Teil 2
Programmfluss

- Gültigkeit von Variablen
- Schleifen
- Wertebereiche & Vorzeichen von Zahlen
- Funktionen
- Überladen
- mathematische Funktionen
- Stack
- Debugging

Gültigkeitsbereiche

```
int main()
{
   int a = 1;
   {
      int b = a + 1;
      cout << a << endl;
   }
   cout << b << endl;

   return a;
}
```

b out of scope !!!

Gültigkeitsbereiche

```
int main()
{
   int a = 1;
   {
      int b = a + 1;
      cout << b << endl;
   }
   cout << a << endl;

   return a;
}
```

Schleifen

`for` (*Initialisierung; Test; Fortsetzung*) *Anweisung;*

- Initialisierung wird einmalig ausgeführt, Deklarationen nur innerhalb der Schleife gültig
- Test (boolscher Ausdruck) wird vor jedem Durchlauf ausgewertet
- Anweisung (oder Block) wird ausgeführt
- Fortsetzung wird ausgeführt

Schleifen

```
int main()
{
   for (int i = 1; i <= 10; ++i)
   {
      cout << i << endl;
   }
   return 0;
}
```

i out of scope

Aufgaben

Aufgabe 5A

- alle zweistelligen geraden Zahlen sollen ausgegeben werden
- durch n (Eingabe) teilbare Zahlen sollen ausgegeben werden

Aufgabe 5B

- eine Folge von Zufallszahlen soll ausgegeben werden (Eingabe der Anzahl)
- das Minimum und Maximum dieser Folge soll ermittelt und ausgegeben werden

Schleifen

```
int main()
{
   int i = 1;
   while (i <= 10)
   {
      cout << i << endl;
      ++i;
   }

   return 0;
}
```

i weiter gültig

Schleifen

```
int main()
{
   int i = 1;
   do
   {
      cout << i << endl;
      ++i;
   }
   while (i <= 10)

   return 0;
}
```

Wertebereiche

- typische Wertebereiche - in C++ nicht exakt festgelegt

`int`	`-2147483648`	bis	`2147483647`
`unsigned int`	`0`	bis	`4294967295`
`signed char`	`-128`	bis	`127`
`unsigned char`	`0`	bis	`255`

```
signed char x = 127;
++x;
```

- kein Schutz vor Überlauf

Binärzahlen

0	0	1000	8
1	1	1001	9
10	2	1010	10
11	3	1011	11
100	4	1100	12
101	5	1101	13
110	6	1110	14
111	7	1111	15

