Control Statements: Looping

COEN 10

C -- Lecture 4

Control Statements: Looping

★3 types of loops

- ◎while
- ◎for
- ◎do while

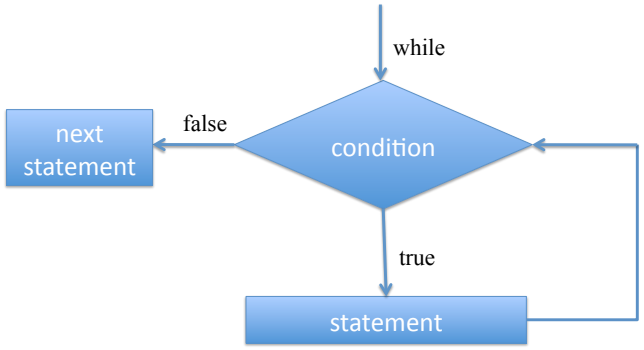
The while Statement

★General form

```
while (expression)
  statement
```

→ Executes the statement while the expression is true or has a value different than zero

The while Statement



while: Terminating the loop

★Crucial

◎A while loop must contain something that changes the value of the test expression so that the loops eventually stops

while: Terminating the loop

★Example

```
i = 1;
while (i < 5)
  printf ("in the while loop ...\n");
```

while: Terminating the loop

★Example

```
i = 1;
while (--i < 5)
  printf ("in the while loop ...\n");
```

while: Terminating the Loop

★The decision to terminate the loop takes place when the test condition is evaluated.

```
n = 5;
while (n < 7)
{
  printf ("n = %d\n", n);
  n++;
  printf ("n = %d\n", n);
}
```

while: An Entry-Condition Loop

★Condition is verified before entering the loop

◎The loop may execute zero or more iterations

while: Syntax Points

★Only one statement, simple or compound, following the test condition is part of the loop.

```
n = 0;
while (n < 3)
  printf ("n = %d\n", n);
  n++;
```

while: Syntax Points

★The while statement is itself a single statement

◎Runs from the while to the first semicolon (simple) or to the terminating brace (block).

while: Syntax Points

★Careful with the semicolon

Example:

```
n = 0;
while (n++ < 3);
  printf ("n = %d\n", n);
```

while: Syntax Points

★If you really want a null statement

Example:

```
num = 0;
while (scanf ("%d", &num) == 1)
  ;     // skip the integer input
```

Relational Operators and Expressions

< → is less than

<= → is less than or equal to

== → is equal to

>= → is greater than or equal to

> → is greater than

!= → is not equal to

Relational Operators and Expressions

★Defined for

◎Integer values

◎Character values

◎Floating point values

✧Careful with the ==

★Not defined for

◎Strings or Arrays

Relational Operators and Expressions

★Value

◎0 → false

◎1 → true

Relational Operators and Expressions

★Example

```
while (1)    // forever
{
  …
}
```

Relational Operators and Expressions

★In fact

- ◎zero → false
- ◎not zero → true

Relational Operators and Expressions

★Example

```
int n = 3;
while (n)
  printf ("n = %d\n", n--);
```

Relational Operators and Expressions

★Example

```
while (n != 0)
```

same as

```
while (n)
```

Relational Operators and Expressions

★Troubles with Truth

```
status = scanf ("%d", &num);
while (status = 1)
{
   sum += num;
   status = scanf ("%d", &num);
}
```

Relational Operators and Expressions

★Idea to avoid the problem

◎Get the constant on the left

```
status = scanf ("%d", &num);
while (1 = status)    // compilation error
{
   sum += num;
   status = scanf ("%d", &num);
}
```

Relational Operators and Expressions

★The new _Bool Type

◎Can only have two values

- ✧1 → true
- ✧0 → false

Relational Operators and Expressions

★C90 defines <stdbool.h>

◎Type - bool

◎Values - true and false

Relational Operators and Expressions

★Precedence

◎Lower than arithmetic operators

◎Higher than assignment operators

Relational Operators and Expressions

★Precedence Rules

1.	++,-- postfix ( )	→ L-R
2.	++,-- prefix +,- (type) sizeof	→ R-L
3.	* / %	→ L-R
4.	+ -	→ L-R
5.	< > <= >=	→ L-R
6.	== !=	→ L-R
7.	=	→ R-L

