Loops

ITCS 2116: C Programming

College of Computing and Informatics

Department of Computer Science

Flow of control

- Flow-of-control statements in C
  - `if-then-else`
  - `conditional operator (?:)`
  - **`for`**
  - **`continue`** and **`break`**
  - **`while`** and **`do-while`**
  - **`switch-case`**
  - `goto`

Constant Expressions

- Constant-valued expressions are used in…
  - case statement labels
  - array bounds
  - bit-field lengths
  - values of enumeration constants
  - initializers of **static** variables

all evaluated at **compile** time, not run time

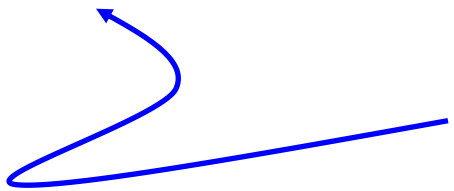
```
static int a = 35 + (16 % (4 | 1));
```

*(**`static`**: variable's value is initialized only once, no matter how many times the block in which it is defined is executed)*

Constant Expressions… (cont'd)

- **Cannot** contain assignments, increment or decrement operators, function calls, …
  - see a C reference manual for all the restrictions
  - basically: nothing that has to be evaluated at **run-time**

```
static int b = a++ - sum();
```



error

Why is Repetition Needed?

- Repetition allows efficient use of variables
- Can input, add and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare one variable for each number, input the numbers and add the variables together
    OR
  - **Create a loop** that reads a number into a variable and adds it to a variable that contains the sum of the numbers

for

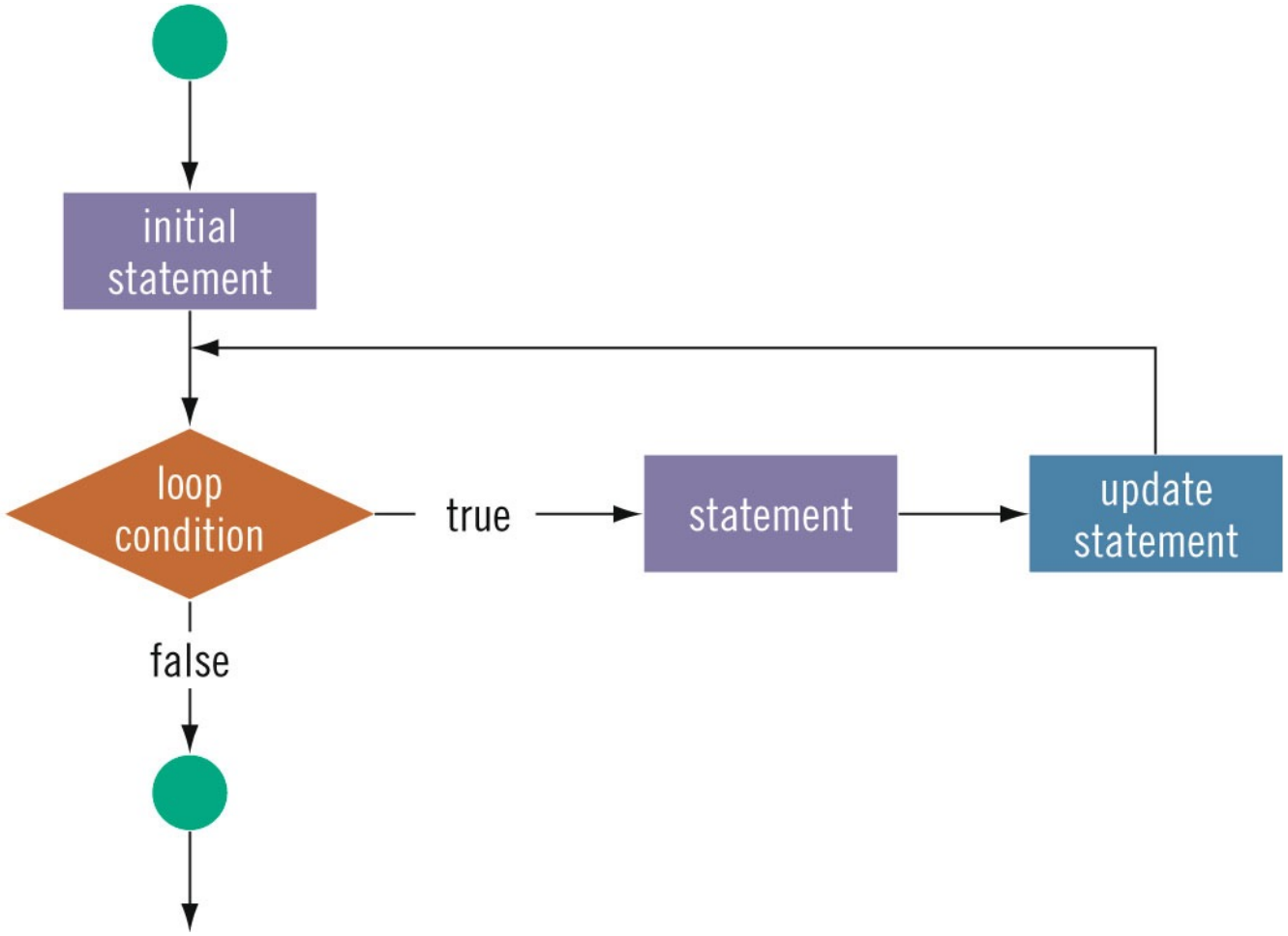
- Used for iterative operations, i.e., instructions that must be executed multiple times
- Syntax:

```
for (init expr; cond expr; incr expr)
   statement;
```

