PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

AGENDA FÜR HEUTE

- Quiz zur Wiederholung
- Klassen (erstmal ohne Vererbung)
  - Destruktiven
  - RAII
- Übungen

WARUM GEBEN WIR HIER RETURN 0 AN?

```
#include <iostream>

int main(){
    printf("Hello World");
    return 0;

}
```

WAS IST HIER DEKLARATION UND WAS DEFINITION?

```
#include <iostream>

void printElements(int *int_array, size_t length);
void printElements(int *int_array, size_t length)
{
    for (int i = 0; i < length; ++i)
    {
        printf("[%d]: %d\n", i, int_array[i]);
    }
}

int main()
{
    int int_array[6] = {13,24,32,42,51,633};
    printElements(int_array,6);
}
```

DARF MAN SO AUF EINEN POINTER ZUGREIFEN?

```
#include <iostream>

void printElements(int *int_array, size_t length);
void printElements(int *int_array, size_t length)
{
    for (int i = 0; i < length; ++i)
    {
        printf("[%d]: %d\n", i, int_array[i]);
    }
}

int main()
{
    int int_array[6] = {13,24,32,42,51,633};
    printElements(int_array,6);
}
```

UNTERSCHIED ZWISCHEN ARRAYS UND POINTER?

```
#include <iostream>

void printElements(int *int_array, size_t length);
void printElements(int *int_array, size_t length)
{
    for (int i = 0; i < length; ++i)
    {
        printf("[%d]: %d\n", i, int_array[i]);
    }
}

int main()
{
    int int_array[6] = {13,24,32,42,51,633};
    printElements(int_array,6);
}
```

WAS IST DAS PROBLEM MIT DIESEM CODE?

```
#include <iostream>
typedef struct MY_STRUCT
{
    int x, y, z;
} my_struct_t;
my_struct_t *createStruct()
{
    my_struct_t my_struct;
    return &my_struct;
}
int main()
{
    my_struct_t *structi = createStruct();
    structi->y = 0x5;
    printf("0x%X", structi->x, structi->y, structi->z);
}
```

JA ES GIBT HIER EIN PROBLEM

```
Torstens-MacBook-Pro:quiz mhptorsten$ g++ Snip03.cpp -o snip3
Snip03.cpp:12:13: warning: address of stack memory associated with local variable 'my_struct' returned [-Wreturn-stack-address]
    return &my_struct;
           ^~~~~~~~~~
1 warning generated.
Torstens-MacBook-Pro:quiz mhptorsten$
```

FUN FACT: DAS FUNKTIONIERT TROTZDEM… WARUM?

```c
#include <iostream>
typedef struct MY_STRUCT
{
    int x, y, z;
} my_struct_t;
my_struct_t *createStruct()
{
    my_struct_t my_struct;
    return &my_struct;
}
int main()
{
    my_struct_t *structi = createStruct();
    structi->y = 0x5;
    printf("0x%X", structi->x, structi->y, structi->z);
}
```

WAS IST EIN INCLUDE GUARD?

