loops

Write a program that prints the integers from -7 to 15 using:

- *for* loop
- *while* loop
- *do...while* loop

How many times does the body of the loop execute?
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><code>\a</code></td>
<td>07</td>
<td>Alarm (Beep, Bell)</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>08</td>
<td>Backspace</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>0C</td>
<td>Formfeed</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>0A</td>
<td>Newline (Line Feed); see notes below</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>0D</td>
<td>Carriage Return</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>09</td>
<td>Horizontal Tab</td>
</tr>
<tr>
<td><code>\v</code></td>
<td>0B</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td><code>\</code></td>
<td>5C</td>
<td>Backslash</td>
</tr>
<tr>
<td><code>\</code></td>
<td>27</td>
<td>Single quotation mark</td>
</tr>
<tr>
<td><code>\&quot;</code></td>
<td>22</td>
<td>Double quotation mark</td>
</tr>
<tr>
<td><code>\?</code></td>
<td>3F</td>
<td>Question mark</td>
</tr>
<tr>
<td><code>\</code>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><code>\</code>xhh</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).
Conversion specifiers for `printf` strings a.k.a. placeholders

<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>Unsigned long long integer</td>
<td>unsigned long long</td>
</tr>
</tbody>
</table>
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  numsc: 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 %f", 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 %f/%f = %f\n", g, f, g / f);
20:       
21:       return 0;
22:    }
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");
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
\n\n\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?
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!)

Input: `scanf`

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("Enter a signed decimal integer (i.e. -123)");
        scanf("%d", \&int_var);
    }
    if (choice == 2)
    {
        puts("Enter a decimal floating-point number\
             (e.g. 1.23)");
        scanf("%f", \&float_var);
    }
    if (choice == 3)
    {
        puts("Enter 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!
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,

```c
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].

Referring to a[6] is a big (although common) mistake: a[6] ++
How arrays are declared and used in a program

```c
float expenses[100];
int a[10];
/* additional statements go here */
expenses[i] = 100;     // i is an integer variable
expenses[2 + 3] = 100; // equivalent to expenses[5]
expenses[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)*

```c
/* expenses.c - Demonstrates use of an array */
#include <stdio.h>

/* Declare an array to hold expenses, and a counter variable */
float expenses[13];
int count;
float year_expenses = 0;

int main( void )
{
    /* Input data from keyboard into array */
    for (count = 1; count < 13; count++)
    {
        printf("Enter expenses for month %d: ", count);
        scanf("%f", &expenses[count]);
    }

    /* Print array contents */
    for (count = 1; count < 13; count++)
    {
        printf("Month %d = $%.2f\n", count, expenses[count]);
        year_expenses += expenses[count];
    }
    printf("Yearly expenses are $%.2f\n", year_expenses);
    return 0;
}
```
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:

    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:

```c
arrayname[row_index][column_index]
```
int b[2][3];

Abstract matrix view

![Diagram of 2D array storage]

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];
}
Initializing 1D-arrays

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' };
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]);
}
```
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`
What does this program do?

```c
void main(){
    float mat1[3][3] = {
        {42.0, 43.0, 44.0},
        {45.0, 46.0, 47.0},
        {48.0, 49.0, 50.0}
    };

    float mat2[3][3] = {
        {1.0, 2.5, 0.0},
        {3.2, 4.2, 0.0},
        {0.0, 0.0, 0.0}
    };

    float mat3[3][3];
    printMat(mat1, 3, 3);
    printMat(mat2, 2, 2);
    addMat(mat1, mat2, mat3, 3, 3);
    printMat(mat3, 3, 3);
}
```
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 Monday, Feb.22