We’re Doomed! Doomed!

- Machine’s execution speed
  - 10,000,000 lines of C per second
- Programmer speed
  - ~12 lines of working C per day
- How many days will it take to write a program that runs for 1 second?

CSE 142
Computer Programming I

Iteration

or... How we really get the computer to do our work for us.

(in which we catch up with Turing!)

Outline

Iteration: why do we need it?
What are loops?
How do we write loops in C?
How do we go about writing loops?
Some examples
Nested loops
Other ways to write loops
Dangers and devices

Chapter 5

Read Sections 5.1-5.6, 5.10
5.1 Introduction
5.2-5.3 While statement
5.4 For statement
5.5-5.6 Loop design
5.7 Nested Loops
5.11 Common errors

Revisiting Our Paper Route

Go to house #1
Deliver a paper to #1
Go to house #2
Deliver a paper to #2
Go to house #3
Deliver a paper to #3
...
This is better than before we had functions.
But, it’s still cumbersome. Why?

Revisiting Our Paper Route

For every house on the block:
Go to the house
Deliver a paper to it
We really want to repeat the same process for each house in order.
We want to loop over the houses.
What are Loops?
One More Type of Control Flow

When we want to repeat a block of code, we use a loop.

Loops

- A "loop" is a repeated ("iterated") sequence of statements
- Like conditionals, loops let us control the flow of our program in powerful ways.
- Like functions, loops take code that has been generalized and execute it many times.

(More) Robust Input

char choice = 'x';
printf("Do you want to switch doors? (y/n) ");
scanf (" %c", &choice);

Test: If it didn’t work, try again until it does.

Getting Loopy

char choice;
printf("Do you want to switch doors? (y/n) ");
scanf (" %c", &choice);

while (choice != 'y' && choice != 'n') {
    printf("Do you want to switch doors?\n");
    printf("Please just enter 'y' or 'n'! ");
    scanf (" %c", &choice);
}

How do we write loops in C?
The while statement.

while ( condition ) {
    statement1;
    statement2; ...
}

Psst... just like with conditionals, a while’s body is really a statement. So, why are there two statements? And, why are those braces there?
A Loopless Problem (?)

Problem: add 4 numbers entered at the keyboard.

```c
int sum; int x1, x2, x3, x4;
printf("Enter 4 numbers: "); scanf("%d%d%d%d", &x1, &x2, &x3, &x4);
sum = x1 + x2 + x3 + x4;
```

This works perfectly!

But... what if we had 14 numbers? or 40? or 4000?

How do we go about writing loops? Generalizing!

Problem: read a series of numbers entered at the keyboard and add all of them.

The key to solving problems with loops is to figure out how to do one or a few concrete steps... then generalize.

Our algorithm for adding four numbers was concrete. It had no repeated statements at all...

But it did have some repetition buried in it.

Let’s rework the algorithm to make the repetition more explicit... then, we can solve the general problem.

Add 4 Numbers, Repetitively

```c
int sum, x;
sum = 0;
printf("Enter 4 numbers: ");
scanf("%d", &x);
sum = sum + x;
scanf("%d", &x);
sum = sum + x;
scanf("%d", &x);
sum = sum + x;
scanf("%d", &x);
```

More General Loop to Add Numbers

```c
int sum, x, count; int number_inputs; /* Number of inputs */
sum = 0; printf("How many numbers? ");
scanf("%d", &number_inputs);
printf("Enter %d numbers: ", number_inputs);
count = 1;
while (count <= number_inputs) {
    scanf("%d", &x);
    sum = sum + x;
    count = count + 1;
}
```
Examples: Compute 7!

What is $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7$? ("seven factorial")

```c
x = 1 * 2 * 3 * 4 * 5 * 6 * 7;
printf ("%d", x);
```

Bite size pieces: More Regular: As a loop:

<table>
<thead>
<tr>
<th>x = 1;</th>
<th>x = 1;</th>
<th>x = 1;</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = x * 2;</td>
<td>x = x * i; i = i + 1;</td>
<td>i = 2;</td>
</tr>
<tr>
<td>x = x * 3;</td>
<td>x = x * i; i = i + 1;</td>
<td>while ( i &lt;= 7 )</td>
</tr>
<tr>
<td>x = x * 4;</td>
<td>x = x * i; i = i + 1;</td>
<td>x = x * i;</td>
</tr>
<tr>
<td>x = x * 5;</td>
<td>x = x * i; i = i + 1;</td>
<td>i = i + 1;</td>
</tr>
<tr>
<td>x = x * 6;</td>
<td>x = x * i; i = i + 1;</td>
<td>x = x * i;</td>
</tr>
</tbody>
</table>
| x = x * 7; | x = x * i; i = i + 1; | }