Vorzeichen

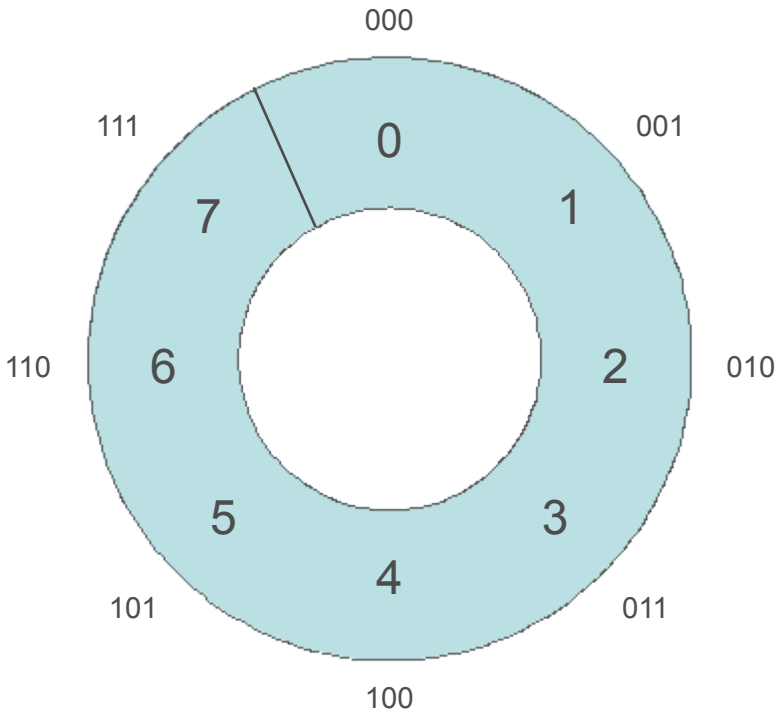
- erstes Bit hält Vorzeichen

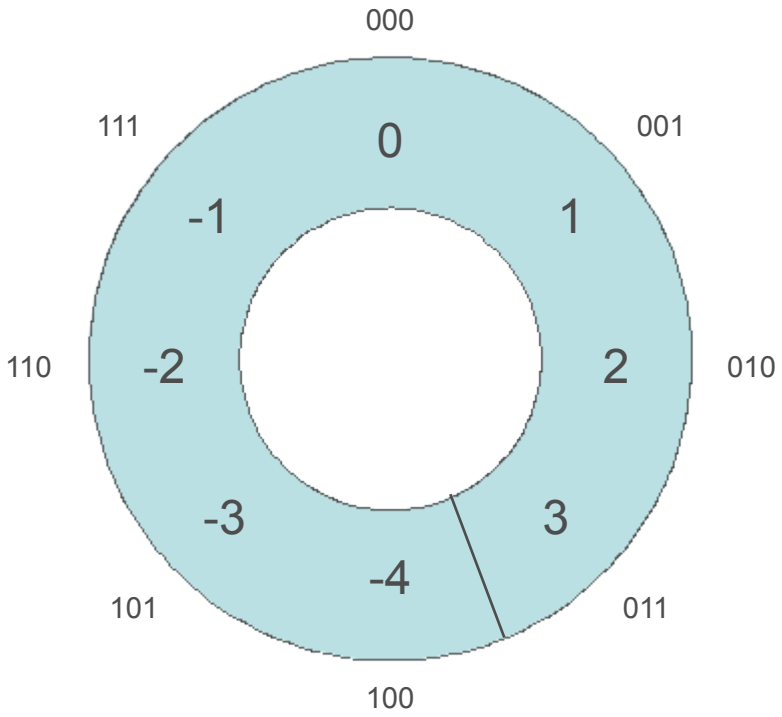
```
00000000    0          10000000  128 bzw. -128
00000001    1          10000001  129 bzw. -127
01111110  126          11111110  254 bzw.   -2
01111111  127          11111111  255 bzw.   -1
```

```
unsigned char x = 255;
signed char y = x;
```

- Compiler-Warnung beim Wechsel von unsigned auf signed oder umgekehrt

Vorzeichen


000
001
010
011
100
101
110
111
0
1
2
3
4
5
6
7


000
001
010
011
100
101
110
111
0
1
2
3
-4
-3
-2
-1

Vorzeichen

```
int main()
{
   unsigned int i = 9;
   while (i >= 0)
   {
      cout << i << endl;
      --i;
   }

   return 0;
}
```

Bedingung immer wahr !!!

Vorzeichen

- Vorzeichen sind Freunde!
- signed und unsigned mischen ist übel
- signed ist fast immer vorzuziehen
  (außer bei IDs, mit denen man nicht rechnen will)
- seltenst reicht der Wertebereich von signed int nicht aus
- wenn der Wertebereich von signed int nicht ausreichen sollte, kommt man mit unsigned int meist auch nicht viel weiter

Funktionen

```
bool isDividable(int a, int b)
{
   int r = a % b;
   return r == 0;
}

int main()
{
   cout << isDividable(42, 7) << endl;
   return 0;
}
```

Funktionen

```
void print(int number)
{
  cout << number << endl;
}

int main()
{
  print(42 / 7);
  return 0;
}
```

Aufgaben

Aufgabe 6A

- eine Funktion soll die Potenz zweier Zahlen berechnen (m^n)
- eine zweite Funktion soll die Fakultät einer Zahl berechnen (n!)
- eine weitere Funktion soll den Binomialkoeffizienten (n über k) einer Zahl berechnen als n! / (k! · (n - k)!)
- das Pascalsche Dreieck soll ausgegeben werden (erste Zeile: 0 über 0, zweite Zeile: 1 über 0 und 1 über 1, dritte Zeile: 2 über 0, 2 über 1, 2 über 2...)

