University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Statements in C

Remember: Statements Tell the Computer What to Do

In **C**, **a statement tells the computer to do something**.

There are **three types of statements**.

But statements can consist of other statements,

which can consist of other statements,

and so forth.

Many Statements are Quite Simple

Here are two of the three types…

```
;       /* a null statement */

/* A simple statement is often an
   expression and a semicolon. */
A = B;  /* simple statements */
printf ("Hello, ECE120!\n");
```

These two types **end with a semicolon** (;).

Compound Statements Consist of Other Statements

Third type: a **compound statement** consists of
- a **sequence of statements**
- **between braces**.

```
{   /* a compound statement */
    radius = 42;
    C = 2 * 3.1416 * radius;
    printf ("C = %f\n", C);
}
```

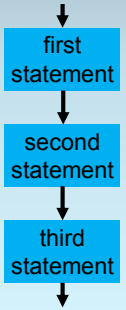
A compound statement may also contain variable declarations for use inside the statement.

A Program is a Sequence of Statements

The function body of **`main`** is a compound statement.

The function body of **`main`** thus **includes a sequence of statements**.

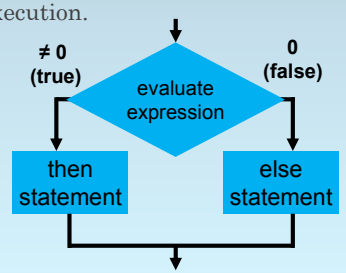
When program is started, **these statements execute in sequential order**.



Simple Statements Can Also Introduce Conditions

Simple statements in **C** can also introduce **conditional** execution.

Based on an expression, the computer executes one of two statements.



C's `if` Statement Enables Conditional Execution

Conditional execution uses the **`if` statement**:

```
if ( <expression> ) {
    /* <expression> != 0:
       execute "then" block */
} else {
    /* <expression> == 0:
       execute "else" block */
}
```

`<expression>` can be replaced with any expression, and **"else { ... }" can be omitted**.

Examples of the `if` Statement

For example,

```
/* Calculate inverse of number. */
if (0 != number) {
    inverse = 1 / number;
} else {
    printf ("Error!\n");
}
```


Examples of the `if` Statement

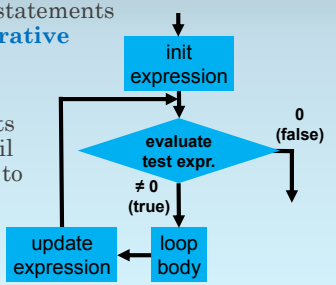
Or,

```
/* Limit size to 42. */
if (42 < size) {
    printf ("Size set to 42.\n");
    size = 42;
}
```


Simple Statements Can Also Be Iterations

Finally, simple statements can describe **iterative** execution.

This type of execution repeats a statement until a test evaluates to false (0).



C's `for` Loop Enables Iterative Execution

The following is called a **`for` loop**:

```
for (<init>; <test>; <update>) {
    /* loop body */
}
```

As shown on the previous slide, the computer:

1. Evaluates `<init>`.
2. Evaluates `<test>`, and stops if it is false (0).
3. Executes the **`loop body`**.
4. Evaluates `<update>` and returns to Step 2.

Iterations are Used for Repeated Behavior

```
/* Print multiples of 42 from
   1 to 1000. */
int N;
for (N = 1; 1000 >= N; N = N + 1) {
    if (0 == (N % 42)) {
        printf ("%d\n", N);
    }
}
```


Let's See How This Loop Works

```
/* Print 20 Fibonacci numbers. */
int A = 1; int B = 1; int C; int D;
for (D = 0; 20 > D; D = D + 1) {
    printf ("%d\n", A);
    C = A + B;
    A = B;
    B = C;
}
```


Another Iterative Construct: the `while` Loop

C provides other loop constructs, but
only the `for` loop is needed for ECE120.

However, we may forget to remove
`while` loops from our example programs.

A `while` loop
- only specifies a `<test>` and a **`loop body`**,
- but is otherwise equivalent to a for loop.

```
while (<test>) {
    /* loop body */
}
```


Easy to Map `while` Loop into `for` Loop

```
while (<test>) {
    /* loop body */
}
```

is completely equivalent to
(with empty `<init>` and `<update>`):

```
for ( ; <test>; ) {
    /* loop body */
}
```


Execution of a `while` Loop

How does the computer execute a `while` loop?

```
while (<test>) {
    /* loop body */
}
```

We can simplify the rules for a `for` loop…

1. ~~Evaluates `<init>`~~. Skip this step.
2. Evaluates `<test>`, and stops if it is false (0).
3. Executes the **`loop body`**.
4. ~~Evaluates `<update>`~~ and returns to Step 2.

Skip this part.

