

Hardwarenahe Software-Entwicklung

Hardwarenahe Software-Entwicklung

Softwaretechnik
Hardwarenahe Software-Entwicklung
Rechnertechnik

Hardwarenahe Software-Entwicklung

Softwaretechnik
Hardwarenahe Software-Entwicklung
Rechnertechnik

Programmierung in C

- kleinster gemeinsamer Nenner für viele Plattformen
- kein „Fallschirm“

1 Einführung in C

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

```
int main (void)
```

```
{  
    printf ("Hello, _world!\n");  
    return 0;  
}
```

```
$ gcc hello-1.c -o hello-1
```

```
$ ./hello-1
```

```
Hello, world!
```

```
$
```

1 Einführung in C

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

```
int main (void)
```

```
{
```

```
    double a;
```

```
    printf ("Bitte_eine_Zahl_eingeben:_");
```

```
    scanf ("%lf", &a);
```

```
    a = 2 * a;
```

```
    printf ("Das_Doppelte_der_Zahl_ist:_%lf\n", a);
```

```
    return 0;
```

```
}
```

```
$ gcc mathe-3b.c -o mathe-3b
```

```
$ ./mathe-3b
```

```
Bitte eine Zahl eingeben: 23
```

```
Das Doppelte der Zahl ist: 46.000000
```

```
$
```

1 Einführung in C

if-Verzweigung

```
if (b != 0)
    printf ("%d\n", a / b);
```

while-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

1 Einführung in C

if-Verzweigung

```
if (b != 0)
    printf ("%d\n", a / b);
```

1.6 Schleifen

while-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

1 Einführung in C

if-Verzweigung

```
if (b != 0)
    printf ("%d\n", a / b);
```

while-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

1.6 Schleifen

for-Schleife

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


1 Einführung in C

if-Verzweigung

```
if (b != 0)
    printf ("%d\n", a / b);
```

while-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

1.6 Schleifen

for-Schleife

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

do-while-Schleife

```
a = 1;
do
{
    printf ("%d\n", a);
    a = a + 1;
}
while (a <= 10);
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)  
{  
    printf ("\n%d", 42);  
    "\n" ;return 0;  
}
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{
```

```
    printf ("%d\n", 42);
```

```
    "\n";
```

```
    return 0;
```

```
}
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{
```

```
    printf ("%d\n", 42);
```

```
    "\n";
```

← Ausdruck als Anweisung: Wert wird ignoriert

```
    return 0;
```

```
}
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{
```

```
    printf ("%d\n", 42);
```

```
    "\n";
```

```
    return 0;
```

```
}
```

← Ausdruck als Anweisung: Wert wird ignoriert

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

```
$ gcc -Wall -O side-effects-1.c -o side-effects-1
```

```
$ ./side-effects-1
```

```
42
```

```
3
```

```
$
```

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.
- „Haupteffekt“: Wert zurückliefern
(hier: Anzahl der ausgegebenen Zeichen)

1 Einführung in C

1.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.
- „Haupteffekt“: Wert zurückliefern
(hier: Anzahl der ausgegebenen Zeichen)
- *Seiteneffekt*: Ausgabe

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `-foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ - * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `-foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ - * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `-foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i += 1;  
}
```

Binäre Operatoren:

- Rechnen: `+` `-` `*` `/` `%`
- Vergleich: `==` `!=` `<` `>` `<=` `>=`
- Zuweisung: `=` `+=` `-=` `*=` `/=` `%=`

rot = mit Seiteneffekt

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `¬foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+` `-` `*` `/` `%`
- Vergleich: `==` `!=` `<` `>` `<=` `>=`
- Zuweisung: `=` `+=` `-=` `*=` `/=` `%=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

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

```
    i += 1;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

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

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo—`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `—foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i += 1;  
}
```

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

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

1 Einführung in C

1.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo—`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `—foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i += 1;  
}
```