Aufgabe 6B

- eine Funktion soll alle Teiler einer Zahl ausgeben
- eine zweite Funktion soll die Anzahl der Teiler einer Zahl ermitteln
- eine weitere Funktion soll prüfen, ob eine Zahl eine Primzahl ist
- in einer Schleife sollen alle Primzahlen bis 100 ausgegeben werden

Überladen

```
void print(int number)
{
  cout << number << endl;
}

void print(string text)
{
  cout << text << ": ";
}

int main()
{
  print("die Antwort lautet");
  print(42);
  return 0;
}
```

Funktionssignatur

```
void print(int number)
{
  cout << number << endl;
}

void print(string text)
{
  cout << text << ": ";
}

int print(string text)
{
  cout << text << ": ";
  return text.length();
}
```

unmöglich, da identische Signatur

Parameter mit Defaultwerten

```
void print(double value, string unit = "m")
{
  cout << value << " " << unit;
}

int main()
{
  print(250.0);
  print(500.0);
  print(2.5, "km");
  return 0;
}
```

wird vom Aufrufer ergänzt

mathematische Funktionen

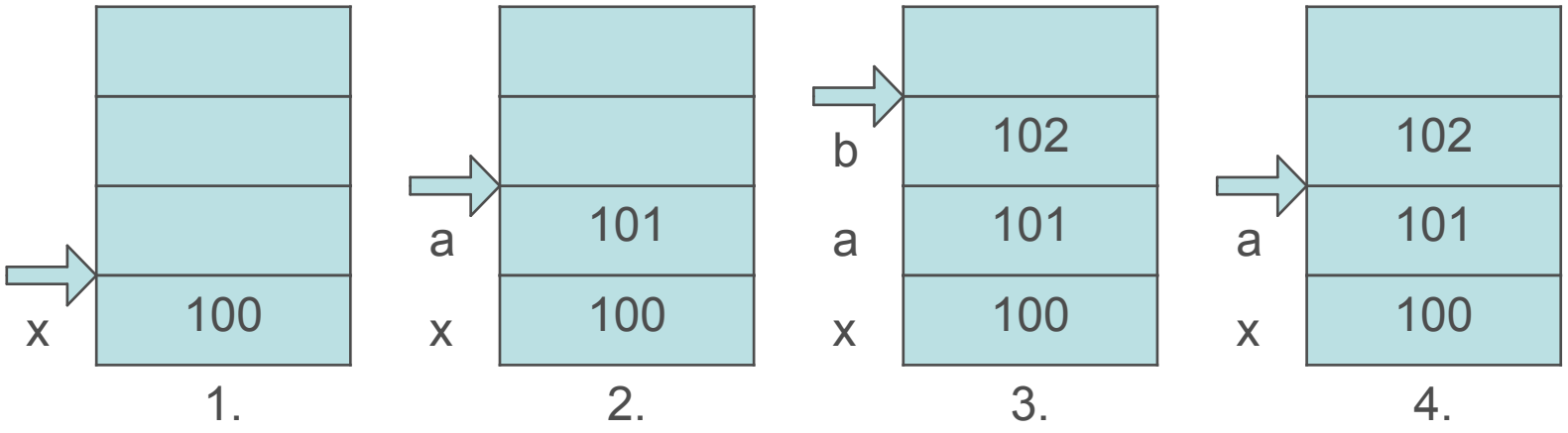
```
#include <cmath>

const double pi = acos(-1.0);
double x = cos(0.25 * pi);
double y = sin(0.25 * pi);
double delta = fabs(a - b);
double dist = sqrt(x * x + y * y);
double rest = fmod(7.5, 5.0);
double million = pow(10.0, 6.0);
double magnitude = log10(r);
double angle = atan2(y, x);
```

- weitgehend kompatibel mit den C-Funktionen aus math.h

```
#include <math.h> // C
#include <cmath> // C++
```