  - The *increment expression* is also known as the *loop expression* or *update statement*.
- Example:

```
for ( int i = 0; i < 10; i++ )
   printf("i=%d\n", i);
```

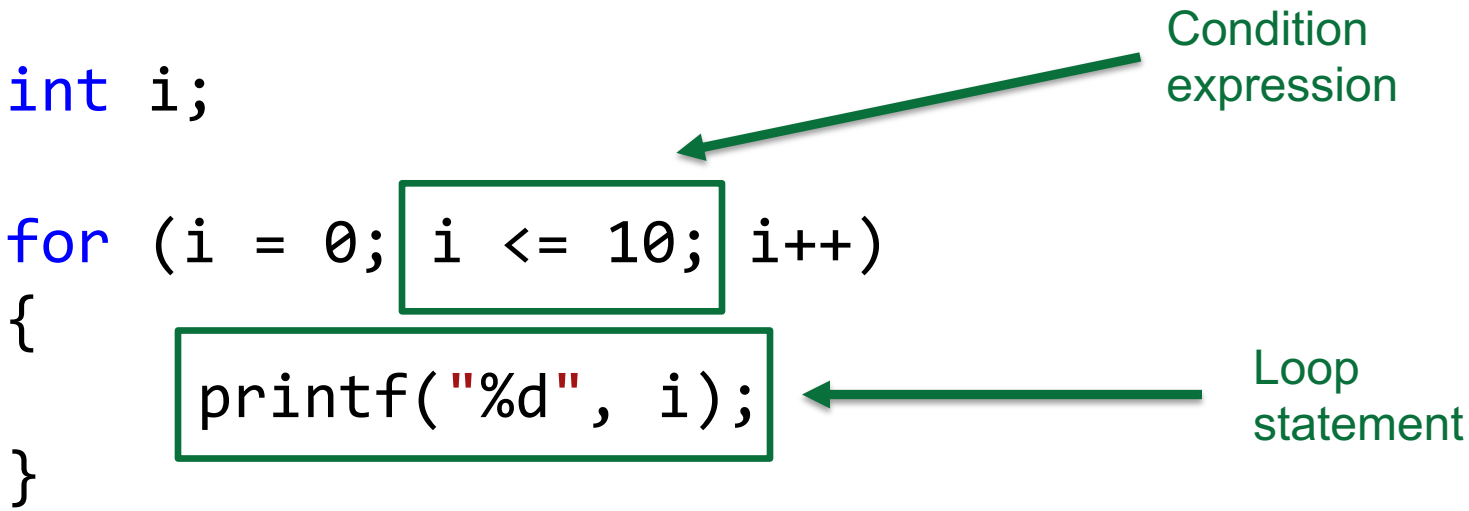
`for` loop (Repetition) Structure



`for` loop (Repetition) Structure Example

```
int i;

for (i = 0; i <= 10; i++)
{
    printf("%d", i);
}
```



- The statement will execute as long as the condition evaluates to true.
- How many times will the statement execute?

(see **`square2.c`** in *Code Samples and Demonstrations* in *Canvas*)

Reminder: Compound Statements

- Any group of statements that is surrounded by braces will be handled by the C compiler as a single statement.
- Syntax:

```
{
    statement1;
    statement2;
    …
    statementn;
}
```

`for` loop (Repetition) Structure Example

```
int i;

for (i = 0; i < 5; i++)
{
      printf("Hello\n");
      printf("*\n");
}
```

This loop outputs (prints) Hello and a star (on separate lines) five times.

