



UNIVERSITY OF MASSACHUSETTS  
DARTMOUTH

# ECE160: Foundations of Computer Engineering I

## Lecture #13 – **Loops**

Instructor: Dr. Liudong Xing  
SENG-213C, [lxing@umassd.edu](mailto:lxing@umassd.edu)  
ECE Dept .



# Administrative Issues

- Homework #3 assigned today
  - Due 9am, Friday, March 3
- Lab #6 starts on Monday, Feb. 27
  - Due 5pm, Wednesday, March 1

# Review of Lecture #12

- Multi-way selection using
  - *switch statement*: can be used only when the selection condition can be reduced to an integral expression!
  - *if-else-if control structure*: no the above limitation

# Outline (Loops)

- Basic concepts and forms
- Three C loop statements
  - *while* loops
  - *do...while* loops
  - *for* loops
- *break/continue* statements

# Loops

- A **loop** is a group of instructions that the computer executes repeatedly while some condition stays true.
- Two basic forms of loops:
  - Counter-controlled repetition.
  - Event/sentinel-controlled repetition.

# Counter-Controlled Loops

- A **control variable** is used to count the number of repetitions.

```
#include <stdio.h>
void main(void)
{
    int counter; /* control variable*/
    counter=1;
    while (counter <= 23)
    {
        printf("The value of mycounter is:%d\n", counter);
        ++counter;
    }
}
```

# Event/Sentinel-Controlled Loops

- Used when we don't know in advance the number of repetitions
- The loop includes statements that obtain data every time the loop is executed.
- The loop ends when **the loop control expression** changes from **true** to **false**

# An Example of Sentinel/Event-Controlled Repetition

```
#include <stdio.h>
void main(void)
{
    int x=0;
    int sum = 0;
    printf("Enter your numbers to add. Enter <EOF> if you wish to stop \n");
    /* EOF is <ctrl+z>*/
    while(scanf("%d", &x) != EOF)
        sum+=x;
    printf(" I am out of the loop \n");
    printf("The total is %d\n", sum);
}
```



# Agenda

- ✓ Basic concepts and forms
- **Three C loop statements**
  - *while* loops
  - *do...while* loops
  - *for* loops
- *break/continue* statements

# The *while* Loop

- Syntax

```
while (expression)
{
    statement-1;
    statement-2;
    .....
    statement-n;
}
```

- A **pretest** loop: in each iteration, the loop control expression is tested first. If it's **true**, the loop body (statements between the braces) is executed. If it's **false**, the loop is terminated
- Braces are not required if the loop body consists of only one statement
- **No semicolon** is needed at the end of the while statement!

# Exercises (1)

- Assume *int b=1*; find error(s), if any, in the following while statements

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

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

# Exercises (2)

- Assume *int b=1;* find error(s), if any, in the following while statements

```
while (7)
    printf("hello\n");
```

```
while (3+1==7)
{
    printf("b=%d\n", b);
    b++;
}
```

# Operator Precedence (in descending order)

Postfix operators: ++, --, ..

Prefix operators: ++, --, ..

sizeof

Plus/minus signs: +,-

Logical NOT: !

Type cast: ()

Multiplicative operators: \*, /, %

Addition: +, -

Shift: << , >>

Relation: < , <=, >, >= ..

Equality operations: ==, !=

Bitwise/Boolean AND: &

Bitwise/Boolean XOR: ^

Bitwise/Boolean OR: |

Logical AND: &&

Logical OR: ||

Ternary conditional operator: ?:

Assignment: = , +=, -=, etc..

## Exercise (3)

- Write a program that reads 4 integers from the keyboard, compute their sum and prints it.

Using counter-controlled Loop!

## Exercise (4)

- Write a program that computes the sum for **any number** of integers entered from the keyboard.

Using event-controlled Loop!

# C Loop Statements

- ✓ *while* loops
- *do...while* loops
- *for* loops



# The *do...while* Loop

- Syntax

```
do
{
    statement-1;
    statement-2;
    .....
    statement-n;
} while (expression);
```

- A **post-test** loop: in each iteration, the loop body is executed. Then the loop control expression is tested. If it's true, a new iteration is started; otherwise, the loop terminates
- Braces are not required if the loop body consists of only one statement
- The loop body is executed at least once
- **Semicolon is needed** at the end of the *do...while* statement!!

