

# Vorlesung2

---

- Einfaches Programm ohne Funktion.
- Funktion mit einfachen Parametern. (Call by Value Übergabe per Wert)
- Übergabe von Arrays an Funktionen. (Call by Reference Übergabe per Referenz).

## *Uebung1\_Aufgabe3*

- Gültigkeitsbereich einer Variable.
- Die Speicheradresse.

# Einfaches Programm ohne Funktion

---

- Lineare Form des Quellcodes.
  - Viele Programme aus dem ersten Semester

*Vorlesung2\_1.cpp*

# Funktion mit einfachen Parametern.

---

## *Call by Value*

- Wenn an Funktionen Parametern übergeben werden, gibt es zwei verschiedene Aufgabestellungen.
  - Im ersten Fall werden die Parameter für Berechnungen verwendet und das Ergebnis der Berechnung z.B per „return“ an das aufrufende Programm zurückgegeben. Die Parameter werden in der Funktion nicht verändern.

# Funktion mit einfachen Parametern.

---

*Call by Value*

- Siehe `ausgabe.cpp`

# Funktion mit Array als Parametern

---

- Siehe `ausgabe2.cpp`
- Analyse der Parameterübergabe.
- Analyse des Funktionsaufrufs.

# Aufgabe2: vom Problem zum Programm

---

Aufgabe 2: Es soll ein Programm entwickelt werden, das

- 5 ganze Zahlen einliest
- 5 weitere Zahlen einliest
- und die Summe der ersten 5 Zahlen ausgibt
- und die Summe der zweiten 5 Zahlen ausgibt

# Lösungsweg

---

- 2 Arrays mit 5 Zahlen vereinbaren  
a[0], a[1]...a[4]  
b[0], b[1]...b[4]
- Werte in jedem Array einlesen
- **Aufruf** der Funktion Summe mit 'a' als Parameter
- **Aufruf** der Funktion Summe mit 'b' als Parameter

# Funktion im Programm

---

- Parameter: Ein Array von *int*
- Name: *ElementeSumme*
- Rückgabe: Ein *int*

$$\begin{array}{ccc} \overbrace{a}^{\text{ÜBERGABE}} & \rightarrow & \overbrace{a_1 + a_2 + \dots + a_5}^{\text{RÜCKGABE}} \end{array}$$

***int ElementeSumme ( int v[] );***

*Vorlesung1\_6.cpp*



# Arrays als Parameter einer Funktion

---

- Parameter: Array ***w*** von Datentyp ***Typ***
- Name der Funktion: ***Funktionsname***
- Datentyp der Rückgabe: ***Rückgabetyt***

<b><i>Rückgabetyt Funktionsname ( Typ w[] )</i></b>
---

# Aufgabe 3: vom Problem zum Programm

---

Aufgabe 3: Es soll ein Programm,entwickelt werden,das

- 1 Array einliest
- Den Array mit 2 multipliziert
- Den Array ausgibt

*Vorlesung2\_1.cpp*

# Lösungsweg

---

- 1 Arrays mit 5 Zahlen vereinbaren  
 $a[0]$ ,  $a[1]$ ... $a[4]$
- Werte im Array einlesen
- **Aufruf** der Funktion ***doppel*** mit 'a' als Parameter und Rückgabe...

Ein Array als Rückgabe??

- Ausgabe der Werte durch Bildschirm

# Array als Rückgabe

---

- Parameter: Ein Array von *int*
- Name: *doppel*
- Rückgabe: ???

$\underbrace{\text{ÜBERGABE ARRAY}}_a \rightarrow \underbrace{\text{RÜCKGABE Verdoppelter Array}}_{2 \cdot a}$

      ? *doppel* ( *int* *v[]* )

# Array als Rückgabe

---

- Parameter: Ein Array von *int*
- Name: *doppel*
- Rückgabe: **void**

$\overbrace{a}^{\text{ÜBERGABE ARRAY}} \rightarrow \overbrace{2 \cdot a}^{\text{ÜBERGABE Verdoppelter Array}}$

***void doppel ( int v[] )***

*Vorlesung2\_2.cpp*

# Array als Rückgabe

---

- Ein übergebener Array kann in einer Funktion verändert werden ohne Rückgabe.
- Ein grundlegender Datentyp kann nicht ohne Rückgabe verändert werden

*Vorlesung2\_3.cpp*

# Gültigkeitsbereich einer Variable

---

- Warum kann auf einer Variablen des *Hauptprogramms* nicht in einer Funktion zugegriffen werden?
- Warum kann auf einer Variablen einer beliebigen Funktion im Hauptprogramm nicht zugegriffen werden?
- Warum können 2 verschiedene Variablen gleich genannt werden?