`for` loop (Repetition) Structure Example

```
int i;

for (i = 0; i < 5; i++)
     printf("Hello\n");
     printf("*\n");
```

This loop outputs (prints) Hello five times and a star only once.

`for` loop (Repetition) Structure (cont'd.)

- The following is a semantic error:

```
for (i = 0; i < 5; i++);
      printf("%d ", i);
```

  The semicolon at the end of the **`for`** statement terminates the loop. The action for this loop is empty.

- The following is a legal (but infinite) **`for`** loop:

```
for (;;)
    printf("Hello\n");
```

`for` loop (Repetition) Structure Counting Backward

Consider the following code:

```
for (int j = 10; j >= 1; j--)
{
      printf("%d ", j);
}
```

Sample output:

```
 10   9   8   7   6   5   4   3   2   1
```

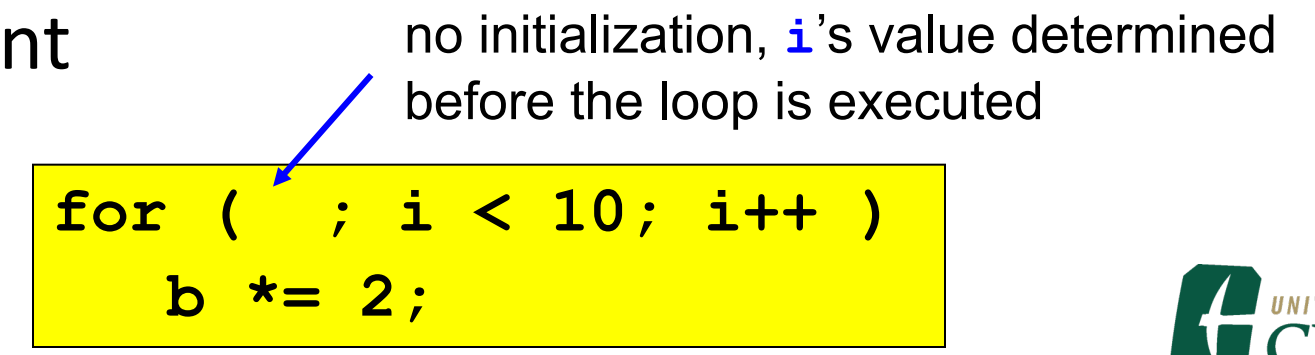
The variable `j` is initialized to 10. After each iteration of the loop, `j` is decremented by 1. The loop continues to execute as long as `j >= 1` evaluates to true

for

- The value of the counter after the loop is exited **is valid** and can be tested or used
  - In C99 you can declare your counter in the for loop

```
for ( i = 0; i < 10; i++ )
   b *= 2;
printf("b was doubled %d times\n", i);
```

