Inhaltsverzeichnis

Gültigkeitsbereich einer Variable

- Warum kann keine Variable ***des Hauptprogramms*** in einer Funktion abgerufen werden?
Siehe: *summe_var_main_funkt.cpp*

- Warum kann keine Variable ***einer Funktion*** im Hauptprogramm abgerufen werden?
Siehe: *summe_var_funkt_main.cpp*

- Warum können 2 verschiedene Variablen in Funktion und im main den selben Namen haben?
Siehe: *summe_var_gleiche_namen.cpp*

ANTWORT: GÜLTIGKEITSBEREICH

Gültigkeitsbereich einer Variable

Definition und Erkennung

Gültigkeitsbereich einer Variable ist das/die Programmblock/-blöcke, für den/die die Variable **nutzbar** und sichtbar ist.

Eine Variable ist nutzbar im Bereich wo sie deklariert ist.

Also ist der **Gültigkeitsbereich** der Bereich, in dem eine Variable deklariert ist.

zwei **lokale** Gültigkeitsbereiche

Hauptprogramm und Funktion

```
#include <iostream>
using namespace std;
 ...
```

HAUPTPROGRAM

```
main()
{
  int a;

  funkt (5);
  ...

  cout << a;
}
```

FUNKTION

```
void funkt(int b)
{
  int c;

  ...

  cout << b;
  cout << c;
}
```

Gültigkeitsbereiche: LOKAL

```
#include <iostream>
using namespace std;
...
```

HAUPTPROGRAM

```
main()
{
  int a;

  funkt (a);
  ...

  cout << a;
}
```

Gültigkeitsbereich von *a*

FUNKTION

```
void funkt(int b)
{
  int c;

  ...
  cout << b;
  cout << c;
}
```

Gültigkeitsbereich von *b und c*

Zugriff zwischen Gültigkeitsbereichen

Wie oben gesehen, können die in *main* deklarierten Variablen in der Funktion nicht anhand des Namens verwendet werden.

umgekehrt ist auch wahr:
die in der Funktion deklarierten Variablen können im *main* nicht anhand des Namens verwendet werden.

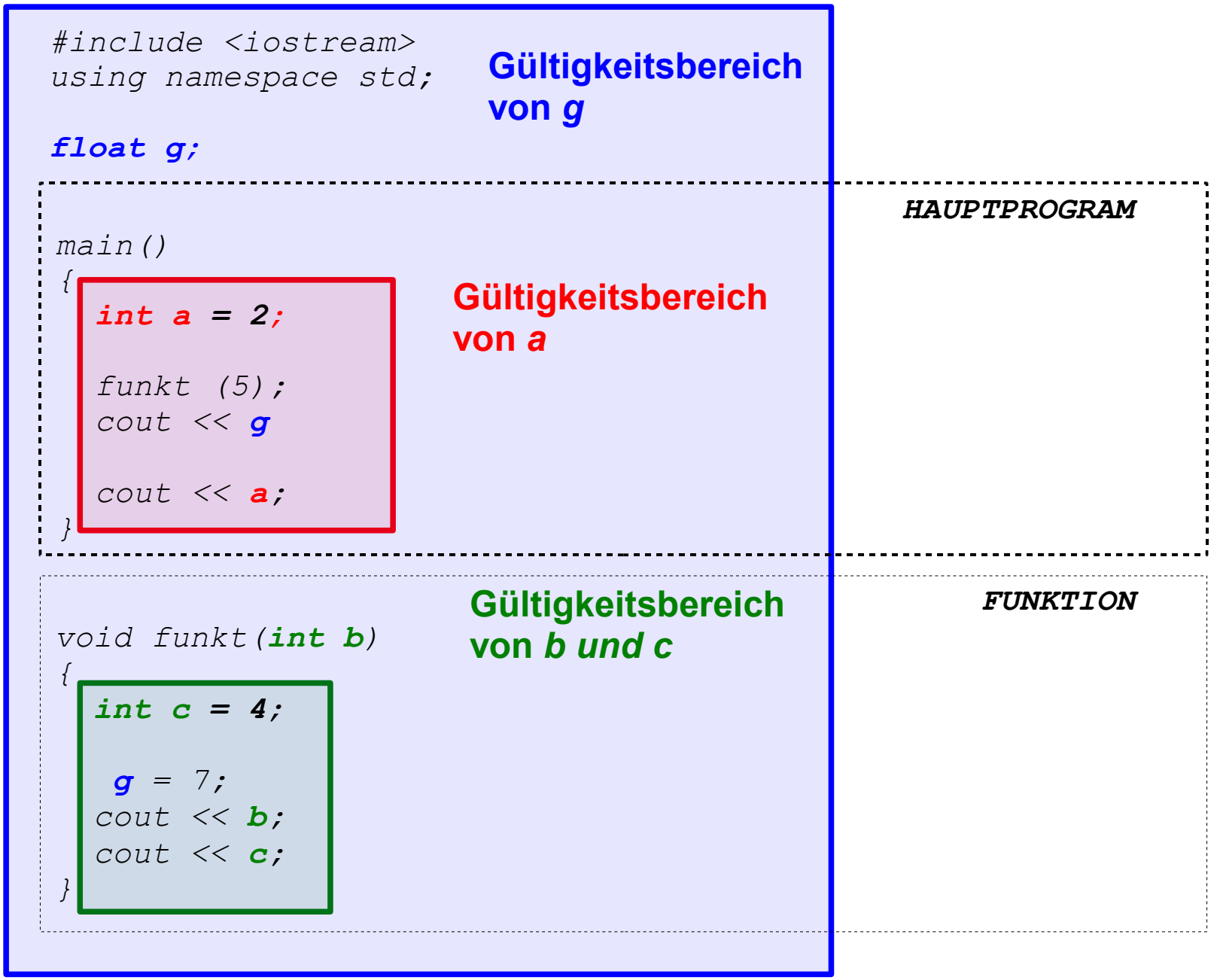
Warum?: Weil der Gültigkeitsbereich einer Variable ist die Funktion wo sie deklariert ist. Anders ausgedrückt: „Eine Variable ist lokal zu ihrer Funktion“

