



SoftUni Team
Technical Trainers
Software University
<http://softuni.bg>



Repeating Code Multiple Times

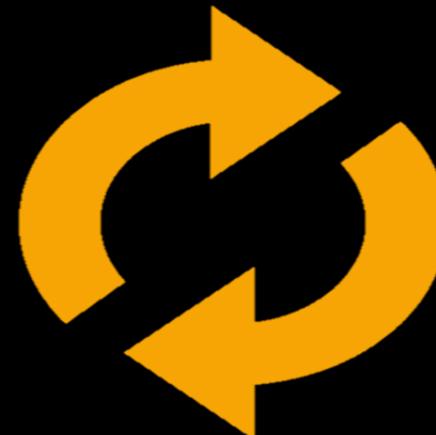


C Loops



Table of Contents

1. What is a Loop?
2. Loops in C
 - **while** loops
 - **do ... while** loops
 - **for** loops
3. Special loop operators
 - **break**, **continue**, **goto**
4. Nested loops



Loop: Definition

- A **loop** is a control statement that repeats the execution of a block of statements

```
while (condition)
{
    statements;
}
```



- May execute a code block fixed number of times
- May execute a code block while given condition holds
- Loops that never end are called **infinite loops**

Using `while(...)` Loop

**Repeating a Statement While
Certain Condition Holds**

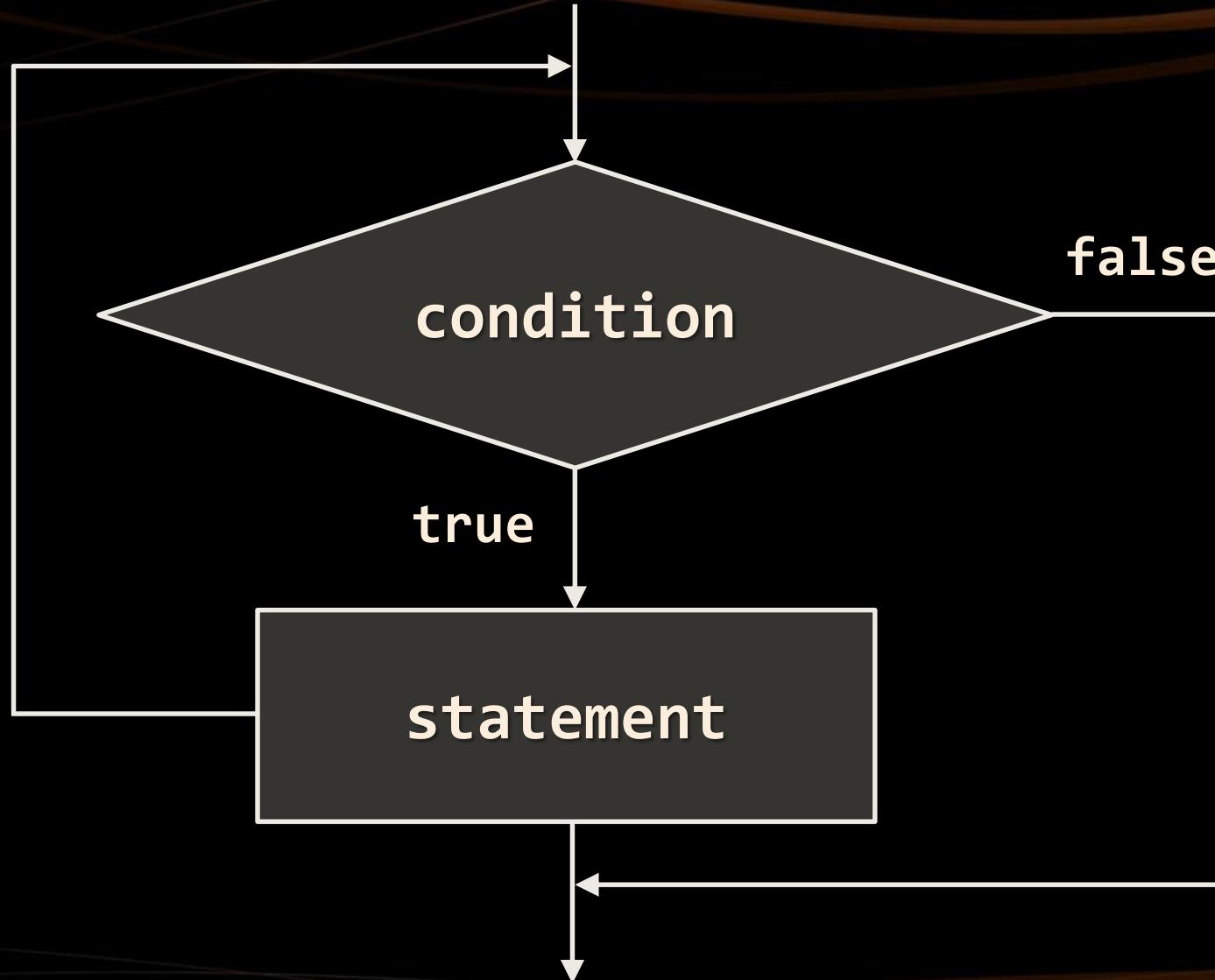
How To Use While Loop?