- Some parts of the expression can be missing; default to null statement



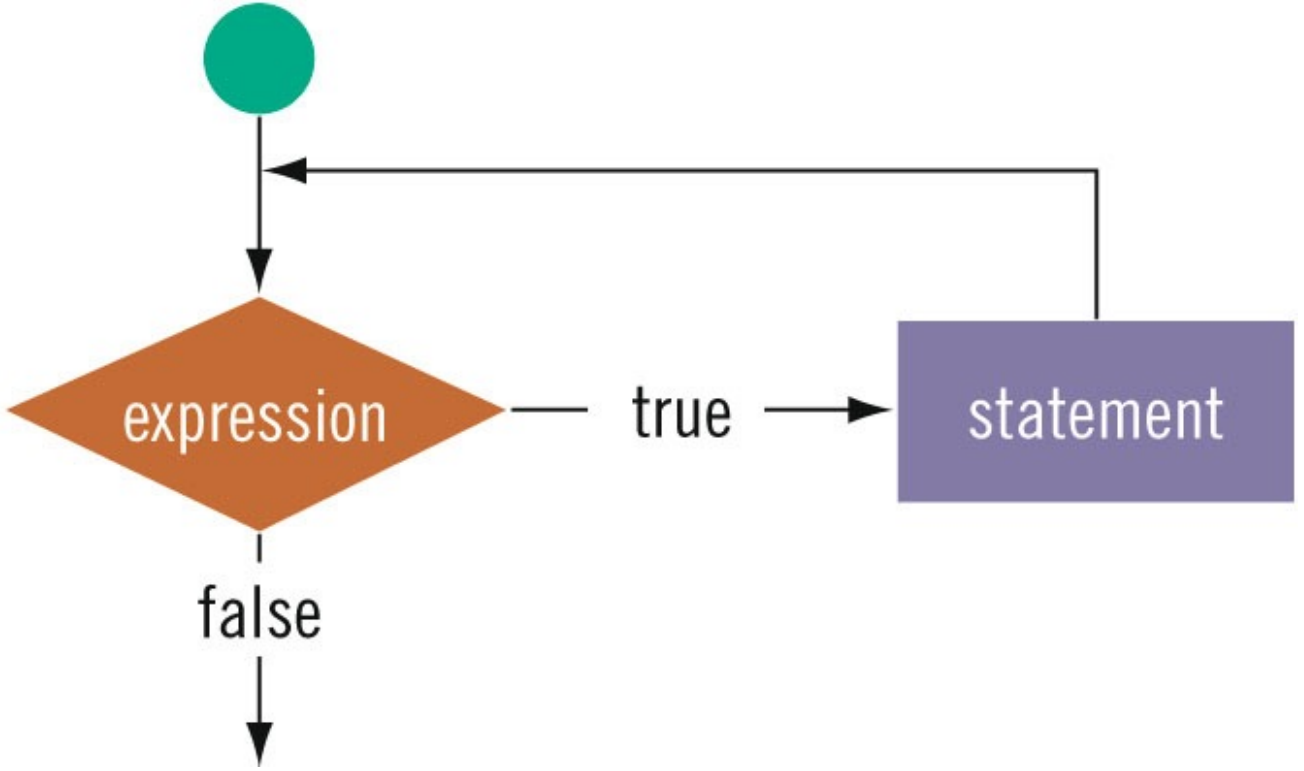
`while` loop (Repetition) Structure

- Syntax of the **`while`** statement:

```
while (expression)
    statement
```

- `statement` can be simple or compound
- `expression` acts as a decision maker and is usually a logical expression
- `statement` is called the body of the loop
- The parentheses are part of the syntax

`while` loop (Repetition) Structure (cont'd.)



`while` loop (Repetition) Structure Example

Consider the following code:

```
int i = 0;

while (i <= 10)
{
      printf("%d ", i);

      i = i + 1;
}
```

Sample output:

```
0   1   2   3   4   5   6   7   8   9   10
```

`while` loop (Repetition) Structure Example (cont'd.)

- The variable `i` in the loop is called the **loop control variable**
- **Infinite loop**: continues to execute endlessly
  - Avoided by including statements in loop body that assure the exit condition is eventually false

General Form of a **`while`** Loop

```
// Initialize the loop control variable(s)

while (expression)    //expression tests the LCV
{
    .
    .
    .
    // Update the LCV
    .
    .
    .
}
```

(see **`sum.c`** in *Code Samples and Demonstrations* in *Canvas*)

Various Forms of `while` Loops

- Counter-controlled **`while`** loop
- Sentinel-controlled **`while`** loop

Counter-Controlled `while` Loops

Use when you know exactly how many times the statements need to be executed

```
counter = 0;          //initialize the loop control variable

while (counter < N) //test the loop control variable
{
      .
      .
      .
      counter++;      //update the loop control variable
      .
      .
      .
}
```

(see **square.c** in *Code Samples and Demonstrations* in *Canvas*)

Sentinel-Controlled `while` Loops

- **Sentinel** variable is tested in the condition
- Loop ends when sentinel is encountered

```
int num = 0;
int sum = 0;          //  Initialize the loop control variable

while (num != -1)     //  Test the loop control variable
{
    sum += num;

    scanf("%d", &num);  //  Update the loop control variable
}

printf("%d", sum);
```

(see **sum.c** in *Code Samples and Demonstrations* in *Canvas*)

`do...while` loop (Repetition) Structure

- Syntax of a **`do...while`** loop:

```
do
   statement
while (expression);
```