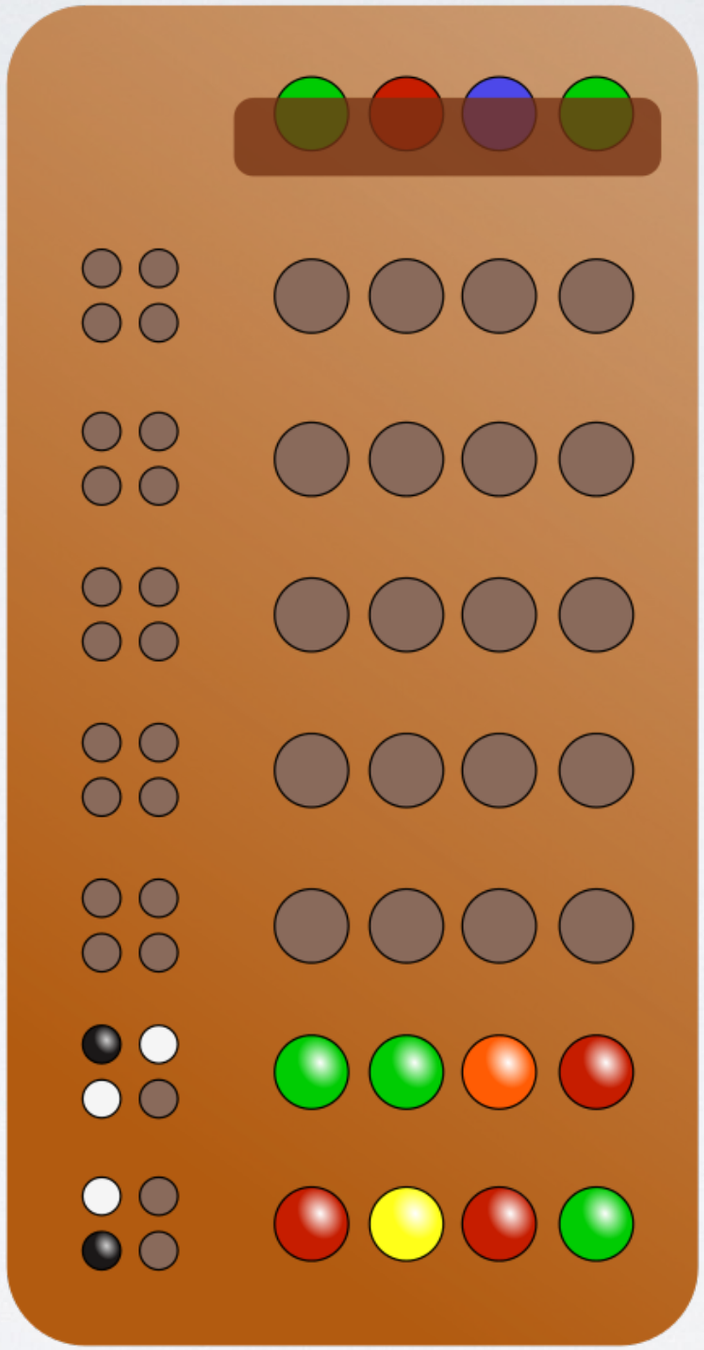
WIE GROß IST DIESER UNION?
UND WAS IST EIN UNION?

```
union my_first_union
{
    int x;
    char c;
    int64_t longi
}
```

