

**CONFIDENTIAL**

# **C Programming Introduction**

## **Week 7: Loops**

**Lecturer : Do Quoc Huy**  
**Dept of Computer Science**  
**Hanoi University of  
Technology**



# Topic of this week

- Loops
  - Class Lecture Review
    - The for Repetition Structure
    - Notes and Observations
  - Programming Exercises

# The for Repetition Structure

- Format when using **for** loops

**for** ( *initialization* ; *loopContinuationTest* ; *increment* )  
*statement*

Example:

```
for( int counter = 1; counter <= 10; counter++ )  
    printf( "%d\n", counter );
```

- Prints the integers from one to ten.

No  
semicolon  
after last  
expression

# The for Repetition Structure (II)

- **for** loops can usually be rewritten as **while** loops:

```
initialization;  
while ( loopContinuationTest ) {  
    statement  
    increment;  
}
```

- Initialization and increment
  - Can be comma-separated lists

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

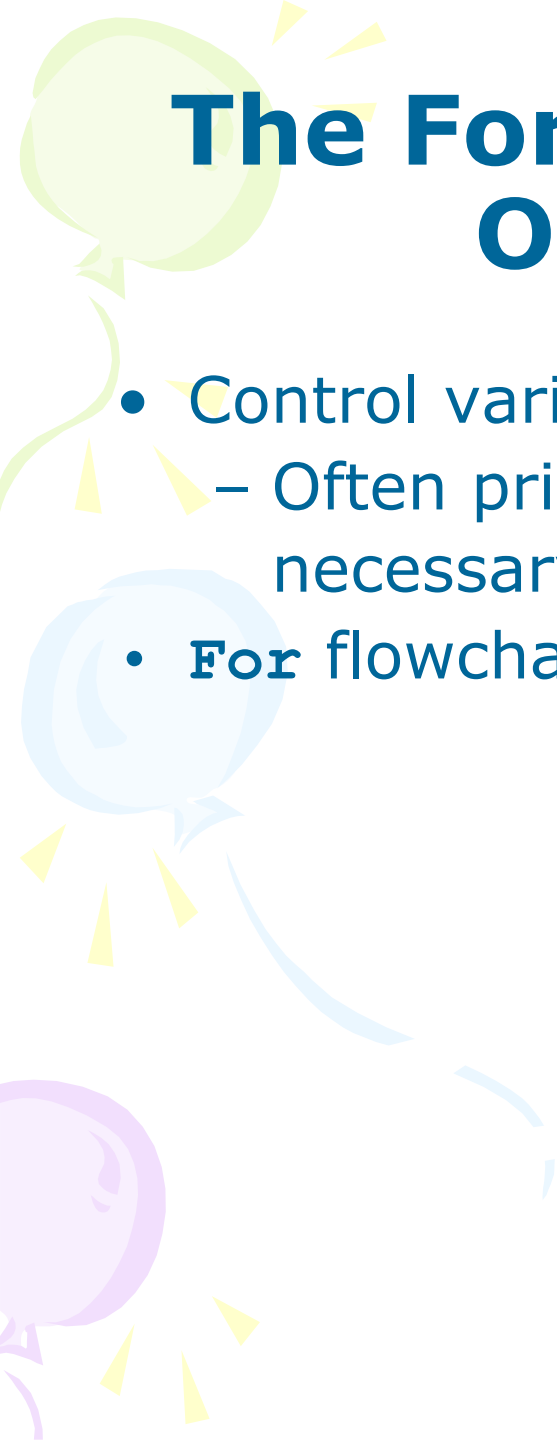
# The For Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If  $x = 2$  and  $y = 10$   

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to  

```
for ( j = 2; j <= 80; j += 5 )
```
- "Increment" may be negative (decrement)
- If loop continuation condition initially **false**
  - Body of **for** structure not performed
  - Control proceeds with statement after **for** structure



# The For Structure: Notes and Observations (II)

- Control variable
  - Often printed or used inside `for` body, but not necessary
- **For** flowcharted like `while`

# Example

- Example of **for**

```
for (i=1;i<=100;i++) {  
    x += i;  
    if ((x % i) == 0) { i--; }  
}
```

```
for (i=0, j=strlen(s)-1; i<j; i++,j--)  
    { c = s[i], s[i] = s[j], s[j] = c; }
```

```
char c;  
int count;  
for (count=0; (c=getchar()) != '\n'); count++)  
    { }  
printf("Number of characters is %d\n", count);
```



# Exercise 7.1

- Write a program that prints ten integers and their squares.

1	1
2	4
3	9
...	
10	100





# Exercise 7.2

- Write a program that prints out a triangle like:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

Three balloons (green, blue, and purple) are positioned on the left side of the slide, each with a string and small yellow triangular flags.

## Exercise 7.3

- Write a program that lists numbers which is greater than 27 from 1 to 100.



## Exercise 7.4

- Write a program that lists prime numbers which is smaller than 100.
- Use `math.h` library to use some mathematical functions: `sqrt`,...



## Exercise 7.5

- Alter the exercise 7.4 above by eliminating the even numbers to avoid calling sqrt function many times.



# Exercise 7.6

- Try the following program in your compiler.

```
/* Counting down to blast-off */  
#include <stdio.h>
```

```
int main(void)  
{  
    int time, start;  
  
    printf("Enter starting time (an integer) in seconds> ");  
    scanf("%d", &start);  
    printf("\nBegin countdown\n");  
  
    for (time = start; time > 0; time = time - 1)  
    {  
        printf("T - %d\n", time);  
    }  
  
    printf("Blast-off!\n");  
    return (0);  
}
```



## Exercise 7.7

- Write a program that converts temperatures from Celsius to Fahrenheit.
- Notice the conditions of the loops continuation and the way in which `#define` macros are used to set constant values.
- `fahrenheit = 1.8 * celsius + 32.0;`



## Exercise 7.8

- Sometimes we need to have loops within loops, this is called nested loops. This program demonstrates how this works. By running it you should see in what sequence the code is called.



# exercise7\_8.c

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

```
int main(void)
{
```

```
    int i, j;
```

```
    printf("      I   J\n");
```

```
    for (i = 1; i < 4; i = i + 1)
    /*
```

```
    {
```

```
        printf("Outer %6d\n", i);
```

```
        for (j = 0; j < i; j = j + 1)
        {
```

```
            printf("  Inner%9d\n", j);
```

```
        }
```

```
    }
```

```
    return (0);
```

```
}
```

```
/* loop control variables */
```

```
/* prints column labels */
```

```
/* heading of outer for loop
```

```
/* heading of inner loop */
```

```
/* end of inner loop */
```

```
/* end of outer loop */
```





# Exercise 7.9

- Write a program that uses *for* *structure* to calculate the value of  $n!$ .
- Some outputs:

## Results

Enter n: 4

4! = 24

## Results

Enter n: 0

0! = 1



# Exercise 7.10

- In mathematics, a **perfect number** is defined as a positive integer which is the sum of its proper positive divisors, that is, the sum of the positive divisors not including the number itself. E.g:  $6 = 1 + 2 + 3$
- Write a program that lists perfect numbers which is smaller than inputted N.