



for Loop Lesson 2 Outline

1. for Loop Lesson 2 Outline
2. for Loop Application
3. Factorial
4. Factorial Program #1
5. Factorial Program #2
6. for Loop With Implicit Increment
7. for Loop With Explicit Increment #1
8. for Loop With Explicit Increment #2
9. for Loop With Explicit Increment #3
10. for Loop with Negative Increment
11. for Loop with Decrement Example #1
12. for Loop with Decrement Example #2
13. for Loop with Decrement Example #3
14. for Loop with Named Constants
15. for Loop w/Named Constants Example #1
16. for Loop w/Named Constants Example #2
17. for Loop with Variables
18. for Loop with Variables Example #1
19. for Loop with Variables Example #2
20. for Loop with Expressions
21. for Loop with Expressions Example #1
22. for Loop with Expressions Example #2





for Loop Application

Suppose that there's a line of a dozen students waiting for tickets for the next OU-Texas football game.

How many different orders can they have in the line?

- The head of the line could be any student.
- The 2nd position in line could be any student except the student at the head of the line.
- The 3rd position in line could be any student except the student at the head of the line or the student in the 2nd position.
- And so on.





Factorial

Generalizing, we have that the number of different orders of the 12 students is:

$$12 \cdot 11 \cdot 10 \cdot \dots \cdot 2 \cdot 1$$

We can also express this in the other direction:

$$1 \cdot 2 \cdot 3 \cdot \dots \cdot 12$$

In fact, for any number of students n , we have that the number of orders is:

$$1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

This arithmetic expression is called *n factorial*, denoted *n!*

There are $n!$ *permutations* (orderings) of the n students.





Factorial Program #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int program_success_code = 0;
    int number_of_students;
    int permutations;
    int count;

    printf("How many students are in line for tickets?\n");
    scanf("%d", &number_of_students);
    permutations = 1;
    for (count = 1; count <= number_of_students; count++) {
        permutations = permutations * count;
    } /* for count */
    printf("There are %d different orders in which\n");
    printf("  the %d students can stand in line.\n",
        number_of_students);
    return program_success_code;
} /* main */
```





Factorial Program #2

```
% gcc -o permute permute.c
```

```
% permute
```

```
How many students are in line for tickets?
```

```
12
```

```
There are 479001600 different orders in which  
the 12 students can stand in line.
```



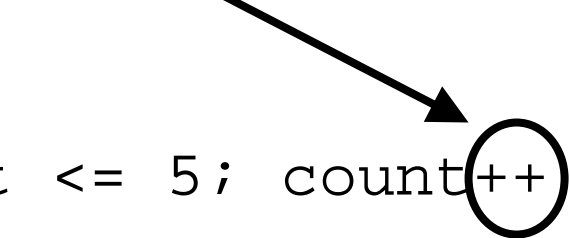
for Loop With Implicit Increment

The most common increment in a `for` loop is 1.

For convenience, therefore, we typically use the increment operator `++` in our loop change

statement. For example:

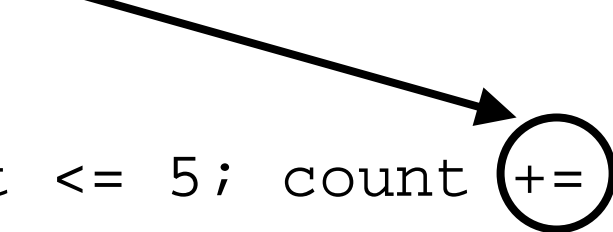
```
int product = 1;
int count;
for (count = 1; count <= 5; count++) {
    product *= count;
} /* for count */
```



for Loop With Explicit Increment #1

We could state the loop increment explicitly in the for statement, by using, for example, an addition assignment operator += :

```
int product = 1;
int count;
for (count = 1; count <= 5; count += 1) {
    product *= count;
} /* for count */
```



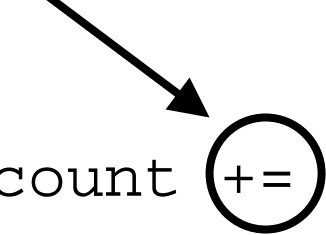
The above program fragment behaves **identically** to the one on the previous slide. Notice that both of the above loops have 5 iterations: count of 1, 2, 3, 4, 5.



for Loop With Explicit Increment #2

On the other hand, if the loop increment is not 1, then it **MUST** be explicitly stated, using, for example, an addition assignment operator += :

```
int product = 1;
int count;
for (count = 1; count <= 5; count += 2) {
    product *= count;
} /* for count */
```



Notice that the above loop has only 3 iterations:
count of 1, 3, 5.



for Loop With Explicit Increment #3

```
int product = 1;
int count;
for (count = 1; count <= 5; count += 2) {
    product *= count;
} /* for count */
```

The above program fragment behaves **identically** to:

```
int product = 1;
int count;
count = 1;          /* count == 1, product == 1 */
product *= count;  /* count == 1, product == 1 */
count += 2;        /* count == 3, product == 1 */
product *= count;  /* count == 3, product == 3 */
count += 2;        /* count == 5, product == 3 */
product *= count;  /* count == 5, product == 15 */
count += 2;        /* count == 7, product == 15 */
```





for Loop with Negative Increment

Sometimes, we want to loop backwards, from a high initial value to a low final value. To do this, we use a negative loop increment; that is, we use the decrement operator `--` :