# An Example

```
#include <stdio.h>
```

```
void main(void)
```

```
{
```

```
    int counter;
```

```
    counter=1;
```

```
    do
```

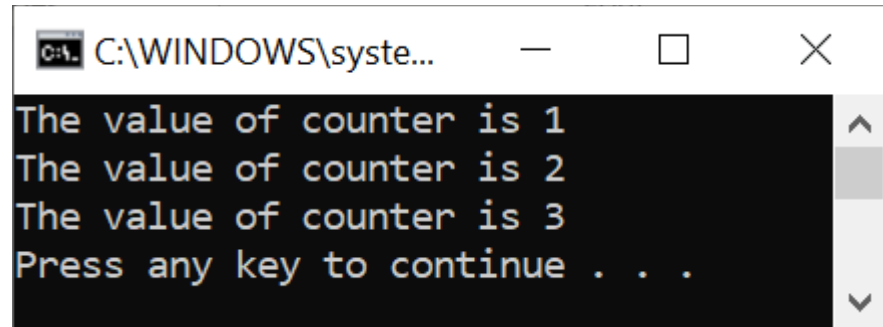
```
    {
```

```
        printf("The value of counter is %d\n", counter);
```

```
    } while(++counter <= 3);
```

```
}
```

**Not required!**



```
C:\WINDOWS\system...
The value of counter is 1
The value of counter is 2
The value of counter is 3
Press any key to continue . . .
```

# *while vs. do...while*

```
while (expression)
{
    statement-1;
    statement-2;
    .....
    statement-n;
}
```

```
do
{
    statement-1;
    statement-2;
    .....
    statement-n;
} while (expression);
```

- Pre-test: loop-continuation condition is tested before the loop.
- Post-test: loop-continuation condition is tested after the loop.

Braces are not required if the loop body consists of only one statement

# C Loop Statements

- ✓ *while* loops
- ✓ *do...while* loops
- *for* loops

# The for Loop

```
for(statement1; statement2; statement3)
{
    loop_body
}
```

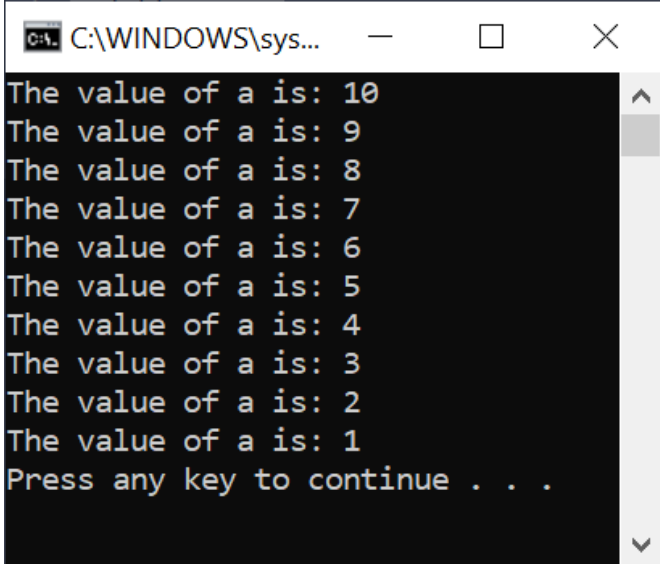
- statement1: contains the **initial value** of the control variable
- statement2: contains the **final value** of the control variable
- statement3: **increments/decrements** the control variable
- **Braces are not required if the loop body consists of only one statement**
- **Pre-test: loop-continuation condition (statement2) is tested before the loop.**
- Example:

```
for(counter=1; counter <= 15; counter++)
    printf("hello %d\n", counter);
```

# An Example

```
#include <stdio.h>

void main(void)
{
    int a;
    for(a=10; a>=1; a--)
        printf("The value of a is: %d\n",a);
}
```

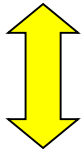


```
C:\WINDOWS\sys...
The value of a is: 10
The value of a is: 9
The value of a is: 8
The value of a is: 7
The value of a is: 6
The value of a is: 5
The value of a is: 4
The value of a is: 3
The value of a is: 2
The value of a is: 1
Press any key to continue . . .
```

In a for loop, the starting counter value can be larger than the ending counter value!