- The simplest and most frequently used loop

```
while (condition)
{
    statements;
}
```

- The repeat condition
 - Returns a integer result of **0** or **1**
 - Also called loop condition

While Loop: How It Works?

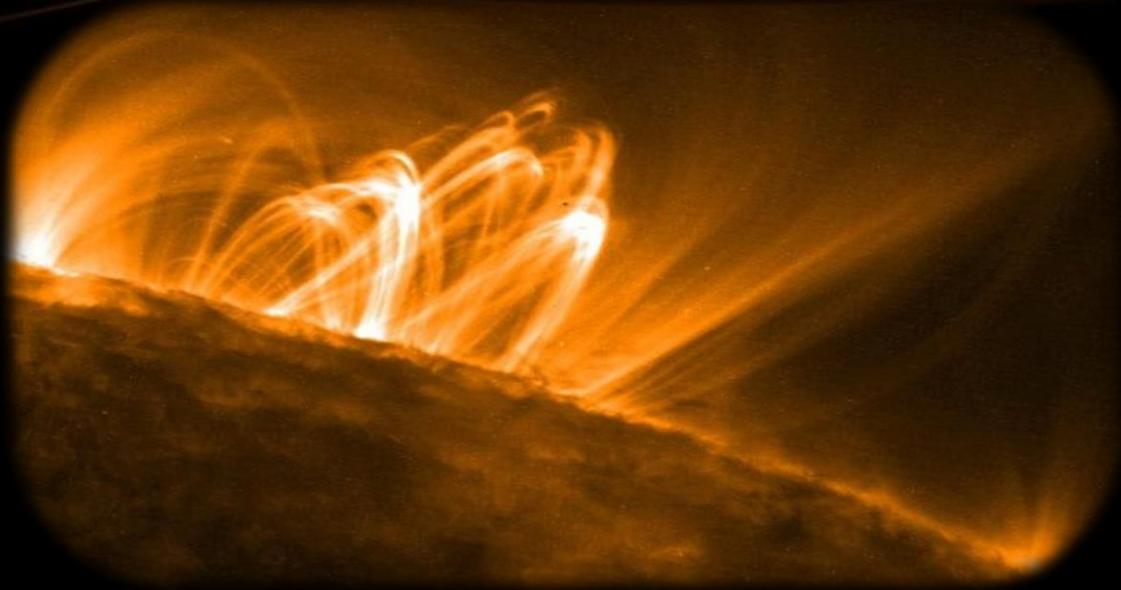


While Loop – Example

```
int counter = 0;  
while (counter < 10)  
{  
    printf("Number : %d\n", counter);  
    counter++;  
}
```

Loops

```
Number : 0  
Number : 1  
Number : 2  
Number : 3  
Number : 4  
Number : 5  
Number : 6  
Number : 7  
Number : 8  
Number : 9  
  
Process returned 0 (0x0)    execution time : 0.001 s  
Press ENTER to continue.
```



while(...) Loop

Examples

Sum 1..N – Example

- Calculate and print the sum of the first N natural numbers

```
int n;  
printf("n = ");  
scanf("%d", &n);  
int number = 1, sum = 1;  
printf("The sum 1");  
while (number < n)  
{  
    number++;  
    sum += number ;  
    printf("+%d", number);  
}  
printf(" = %d\n", sum);
```

Calculating Sum 1..N

Live Demo



```
C:\WINDOWS\system32\cmd.exe
n = 5
The sum 1+2+3+4+5 = 15
Press any key to continue . . .
```