Tracing the Loop

```
/* What is 1 * 2 * 3 * ...*7 */
x = 1 ; /* A */
i = 2 ; /* B */
while ( i <= 7 )  {  /* C */
x = x * i ; /* D */
i = i + 1 ; /* E */
} /* F */
printf ( "%d", x ) ; /* G */
```

Double Your Money

/* Suppose your $1,000 is earning interest at 5% per year. How many years until you double your money? */

```c
my_money = 1000.0;
n = 0;
while ( my_money < 2000.0 ) {
    my_money = my_money * 1.05;
    n = n + 1;
}
printf( "My money will double in %d years.", n);
```

Average Inputs

```c
printf ( "Enter values to average, end with -1.0 \n" ) ;
sum = 0.0 ;
count = 0 ;
sentinel
scanf ( "%f", &next ) ;
while ( next != -1.0 )  {
    sum = sum + next ;
    count = count + 1 ;
    scanf ( "%f", &next ) ;
}
if (count > 0 )
    printf( "The average is %f. \n", sum / (double) count ) ;
```

Printing a 2-D Figure

How would you print the following diagram?

```
∗∗∗∗∗
∗∗∗∗∗
∗∗∗∗∗
```

repeat 3 times
print a row of 5 stars
repeat 5 times

It seems as if a loop within a loop is needed

Is that allowed?
What can go inside loops?

Nested Loop

```c
#define ROWS 3
#define COLS 5
...
row = 1;
while ( row <= ROWS ) {
    /* print a row of COLS *'s */
    ...
    row = row + 1;
}
```
Nested Loop

```c
row = 1;
while (row <= ROWS) {
    /* print a row of COLS "s" */
    col = 1;
    while (col <= COLS) {
        printf("*");
        col = col + 1;
    }
    printf("
");
    row = row + 1;
}
```

Print a Multiplication Table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

Print Row 2

```c
col = 1;
while (col <= 3) {
    printf("%4d", 2 * col);
    col = col + 1;
}
printf("
");`
Notes About Loop Conditions

They offer all the same possibilities as conditions in if-statements
Can use &&, ||, !
Condition is reevaluated each time through the loop
A common loop condition: checking the number of times through the loop

Counting Loops

A common loop condition: checking the number of times through the loop
Requires keeping a "counter"
This pattern occurs so often there is a separate statement type based on it: the for-statement

A for Loop

"What is 1 * 2 * 3 * ... * n ?/
x = 1;
i = 2;
while ( i <= n ) {
    x = x * i;
    i = i + 1;
}
printf ( "%d", x );

for Statement Syntax

for ( initialization;
     condition;
     update expression) {
   statement1;
   statement2;
   ...;
}

for Loop Control Flow

Control Flow: for
**for Loops vs while Loops**

Any for loop can be written as a while loop.

These two loops mean exactly the same thing:

```c
for (initialization; condition; update)
    statement;
```

```c
initialization;
while (condition) {
    statement;
    update;
}
```

**Counting in for Loops**

/* Print n asterisks */
for (count = 1; count <= n; count = count + 1) {
    printf("\n");
}

/* Different style of counting */
for (count = 0; count < n; count = count + 1) {
    printf("\n");
}

**"3 Rows of 5" as a Nested for Loop**

```c
#define ROWS 3
#define COLS 5
...
for (row = 1; row <= ROWS; row = row + 1) {
    for (col = 1; col <= COLS; col = col + 1) {
        printf("\n");
    }
    printf("n");
}
```

**Yet Another 2-D Figure**

How would you print the following diagram?

```
*  
*  
*  
*  
*  
```

For every row (row = 1, 2, 3, 4, 5)
Print row stars

**Solution: Another Nested Loop**

```c
#define ROWS 5
...
int row, col;
for (row = 1; row <= ROWS; row = row + 1) {
    for (col = 1; col <= COLS; col = col + 1) {
        printf("\n");
    }
    printf("n");
}
```

**Yet One More 2-D Figure**

How would you print the following diagram?

```
   *  
   *  
   *  
```

For every row (row = 0, 1, 2, 3, 4)
Print row spaces followed by (5 - row) stars

---

H1-7
Yet Another Nested Loop

```c
#define ROWS 5

int row, col;
for (row = 1; row <= ROWS; row = row + 1) {
    for (col = 1; col <= row - 1; col = col + 1) {
        printf(" ");
    }
    for (col = row; col <= ROWS; col = col + 1) {
        printf("* ");
    }
    printf("\n");
}
```

Generalizing Ever More

```c
/* Print character ch n times */
void RepeatChars ( int n, char ch) {
    int i;
    for ( i = 0 ; i < n ; i = i + 1 ) {
        printf("%c", ch);
    }
    ...
    for ( row = 1 ; row <= ROWS ; row = row + 1 ) {
        RepeatChars ( row - 1, ' ');
        RepeatChars ( ROWS - row + 1, '*');
        printf("\n");
    }
}
```

Remember PrintBannerLines?

```c
/* Print N rows of asterisks */
void PrintBannerLines ( int lines ) {
    int i;
    for ( i = 0 ; i < lines ; i = i + 1 ) {
        RepeatChars(20, '*');
    }
}
```

Some Loop Pitfalls

```c
while ( sum < 10 ) {
    sum = sum + 2;
    for ( i = 1; i <= 10; i = i + 1) {
        sum = sum + i;
    }
    for ( i = 1; i != 10 ; i = i + 2 )
        sum = sum + i;
}
```

Double Danger

```c
double x;
for ( x = 0.0 ; x < 10.0 ; x = x + 0.2 )
    printf("%.18f", x);

Seems harmless...
```

Double Danger

<table>
<thead>
<tr>
<th>What you expect:</th>
<th>What you might get:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>0.0000000000000000</td>
<td>0.0000000000000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9.0000000000000000</td>
<td>8.9999999999999999</td>
</tr>
<tr>
<td>9.2000000000000000</td>
<td>9.1999999999999999</td>
</tr>
<tr>
<td>9.4000000000000000</td>
<td>9.3999999999999999</td>
</tr>
<tr>
<td>9.6000000000000000</td>
<td>9.5999999999999999</td>
</tr>
<tr>
<td>9.8000000000000000</td>
<td>9.7999999999999999</td>
</tr>
<tr>
<td>9.9999999999999999</td>
<td>9.9999999999999999</td>
</tr>
</tbody>
</table>
Use *ints* as Loop Counters

```c
int i;
double x;
for ( i = 0 ; i < 50 ; i = i + 1 )
{
    x = (double) i / 5.0;
    printf("%.18f", x);
}
```

Counting in Loops

Counting up by one or down by one:

```c
for ( i = 1 ; i <= limit ; i = i+1 ) { ... }
times_to_go = limit;
while ( times_to_go > 0 ) {
    ... 
times_to_go = times_to_go - 1;
}
```

Counting Up or Down by 1

This pattern is so common there is special jargon and notation for it

To "increment:" increase (often by 1)
To "decrement:" decrease (often by 1)

C operators:
Post-increment ( x++ ): add 1
Post-decrement ( x-- ): subtract 1

Handy Shorthand  

```
x++   x--
```

Used by itself,
```
x++ means the same as x = x+1
x-- means the same as x = x-1
```

Very often used with loop counters:
```
for ( i=1 ; i <= limit ; i++ ) { ... }
times_to_go = limit;
while ( times_to_go > 0 ) {
    ... 
times_to_go-- ...
```

Surgeon General’s Warning

++ and -- are unary operators.
Pre-increment (++x) and pre-decrement (--x) exist, too.
In this course, use ++ and -- only in isolation.
Don’t combine these with other operators in expressions! E.g., don’t try
```
x = y++ / (3 * --x--)
```

Iteration Summary

General pattern:
```
Initialize, test, do stuff, repeat . . .
```

“while” and “for” are equally general in C
Use “for” when initialize/test/update are closely related and simple, especially when counting
Looking Ahead

We’ll talk more about how to design loops
We’ll discuss complex conditional expressions
Can be used with loops as well as in conditional statements
We’ll see “arrays”, a powerful new way of organizing data
Very often used with loops

QOTD:
Counting Crows

A vital part of using loops is to understand the structure of the data you’re looping over.
In particular, the loops we look at require “serializing” the data: giving it an order and going through it one at a time.
Serialize the following data:
- A murder of crows on a wire
- The rooms on your house (for vacuuming!)
- The integers (to check for primes!)
- All integral, positive (x, y) coordinates

There’s something tricky about the last one.
What is it???