Gültigkeitsbereich

Frage: könnte ausnahmsweise genau die selbe Variable im *main* und in einer Funktion abgerufen werden?

Antwort: Ja. Solche Variablen mit allgemeinem Gültigkeitsbereich heißen **globale** Variablen.

Gültigkeitsbereiche: GLOBAL



Über GLOBALE Variablen

- Die Verwendung von globalen Variablen ist **streng abgeraten**.
- Aus den folgenden Gründen:
  - Verlust an Übersichtlichkeit
  - Unerwartete Nebenwirkungen

Funktionen und Gültigkeitsbereich

- Wie vorhin gesehen, ist jede Variable lokal zu einer Funktion. Nur in der eigenen Funktion abrufbar.

  Das führt zu der Feststellung einer wichtigen Eigenschaft der Funktionen:

  die Funktionen sind **dicht** für die Variablen.

Funktionen, Gültigkeitsbereich, Übergabe und Rückgabe

- Da die Funktionen **dicht** für die Variablen sind, muss ich etwas tun um Werte zwischen Funktionen und *main* auszutauschen:
  - ÜBERGABE: Die Werte werden in die Funktion geliefert.
  - RÜCKGABE: Ein Wert wird aus der Funktion zurück.

Arrays und Funktionen

Der Sonderfall der Übergabe eines Arrays

Array als Rückgabe

Tatsachen

- Ein übergebener Array (`int []`, `float[]`, `char[]`, `...`) oder eine übergebene Matriz (`int [3][3]`, `float[2][2]`, `...`) können in einer Funktion verändert werden ohne Rückgabe.
- Ein grundlegender Datentyp (`int`, `float`, `char`, ...) ***kann nicht ohne Rückgabe*** verändert werden.
  Siehe unten den Begriff „Gültigkeitsbereich“

Parameterübergabe in einer Funktion

zwei Art Übergaben

- **Per Wert**
  WERT wird übergeben
  (Grundlegende Typen)

- **Per Referenz**
  SPEICHERADRESSE wird übergeben
  (Arrays und Zeiger)

Übergabe per Wert

Per **Wert**:

Eine Kopie der Wert wird übergeben bei dem Aufruf.

```
int main(void)
{
float a = 3.2;
void doppel (a);
cout << a;
return 0;
}
```


```
void doppel (int a)
{
  a= 2*a;
}
```

Siehe: verdoppelt_Wert.cpp

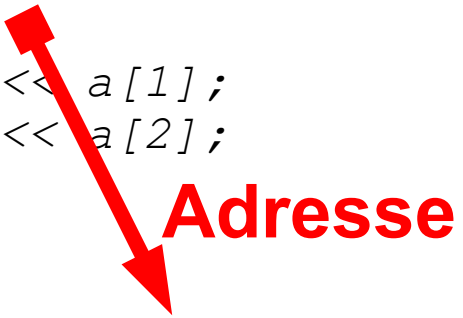
Übergabe per Referenz

Per **Referenz**:

Die Speicheradresse des Arrays wird übergeben.

```
int main(void)
{
float a[3] = {3.1, 2.0, 1.5};
void doppel (a);
cout << a[0];
cout << endl << a[1];
cout << endl << a[2];
return 0;
}

void doppel (float c[])
{
  int i;
  for (i = 0; i<3; i++ )
  c[i]= 2*c[i];
}
```



Siehe: verdoppelt_Vekt.cpp