Prime Number Check – Example

```
printf("Enter a positive integer number: ");
unsigned int number;
scanf("%du", &number);
unsigned int divider = 2;
unsigned int maxDivider = (unsigned int) sqrt(number);
int isPrime = 1;
while (isPrime && (divider <= maxDivider))
{
    if ((number % divider) == 0)
    {
        isPrime = 0;
    }
    divider++;
}
printf("Prime? %s", isPrime ? "yes" : "no");
```



Checking Whether a Number Is Prime

Live Demo

X	2	3	X	5	X	7	X	X	X
11	X	13	X	X	X	17	X	19	20
X	X	23	X	X	X	X	X	29	30
31	X	X	X	X	X	37	X	X	38
41	X	43	X	X	X	47	X	X	X
X	X	53	X	X	X	X	X	59	60
61	X	X	X	X	X	67	X	X	68
71	X	73	X	X	X	X	X	79	80
X	X	83	X	X	X	X	X	89	90
X	X	93	X	X	X	X	X	97	98



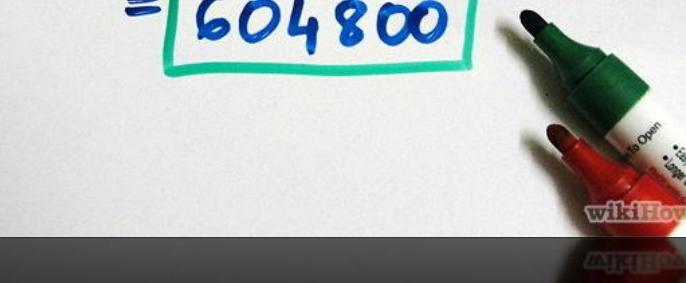
Using the **break** Operator

- The **break** operator exits the inner-most loop

```
int n;  
scanf("%d", &n);  
  
// Calculate n! = 1 * 2 * ... * n  
int result = 1;  
while (1)  
{  
    if (n == 1)  
        break;  
    result *= n;  
    n--;  
}  
printf("n! = %d", result);
```



$$\begin{aligned}5! &= 5 \times 4 \times 3 \times 2 \times 1 \\&= 120\end{aligned}$$
$$\begin{aligned}7! &= 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \\&= 5040\end{aligned}$$
$$\begin{aligned}5! \times 7! &= 120 \times 5040 \\&= 604800\end{aligned}$$



