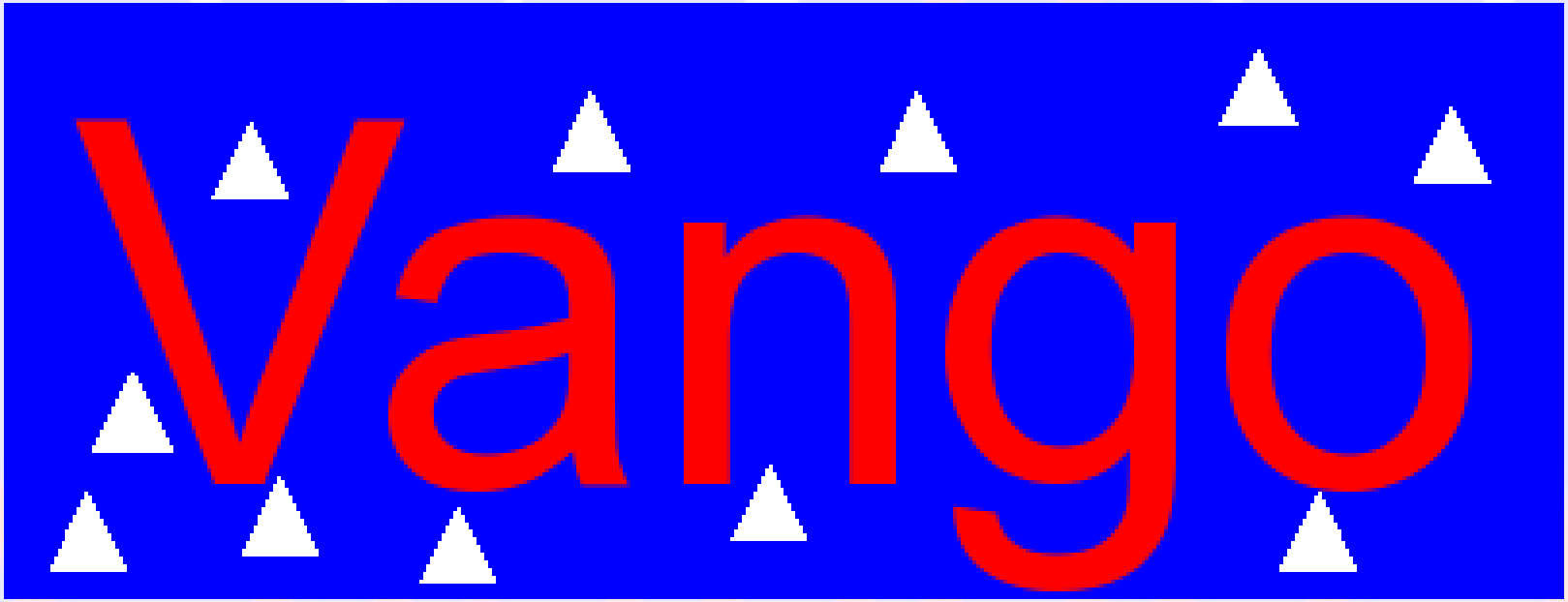


C-Programmierung (Teil 1)

Themenüberblick

1. Grundlagen

- Funktionen (Eingabeparameter und Rückgabewerte)
- main()-Funktion
- Datentyp int (Integer)
- Variablen
- printf()-Funktion
- Kommentare

2. Die wichtigsten Datentypen

- Eigenschaften
- Formatierte Ausgabe mit printf()
- Typkonvertierung
- Arrays und Strings