ANTWORT: GÜLTIGKEITSBEREICH

*Vorlesung2\_4.cpp*

# Gültigkeitsbereiche

---

```
#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;
```

```
    ...
```

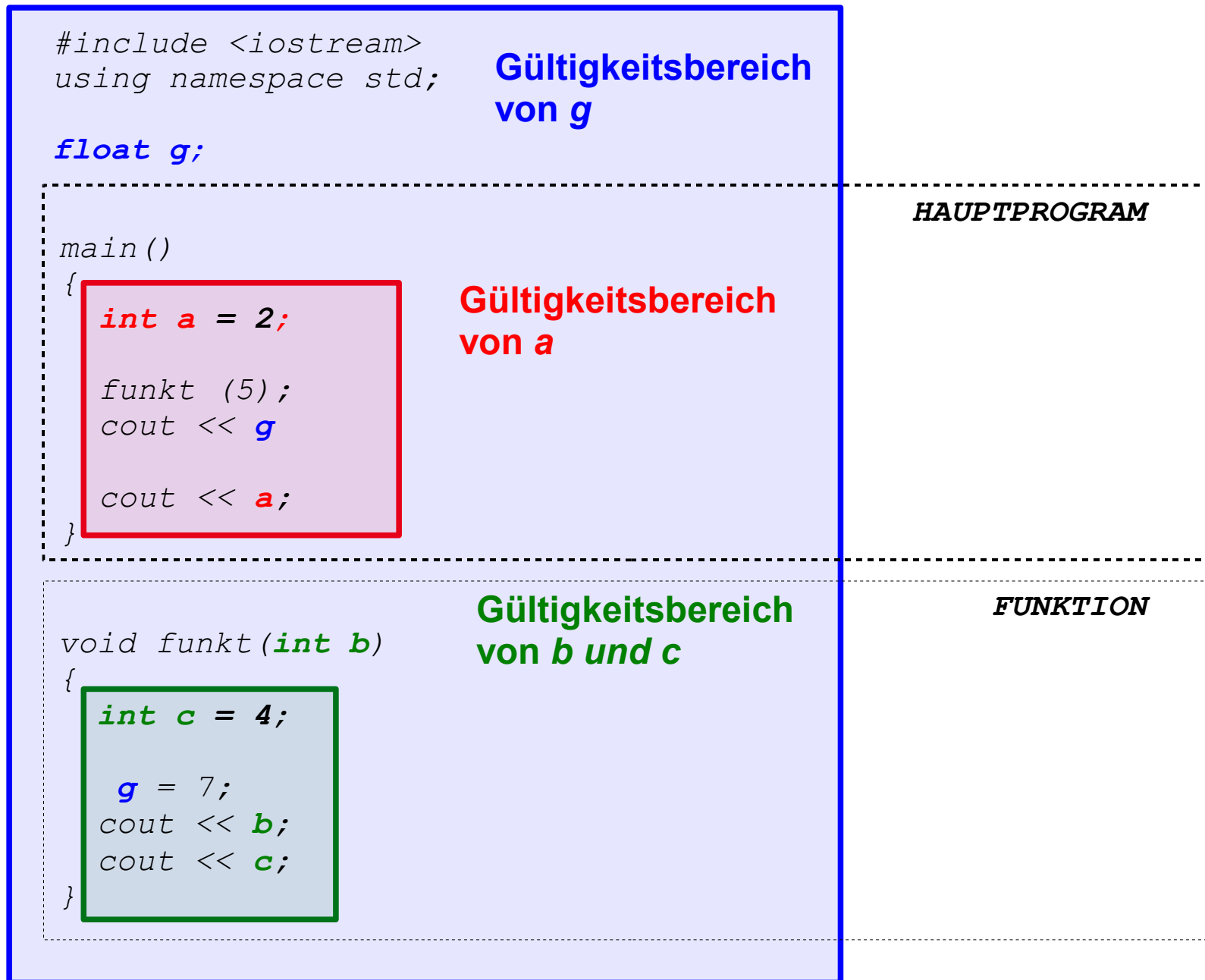
```
    cout << b;
```

```
    cout << c;
```

```
}
```

**Gültigkeitsbereich  
von b und c**

# Gültigkeitsbereiche: GLOBAL



# Veränderung

---

Problem: Kann man eine Variable von main direkt in einer Funktion verändern?

# 2 Arten von Übergabe

---

- 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);
    return 0;
}
```

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

# Gültigkeitsbereich einer Variable

---

Eine Variable ist abrufbar (verwendbar) im Bereich wo sie deklariert wurde, nämlich ihrem Gültigkeitsbereich.

# 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);
    return 0;
}

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

# Die Speicheradresse

---

- Jede Speicheradresse entspricht einem Byte (8 Bits) des Speicherplatzes
- Jede Variable besitzt eine eigene Speicheradresse.
- Die Speicheradresse ist eindeutig für jede Variable.

Variable  Speicheradresse

Speicheradresse

FF02

Wert

-1023



# Arrays und Speicheradresse

---

```
int a[7] = {234, -872, 0, 2, 1009, 2191, -7427}
```

WERTE	234	-872	0	2	1009	2191	-7427
ADRESSE	FF02	FF06	FF0A	FF0E	FF12	FF16	FF1A
POSITION	0	1	2	3	4	5	6

# Ermittlung der Adresse

---

Wie kann man die Adresse einer Variable ermitteln?

Durch den **Operator &**

***&Variablenname***

*Vorlesung2\_5.cpp*

# Veränderung einer Variable ohne Rückgabe

---

Übergabe per Referenz (Übergabe der Adresse)

&a

*Vorlesung2\_6.cpp*