

C++ für Promovierende

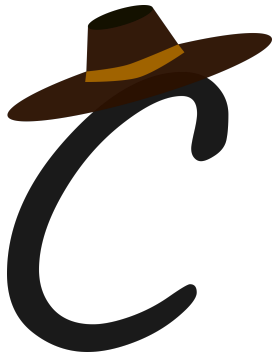
Moritz Gobbert
(gobbert@uni-trier.de)

FB IV – Datenstrukturen und Effiziente Algorithmen

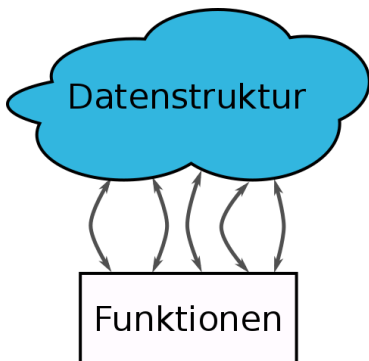
15. März 2017

- 1 Allgemeines über C++
- 2 Erste Schritte
- 3 Weitere Unterschiede zu anderen Programmiersprachen
- 4 Pointer
- 5 Objektorientiertes Programmieren
- 6 Generisches Programmieren
- 7 Aufgaben

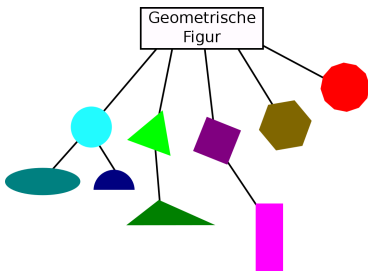
Allgemeines über C++



- „ein besseres C“

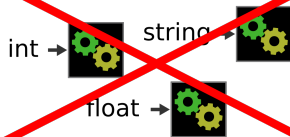


- „ein besseres C“
- Datenabstraktion



- „ein besseres C“
- Datenabstraktion
- Objektorientiert

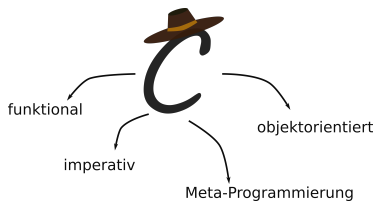
Designziele



Datentyp → [gear icon]

[Quelle: Wikimedia]

- „ein besseres C“
- Datenabstraktion
- Objektorientiert
- Generisch



- „ein besseres C“
- Datenabstraktion
- Objektorientiert
- Generisch
- Multi-Paradigmen-Sprache



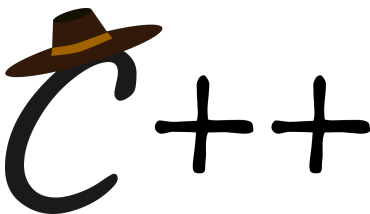
[Quelle: Bjarne Stroustrup's Homepage]

- Bjarne Stroustrup

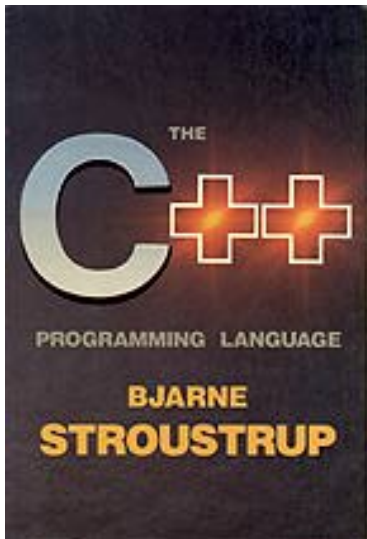


[Quelle: Daria]

- Bjarne Stroustrup
- 1979: „C with classes“

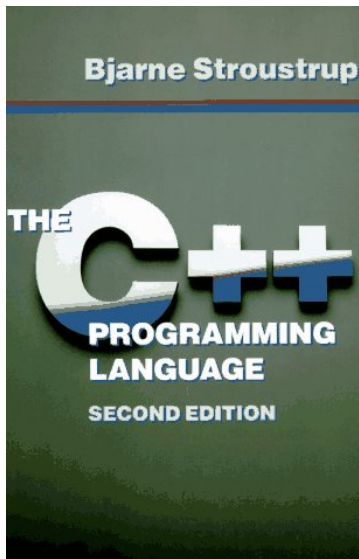


- Bjarne Stroustrup
- 1979: „C with classes“
- 1983: C++ als Name gewählt



[Quelle: Bjarne Stroustrup's Homepage]

- Bjarne Stroustrup
- 1979: „C with classes“
- 1983: C++ als Name gewählt
- 1985: „The C++ Programming Language“ (TC++PL)



[Quelle: Amazon]

- 1991: TC++PL, zweite Ausgabe



[Quelle: Wikimedia]

- 1991: TC++PL, zweite Ausgabe
- 1993: „Boolean“ und „namespaces“

Geschichte nach 1990



[Quelle: Adventure Time Wiki]

- 1991: TC++PL, zweite Ausgabe
- 1993: „Boolean“ und „namespaces“
- 1994: „The STL“



[Quelle: Standard C++]

- 1991: TC++PL, zweite Ausgabe
- 1993: „Boolean“ und „namespaces“
- 1994: „The STL“
- 1998: C++98

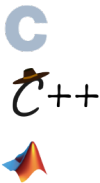


- 1983: C++



- 1972: C
- 1983: C++

[Quelle: Wikimedia]



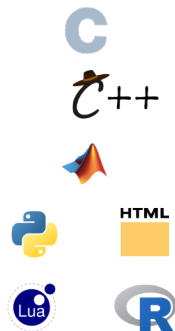
- 1972: C
- 1983: C++
- 1984: MATLAB

[Quelle: MATLAB - MathWorks]



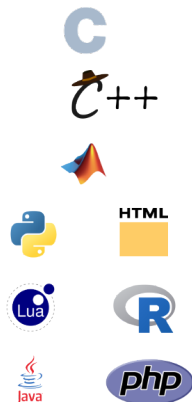
- 1972: C
- 1983: C++
- 1984: MATLAB
- 1991: PYTHON & HTML

[Quelle: Wikimedia] & [Quelle: Wikimedia]



- 1972: C
- 1983: C++
- 1984: MATLAB
- 1991: PYTHON & HTML
- 1993: LUA & R

[Quelle: Wikimedia] & [Quelle: Wikimedia]



- 1972: C
- 1983: C++
- 1984: MATLAB
- 1991: PYTHON & HTML
- 1993: LUA & R
- 1995: JAVA & PHP

[Quelle: Wikipedia] & [Quelle: Wikimedia]



[Quelle: Wikimedia]

- 1972: C
- 1983: C++
- 1984: MATLAB
- 1991: PYTHON & HTML
- 1993: LUA & R
- 1995: JAVA & PHP
- 2000: C#

Wo nutzt man C++?



