

## CSE 142 Computer Programming I

### Iteration

© 2000 UW CSE

H1-1

## Overview

### Concepts this lecture

- Iteration - repetitive execution
- Loops and nested loops
- while statements
- for statements

H1-2

## 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

H1-3

## An Old Friend: Fahrenheit to Celsius

```
#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;
}
```

H1-4

## 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?

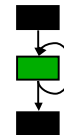
Program ends immediately if user types a bad input

Wouldn't it be nice the program politely asked the user again (and again, etc. if necessary)?

H1-5

## One More Type of Control Flow

Sometimes we want to repeat a block of code. This is called a *loop*.



H1-6

## 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

H1-7

## 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?**

H1-8

## 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

H1-9

## 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;
```

H1-10

## 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;
```

```
int sum, x;
int count;

sum = 0;
printf("Enter 4 numbers:");
```

```
count = 1;
while (count <= 4) {
    scanf("%d", &x);
    sum = sum + x;
    count = count + 1;
}
```

H1-11

## while Statement Syntax

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

Loop condition

Loop body: Any statement, or a compound statement

H1-12

## More General Loop to Add Numbers

```
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;
}
```

H1-13

## Compute 7!

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

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

H1-14

## Compute 7!

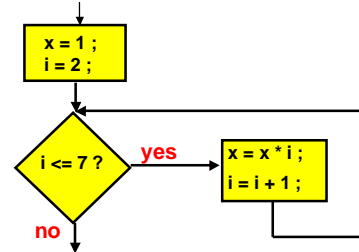
What is  $1 * 2 * 3 * 4 * 5 * 6 * 7$ ? ("seven factorial")

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

Bite size pieces:	More Regular:	As a loop:
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 * 7;	x = x * i; i = i + 1;	

H1-15

## while Loop Control Flow



H1-16

## Tracing the Loop

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

H1-17

## Double Your Money

/\* 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);
```

H1-18

## Average Inputs

```
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 %. \n",
           sum / (double) count );
```

↑  
sentinel

H1-19

## 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

H1-20

## Nested Loop

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

H1-21

## Nested Loop

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

outer loop: print 3 rows

inner loop: print one row

H1-22

## Trace

```
row: 1      2      3      4
col: 123456 123456 123456
```

output: \* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

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

H1-23

## Print a Multiplication Table

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9
4	4	8	12

	1	2	3
1	1 * 1	1 * 2	1 * 3
2	2 * 1	2 * 2	2 * 3
3	3 * 1	3 * 2	3 * 3
4	4 * 1	4 * 2	4 * 3

H1-24

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9
4	4	8	12

Print Row 2

```
col = 1;
while (col <= 3) {
    printf("%4d", 2 * col);
    col = col + 1;
}
printf("\n");
```

row number

### Nested Loops

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

Print 4 rows

Print one row

### Loop Trace

row col	print	row col	print
1 1	1	3 1	3
1 2	2	3 2	6
1 3	3	3 3	9
1	print \n	3	print \n
2 1	2	4 1	4
2 2	4	4 2	8
2 3	6	4 3	12
2	print \n	4	print \n

### 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 );
```

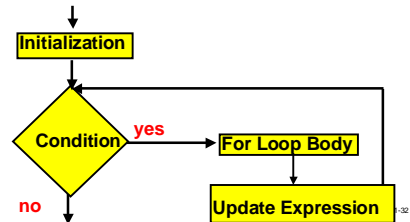
```
x = 1;
for ( i = 2; i <= n; i = i+1 ) {
    x = x * i;
}
printf ( "%d", x );
```

## for Statement Syntax

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

H1-31

## for Loop Control Flow



## 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;
```

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

H1-33

## 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 ( "*" );  
}
```

H1-34

## "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" );  
}
```

inner loop: print one row  
outer loop: print 3 rows

H1-35

## Yet Another 2-D Figure

How would you print the following diagram?

```
*  
* *  
* * *  
* * * *  
* * * * *
```

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

H1-36

## Solution: Another Nested Loop

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

H1-37

## 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

H1-38

## Yet Another Nested Loop

```
#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" );
}
```

H1-39

## The Appeal of Functions

```
/* 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" );
}
```

H1-40

## Some Loop Pitfalls

```
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 ;
```

H1-41

## Double Danger

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

Seems harmless...

H1-42

## Double Danger

What you expect:	What you might get:
0.0000000000000000	0.0000000000000000
0.2000000000000000	0.2000000000000000
0.4000000000000000	0.4000000000000000
...	...
9.0000000000000000	8.9999999999999997
9.2000000000000000	9.1999999999999996
9.4000000000000000	9.3999999999999996
9.6000000000000000	9.5999999999999996
9.8000000000000000	9.7999999999999996
	9.9999999999999996

## Use *ints* as Loop Counters

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

H1-44

## Counting in Loops

Counting up by one or down by one:

```
for ( i = 1 ; i <= limit ; i = i+1 ) { ... }
```

```
times_to_go = limit;
while ( times_to_go > 0 ) {
    ...
    times_to_go = times_to_go - 1;
}
```

H1-45

## 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

H1-46

## 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--
```

H1-47

## 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--)
```

H1-48

## 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

H1-49

## 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

H1-50