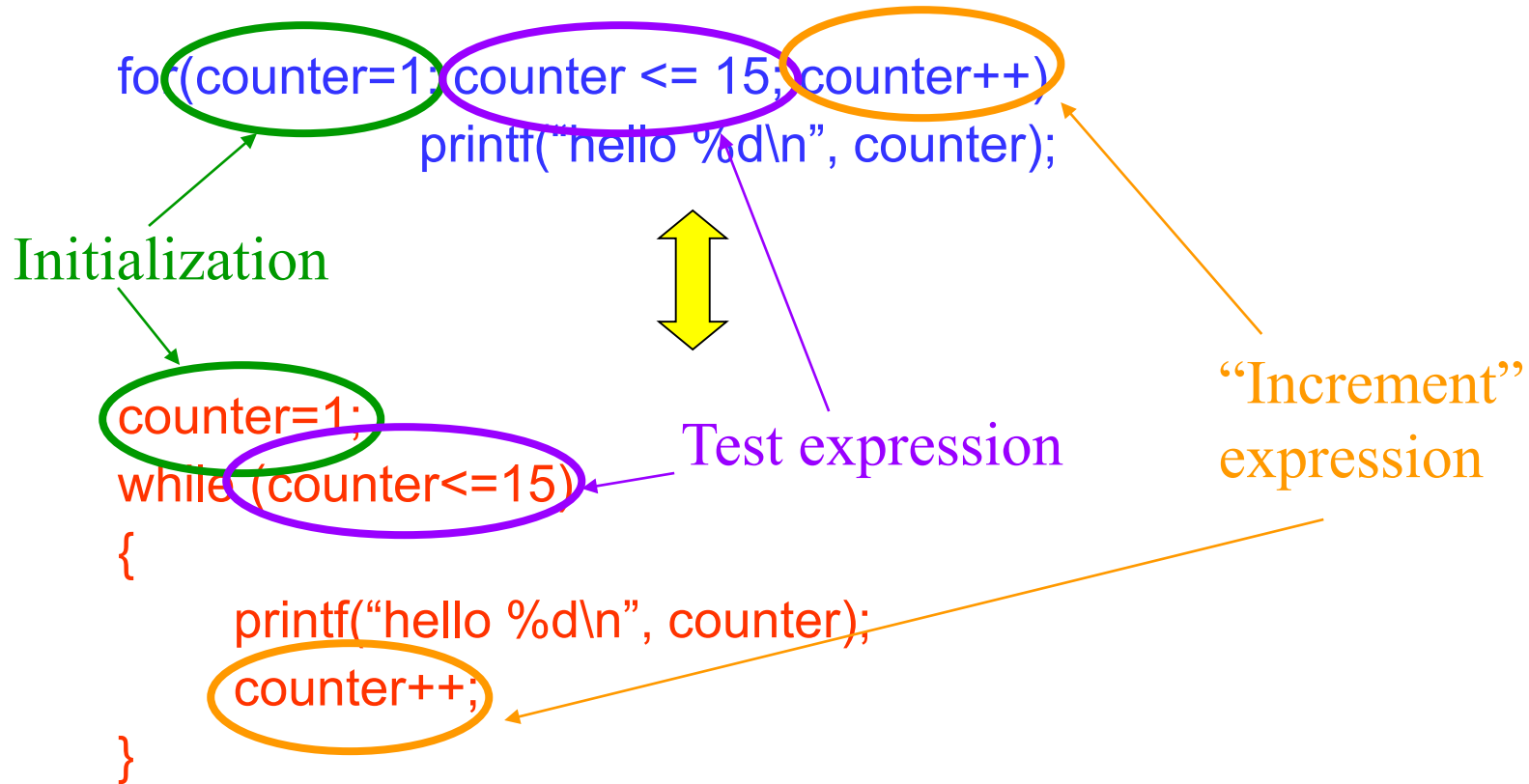
# Equivalent to a *while* Loop

```
for(counter=1; counter <= 15; counter++)  
    printf("hello %d\n", counter);
```



Equivalent while loop

# Equivalent to a *while* Loop





# Note on for loops!

- The for loop is used when your loop is to be executed a known number of times!
- The 3 expressions in the *for* structure are optional. The two semicolons are required.
  - If the counter variable is being initialized elsewhere in the program, then *statement1* can be omitted.
  - If *statement2* is omitted, then C assumes the testing condition is true and creates an infinite loop.
  - *Statement3* can be omitted if the counter variable is being incremented (or decremented) elsewhere in the program.