Der Stack

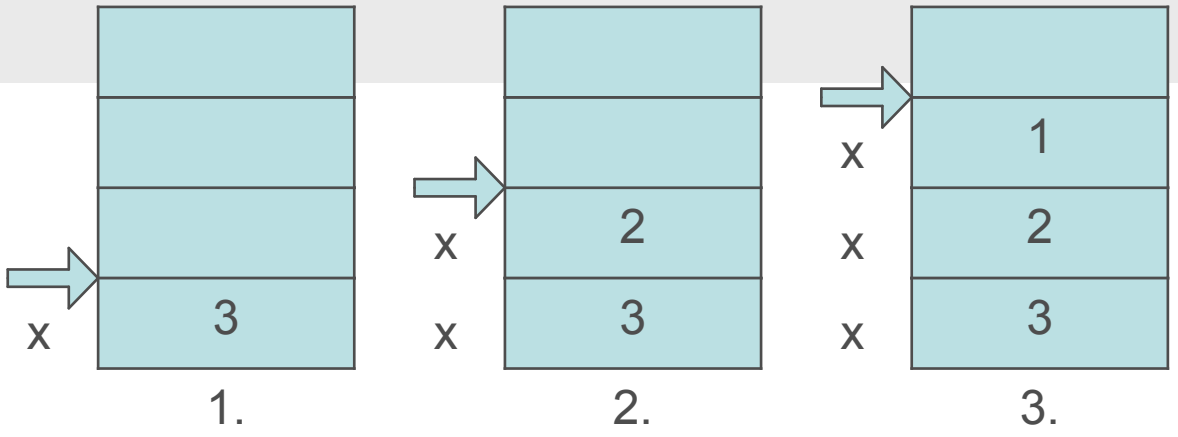
```
int foo(int x)
{
  int a = x + 1;
  {
    int b = a + 1;
  }
  return a;
}
```