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

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

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


1 Einführung in C

1.8 Funktionen

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

```
int answer (void)
```

```
{  
    return 42;  
}
```

```
void foo (void)
```

```
{  
    printf ("%d\n", answer ());  
}
```

```
int main (void)
```

```
{  
    foo ();  
    return 0;  
}
```

1 Einführung in C

1.8 Funktionen

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

```
int answer (void)
{
    return 42;
}
```

```
void foo (void)
{
    printf ("%d\n", answer ());
}
```

```
int main (void)
{
    foo ();
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}

1 Einführung in C

1.8 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}

1 Einführung in C

1.8 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void** steht für „nichts“

1 Einführung in C

1.8 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void**
steht für „nichts“
und kann ignoriert werden.

1 Einführung in C

1.8 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void**
steht für „nichts“
und muß ignoriert werden.

1 Einführung in C

1.8 Funktionen

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

```
int a, b = 3;
```

```
void foo (void)
```

```
{  
    b++;  
    static int a = 5;  
    int b = 7;  
    printf ("foo():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    a++;  
}
```

```
int main (void)
```

```
{  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    foo ();  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    a = b = 12;  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    foo ();  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    return 0;  
}
```

1 Einführung in C

1.9 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```


1 Einführung in C

1.9 Zeiger

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

```
void calc_answer (int *a)
```

```
{  
    *a = 42;  
}
```

- `*a` ist eine `int`.

```
int main (void)
```

```
{  
    int answer;  
    calc_answer (&answer);  
    printf ("The_answer_is_%d.\n", answer);  
    return 0;  
}
```

1 Einführung in C

1.9 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Derferenzierung

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

1 Einführung in C

1.9 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- ***a** ist eine **int**.
- unärer Operator *****:
Pointer-Derferenzierung

→ **a** ist ein Zeiger (Pointer) auf eine **int**.

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

1 Einführung in C

1.9 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Derferenzierung

→ `a` ist ein Zeiger (Pointer) auf eine **int**.

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

- unärer Operator `&`: Adresse

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable.

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

- `prime` ist eine Ansammlung von fünf ganzen Zahlen.

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.

1 Einführung in C

1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.

1 Einführung in C


1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.

1 Einführung in C


1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.

1 Einführung in C


1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`

1 Einführung in C


1.10 Arrays

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int i = 0;  
    while (i < 5)  
        printf ("%d\n", prime[i++]);  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    int i = 0;
```

```
    while (hello_world[i] != 0)
```

```
        printf ("%d", hello_world[i++]);
```

```
    return 0;
```

```
}
```

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    int i = 0;
```

```
    while (hello_world[i])
```

```
        printf ("%d", hello_world[i++]);
```

```
    return 0;
```

```
}
```


1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%d", *p++);
```

```
    return 0;
```

```
}
```

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**.

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars**.

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars**
also ein Zeiger auf (kleinere) Integer.

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Die Formatspezifikation entscheidet über die Ausgabe:
 - %d** dezimal
 - %x** hexadezimal
 - %c** Zeichen

1 Einführung in C

1.10 Arrays

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Die Formatspezifikation entscheidet über die Ausgabe:
 - %d** dezimal
 - %x** hexadezimal
 - %c** Zeichen
 - %s** String

Aufgaben

Aufgabe 1: Multiplikationstabelle

Geben Sie mit Hilfe einer Schleife ein „Einmaleins“ aus:

1 * 7 = 7

2 * 7 = 14

...

10 * 7 = 70

Hinweis: Verwenden Sie Formatspezifikationen wie z. B. `%3d` (siehe die Dokumentation zu `printf()`)

Aufgabe 2: Fibonacci-Zahlen berechnen

1. Zahl: 1

2. Zahl: 1

nächste Zahl = Summe der beiden vorherigen

Aufgabe 3: Schaltjahr ermitteln

Jahreszahl erfragen

Wenn die Zahl durch 4 teilbar ist, ist es ein Schaltjahr.

Wenn die Zahl durch 100 teilbar ist, ist es kein Schaltjahr.

Wenn die Zahl durch 400 teilbar ist,
ist es doch wieder ein Schaltjahr.