Calculating Factorial

Live Demo

```
do { ... }  
while (...)  
Loop
```



Using Do-While Loop

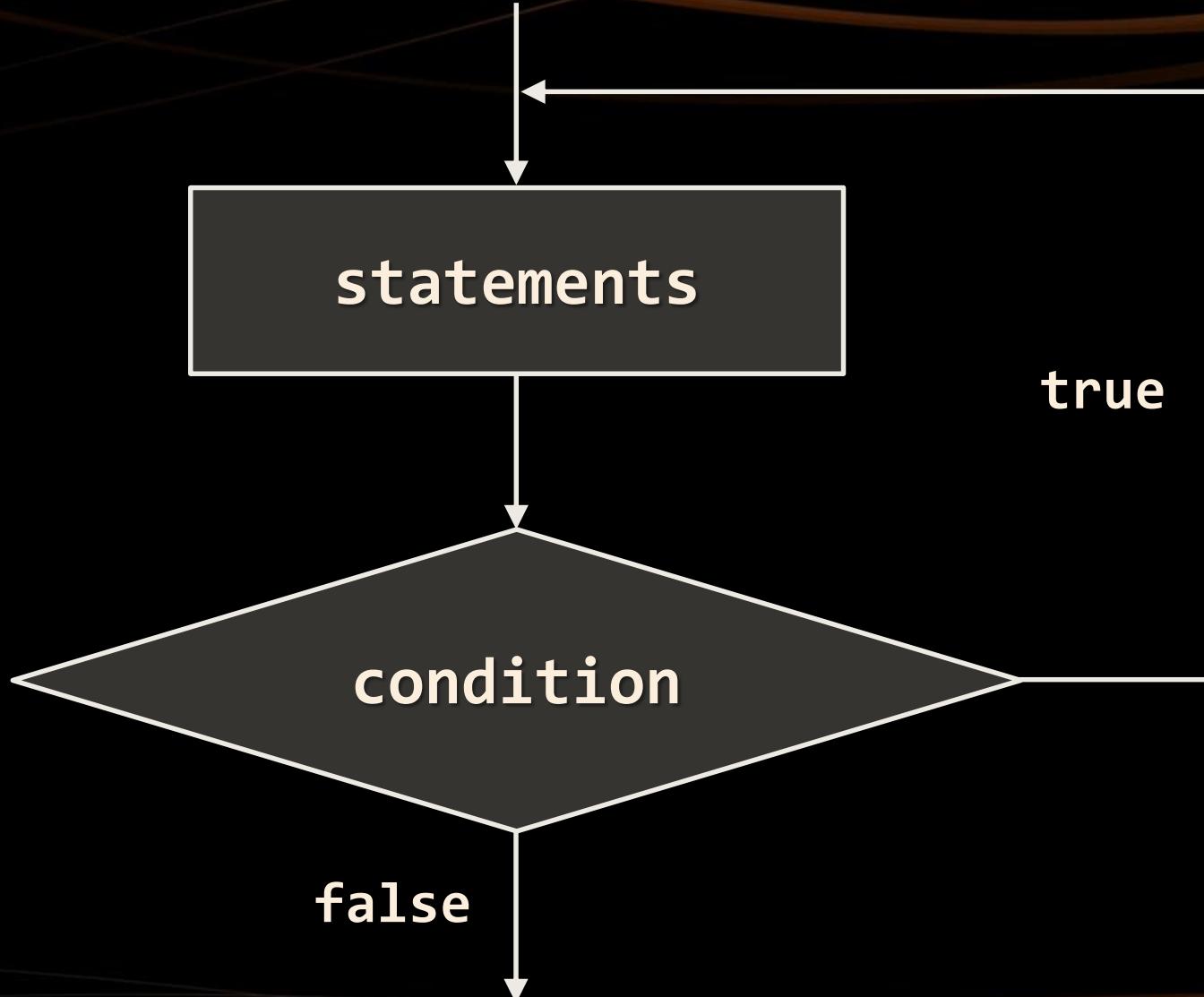
- Another classical loop structure is:

```
do
{
    statements;
}
while (condition);
```



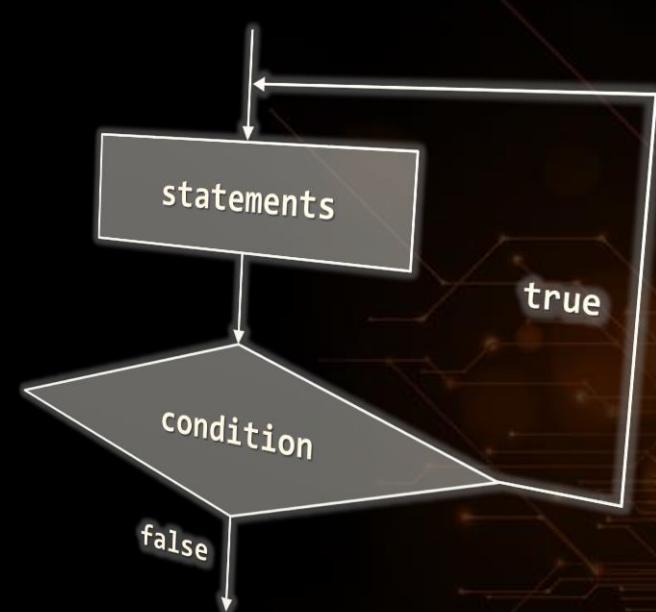
- The block of statements is repeated
 - While the boolean loop condition holds
- The loop is executed at least once

Do-While Statement: How It Works?