```
count--
```





for Loop with Decrement Example #1

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

int main ()
{ /* main */
    const int input_digits          = 4;
    const int base                  = 10;
    const int program_success_code = 0;
    int base_power, input_value;
    int base_digit_value, output_digit;

    printf("Input an integer of no more ");
    printf("than %d digits:\n", input_digits);
    scanf("%d", &input_value);
```





for Loop with Decrement Example #2

```
for (base_power = input_digits - 1;
     base_power >= 0; base_power--) {
    base_digit_value = pow(base, base_power);
    if (input_value >= base_digit_value) {
        output_digit =
            input_value / base_digit_value;
        printf("%2d^%1d: %1d\n",
              base, base_power, output_digit);
        input_value =
            input_value -
            output_digit * base_digit_value;
    } /* if (input_value >= ...) */
} /* for base_power */
return program_success_code;
} /* main */
```





for Loop with Decrement Example #3

```
% gcc -o decimaldigits decimaldigits.c -lm
```

```
% decimaldigits
```

```
Input an integer of no more than 4 digits:
```

```
3984
```

```
103: 3
```

```
102: 9
```

```
101: 8
```

```
100: 4
```





for Loop with Named Constants

For the loop lower bound and upper bound, and the stride if there is one, we can use int named constants.



for Loop w/Named Constants Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int initial_value      = 1;
    const int final_value       = 20;
    const int stride             = 3;
    const int program_success_code = 0;
    int count, sum;

    sum = 0;
    for (count = initial_value;
        count <= final_value; count += stride) {
        sum = sum + count;
        printf("count = %d, sum = %d\n",
            count, sum);
    } /* for count */
    printf("After loop, count = %d, sum = %d.\n",
        count, sum);
    return program_success_code;
} /* main */
```



for Loop w/Named Constants Example #2

```
% gcc -o loopbndconsts loopbndconsts.c
% loopbndconsts
count = 1, sum = 1
count = 4, sum = 5
count = 7, sum = 12
count = 10, sum = 22
count = 13, sum = 35
count = 16, sum = 51
count = 19, sum = 70
After loop, count = 22, sum = 70.
```

In fact, we should use `int` named constants rather than `int` literal constants: it's much better programming practice, because it makes it much easier to change the loop bounds and the stride.





for Loop with Variables

For the loop lower bound, loop upper bound and loop stride, we can use int variables.





for Loop with Variables Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int program_success_code = 0;
    int initial_value, final_value, stride;
    int count, sum;

    printf("What are the initial, final and ");
    printf("stride values?\n");
    scanf("%d %d %d",
          &initial_value, &final_value, &stride);
    sum = 0;
    for (count = initial_value;
         count <= final_value; count += stride) {
        sum = sum + count;
        printf("count = %d, sum = %d\n", count, sum);
    } /* for count */
    printf("After loop, count = %d, sum = %d.\n",
          count, sum);
    return program_success_code;
} /* main */
```





for Loop with Variables Example #2

```
% gcc -o loopbndvars loopbndvars.c
```

```
% loopbndvars
```

What are the initial, final and stride values?

```
1 7 2
```

```
count = 1, sum = 1
```

```
count = 3, sum = 4
```

```
count = 5, sum = 9
```

```
count = 7, sum = 16
```

```
After the loop, count = 9, sum = 16.
```





for Loop with Expressions

If we don't happen to have a variable handy that represents one of the loop bounds or the stride, then we can use an expression.



for Loop with Expressions Example #1

```
#include <stdio.h>
int main ()
{ /* main */
    const int program_success_code = 0;
    int initial_value, final_value, stride;
    int count, sum;

    printf("What are the initial, final and ");
    printf("multiplier values?\n");
    scanf("%d %d %d",
        &initial_value, &final_value, &multiplier);
    sum = 0;
    for (count = initial_value * multiplier;
        count <= final_value * multiplier;
        count += multiplier - 1) {
        sum = sum + count;
        printf("count = %d, sum = %d\n", count, sum);
    } /* for count */
    printf("After loop, count = %d, sum = %d.\n",
        count, sum);
    return program_success_code;
} /* main */
```



for Loop with Expressions Example #2

```
% gcc -o loopbndexprs loopbndexprs.c
```

```
% loopbndexprs
```

```
What are the initial, final and multiplier values?
```

```
1 7 2
```

```
count = 2, sum = 2
```

```
count = 3, sum = 5
```

```
count = 4, sum = 9
```

```
count = 5, sum = 14
```

```
count = 6, sum = 20
```

```
count = 7, sum = 27
```

```
count = 8, sum = 35
```

```
count = 9, sum = 44
```

```
count = 10, sum = 54
```

```
count = 11, sum = 65
```

```
count = 12, sum = 77
```

```
count = 13, sum = 90
```

```
count = 14, sum = 104
```

```
After the loop, count = 15, sum = 104.
```