Counting Loops

★for loops

◎the natural option for counting

★while loops

◎better for indefinite loops

Counting Loops

★for loops

◎Gathers all three actions into one place

✧Initializing

✧Testing

✧Updating

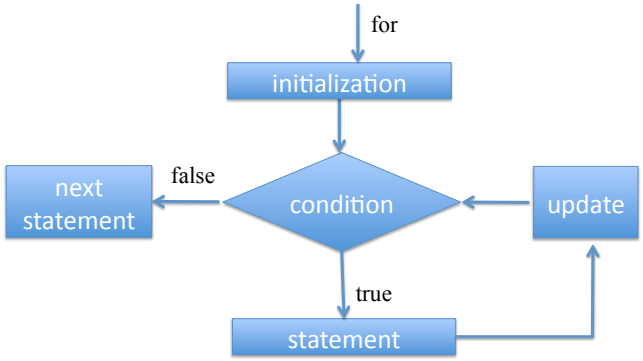
Counting Loops

★for loops

◎Format

for (initialization; condition; update)
  statement

The for Statement



Counting Loops

★for flexibility

◎Count up and down

◎Count by different amounts

◎Count by characters

◎Test any condition

◎Update geometrically

◎Use any expression for updating

Counting Loops

★for flexibility

◎Leave one or more of the expressions empty

```
for ( ; ; )    // forever loop
```

◎The first expression can be any expression that will execute only once

Counting Loops

★for flexibility

◎The parameters of the loop can be modified during the loop execution

Counting Loops

```
n = 10;
delta = 1;
for (i = 0; i < n; i += delta)
{
    …
    if (x > 100)
    {
        n *= 100;
        delta *= 2;
    }
    …
}
```

More Assignment Operators

★To update variables

+= → add to

-= → decrease from

*= → multiply to

/= → divide from

%= → module from

The Comma Operator

★Extends the flexibility of the for loop

◎Enables more than one initialization and update

◎Example

```
for (i = 0, j = 0; I + j < x; i++, j += 2)
{
    …
}
```

The Comma Operator

★In expressions

◎The comma is a sequence point and guarantees that operations happen from left to right

Example

```
x++, y = x * 10;
```

Don't do that!!

The Comma Operator

★In expressions

◎The value of the whole comma expression is the value of the right hand member

Example

```
x = (y = 3, (z = ++y + 2) + 5);
```

Don't do that!!

The Comma Operator

★Also used in as a separator

◎Arguments in function calls

Example

```
printf ("x = %d, y = %d\n", x, y);
```

◎Declaration of multiple variables

Example

```
int  x, y;
char a, b = 'c';
```

An Exit-Condition Loop

★do-while loop

◎The condition is checked after each iteration of the loop

◎Statements always execute at least once

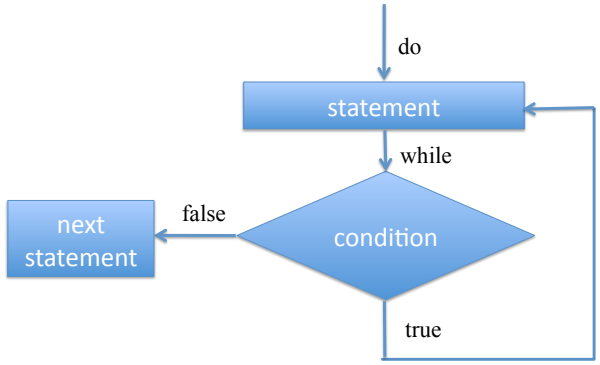
The do-while Statement

★General form

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

→ Re-executes the statement while the expression is true or has a value different than zero

The do-while Statement



Which loop?

★for and while

◎Can be used interchangeably

Which loop?

★for

◎Counting

```
for (i = 0; i < value; i++)
```

★while

◎Condition

```
while (scanf ("%d", &number) == 1)
```

Nested Loops

- A loop inside another loop
- Can have several levels
- Example with two levels
  - outer loop and inner loop
  - For each iteration of the outer loop, the program executes all the iterations of the inner loop

Nested Loops

- Note
  - The inner loop iterations may depend on the outer loop

Intro to Arrays

- Arrays enable the "grouping" of several items of related information

Intro to Arrays

- An array is a series of values of the same type, stored sequentially
  - The whole array has a single name
  - Individual elements are accessed with an integer index

Intro to Arrays

★Declaration

type name[size];

or

type name[size] = {value1, value2, …};

or

type name[ ] = {value1, value2, …};

Intro to Arrays

★Example

int numbers[10];

→Declares an array called numbers, with 10 elements, each of which holds an integer

→The first element is numbers[0], the second is numbers[1], and so on

→The last element is numbers[9]

Intro to Arrays

★An array can be of any type

★An array element can be used in the same way as a variable of the same type

◎assignments and expressions

◎scanf and printf

Intro to Arrays

★Using a for loop to initialize an array with a pattern: 0, 1, 0, 1, …

```
#define SIZE 100
…
int x[SIZE];
…
for (i = 0; i < SIZE; i++)
   x[i] = i % 2;
```

Intro to Arrays

★Using a for loop to initialize an array with values obtained with scanf

```
#define SIZE 100
…
int x[SIZE];
…
for (i = 0; i < SIZE; i++)
   scanf ("%d", &x[i]);
```

Intro to Arrays

★Using a for loop to output the values of an array

```
#define SIZE 100
…
int x[SIZE];
…
for (i = 0; i < SIZE; i++)
   printf ("%d\n", x[i]);
```

Intro to Arrays

★Careful with ranges!