

# Einführung in C - Introduction to C

## 4. Control flow

Prof. Dr. Eckhard Kruse

DHBW Mannheim

Every C **statement** is terminated by a semicolon: `statement;`

Multiple statements can be grouped together with curly brackets. This is called a **block**. Syntactically a block is treated like a single statement (e.g. in `if else` constructs):

```
{  
    statement1;  
    statement2;  
    statement3;  
}
```

Note: Behind the closing bracket of a block there is no semicolon.

For improved readability, statements within a block should be indented.

# Control flow

**Statements** are executed sequentially one after the other:

```
statement1;  
statement2;  
statement3;
```



How can this sequence be influenced?

- Decide about execution / non-execution of statements dynamically, i.e. based on results obtained during runtime.
- Execute the same statements more than once (without copying code), a number of times decided during program execution.
- Generalize/reuse by applying the same statements on different input values

```
if ... else  
switch ... case
```

```
while  
do ... while  
for
```

```
function(...)
```

```
if( expression )
    statement1;
else
    statement2;
```

If *expression* is not zero *statement1* is executed otherwise *statement2* is executed. The **else** *statement2*; part is optional and can be omitted. It is possible to have sequences of **if-else** constructs.

```
if( x<0 )
{
    x=-x;
    ...
}
```

```
if( i )
    printf("i is not 0");
```

```
if( x>0 )
    printf("positive");
else if( x<0 )
    printf("negative");
else
    printf("null");
```

```
switch( expression )
{
    case constant1: statement1; break;
    case constant2: statement2; break;
    ...
    default: default-statement; break;
}
```

If *expression* is equal to one of the **case constants**, the according *statement* is executed. Otherwise the *statement* behind **default** is executed. This is optional.

With **break**; the **switch** block is left. If it is omitted, also the subsequent case commands are executed (which usually is not wanted).

# if → switch: Asciiburger

Exercise?

`switch-case.c`

Code snippet  
401

### switch – case

Have a look at the previous programs  
(e.g. devowelizer).

Replace if-else commands with switch-case.

```
while( expression )  
    statement;
```

The *expression* is evaluated. If it is not 0, *statement* is executed. This cycle is repeated until *expression* becomes 0.

```
int completed=0;  
while(!completed)  
{  
    do_something();  
    completed = ...  
}
```



```
do  
    statement;  
while( expression );
```

*statement* is executed. Then the *expression* is evaluated. If it is not 0, the cycle is repeated.

```
int completed;  
do  
{  
    do_something();  
    completed = ...  
} while(!completed);
```

What is the major  
difference between  
**while** and **do-while**?

```
for ( init-statement; expression; reinit-statement )  
    for-statement;
```

First and only once, *init-statement* is executed.  
Then *expression* is evaluated. If it is not 0, *for-statement* and then *reinit-statement* are executed. The cycle repeats by again evaluating *expression*, it ends when *expression* becomes zero.

```
int i;  
for(i=0; i<100; i=i+1)  
    do_something();
```

# Convenient abbreviations

`i = i + 1;`      `i++;`   or   `++i;`

`i = i - 1;`      `i--;`   or   `--i;`

As a counter in a for-loop `i++` and `++i` work the same.

`i = i + 5;`    $\rightarrow$    `i+=5;`

`i = i * 3;`    $\rightarrow$    `i*=3;`

etc. (works for most operators)

Difference:

`c=0; i=1; c=i++;`    $\rightarrow$    `c=1, i=2`

`c=0; i=1; c=++i;`    $\rightarrow$    `c=2, i=2`

# Break and continue in loops

## Definition

The `break` statement can be used to force leaving loops immediately.

The `continue` statement can be used to omit the remainder of the execution of the loop statement block and jump to the next evaluation of the loop expression.

```
int i;
for(i=0; i<100; i=i+1) {
    do_something();
    if(special_exception==1)
        break;
}
```

```
for(i=-10; i<= 10; i++){
    if(i==0) // Avoid div by 0
        continue;
    printf("%d", 20/i);
}
```

- `break` is used now and then.
- `continue` is used quite rarely.
- both can be avoided by using if clauses.

How?

(and there is also `goto` ...)

### for – while – do while

Have a look at the previous programs:

- a) Replace for-loops with while-loops.
- b) Replace for-loops with do-while-loops.
- c) Replace while-loops for for-loops.

# Number puzzle

Exercise?



`number_puzzle.c`

Code snippet  
402