Eric Ceglie

Informatik Übungsstunde – Woche 4

Expressions

Ähnlich wie bei den Operatoren haben in C++ auch die verschiedenen Datentypen Präzedenzen, auf die man achten muss! Folgende Reihenfolge sollte man sich merken:

kommt in zukünftiger Vorlesung

bool < char < int < unsigned int < float < double

Faustregel: Es wird in die "weniger aufwändige" Richtung konvertiert.

Bsp:

- 5 / 2 == 2 (int, int ⇝ int)
- 5 / 2.0 == 2.5 (int < double ⇝ double)
- 3.0 + 5 / 2 == 5

  3.0 + (5/2) == 3.0 + 2 == 5

  ⇝ Präzedenz der Operatoren nicht vergessen!

- 3.0 + true / 2.0 * 5 − 6.0f / 3 == 3.5

  3.0: double; true: bool; 2.0: double; 5: int; 6.0f: float; 3: int; 3.5: double

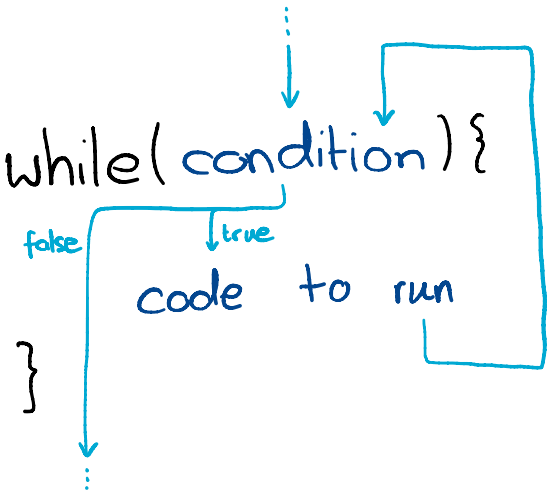
  (true / 2.0) * 5 == 2.5

  0.5; double

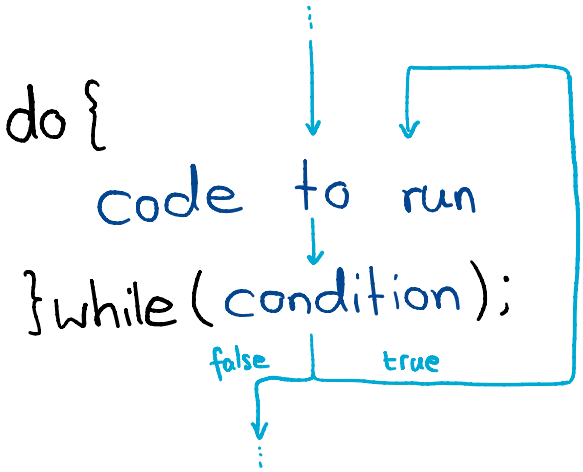
  6.0f / 3 → 2.0f; float

(Do-) While-Loops

Allgemeiner While-Loop:



Allgemeiner Do-While-Loop:



Nun bemerke man, dass folgende Codes äquivalent sind:

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

=

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

=

```
code;
for(;condition;){
    code;
}
```

Vermeiden

Scopes

Ein Scope (Gültigkeitsbereich) wird meistens (nicht immer!) mit {} gekennzeichnet.
Wird eine Variable in einem Scope deklariert, so ist sie innerhalb des Scopes und auch in allen darin verschachtelten Scopes sichtbar.
Am Ende des Scopes wird der Speicherplatz wieder freigegeben und die Variable ist nicht mehr sichtbar.

Bsp:

```
int main() {

  int x = 1;

  for(int i = 0; i < 5; i++){
    x *= 2;
    std::cout << i;   ⇝ Ok.
  }

  std::cout << x;   ⇝ Ok.

  std::cout << i;   ⇝ Fehler

  return 0;
}
```

Scope der Variable i

Scope der Variable x

Zudem hat jeder Scope seine "eigenen Variablen". Wird sehr gefährlich bei verschachtelten Scopes!

Bsp:

```
int main() {

  int n = -3;

  for(int n = 0; n < 5; n++){
    std::cout << n << "\n";
  }                  ↑ n ∈ {0,1,2,3,4}
  std::cout << n << "\n";
                     ↑ n = -3

  return 0;
}
```

verschachtelter Scope
⇝ n wird hier neu deklariert und am Ende von diesem Scope wieder freigegeben

äusserer Scope