Adobe

[Quelle: Wikimedia]

- **Adobe:** Photoshop, Acrobat, ...

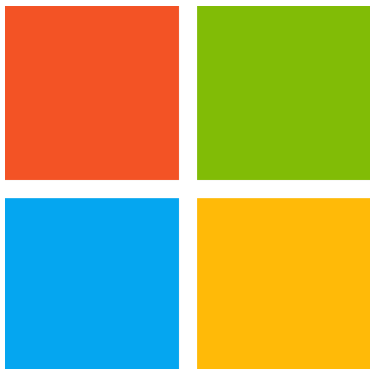
Wo nutzt man C++?



[Quelle: Wikimedia]

- **Adobe**: Photoshop, Acrobat, ...
- **Apple**: iOS, macOS, ...

Wo nutzt man C++?



[Quelle: Wikimedia]

- **Adobe**: Photoshop, Acrobat, ...
- **Apple**: iOS, macOS, ...
- **Microsoft**: Internet Explorer, C#-Compiler, ...

Wo nutzt man C++?



[Quelle: NASA Jet Propulsion Laboratory]

- **Adobe:** Photoshop, Acrobat, ...
- **Apple:** iOS, macOS, ...
- **Microsoft:** Internet Explorer, C#-Compiler, ...
- **NASA:** Mars-Rover Steuerung, ...

Wo nutzt man C++?



[Quelle: Wikipedia]

- **Adobe**: Photoshop, Acrobat, ...
- **Apple**: iOS, macOS, ...
- **Microsoft**: Internet Explorer, C#-Compiler, ...
- **NASA**: Mars-Rover Steuerung, ...
- **Computerspiele**: Unreal Engine, Source, Elder Scrolls, ...

Erste Schritte



[Quelle: Wikimedia]

- „Standard“-Compiler: **GCC** (GNU Compiler Collection)



[Quelle: Wikimedia]

- „Standard“-Compiler: **GCC** (GNU Compiler Collection)
- C, C++, JAVA, FORTRAN, ...



[Quelle: Wikimedia]

- „Standard“-Compiler: **GCC** (GNU Compiler Collection)
- C, C++, JAVA, FORTRAN, ...
- Varianten für Windows



[Quelle: Wikimedia]

- „Standard“-Compiler: **GCC** (GNU Compiler Collection)
- C, C++, JAVA, FORTRAN, ...
- Varianten für Windows
- **GCC** ist geschrieben in C++

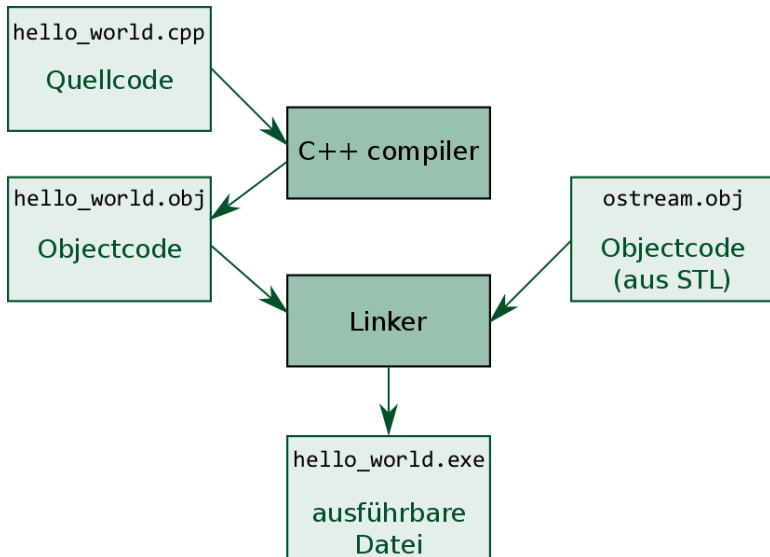
Hello, world!

hello_world.cpp:

```
#include <iostream>

int main(){
    std::cout << "Hello, world!\n";
    return 0;
}
```

Kompilieren & Linken



`g++ hello_world.cpp`: kompiliert „hello_world.cpp“ und erzeugt eine ausführbare Datei „a.out“.

<code>-o outfile</code>	Benennt die ausführbare Datei <code>outfile</code> statt <code>a.out</code>
<code>-O3</code>	Optimiert den Code vor dem Kompilieren – erhöht sowohl Dauer des Kompilierens, als auch Performance des Programms
<code>-g</code>	Erzeugt Debug-Informationen, die u. a. von <code>GDB</code> (GNU Project Debugger) genutzt werden
<code>-Wall</code>	Aktiviert allgemeine Warnungen, die von GCC unterstützt werden
<code>-std=c++14</code>	Kompiliert unter Nutzung des „2014 ISO C++“ Standards – benötigt passende Version des Compilers


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

int main(){
    cout << "Was ist deine Lieblingsfarbe? ";
    string farbe;
    cin >> farbe;
    cout << farbe +
        " ist eine schoene Farbe!" << endl;