## Exercise (5)

Find error(s), if any, in the following for statements

- `for (day=1,day<3,day++) printf(“good morning\n”);`
- `for (day=3;day<=10;day++) ;`
- `for (day=7;day<3;day++) printf(“good morning\n”);`
- `for (day=7;day<7;day--) printf(“good morning\n”);`

# The Nested for Loop

- Loop(s) within a loop
- What's the output of this program?

```
#include <stdio.h>
void main(void)
{
    int a;
    int b;
    for(a =1; a <= 3; a++)
        {
            for (b=1; b<=4; b++)
                printf("%d", a);
            printf("\n");
        }
}
```

# Agenda

- Basic concepts and forms
- Three C loop statements
  - *while* loops
  - *do...while* loops
  - *for* loops
- **break/continue statements**

# break/continue

- The *break* and *continue* statements are used in loops to change the flow of control.
- *break* is used to escape from a loop (causes a loop to terminate).
- *continue* is used to skip the remaining statements in the body of a structure and skip to the next iteration.

# *break* example

```
#include "stdio.h"
```

```
void main(void)
```

```
{
```

```
    int a;
```

```
    for(a =1; a <= 7; a++)
```

```
    {
```

```
        if(a == 4)
```

```
            break;
```

```
        printf("%d\n", a);
```

```
    }
```

```
    printf("I got out of the loop at a==%d\n",a);
```

```
}
```

What is the output of the program?

For good programming style, *break* statements should be avoided!

# Note on *break*!


- The *break* statement terminates only the inner loop – the one you are currently in if you are in a series of **nested** loops!

The *break* statement takes you out of the inner *for* loop. The *while* loop is still alive!

```
while (condition1)
{
    .....
    /*some while processing*/

    for (s1;s2;s3)
    {
        ...
        if (condition2)
            break;
        ...
    } /*for loop ends here*/

    .....
    /*more while processing*/
} /*while loop ends here*/
```



## *continue* statement

- *continue* is used to skip the remaining statements in the body of a structure and skip to the next iteration.
- It does not terminate the loop but simply transfers to the testing expression in *while* and *do...while* statements and transfers to the updating expression *statement3* in a *for* loop.



# *continue* Example (1)

```
#include <stdio.h>
void main(void)
{
    int a;
    for(a =1; a <= 7; a++)
    {
        if (a == 4)
            continue;
        printf("%d\n",a);
    }
}
```

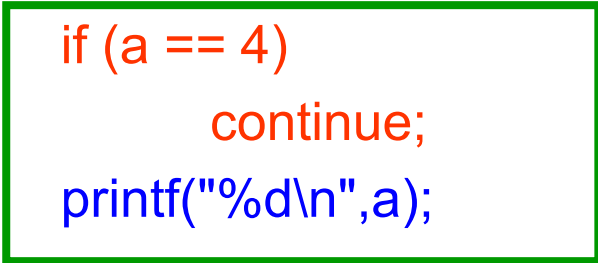
What is the output of the program?

# Note on *continue*

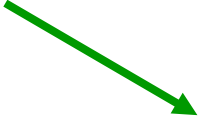
```
#include "stdafx.h"
void main(void)
{
    int a;
    for(a =1; a <= 7; a++)
    {
        if (a == 4)
            continue;
        printf("%d\n",a);
    }
}
```

For good programming style, *continue* should be avoided!

You can eliminate the need for *continue* by simply reversing the conditional test of *if*!



```
if (a == 4)
    continue;
```



```
if (a != 4)
    printf("%d\n", a);
```

# Good Programming Practices

- Use integer variables in controlling loops.
- Indent appropriately.
- Avoid using `break;` or `continue;` in the loops
- Do not use loop nesting more than 3 levels deep.

# Summary of Lectures #13

- Counter-controlled repetition vs. sentinel/event-controlled repetition
- Three C loop statements
  - *while* loops (pre-test)
  - *do...while* loops (post-test)
  - *for* loops (pre-test)
- *break/continue* statements can be used to change the flow of control in loops

# Things To Do

- Review Lecture Notes
- Run the programs in the exercises and examples

# Next Topic

- Functions