

# Chapter 2, Flow Control

Programming Concepts in Scientific Computing  
EPFL, Master class

September 27, 2023

## A single if Statement

```
if (p > q) {  
    /*  
    Statement1;  
    Statement2;  
    */  
}
```

## A single if Statement

```
if (p > q) {  
    /*  
    Statement1;  
    Statement2;  
    */  
}
```

```
if (p > q)  
    Statement();
```

## Example

```
double x = -2.0;
```

```
if (x < 0.0) {  
    x = 0.0;  
}
```

## if-else statement

```
if (i > 0) {  
    y = 2.0;  
} else {  
    // When i <= 0  
    y = 10.0;  
}
```

## Multiple if statements

```
if (i > 100) {  
    y = 2.0;  
} else if (i < 0) {  
    y = 10.0;  
} else {  
    // When 0 <= i <= 100  
    y = 5.0;  
}
```

## Nested if statements

```
if (x > z) {  
    if (p > q) {  
        // Both conditions have been met  
        y = 10.0;  
    }  
}
```

# The switch statement

```
int i;
switch (i) {
case 1:
    std::cout << "i = 1\n";
case 20:
    // The following line is executed also in case i == 1!
    std::cout << "i = 1 or i = 20\n";
    break;
case 30:
    std::cout << "i = 30\n";
    break;
default:
    std::cout << "i is not 1, 20 nor 30\n";
}
```



## Logical conditions

```
bool flag = true;

if (flag) {
    std::cout << "This will be printed\n";
} else {
    // flag is false
    std::cout << "This won't be printed\n";
}
```

# Logical and Relational Operators

Logical Condition	Operator
AND	&&
OR	
NOT	!

```
if ((x > z) && (p > q)) {  
    // Both conditions have been met  
    y = 10.0;  
}
```

# Logical and Relational Operators

Logical Condition	Operator
AND	&&
OR	
NOT	!

```
if (!flag) {  
    // !flag is true when flag is false  
    i += 2;  
}
```

# Logical and Relational Operators

Relation	Operator
Equal to	== (note that it is not '=')
Not equal to	!=
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=

```
if ((y > q) || (i != 1)) {  
    // One or both conditions have been met  
    y = 10.0;  
}  
else {  
    // Neither condition has been met:  
    // y<=q and i==1  
    y = -10.0;  
}
```

# The while statement

```
double x = 10.0;
int count = 0;
while (x > 1.0) {
    // This loop will execute while x > 1, so if the
    // value of x does not decrease then it will not
    // terminate.

    x *= 0.5;
    std::cout << "x = " << x << ", count = " << count << "\n";
    count++;
    std::cout << "Reached bottom of while loop\n";
}
// Here we know the guard (x > 1.0) has broken.
// This means that after the loop, x <= 1.0
std::cout << "x = " << x << ", count = " << count << "\n";
```

# The do-while statement

```
double x = .8;
int count = 0;

do {
    x *= 0.5;
    std::cout << "x = " << x << ", count = " << count << "\n";
    count++;
    std::cout << "Reached bottom of while loop\n";
} while (x > 1.0);

std::cout << "count = " << count << "\n";
```

## Loops using the for statement

```
for (int i = 0; i < 5; i++) {  
    for (int j = 5; j > i; j--) {  
        std::cout << "i = " << i << " j = " << j << "\n";  
    }  
}
```

## Example: Calculating the scalar product of two vectors

We want to compute the scalar product between the two vectors

$$\mathbf{u} = \begin{pmatrix} 0.5 \\ -2.3 \end{pmatrix} \quad \mathbf{v} = \begin{pmatrix} 34.2 \\ 0.015 \end{pmatrix} \quad s = \mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^2 u_i v_i$$



## Example: Calculating the scalar product of two vectors

We want to compute the scalar product between the two vectors

$$\mathbf{u} = \begin{pmatrix} 0.5 \\ -2.3 \end{pmatrix} \quad \mathbf{v} = \begin{pmatrix} 34.2 \\ 0.015 \end{pmatrix} \quad s = \mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^2 u_i v_i$$

```
double vector1[2], vector2[2];
```

```
// The indices of an array start at 0, not at 1!
```

```
vector1[0] = 0.5;
```

```
vector1[1] = -2.3;
```

```
vector2[0] = 34.2;
```

```
vector2[1] = 0.015;
```

```
double scalar_product = 0.0;
```

```
for (int i = 0; i < 2; i++) {  
    scalar_product += vector1[i] * vector2[i];  
}
```

## Tips

```
double x = 2.0;
for (int i=0; i<5; i++);
{
    x *= 2.0;
    std::cout << "x = " << x << "\n";
}
```

## Tips

```
double x = 2.0;
for (int i=0; i<5; i++);
{
    x *= 2.0;
    std::cout << "x = " << x << "\n";
}
```

### Missing for-loop

```
double x = 2.0;
for (int i = 0; i < 5; i++)
;
{
    x *= 2.0;
    std::cout << "x = " << x << "\n";
}
```

```
int x;
```

```
x == 2 + 2;
```

```
// This erroneous line has no effect
```

```
int x;
```

```
x == 2 + 2;
```

```
// This erroneous line has no effect
```

```
// After testing x against the value 4, the
```

```
// answer is discarded.
```

```
x = 4; // This is correct
```

```
int x;  
if (x = 4) {  
    x = 6;  
}
```

```
int x;  
if (x = 4) {  
    x = 6;  
}
```

$x = 4$ ; will alter the value of  $x$

# Tips

```
double max = 0.0;
int count = 0;
double positive_numbers[4] = {1.0, 5.65, 42.0, 0.01};

while (count < 4) {
    if (positive_numbers[count] > max) {
        max = positive_numbers[count];
    }
}
```



# Tips

- ▶ When comparing two floating point numbers for equality, you have to consider that there can be round-off errors.

# Tips

- ▶ When comparing two floating point numbers for equality, you have to consider that there can be round-off errors.
- ▶ Introduce a small tolerance.

# Tips

- ▶ When comparing two floating point numbers for equality, you have to consider that there can be round-off errors.
- ▶ Introduce a small tolerance.

```
double p, q;  
double tolerance = 1.0e-8;  
int k;  
  
if (std::fabs(p - q) < tolerance) {  
    k = 0;  
}
```

# Flow control

## Take away message

- ▶ **if/else**: simple condition
- ▶ **switch/case**: case execution (beware the **break**)
- ▶ **(do-)while** loop: conditional loop
- ▶ **for** loop: conditional loop, together with init and increment (preferred for scientific work)
- ▶ **floating point numbers**: Use threshold for conditions on float/double numbers