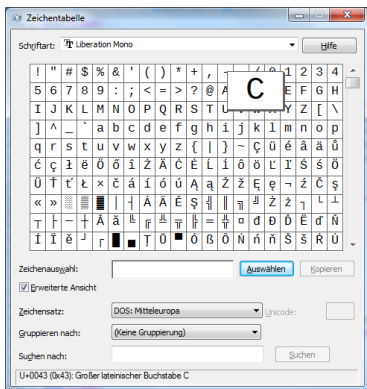
    return 0;
}
```



[Quelle: Wikimedia]

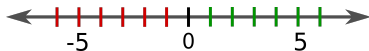
- `bool`

Datentypen



- bool
- char

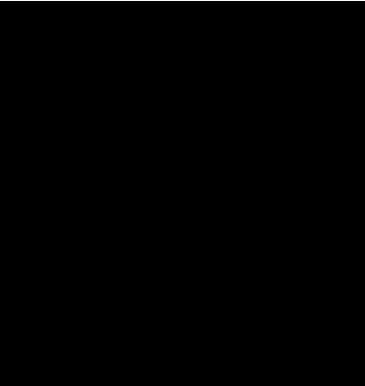
Datentypen



- `bool`
- `char`
- `int` (inkl. Varianten)

17.5923

- `bool`
- `char`
- `int` (inkl. Varianten)
- `float` (und `double`)

- 
- `bool`
 - `char`
 - `int` (inkl. Varianten)
 - `float` (und `double`)
 - `void`

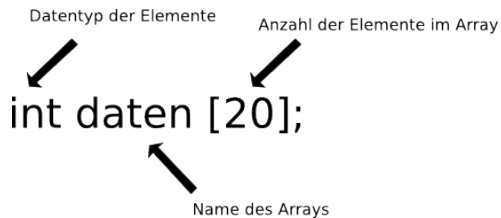
- Sammlung von Elementen gleichen Typs
- kein eigener Typ, kein Objekt
- statische Länge
- „gültige“ Indizes: 0 bis Länge -1
- keine Laufzeitüberprüfung auf gültigen Index

Initialisierung von Arrays

Datentyp der Elemente Anzahl der Elemente im Array

int daten [20];

Name des Arrays

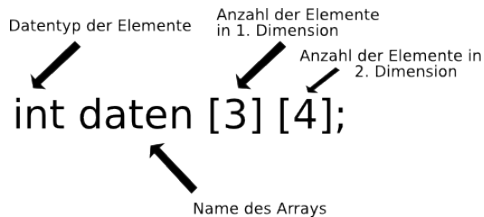


Initialisierung von Arrays

Datentyp der Elemente Anzahl der Elemente in 1. Dimension Anzahl der Elemente in 2. Dimension

`int daten [3] [4];`

Name des Arrays



Array-Beispiel

```
int zahlen[5]; // Array mit Positionen 0 bis 4

zahlen[2] = 674; // weist dem dritten Element
                 den Wert 674 zu
cout << zahlen[2] << endl;

// das Element wurde nicht initialisiert
cout << zahlen[0] << endl;
// Indizes liegen ausserhalb der Grenzen
cout << zahlen[9] << endl;
cout << zahlen[-5] << endl;
```

Array-Beispiel

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ -o arrays.o arrays.cpp
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./arrays.o
674
4196624
0
32693
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./arrays.o
674
4196624
0
32571
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./arrays.o
674
4196624
0
32662
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./arrays.o
674
4196624
0
32557
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

Array-Beispiel



Eigenschaften von Strings

- Variable Länge
- Konkatenation mit „+“
- „Array“ von Zeichen
- Vergleichsoperationen anwendbar (==, <, <=, ...)
- Member-Funktionen (`substr()`, `find()`, ...)

String-Beispiel

```
string s, r;

cout << "Bitte ein Wort eingeben: ";
cin >> s;

for(int i = s.length()-1; i >= 0; i--){
    r += s[i];
}

if(s == r){
    cout << s + " ist ein Palindrom!" << endl;
}else{
    cout << s + " ist kein Palindrom!" << endl;
}
```

String-Beispiel

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ -o palindrom.o palindrom.cpp
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./palindrom.o
Bitte ein Wort eingeben: Palindrom
Palindrom ist kein Palindrom!
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./palindrom.o
Bitte ein Wort eingeben: reliefpfeiler
reliefpfeiler ist ein Palindrom!
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

- Funktionen überladen
- Default-Parameter
- *call-by-reference*: eigentliches Objekt übergeben
- *call-by-value*: Kopie eines Objekts/Werts übergeben

Überladungs-Beispiel

```
int summe(int a, int b, int c){
    return a + b + c;
}

int summe(int a, int b){
    return a + b;
}

int main(){
    int i = 1, j = 2, k = 3;

    cout << summe(i,j,k) << endl; // 1 + 2 + 3 = 6
    cout << summe(i,j) << endl;   // 1 + 2 = 3

    return 0;
}
```

Default-Parameter-Beispiel

```
int summe(int a, int b, int c = 0){  
    return a + b + c;  
}  
  
int main(){  
    cout << summe(1, 2, 3) << endl;    // 1+2+3 = 6  
    cout << summe(1, 2) << endl;        // 1+2+0 = 3  
    cout << summe(1) << endl; // error: "too few  
        arguments"  
  
    return 0;  
}
```

Parameterübergabe-Beispiel

```
void by_value(int i, int j){  
    i += j;  
    cout << i << endl;  
}  
  
void by_reference(int& i, int& j){  
    i += j;  
    cout << i << endl;  
}
```

Parameterübergabe-Beispiel

```
int main(){  
    int i = 5, j = 3;  
  
    by_value(i, j);  
    cout << i << endl;  
  
    by_reference(i, j);  
    cout << i << endl;  
  
    return 0;  
}
```

Parameterübergabe-Beispiel

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ fkt_parameter.cpp -o fkt_parameter.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./fkt_parameter.o
8
5
8
8
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

Weitere Unterschiede zu anderen Programmiersprachen

„Kleinstes“ Programm

hello_world.cpp:

```
int main(){  
    std::cout << "Hello, world  
    !\n";  
    return 0;  
}
```

- C++ : Funktion

„Kleinstes“ Programm

hello_world.java:

```
class hello_world{  
    public static void main(  
        String[] args){  
        System.out.println("Hello  
            , world!");  
    }  
}
```

- C++ : Funktion
- JAVA : Klasse

„Kleinstes“ Programm

hello_world.lua/.py/.r:

```
print("Hello, world!")
```

- C++ : Funktion
- JAVA : Klasse
- LUA : Anweisung
- PYTHON : Anweisung
- R : Anweisung

„Kleinstes“ Programm

hello_world.php:

```
<?php
    echo "Hello , world!"
?>
```

- C++ : Funktion
- JAVA : Klasse
- LUA : Anweisung
- PYTHON : Anweisung
- R : Anweisung
- PHP : Anweisung (+ PHP-Tag)

„Kleinstes“ Programm

hello_world.m:

```
disp("Hello , world!")
```

- C++ : Funktion
- JAVA : Klasse
- LUA : Anweisung
- PYTHON : Anweisung
- R : Anweisung
- PHP : Anweisung (+ PHP-Tag)
- MATLAB : Anweisung

C++ unterstützt zusammengesetzte Operatoren:

```
int z = 22;
```

```
z += 5; // z = z + 5; (z == 27)
```

```
z -= 7; // z = z - 7; (z == 20)
```

```
z *= 2; // z = z * 2; (z == 40)
```

```
z /= 4; // z = z / 4; (z == 10)
```

C++ unterstützt zusammengesetzte Operatoren:

```
int z = 22;
```

```
z += 5; // z = z + 5; (z == 27)
```

```
z -= 7; // z = z - 7; (z == 20)
```

```
z *= 2; // z = z * 2; (z == 40)
```

```
z /= 4; // z = z / 4; (z == 10)
```

Diese existieren nicht in **LUA**, **R** oder **MATLAB**

Inkrement und Dekrement

C++ unterstützt **Inkrement**- und **Dekrement**-Operatoren – jeweils in der Präfix- und der Postfix-Variante.

```
int i = 1, j = 1;
```

```
int a, b, c, d;
```

```
a = i++; // a = i; i = i+1; (a == 1, i == 2)
```

```
b = i--; // b = i; i = i-1; (b == 2, i == 1)
```

```
c = ++j; // j = j+1; c = j; (c == 2, j == 2)
```

```
d = --j; // j = j-1; d = j; (c == 1, j == 1)
```

Inkrement und Dekrement

C++ unterstützt **Inkrement**- und **Dekrement**-Operatoren – jeweils in der Präfix- und der Postfix-Variante.

```
int i = 1, j = 1;
```

```
int a, b, c, d;
```

```
a = i++; // a = i; i = i+1; (a == 1, i == 2)
```

```
b = i--; // b = i; i = i-1; (b == 2, i == 1)
```

```
c = ++j; // j = j+1; c = j; (c == 2, j == 2)
```

```
d = --j; // j = j-1; d = j; (c == 1, j == 1)
```

Diese existieren nicht in PYTHON, LUA, R oder MATLAB

- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht

Operatoren allgemein

- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht
- Modulo (%) $\rightarrow$ (R: %%)

Operatoren allgemein

- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht
- Modulo (%) → (R: %%)
- Gleichheit (==)

- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht
- Modulo (%) → (R: %%)
- Gleichheit (==)
- „Ungleichheit“ (!=) → (MATLAB, LUA: ~=)

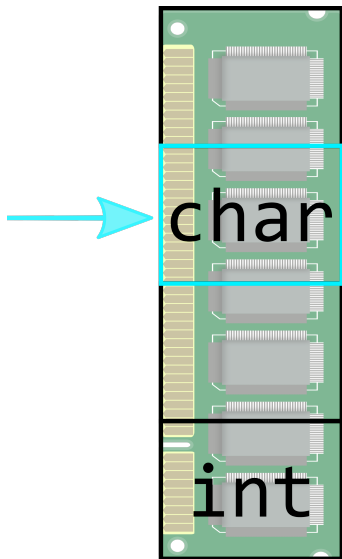
- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht
- Modulo (%) → (R: %%)
- Gleichheit (==)
- „Ungleichheit“ (!=) → (MATLAB, LUA: ~=)
- Vergleiche (<, >, <=, >=)

Operatoren allgemein

- Arithmetik (+, -, *, /); Potenzierung (^, **) existiert nicht
- Modulo (%) → (R: %%)
- Gleichheit (==)
- „Ungleichheit“ (!=) → (MATLAB, LUA: ~=)
- Vergleiche (<, >, <=, >=)
- Logik (&&, ||, !) → (PYTHON, LUA: and, or, not; R: &, |, !; MATLAB: &, |, ~)

Pointer

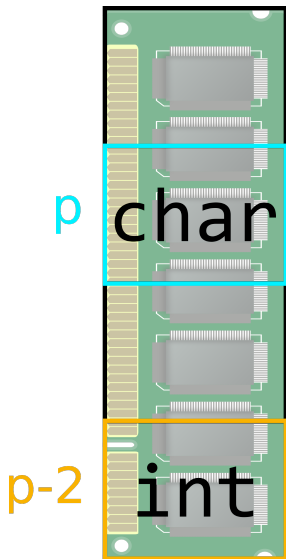
Pointer in C++



- Adressen von Variablen

[Quelle: Wikimedia]

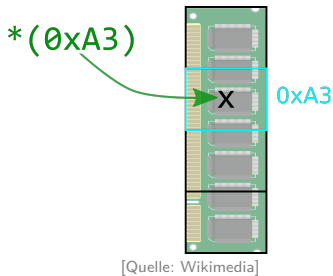
Pointer in C++



[Quelle: Wikimedia]

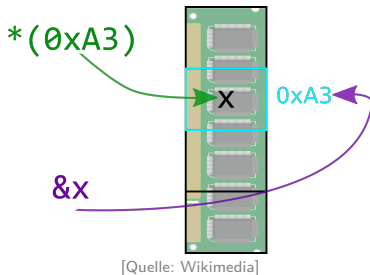
- Adressen von Variablen
- „Pointerarithmetik“

Pointer in C++



- Adressen von Variablen
- „Pointerarithmetik“
- `*`: Adresse $\rightarrow$ Variable

Pointer in C++



- Adressen von Variablen
- „Pointerarithmetik“
- `*`: Adresse $\rightarrow$ Variable
- `&`: Variable $\rightarrow$ Adresse

Pointer-Beispiele