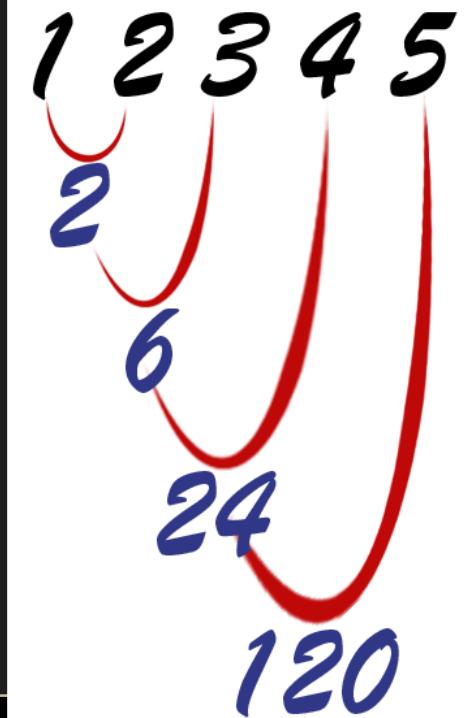
```
do { ... }  
while (...)
```

Examples



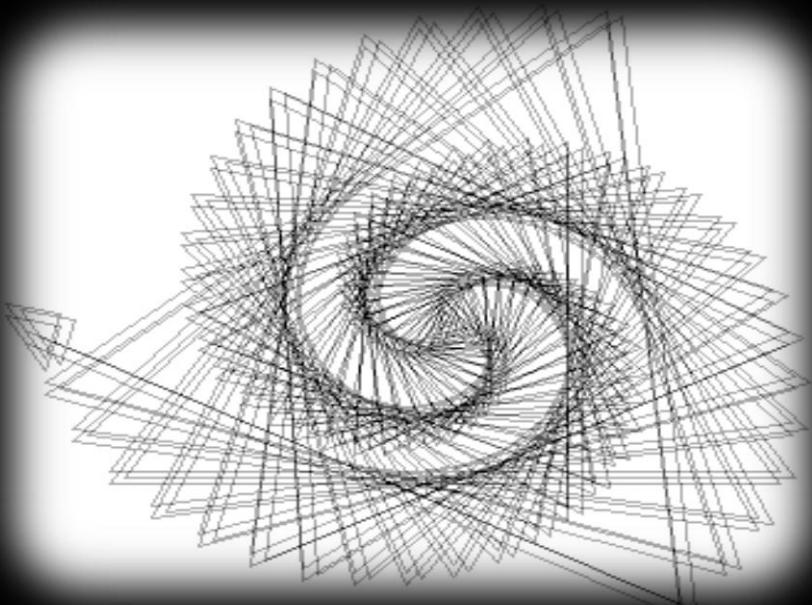
Calculating N Factorial – Example

```
int n;  
scanf("%d", &n);  
int factorial = 1;  
do  
{  
    factorial *= n;  
    n--;  
}  
while (n > 0);  
  
printf("n! = %d", factorial);
```



Factorial (do ... while)

Live Demo



Product of Integers [N..M] – Example

- Calculating the product of all integers in the interval [n..m]:

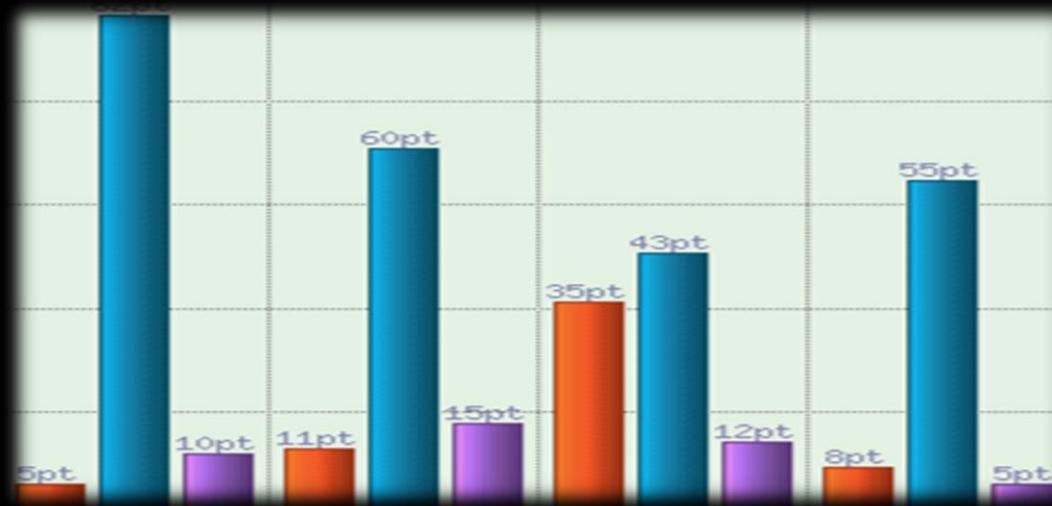
```
int n, m;
scanf("%d %d", &n, &m);
int number = n;
long product = 1;

do
{
    product *= number;
    number++;
}
while (number <= m);

printf("product[%d..%d] = %ld", n, m, product);
```