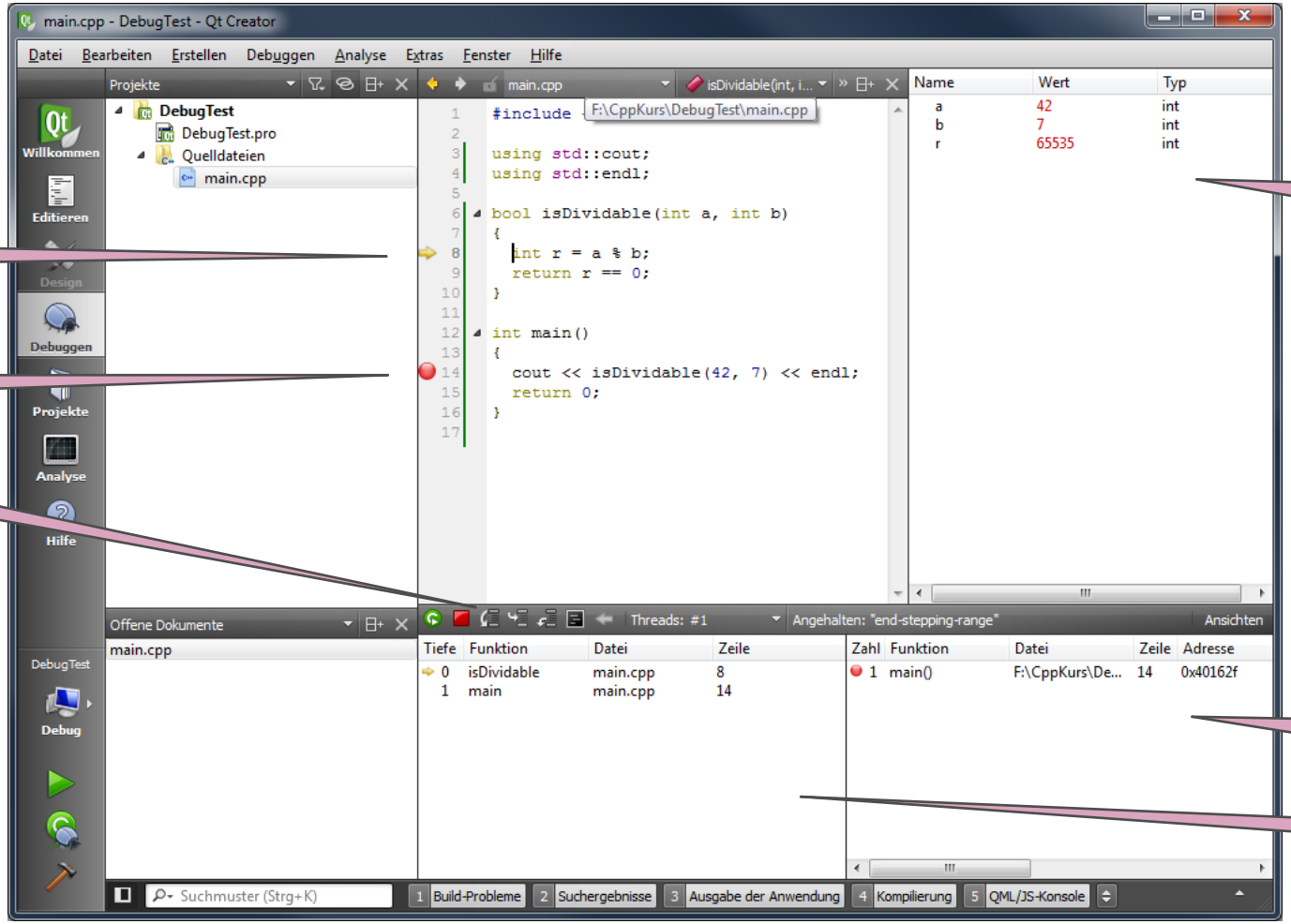
Rekursion

- der Stack ermöglicht auch Rekursion, d.h. Funktionen, die sich selbst aufrufen
- Achtung! Der Stack ist begrenzt. Bei Stack Overflow wird das Programm vom Betriebssystem beendet.

```
int fakultaet(int x)
{
  if (x == 1) return 1;
  else return x * fakultaet(x - 1);
}
```



Debugging


main.cpp - DebugTest - Qt Creator
Datei Bearbeiten Erstellen Debuggen Analyse Extras Fenster Hilfe
Projekte
DebugTest
DebugTest.pro
Quelldateien
main.cpp
Willkommen
Editieren
Design
Debuggen
Projekte
Analyse
Hilfe
DebugTest
Debug
main.cpp
isDividable(int, i...
F:\CppKurs\DebugTest\main.cpp
#include
using std::cout;
using std::endl;
bool isDividable(int a, int b)
{
int r = a % b;
return r == 0;
}
int main()
{
cout << isDividable(42, 7) << endl;
return 0;
}
Name Wert Typ
a 42 int
b 7 int
r 65535 int
Offene Dokumente
main.cpp
Threads: #1
Angehalten: "end-stepping-range"
Ansichten
Tiefe Funktion Datei Zeile
0 isDividable main.cpp 8
1 main main.cpp 14
Zahl Funktion Datei Zeile Adresse
1 main() F:\CppKurs\De... 14 0x40162f
Suchmuster (Strg+K)
1 Build-Probleme
2 Suchergebnisse
3 Ausgabe der Anwendung
4 Kompilierung
5 QML/JS-Konsole
nächste Zeile
Breakpoint
Steuerung
Variablen
Breakpoints
Call Stack

Aufgaben

Aufgabe 7A

- Breakpoints sollen sinnvoll in den zuvor programmierten Funktionen verteilt werden
- das Programm soll schrittweise abgearbeitet werden, dabei sollen der Stack und die Variablen beobachtet werden
- Kann man Variablen zur Laufzeit im Debugger verändern? Was passiert?

Aufgabe 7B

- Breakpoints sollen sinnvoll in den zuvor programmierten Funktionen verteilt werden
- das Programm soll schrittweise abgearbeitet werden, dabei sollen der Stack und die Variablen beobachtet werden
- Lässt sich der Breakpoint so verändern, dass er die Ausführung erst bei der Zahl 13 anhält?