3. Eingabe mit scanf()

1. Grundlagen

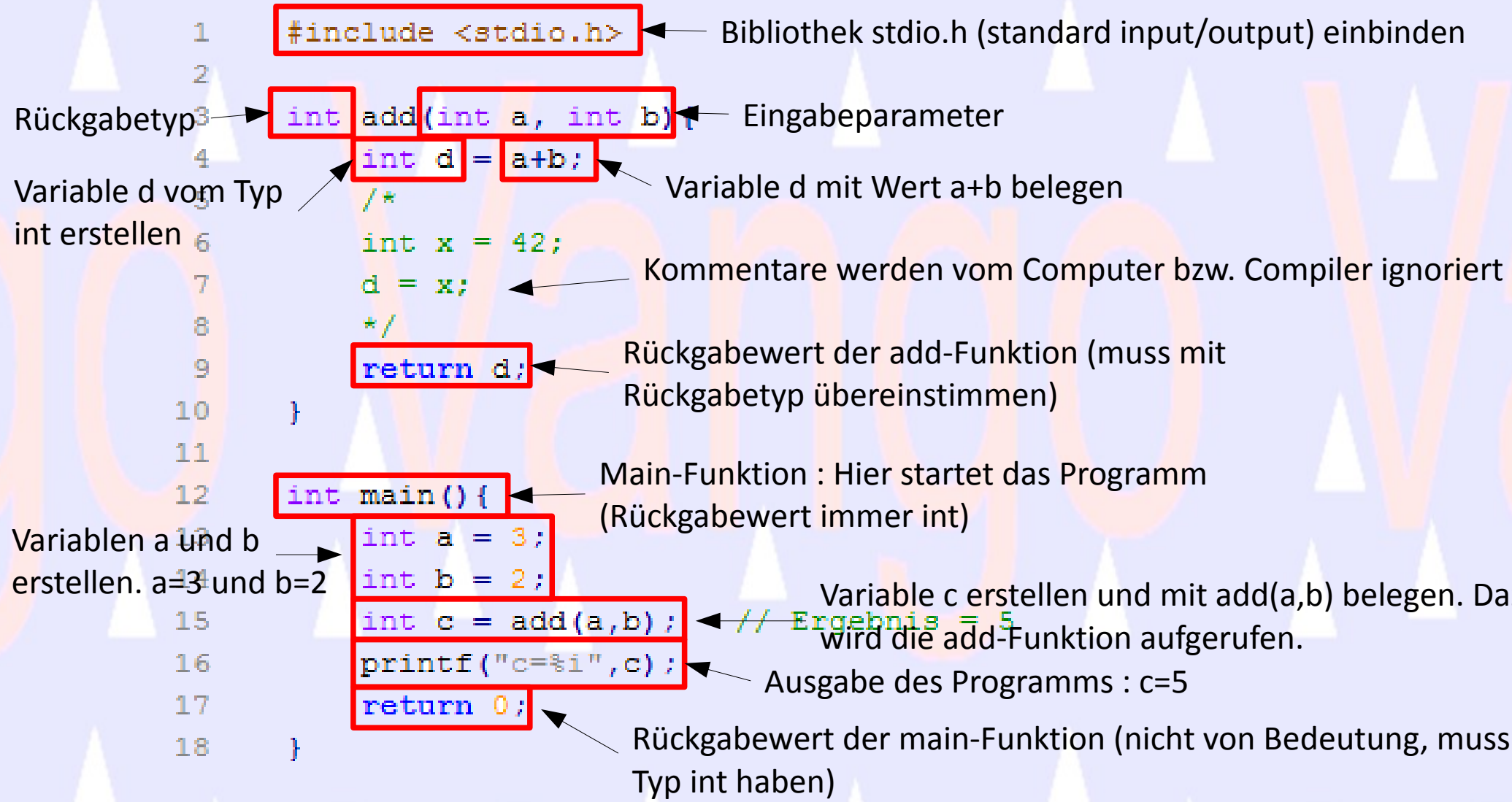
Grundraster-Programm

```c
#include <stdio.h>

int add(int a, int b){
    int d = a+b;
    /*
    int x = 42;
    d = x;
    */
    return d;
}

int main(){
    int a = 3;
    int b = 2;
    int c = add(a,b);    // Ergebnis = 5
    printf("c=%i",c);
    return 0;
}
```

Grundraster-Programm



2. Die wichtigsten Datentypen

Integer

- Ganze Zahlen
- Wertebereich : -2147483648 bis 2147483647
- Name : `int`
- Formatzeichen : %d oder %i
- Beispiel :

```c
#include <stdio.h>

int main(){
    int a = 194;
    int b = 74;
    int c = a-b;
    printf("%i - %d = %i",a,b,c);
    return 0;
}
```

long

- Ganze Zahlen
- Wertebereich : -2147483648 bis 2147483647
- Name : `long`
- Formatzeichen : `%ld` oder `%li`
- Beispiel :

```c
#include <stdio.h>

int main(){
    long a = 194;
    long b = 74;
    long c = a-b;
    printf("%li - %ld = %li",a,b,c);
    return 0;
}
```

short

- Ganze Zahlen
- Wertebereich : -32768 bis 32767
- Name : `short`
- Formatzeichen : %d oder %i
- Beispiel :

```c
#include <stdio.h>

int main(){
    short a = 194;
    short b = 74;
    short c = a-b;
    printf("%i - %d = %i",a,b,c);
    return 0;
}
```

double

- Dezimalzahlen
- Genauigkeit : 15-stellig
- Name : `double`
- Formatzeichen : `%lf`
- Beispiel :

```c
#include <stdio.h>

int main(){
    double a = 45.72;
    double b = 19.3;
    double c = a+b;
    printf("%lf + %lf = %lf",a,b,c);
    return 0;
}
```

float

- Dezimalzahlen
- Genauigkeit : 6-stellig
- Name : `float`
- Formatzeichen : `%f`
- Beispiel :

```c
#include <stdio.h>

int main(){
    float a = 45.72;
    float b = 19.3;
    float c = a+b;
    printf("%f + %f = %f",a,b,c);
    return 0;
}
```

char

- Einzelne Zeichen
- Wertebereich : -128 bis 127
- Name : `char`
- Formatzeichen : `%c`
- Jedem ASCII-Zeichen wird eine Zahl zwischen -128 und 127 zugeordnet
  - char kann als Zahl oder Zeichen interpretiert werden
  - Formatausgabe mit `%c` interpretiert char als Zeichen
  - Formatausgabe mit `%i` interpretiert char als ganze Zahl

char - Beispiele

```c
#include <stdio.h>

int main(){
    char a = 'a';
    char b = 4;
    char c = a+b;
    printf("%c + %i = %i",a,b,c);
    return 0;
}
```

Ausgabe :

a + 4 = 101

```c
#include <stdio.h>

int main(){
    char a = 'a';
    char b = 4;
    char c = a+b;
    printf("%c + %i = %c",a,b,c);
    return 0;
}
```

Ausgabe :

a + 4 = e

Vorzeichenlos

- Standardmäßig sind alle Typen auf signed (mit Vorzeichen) eingestellt
- `unsigned` schließt negative Zahlen aus
- Wertebereich ist damit von 0 bis |negative| + positive
- Existiert für `int`, `long`, `short` und `char`
- Beispiel :

```
unsigned int x = 3;
unsigned char y = 'd';
```

Zahlenüberlauf

- Liegt eine Zahl außerhalb des Wertebereichs, kommt es zu einem Überlauf

- Sei *max* die größte Zahl im Wertebereich, *min* die kleinste. Dann folgt :

  → max + 1 = min

  → min – 1 = max

- Kommt bei `char` und `short` besonders oft vor

Zahlenüberlauf – Beispiel

```
1    #include <stdio.h>
2
3    int main(){
4        char a = 122;
5        char b = 8;
6        char c = a+b;
7        printf("%i + %i = %i",a,b,c);
8        return 0;
9    }
```

Ausgabe :

122 + 8 = -126

Typkonvertierung

- Variablen mit unterschiedlichem Typen werden in „genaueren“ Typen konvertiert
- Verlust von Genauigkeit, wenn genauerer Typ als ungenauerer Typ gespeichert wird.
- Beispiel :

```
1   #include <stdio.h>
2
3   int main(){
4       int a = -12;
5       double b = 3.5;
6       int c = a+b;
7       double d = a+b;
8       printf("c = %i\n",c);
9       printf("d = %lf",d);
10      return 0;
11  }
```

Ausgabe :

c = -8
d = -8.500000

printf-Ausgaben formatieren

- Anzahl der Nachkommastellen in der Ausgabe kann manuell eingestellt werden (der Rest wird mit 0en aufgefüllt)

```
double a = 0.14295;
printf("%.3lf“,a);
```

Ausgabe = 0.142 (3 Nachkommastellen)

- Anzahl der überflüssigen Dezimalstellen kann auch eingestellt werden (der Rest wird mit Leerzeichen (%xi) oder Nullen (%0xi) aufgefüllt)

```
int a = 12;
printf("%04i“,a);
```

Ausgabe = 0012 (4 Dezimalstellen)

Arrays

- Feld mit mehreren Variablen
- Vorstellbar wie ein Vektor
- Initialisierung :
  - `int NameDesArrays[Größe];`
- Initialisieren und Werte zuweisen
  - `int array[] = {1, 5, 3, 9};`
- Zugriff auf das i-te Element :
  - `array[i-1]`
  - `int c = array[1];` → c = 5
  - Fehlermeldung wenn es das Element nicht gibt.

Arrays – Beispiel

```
1  #include <stdio.h>
2
3  int main(){
4      int array[5] = {1, 3, 4, 2, 5};
5      printf("%i\n",array[0]);
6      printf("%i\n",array[3]);
7      return 0;
8  }
```

- Ausgabe :

  1

  2

Strings (1)

- Array aus **char**
- Vorstellbar wie eine Zeichenkette
- Initialisierung :
  - `char NameDesStrings[Größe];`
- Initialisieren und Werte zuweisen :
  - `char NameDesStrings[] = "Hallo";`

Strings (2)

- Zugriff auf das i-te Zeichen genauso wie bei Arrays :
  - `NameDesStrings[i-1]`
- Formatzeichen : `%s`
- Beispiel :

```c
#include <stdio.h>

int main(){
    char string[] = "Hallo";
    printf("%s\n",string);
    printf("%c\n",string[3]);
    return 0;
}
```

Ausgabe :

```
Hallo
l
```

Strings (3)

- Zeichen `\0` beendet einen String
  - Wird bei direkter Zuweisung von Wert automatisch hinzugefügt
  - Bei Deklaration als Array aber nicht!

- Zuweisung von Werten
  - Variablen kann leicht Wert von anderen Variablen zugewiesen werden. `int c=3;` `int d=c;`
  - Bei Arrays geht das im Allgemeinen nicht direkt
  - Und bei Strings?

Strings (4)

- Dazu wird eine weitere Bibliothek benötigt :
  - `#include <string.h>`

- `strcpy(Ziel, Quelle)`
  - Kopiert String von Quelle zu Ziel.
  - Sowohl Quelle als auch Ziel müssen Strings (bzw. char-Arrays) sein.

Strings (5)

- Beispiel :

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  int main(){
 4      char Text[] = "Hallo";
 5      char buffer[6];
 6      strcpy(buffer,Text);    // kopiert "Hallo" in buffer
 7      printf("%s\n",buffer);
 8      strcpy(buffer,"Hi"); // kopiert "Hi" in buffer
 9      printf("%s\n",buffer);
10      int i;
11      for(i=0;i<6;i++) printf("%c",buffer[i]);
12      return 0;
13  }
```

Ausgabe :

```
Hallo      H  a  l  l  o  \0
Hi         H  i  \0 l  o  \0
Hilo       H  i  \0 l  o  \0
```

Pufferüberlauf (buffer overflow)

- String wird in Puffer kopiert, welcher kürzer ist als die Länge des Strings
- Es wird in nicht dafür vorgesehenen Speicher geschrieben
  - → Es können Daten überschrieben werten

```
1  #include <stdio.h>
2  #include <string.h>
3  int main(){
4      char Zeichen = 'R';
5      char buffer[5];
6      strcpy(buffer,"Vangotest");
7      printf("%s\n",buffer);
8      printf("%c\n",Zeichen);
9      return 0;
10 }
```

Ausgabe :

```
Vangotest
t
```

3. Eingabe mit scanf()

scanf() (1)

- `scanf("Formatzeichen",Adresse);`
  - Wartet auf Eingabe und speichert sie an Adresse im Speicher
  - Adresse einer Variablen v : `&v`
  - Adresse eines Strings s : `s`

- Beispiel :

```
int i;
scanf("%d",&i);
```

scanf() (2)

- Größeres Beispiel :

```
 1  #include <stdio.h>
 2
 3  int main(){
 4      int a;
 5      double b;
 6      char c[10];
 7      printf("Bitte a eingeben\n");
 8      scanf("%i",&a);
 9      printf("a = %i\n",a);
10      printf("Bitte b eingeben\n");
11      scanf("%lf",&b);
12      printf("b = %lf\n",b);
13      printf("Bitte c eingeben\n");
14      scanf("%s",c);
15      printf("c = %s",c);
16      return 0;
17  }
```

Danke für die Aufmerksamkeit!