Product of the Integers in the Interval [n..m]

Live Demo



for Loops



For Loops

- The typical **for** loop syntax is:

```
for (initialization; test; update)
{
    statements;
}
```

- Consists of
 - Initialization statement
 - Boolean **test** expression
 - Update statement
 - Loop **body** block

The Initialization Expression

```
for (int number = 0; ...; ...)  
{  
    // Can use number here  
}  
// Cannot use number here (out of scope)
```

- The initialization expression
 - Executed once, just before the loop is entered
 - Like it is out of the loop, just before it
- Typically used to declare a counter variable

The Test Expression

```
for (int number = 0; number < 10; ...)  
{  
    // Can use number here  
}  
// Cannot use number here (out of scope)
```

- The **test expression** is evaluated before each loop iteration
 - If **true**, the loop body is executed
 - If **false**, the loop finishes (and the loop body is skipped)
- Used as a **loop condition**

The Update Expression

```
for (int number = 0; number < 10; number++)
{
    // Can use number here
}
// Cannot use number here (out of scope)
```

- The update expression
 - Executed at each iteration after the body of the loop is finished
 - Typically used to update the loop counter
 - Can update multiple variables

for-loop Variants

C89 Standard

```
int i;  
for (i = 0; i < 10; i++)  
{  
    // ...  
}
```

C99 Standard

```
for (int i = 0; i < 10; i++)  
{  
    // ...  
}
```

- To use the C99 standard, add the `-std=c99` compiler flag, e.g.:

```
gcc -std=c99 main
```

for Loop

Examples



Simple for Loop – Example

- A simple **for**-loop to print the numbers 0...9:

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

- A simple **for**-loop to calculate n!:

```
long factorial = 1;
for (int i = 1; i <= n; i++)
{
    factorial *= i;
}
```

Complex for Loop – Example

- A complex **for**-loop could have several counter variables:

```
for (int i = 1, sum = 1; i <= 128; i = i * 2, sum += i)
{
    printf("i=%d, sum=%d", i, sum);
}
```

- Result:

```
i=1, sum=1
i=2, sum=3
i=4, sum=7
i=8, sum=15
...
...
```



For Loops

Live Demo

N^M – Example

- Calculating **n** to power **m** (denoted as **n^m**):

```
static void Main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    long result = 1;
    for (int i = 0; i < m; i++)
    {
        result *= n;
    }
    printf("n^m = %d", result);
}
```

Calculating N^M

Live Demo



Using the `continue` Operator

- `continue` bypasses the iteration of the inner-most loop
- Example: sum all odd numbers in [1...n], not divisors of 7:

```
int n, sum = 0;
scanf("%d", &n);
for (int i = 1; i <= n; i += 2)
{
    if (i % 7 == 0)
    {
        continue;
    }
    sum += i;
}
printf("sum = %d", sum);
```

Using the continue Operator

Live Demo



Nested Loops

Using a Loop Inside a Loop



What Is Nested Loop?

- A composition of loops is called a nested loop
 - A loop inside another loop

```
for (initialization; test; update)
{
    for (initialization; test; update)
    {
        statements;
    }
    ...
}
```

Nested Loops

Examples



Triangle – Example

- Print the following triangle of numbers:

```
1
1 2
...
1 2 3 ... n
```

```
int n;
scanf("%d", &n);
for (int row = 1; row <= n; row++)
{
    for (int column = 1; column <= row; column++)
    {
        printf("%d ", column);
    }
    printf("\n");
}
```

Triangle of Numbers

Live Demo



Primes in the Range [N ... M] – Example

```
int n, m;
scanf("%d %d", &n, &m);
for (int number = n; number <= m; number++)
{
    int prime = 1;
    int divider = 2;
    int maxDivider = (int) sqrt(number);
    while (divider <= maxDivider)
    {
        if (number % divider == 0)
        {
            prime = 0;
            break;
        }
        divider++;
    }
    if (prime)
        printf("%d ", number);
}
```



Primes in the Range [n, m]

Live Demo



C Jump Statements

- Jump statements are:
 - **break, continue, goto**
- How **continue** works?
 - In **while** and **do-while** loops jumps to the test expression
 - In **for** loops jumps to the update expression
- To exit the most-inner loop use **break**
- To exit from an outer loop use **goto** with a label
 - Note: avoid using **goto!** (it is considered harmful)