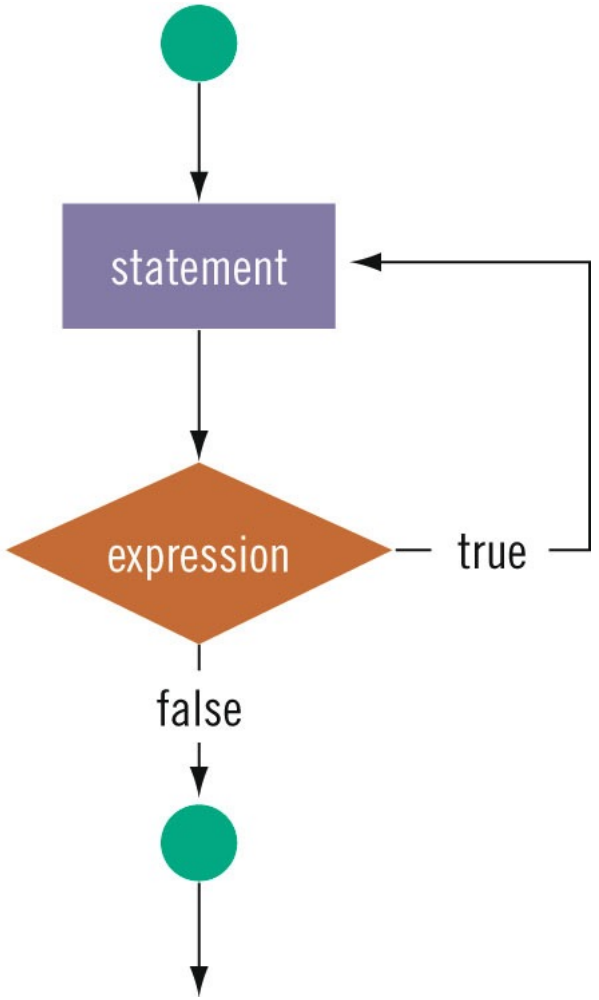
- The `statement` executes first, and then the `expression` is evaluated
  - As long as `expression` is true, the loop continues
- To avoid an infinite loop, the body must contain a statement that makes the `expression` false

`do...while` loop (Repetition) Structure (cont'd.)

- The statement can be simple or compound
- Loop always iterates at least once
- Often used for input validation

(see **`numdigits.c`** in *Code Samples and Demonstrations* in *Canvas*)

`do...while` loop (Repetition) Structure (cont'd.)



(see **`numdigits.c`** in *Code Samples and Demonstrations* in *Canvas*)

`do...while` loop (Repetition) Example

Consider the following code:

```
int x = 0;

do
{
   printf ("%d  ",x);
   x = x + 10;
} while (x <= 100);
```

Sample output:

```
0  10  20  30  40  50  60  70  80  90  100
```

`break` and `continue` Statements

- **`break`** and **`continue`** alter the flow of control
- The **`break`** statement is used for two purposes:
  - To exit early from a loop, which can eliminate the use of certain (flag) variables
  - To skip the remainder of a **`switch`** structure
- After **`break`** executes, the program continues with the first statement after the structure

(see **`break_example.c`** in *Code Samples and Demonstrations* in *Canvas*)

`break` and `continue` Statements (cont'd.)

- `continue` is used in `while`, `for`, and `do…while` structures
- The `continue` statement skips any statements that remains in the loop and proceeds with the next iteration of the loop

(see `continue_example.c` in *Code Samples and Demonstrations* in *Canvas*)

`break` Statement

Terminates execution of **closest** enclosing **`for`**, **`while`**, **`do`**, or **`switch`** statement

which loop(s) does this exit?

```
b = 0;
for ( i = 0; i < 10; i++ ) {
   for (j = 0; j < 5; j++) {
      if (a[i][j] > 100)
         break;
      b += a[j];
   }
   printf("b = %d\n", b);
}
```

(see **`break_example.c`** in *Code Samples and Demonstrations* in *Canvas*)

`continue` Statement

- Use to bypass **one (1) iteration** of the innermost loop
  - but **not** exiting the loop altogether
- Example:

```
for ( i = 0; i < 5;) {
    printf("Enter the next number: ");
    scanf("%d", &next_num);

    if (next_num <= 0)
        continue;

    sum += next_num;
    printf("Sum = %d\n", sum);
    i++;
}
```

(see `continue_example.c` in *Code Samples and Demonstrations* in *Canvas*)

Combining Assignment and Condition Checking

Why write this…

```
c = getchar();
while (c != '\n') {
    …do something…
    c = getchar();
}
```

does the same thing!

…when you can write this instead?

```
while ( (c = getchar()) != '\n' ) {
    …do something…
}
```

Nested Control Structures

- To create the following pattern:

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

- We can use the following code:

```
for (i = 1; i <= 5 ; i++)
{
    for (j = 1; j <= i; j++)
            printf("*");
    printf("\n");
}
```

Nested Control Structures (cont'd.)

- What is the result if we replace the first for statement with this?

```
for (i = 5; i >= 1; i--)
```

- *Try to figure it out it before you go to the next slide…*

Nested Control Structures (cont'd.)

- What is the result if we replace the first for statement with this?

```
for (i = 5; i >= 1; i--)
```

- Answer:

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

References

- K. N. King, *C Programming: A Modern Approach*, 2nd Edition. W. W. Norton & Company. 2008.

- D.S. Malik, *C++ Programming: From Problem Analysis to Program Design*, Seventh Edition. Cengage Learning. 2014.