```
int    x; // Integer
int*   p; // Pointer auf Integer

p = &x; // p = Address-of(x)
*p = 1; // Entspricht x = 1
*p = *p + 1; // Entspricht x = x + 1
```

Pointer-Beispiele

```
int    x; // Integer
int*   p; // Pointer auf Integer

p = &x; // p = Address-of(x)
*p = 1; // Entspricht x = 1
*p = *p + 1; // Entspricht x = x + 1
```

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ pointer1.cpp -o pointer1.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./pointer1.o
x: 2
*p: 2
&x: 0x7ffd7dda1244
&p: 0x7ffd7dda1248
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

Pointer-Beispiele

```
int x;    // Integer
int y;    // Integer
int* px;  // Pointer auf Integer
int* py;  // Pointer auf Integer

px = &x;  // px = Address-of(x)
*px = 1;  // entspricht x = 1
py = px+1; // ?
*py = 2;   // ?
```

Pointer-Beispiele

```
int x;    // Integer
int y;    // Integer
int* px;  // Pointer auf Integer
int* py;  // Pointer auf Integer

px = &x;  // px = Address-of(x)
*px = 1;  // entspricht x = 1
py = px+1; // ?
*py = 2;   // ?
```

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ pointer2.cpp -o pointer2.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./pointer2.o
x: 1
y: 2
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

Pointer-Beispiele

```
void tausche(int* a, int* b){  
    int t;  
    t = *a;  
    *a = *b;  
    *b = t;  
}
```

Pointer-Beispiele

```
void tausche(int* a, int* b){  
    int t;  
    t = *a;  
    *a = *b;  
    *b = t;  
}
```

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ tauschen.cpp -o tauschen.o  
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./tauschen.o  
Vorher: x: 2; y: 8  
tausche(&x, &y);  
Nachher: x: 8; y: 2  
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```



```
int a[n];
```

- Array ist **Pointer** auf erstes Element
- $a == \&(a[0])$
- $a[i] == *(a + i)$
- C++ garantiert, dass $a+0$ bis $a+n-1$ nicht überschrieben werden

Objektorientiertes Programmieren

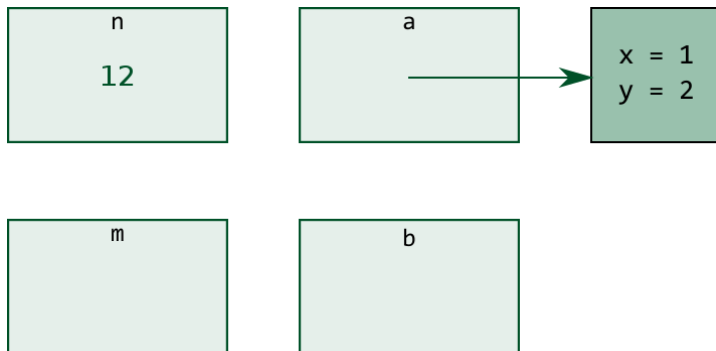
Klasse: Datenstruktur mit Daten und darauf arbeitenden Funktionen

Objekt: Individuelles Exemplar einer Klasse

⇒ Kapselung von Datenstrukturen

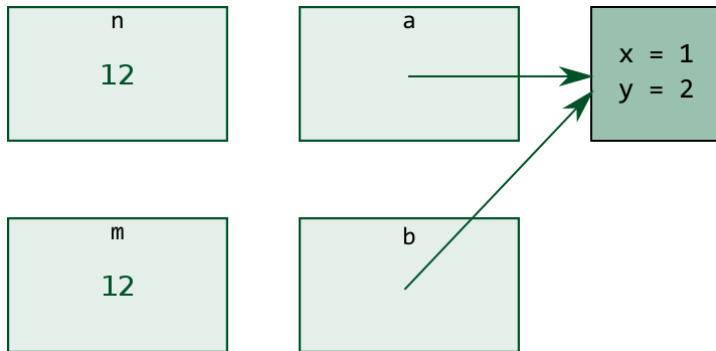
Objekte in JAVA

```
int n = 12;  
int m;  
point a = new point(1,2);  
point b;
```



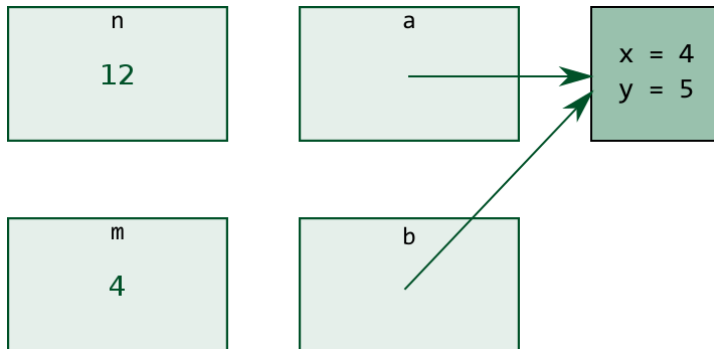
Objekte in JAVA

```
m = n;  
b = a;
```



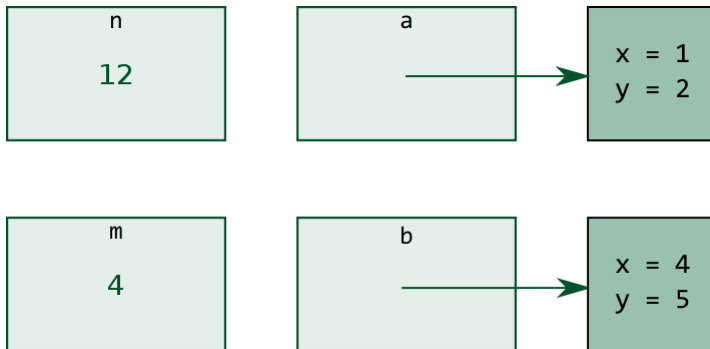
Objekte in JAVA

```
m = 4;  
b.x = 4;  
b.y = 5;
```



Objekte in C++

Gleiche Anweisungen wie vorhin:



Objekte in JAVA und C++ im Vergleich

JAVA:

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ javac objekte.java
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ java objekte
n = 12
m = "error: variable m might not have been initialized"
a = point@75ba3523 (1, 2)
b = "error: variable b might not have been initialized"

n = 12
m = 4
a = point@75ba3523 (4, 5)
b = point@75ba3523 (4, 5)
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

C++:

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ objekte.cpp -o objekte.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ ./objekte.o
n = 12
m = 0
a = 0x7ffde69e3f80 (1, 2)
b = 0x7ffde69e3f90 (4197440, 0)

n = 12
m = 4
a = 0x7ffde69e3f80 (1, 2)
b = 0x7ffde69e3f90 (4, 5)
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```


- Objektdекlaration erzeugt Objekt (ohne `new`)
- Objekt-Variablen enthalten Werte, keine Referenzen
- Keine Standard-Initialisierung
- Keine automatische „Garbage Collection“

```
class Point{  
public:  
    Point();  
    Point(int new_x, int new_y);  
    void move(int dx, int dy);  
    void print();  
  
private:  
    int x;  
    int y;  
};
```

```
class Point{  
public:  
    Point();  
    Point(int new_x, int new_y);  
    void move(int dx, int dy);  
    void print();  
  
private:  
    int x;  
    int y;  
};
```

Keine Funktionsdefinitionen?

Klassen-Beispiel

```
Point::Point(){ x = 0; y = 0; }
```

```
Point::Point(int new_x, int new_y){ x = new_x; y  
    = new_y; }
```

```
void Point::move(int dx, int dy){  
    x = x + dx;  
    y = y + dy;  
}
```

```
void Point::print(){  
    cout << "(" << x << ", " << y << " )";  
}
```

Zugriff auf Daten und Funktionen

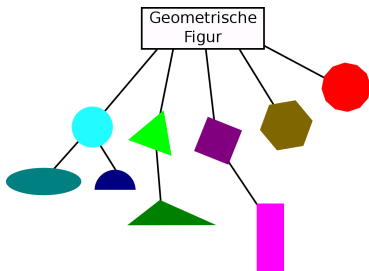
- Daten eines Objekts werden durch den Operator `.` (Punkt) angesprochen:

```
Point a(2, 6);  
a.move(2, -2);
```

- Daten eines Objekt-Pointers werden durch den Operator `->` (Pfeil) angesprochen:

```
Point* a_pointer = &a;  
a_pointer->print();
```

- `p->f()` ist Kurzschreibweise für `(*p).f()`



- Vererbung erweitert Klassen
- „base class“ vererbt Variablen und Methoden an „derived classes“
- Ähnliche Klassen können gleiche Variablen und Methoden aus einer gemeinsamen Oberklasse beziehen

Vererbungs-Beispiel

```
class Polygon{  
protected:  
    double w, h;  
public:  
    void set(double a, double b){  
        w = a;  
        h = b;  
    }  
};
```

Vererbungs-Beispiel

```
class Rechteck: public Polygon{
public:
    double flaeche(){
        return w * h;
    }
};
```

```
class Dreieck: public Polygon{
public:
    double flaeche(){
        return w * h / 2;
    }
};
```



```
Dreieck d;  
Rechteck r;  
  
d.set(2,4); // set() von Polygon geerbt  
r.set(5,3);  
  
cout << "Dreieck: " << d.flaeche() << endl; //  
    gibt 4 aus  
cout << "Rechteck: " << r.flaeche() << endl; //  
    gibt 15 aus
```

Vererbung in anderen Sprachen

- JAVA: `class NewClass extends OldClass { ... }`
- PHP: wie JAVA

Vererbung in anderen Sprachen

- JAVA: `class NewClass extends OldClass { ... }`
- PHP: wie JAVA
- PYTHON: `class NewClass(OldClass): ...`

Vererbung in anderen Sprachen

- JAVA: `class NewClass extends OldClass { ... }`
- PHP: wie JAVA
- PYTHON: `class NewClass(OldClass): ...`
- R: `setClass("NewClass", contains "OldClass")`

Vererbung in anderen Sprachen

- JAVA: `class NewClass extends OldClass { ... }`
- PHP: wie JAVA
- PYTHON: `class NewClass(OldClass): ...`
- R: `setClass("NewClass", contains "OldClass")`
- MATLAB: `classdef NewClass < OldClass`

Befreundete Klassen



[Quelle: Hermann Kern, 1904]

- Befreundete Klassen haben Zugriff auf die privaten Variablen und Methoden ihrer Freunde
- Sowohl ganze Klassen als auch einzelne Methoden können Freunde einer Klasse sein
- Widerspricht dem Prinzip der Datenkapselung

```
class Box{  
private:  
    string inhalt;  
    int position;  
    friend class Regal;  
public:  
    Box(string in){ inhalt = in; position = 0; }  
    void print(){ cout << inhalt << " an Position "  
        << position << endl; }  
};
```

```
class Regal{
public:
    void verschiebe_box(Box& b, int pos){
        b.position = pos; }
    void aendere_inhalt(Box& b, string neu){
        b.inhalt = neu; }
};
```



```
Box kiste("Erdbeeren");  
Regal obst;  
  
kiste.print(); // Erdbeeren an Position 0  
obst.verschiebe_box(kiste, 7);  
obst.aendere_inhalt(kiste, "Orangen");  
kiste.print(); // Orangen an Position 7
```

Freundschafts-Beispiel 2

```
class Box{  
private:  
    string inhalt;  
    int position;  
    friend void verschiebe_box(Box&, int);  
public:  
    Box(string in){ inhalt = in; position = 0; }  
    void print(){ cout << inhalt << " an Position "  
        << position << endl; }  
};
```

Freundschafts-Beispiel 2

```
void verschiebe_box(Box& b, int pos){
    b.position = pos;
}

int main(){
    Box kiste("Erdbeeren");

    kiste.print(); // Erdbeeren an Position 0
    verschiebe_box(kiste, 7);
    kiste.print(); // Erdbeeren an Position 7

    return 0;
}
```

Freundschafts-Beispiel 2

```
void aendere_inhalt(Box& b, string neu){  
    b.inhalt = neu;  
}
```

Freundschafts-Beispiel 2

```
void aendere_inhalt(Box& b, string neu){  
    b.inhalt = neu;  
}
```

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $ g++ freundschaft_methoden.cpp -o freun  
dschaft_methoden.o  
freundschaft_methoden.cpp: In function 'void aendere_inhalt(Box&, std::string)':  
freundschaft_methoden.cpp:6:9: error: 'std::string Box::inhalt' is private  
    string inhalt;  
        ^  
freundschaft_methoden.cpp:19:4: error: within this context  
    b.inhalt = neu;  
    ^  
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/Kleines Studienprojekt $
```

- Aufteilung von Klassen-Definition und -Implementierung
- Definition in Header-Datei (*.h)
- Implementierung in üblicher „implementation“-Datei (*.cpp)

- Aufteilung von Klassen-**Definition** und **-Implementierung**
- Definition in **Header**-Datei (*.h)
- Implementierung in üblicher „implementation“-Datei (*.cpp)
- Einbinden der Header-Datei mit Schlüsselwort `#include`

- Aufteilung von Klassen-**Definition** und **-Implementierung**
- Definition in **Header**-Datei (*.h)
- Implementierung in üblicher „implementation“-Datei (*.cpp)
- Einbinden der Header-Datei mit Schlüsselwort `#include`
- „include guard“ um mehrfaches Einbinden zu verhindern

Header-Beispiel

katze.h

```
#include <iostream>

class Katze{
    private:
        std::string name; // Name der Katze
        int energie; // Zahl zwischen 0 und 60
    public:
        Katze(std::string); // Konstruktor
        void schlafen(int); // erhoeht energie
        void spielen(int); // verringert energie
        std::string status(); // beschreibt energie
};
```

Header-Beispiel

katze.cpp

```
#include "katze.h"
using namespace std;

Katze::Katze(string n = "Kitty"){
    name = n;
    energie = 10;
}

void Katze::schlafen(int dauer){
    // ...
}
// ...
```

Header-Beispiel

katze_test.cpp

```
#include "katze.h"
using namespace std;

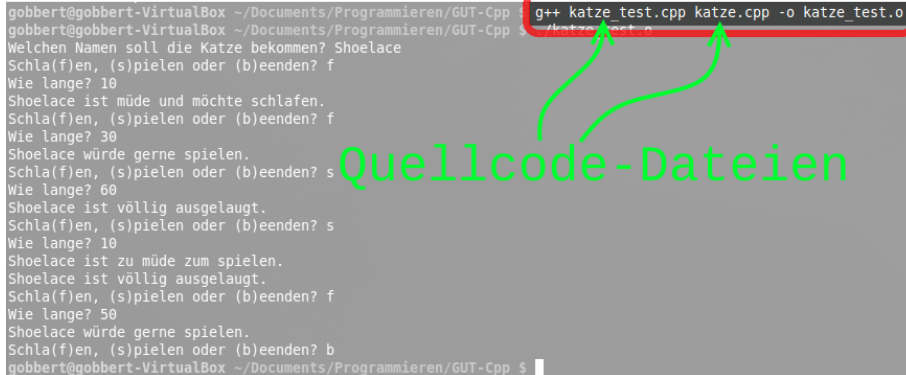
int main(){
    // Variableninitialisierung
    cout << "Wie heisst die Katze? ";
    string name;
    cin >> name;
    Katze k(name);
    string input;
    int dauer;
    // ...
}
```

Header-Beispiel

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $ g++ katze_test.cpp katze.cpp -o katze_test.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $ ./katze_test.o
Welchen Namen soll die Katze bekommen? Shoelace
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 10
Shoelace ist müde und möchte schlafen.
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 30
Shoelace würde gerne spielen.
Schla(f)en, (s)pielen oder (b)eenden? s
Wie lange? 60
Shoelace ist völlig ausgelaugt.
Schla(f)en, (s)pielen oder (b)eenden? s
Wie lange? 10
Shoelace ist zu müde zum spielen.
Shoelace ist völlig ausgelaugt.
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 50
Shoelace würde gerne spielen.
Schla(f)en, (s)pielen oder (b)eenden? b
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $
```

Header-Beispiel

```
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $ g++ katze_test.cpp katze.cpp -o katze_test.o
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $ ./katze_test.o
Welchen Namen soll die Katze bekommen? Shoelace
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 10
Shoelace ist müde und möchte schlafen.
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 30
Shoelace würde gerne spielen.
Schla(f)en, (s)pielen oder (b)eenden? s
Wie lange? 60
Shoelace ist völlig ausgelaugt.
Schla(f)en, (s)pielen oder (b)eenden? s
Wie lange? 10
Shoelace ist zu müde zum spielen.
Shoelace ist völlig ausgelaugt.
Schla(f)en, (s)pielen oder (b)eenden? f
Wie lange? 50
Shoelace würde gerne spielen.
Schla(f)en, (s)pielen oder (b)eenden? b
gobbert@gobbert-VirtualBox ~/Documents/Programmieren/GUT-Cpp $
```



Quellcode-Dateien

Include Guard

grandfather.h

```
class foo {  
    int member;  
};
```

father.h

```
#include "grandfather.h"
```

child.cpp

```
#include "grandfather.h"  
#include "father.h"
```

Include Guard

grandfather.h

```
class foo {  
    int member;  
};
```

father.h

```
#include "grandfather.h"
```

child.cpp

```
#include "grandfather.h"  
#include "father.h"
```

⇒ verletzt „one definition“-Regel (foo doppelt definiert)

Include Guard

grandfather.h (father.h und child.cpp unverändert)

```
#ifndef GF // wenn GF nicht definiert...
#define GF // ...definiere GF und fahre fort

class foo {
    int member;
};

#endif // Ende der Verzweigung
```


Include Guard

grandfather.h (father.h und child.cpp unverändert)

```
#ifndef GF // wenn GF nicht definiert...
#define GF // ...definiere GF und fahre fort

class foo {
    int member;
};

#endif // Ende der Verzweigung
```

Erstes #include "grandfather.h" setzt GF; zweites #include "grandfather.h" tut nichts

Generisches Programmieren

Templates

Motivation: Gleichartige Funktionen (oder Klassen), die sich im Typ unterscheiden, in einer Definition zusammenfassen