C Jump Statements – Example

```
int outerCounter = 0;
for (int outer = 0; outer < 10; outer++)
{
    for (int inner = 0; inner < 10; inner++)
    {
        if (inner % 3 == 0)
            continue; _____↑
        if (outer == 7)
            break; _____↑
        if (inner + outer > 9)
            goto breakOut;
    }
    outerCounter++; ←
breakOut: ←
```

Label

Loops: More Examples



Nested Loops – Examples

- Print all four digit numbers in format **ABCD** such that **A+B = C+D** (known as happy numbers)

```
for (int a = 1; a <= 9; a++)  
    for (int b = 0; b <= 9; b++)  
        for (int c = 0; c <= 9; c++)  
            for (int d = 0; d <= 9; d++)  
                if ((a + b) == (c + d))  
                    printf("%d%d%d%d", a, b, c, d);
```

Can you improve this algorithm to use only 3 nested loops?

Happy Numbers

Live Demo

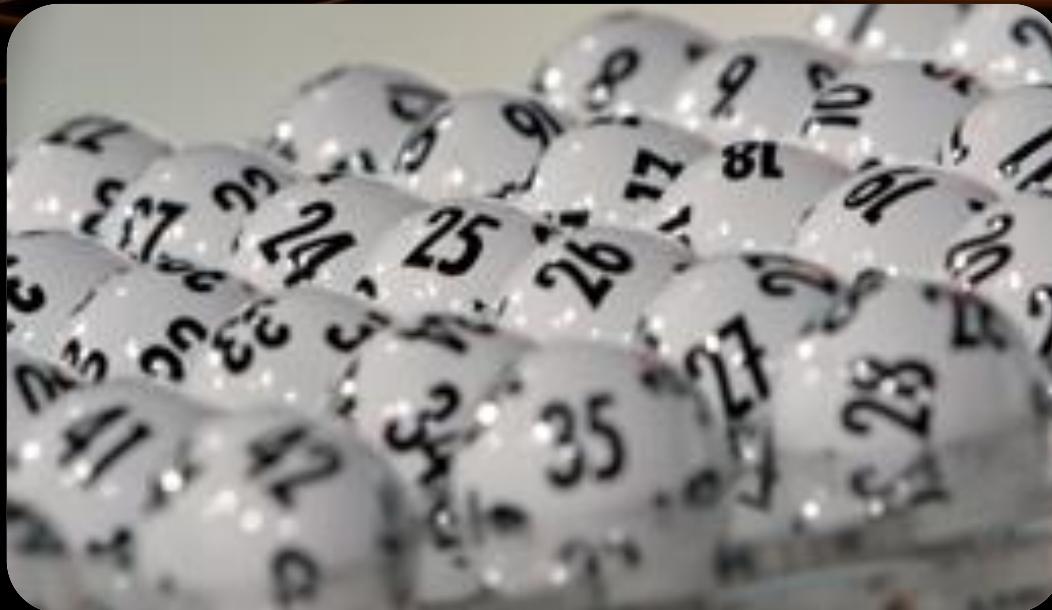


Nested Loops – Examples

- Print all combinations from TOTO 6/49 lottery

```
int i1, i2, i3, i4, i5, i6;
for (i1 = 1; i1 <= 44; i1++)
    for (i2 = i1 + 1; i2 <= 45; i2++)
        for (i3 = i2 + 1; i3 <= 46; i3++)
            for (i4 = i3 + 1; i4 <= 47; i4++)
                for (i5 = i4 + 1; i5 <= 48; i5++)
                    for (i6 = i5 + 1; i6 <= 49; i6++)
                        printf("%d %d %d %d %d %d",
                               i1, i2, i3, i4, i5, i6);
```

Warning: the execution of this code could take a very long time.



TOTO 6/49

Live Demo

Summary

- C supports four types of loops:
 - **while** loops
 - **do-while** loops
 - **for** loops
- Nested loops are used to implement more complex logic
- The operators **continue**, **break** & **goto** can change the default loop execution behavior



C Programming – Loops



Questions?

SUPERHOSTING.BG



License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license



- Attribution: this work may contain portions from
 - "Programming Basics" course by Software University under CC-BY-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University @ YouTube
 - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg

