Overview
Concepts this lecture
- Iteration - repetitive execution
- Loops and nested loops
- while statements
- for statements

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

An Old Friend: Fahrenheit to Celsius
```c
#include <stdio.h>
int main(void)
{
    double fahrenheit, celsius;
    printf("Enter a Fahrenheit temperature: ");
    scanf("%lf", &fahrenheit);
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %f degrees Celsius. ",
            celsius);
    return 0;
}
```

What’s “Wrong” with Fahrenheit/Celsius Program?
User has to rerun the program for every new temperature
Wouldn’t it be nice if the program could process repeated requests?
Program ends immediately if user types a bad input
Wouldn’t it be nice the program politely asked the user again (and again, etc. if necessary)?

One More Type of Control Flow
Sometimes we want to repeat a block of code. This is called a loop.
Loops

A “loop” is a repeated (“iterated”) sequence of statements. Like conditionals, loops (iteration) give us a huge increase in the power of our programs.

Alert: loops are harder to master than if statements. Even experienced programmers often make subtle errors when writing loops.

Motivating Loops

Problem: add 4 numbers entered at the keyboard.

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?

Finding Repeated Code

The key to using loops to solve a problem is to discover steps that can be repeated.

Our first algorithm for adding four numbers had no repeated statements at all.

But it does have some repetition buried in it.

Let’s rework the algorithm to make the repetition more explicit.

Add 4 Numbers, Repetitively

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);
sum = sum + x;

Loop to Add 4 Numbers

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);
sum = sum + x;

while Statement Syntax

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

Loop body: Any statement, or a compound statement.

Loop condition
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;
}
```

Compute 7!

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

Bite size pieces: More Regular: As a loop:

```c
x = 1; x = 1; i = 2; x = 1;
x = x * 2; x = x * i; i = i + 1; i = 2;
x = x * 3; x = x * i; i = i + 1; while ( i <= 7 ) {
x = x * 4; x = x * i; i = i + 1; x = x * i;
x = x * 5; x = x * i; i = i + 1; i = i + 1;
x = x * 6; x = x * i; i = i + 1; x = x * i;
x = x * 7; x = x * i; i = i + 1;
```

Tracing the Loop

```c
/* What is 1 * 2 * 3 * ... * ? */ line i x i?7?
A 1 1
B 2 1 T
while ( i <= 7 ) { /* B */
    i = i + 1;
    x = x * i; /* D */
    printf ("%d", x); /* G */
    while ( i <= 7 ) {
        i = i + 1;
        x = x * i;
        printf ("%d", x);
        G 5040 T
        E 8 720 F
        D 7 720 T
        C 5 5040 T
        B 6 120 T
    }
    i = i + 1;
    printf ("%d", x);
    G 5040 T
    E 8 720 F
    D 7 720 T
    C 5 5040 T
    B 6 120 T
}
```

Double Your Money

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

while Loop Control Flow

```c
x = 1; i = 2;
while ( i <= 7 ) {
    x = x * i;
    i = i + 1;
    if ( i <= 7 ) { /* yes */
        x = x * i;
        i = i + 1;
    }
    if ( i <= 7 ) { /* no */
        x = x * i;
        i = i + 1;
    }
}
```
**Average Inputs**

```c
printf ( "Enter values to average, end with -1.0 \n" );
sum =  0.0 ;
count =  0 ;
scanf ( "%lf", &next ) ;
while ( next  !=  -1.0 ) {
    sum = sum + next ;
    count = count + 1;
    scanf ( "%lf", &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
print *

It seems as if a loop within a loop is needed.

**Nested Loop**

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

**Trace**

```
row:  1  2  3  4
col: 1 2 3 4  1 2 3 4  1 2 3 4
output: * * * * *
        * * * * *
        * * * * *
```

**Print a Multiplication Table**

```
row = 1;
while ( row <= ROWS ) {
    /* print a row of 5 *'s */
    col = 1;
    while (col <= COLS) {
        printf("*");
        col = col + 1;
    }
    printf(" ");
    row =  row + 1;
}
```
Nested Loops

```c
row = 1;
while (row <= 4) {
    col = 1;
    while (col <= 3) {
        printf("%4d", row * col);
        col = col + 1;
    }
    printf("\n");
    row = row + 1;
}
```

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

```c
/* What is 1 * 2 * 3 * ... * n ? */
int i; x = 1;
for (i = 2; i <= n; i = i+1) {
    x = x * i;
}
printf("%d", x);
```
for Statement Syntax

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

for Loop Control Flow

```
Initialization

Condition

yes

For Loop Body

no

Update Expression
```

for Loops vs while Loops

Any for loop can be written as a while loop. These two loops mean exactly the same thing:
```
for (initialization; condition; update) statement;
```
```
while (condition) {
    statement;
    update;
}
```

Counting in for Loops

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

“3 Rows of 5” as a Nested for Loop

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

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("\n");
    printf("\n");
}
```

The Appeal of Functions

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

Some Loop Pitfalls

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

Double Danger

```c
double x;
for (x = 0.0; x < 10.0; x = x + 0.2)
    printf("%.1f", x);

Seems harmless...
```
**Double Danger**

What you expect:  |  What you might get:
--- | ---
0.000000000000000000 | 0.000000000000000000
0.200000000000000000 | 0.200000000000000000
0.400000000000000000 | 0.400000000000000000
... | ...
9.000000000000000000 | 8.999999999999999997
9.200000000000000000 | 9.199999999999999996
9.400000000000000000 | 9.399999999999999996
9.600000000000000000 | 9.599999999999999996
9.800000000000000000 | 9.799999999999999996

**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 ) { . . . }
```

```c
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**

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++ ) { . . . }
```

```c
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