```
void tausche(int* a, int* b){  
    int t = *a; *a = *b; *b = t;  
}  
  
void tausche(double* a, double* b){  
    double t = *a; *a = *b; *b = t;  
}  
  
void tausche(string* a, string* b){  
    double t = *a; *a = *b; *b = t;  
}
```

Templates

Motivation: Gleichartige Funktionen (oder Klassen), die sich im Typ unterscheiden, in einer Definition zusammenfassen

```
template <typename T>
void tausche(T* a, T* b){
    T t = *a; *a = *b; *b = t;
}
```

⇒ Templates sind Schablonen für Funktions- und Klassendefinitionen

Funktions-Templates im Compiler

Wenn im Programm die Funktion aufgerufen wird, definiert der Compiler automatisch die zum Typ passende Funktion:

```
double x, y;  
tausche(x, y);
```

definiert die Funktion

```
void tausche(double* a, double* b){ /* ... */ }
```

- Schablone für Klassen
- Formale Parameter sind Platzhalter für die (in die Schablone einzusetzenden) Daten
- Formale Parameter werden bei Generierung durch konkrete Daten ersetzt

Klassen-Templates (Beispiel)

```
template <typename E>
class Point{
public:
    Point(E new_x, E new_y);
    void move(E dx, E dy);
    void print();

private:
    E x;
    E y;
};
```

Klassen-Templates (Beispiel)

```
template <typename E>
Point<E>::Point(E new_x, E new_y){ x = new_x; y =
    new_y; }

template <typename E>
void Point<E>::move(E dx, E dy){
    x = x + dx;
    y = y + dy;
}

template <typename E>
void Point<E>::print(){
    cout << "(" << x << ", " << y << ")";
}
```


Klassen-Templates (Beispiel)

```
Point<double> a(1.2, 0.7);  
a.move(0.4, 1.2);  
a.print(); // a = (1.6, 1.9)  
  
Point<string> b("ab", "cd");  
b.move("xy", "xz");  
b.print(); // b = (abxy, cdxz)  
  
Point<bool> c(false, false);  
c.move(true, false);  
c.print(); // c = (true, false)
```

Aufgaben

Unterlagen (inkl. Links zu Compilern) finden Sie unter:
<https://www.uni-trier.de/index.php?id=59534>.

Aufgabe 1 – Strings

Schreiben Sie ein Programm, welches ein Wort einliest und zählt, wie oft in diesem Wort die Buchstabenkombination „ei“ vorkommt.

Erweitern Sie das Programm so, dass es eine vom Benutzer eingegebene Buchstabenkombination zählt.

Aufgabe 2 – Arrays

Schreiben Sie ein Programm, welches ein Array `int zahlen[10]` mit Zahlen (Benutzereingabe oder Zufallszahlen) füllt und dann den Mittelwert bestimmt.

Aufgabe 3 – Funktionen

Implementieren Sie sowohl die **Fakultäts-** als auch die **Fibonacci**funktion. Im Hauptprogramm soll der Benutzer gefragt werden, welche der beiden Funktionen er gerne berechnen möchte und für welche Zahl n . Nach der Berechnung soll die Lösung ausgegeben werden.

Zur Erinnerung:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$
$$f_n = \begin{cases} 1, & n \leq 2 \\ f_{n-1} + f_{n-2}, & n > 2 \end{cases}$$

Aufgabe 4 – Funktionsüberladung

Schreiben Sie eine Funktion `max(a, b)`, die für `Integer` `a`, `b` die größere der beiden und für `Strings` `a`, `b` den längeren der beiden zurückgibt.

Aufgabe 5 – Parameterübergabe

Schreiben Sie eine Funktion `void triple()`, die eine aus dem Hauptprogramm übergebene Zahl verdreifacht.

Falls Sie andere Programmiersprachen beherrschen: Wie würden Sie diese Funktion dort implementieren?

Aufgabe 6 – Adressen im Speicher

Initialisieren Sie mehrere Variablen sowohl gleicher als auch verschiedener Typen und geben Sie die jeweiligen Adressen aus. Was fällt Ihnen auf?

(Hinweis: `sizeof(var)` gibt die Größe des zu `var` gehörigen Datentyps zurück.)

Aufgabe 7 – Klassen

Schreiben Sie eine Klasse `Kreis` mit Daten

- `int x` – Position des Kreises
- `int y` – Position des Kreises
- `double r` – Radius des Kreises

und folgenden Funktionen neben den Konstruktoren

- `void move(int int)` – verschiebt den Kreis
- `double umfang()` – berechnet den Umfang
- `double flaeche()` – berechnet die Fläche
- `void scale(double)` – skaliert den Radius

(**Hinweis:** Nutzen Sie für π direkt 3.14 anstatt die Konstante über eine passende Bibliothek einzubinden.)

Aufgabe 8 – Vererbung

Schreiben Sie eine Klasse `Kugel`, die die Variablen und Methoden der Klasse `Kreis` aus der vorigen Aufgabe erbt. Erweitern Sie die Klasse wie folgt:

- `int z` – Position der Kugel
- `double volumen()` – berechnet das Volumen
- `double oberflaeche()` – berechnet die Kugeloberfläche
- `void move(int int int)` – verschiebt die Kugel nun in drei Dimensionen

(**Hinweis:** Es reicht, wenn sie für π Näherungswerte verwenden.)

Aufgabe 9 – Klassen-Templates

Schreiben Sie ein Klassen-Template `stack<T>` mit Daten

- `stack_elem<T>* top` – Verweis auf das oberste Element (schreiben Sie hierzu auch ein Klassen-Template `stack_elem<T>`, welches einen Wert vom Typ `T` und einen Verweis auf das nächste Element speichert)
- `int size` – Anzahl der Element im Stack

und folgenden Funktionen neben den Konstruktoren

- `bool is_empty()` – gibt an, ob der Stack leer ist
- `stack_elem<T> pop` – gibt das oberste Element zurück und entfernt es vom Stack
- `void push(stack_elem<T>)` – legt ein Element auf dem Stack ab