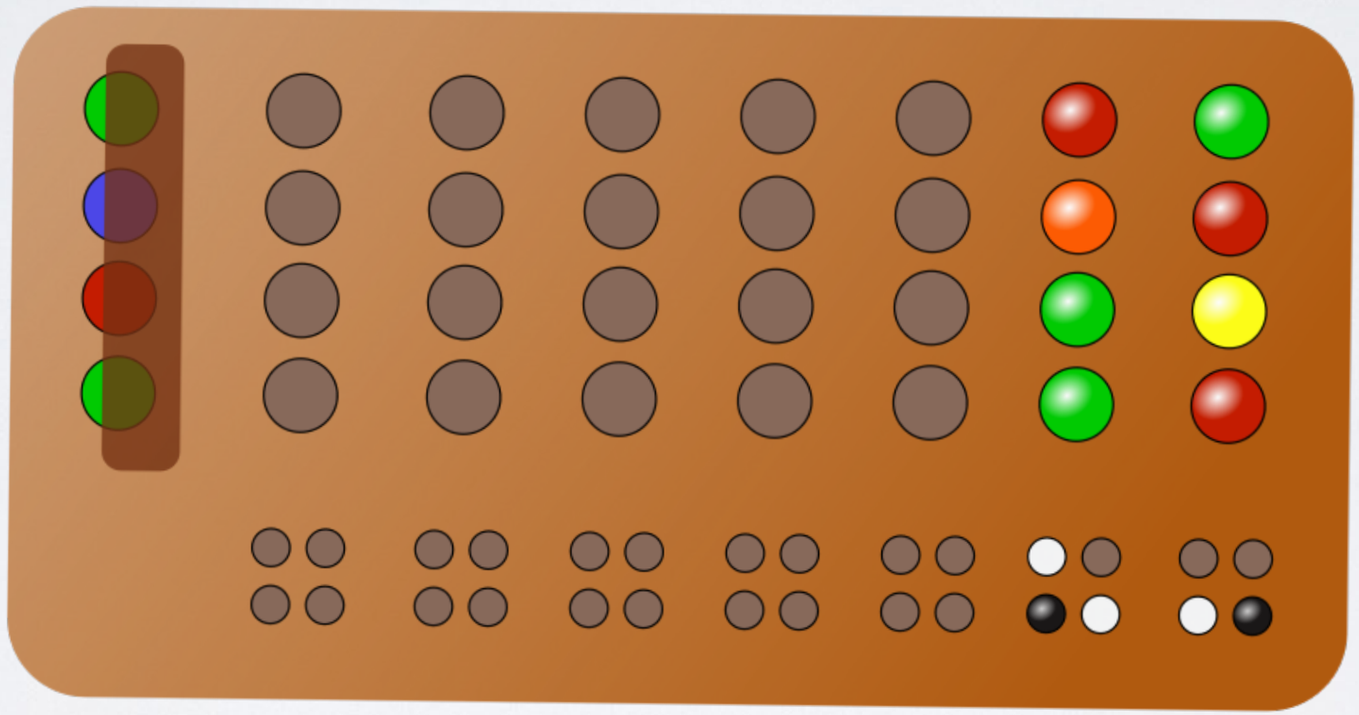
PROGRAMMIERAUFGABE FÜR ZWISCHENDURCH

Ein Spieler (der Codierer) legt zu Beginn einen vierstelligen geordneten Farbcode fest, der aus sechs Farben ausgewählt wird; jede Farbe kann auch mehrmals verwendet werden. Der andere Spieler (der Rater) versucht, den Code herauszufinden. Dazu setzt er einen gleichartigen Farbcode als Frage; beim ersten Zug blind geraten, bei den weiteren Zügen mit Hilfe der Antworten zu den vorangegangenen Zügen.
Auf jeden Zug hin bekommt der Rater die Information, wie viele Stifte er in Farbe und Position richtig gesetzt hat und wie viele Stifte zwar die richtige Farbe haben, aber an einer falschen Position stehen. Ein Treffer in Farbe und Position wird durch einen schwarzen Stift angezeigt, ein farblich richtiger Stift an falscher Stelle durch einen weißen Stift. Es gibt auch Versionen mit roten statt schwarzen Stiften zur Anzeige von Treffern in Farbe und Position. Alle Fragen und Antworten bleiben bis zum Ende des Spiels sichtbar.
Ziel des Raters ist es, den Farbcode mit möglichst wenigen Fragen zu erraten.

PROGRAMMIERAUFGABE FÜR ZWISCHENDURCH



PROGRAMMIERAUFGABE FÜR ZWISCHENDURCH



KLASSENDEKLARATION

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} object_names;
```

KLASSE IN C++ BEISPIEL

```
// classes example
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

ACCESS SPECIFIERS

- **private**
  - Nur die Klasse und ihre „friends“ können es erreichen
- **public**
  - Überall wo sie sichtbar sind
- **protected**
  - selbst, Freunde und alle abgeleiteten Klassen

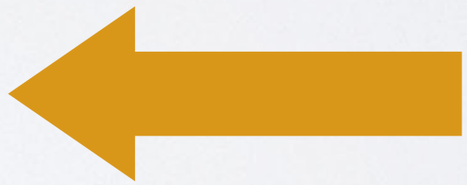
KLASSE IN C++ BEISPIEL

```
// classes example
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

.hpp

KLASSE IN C++ BEISPIEL

```
// classes example
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```



.cpp

KLASSENDEKLARATION

```
#ifndef CUSTOM_CLASS_HPP
#define CUSTOM_CLASS_HPP
class CustomClass{

    public:
    void doSomethigSmart();
};
#endif
```

KLASSENDEFINITION

```
#include "CustomClass.hpp"
#include <iostream>

void CustomClass::doSomethigSmart(){
    printf("That was smart!");
}

int main(){
    CustomClass custom;
    custom.doSomethigSmart();
    return 0x0;
}
```

KONSTRUKTOREN

```
// example: class constructor
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};
Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}
int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

KONSTRUKTOREN

- Deklaration als Funktion mit Namen der Klasse
- Deklaration eigentlich genau wie in Java
- Automatisch generiert:
  - Default Empty Constructor
  - Copy Constructor Class(Class &other)

ERZEUGEN VON OBJEKTEN

```
Rectangle rect (3,4);
Rectangle rectb (5,6);

Rectangle rectb;   // ok called
Rectangle rectc(); // oops
```

SO VIELE MÖGLICHKEITEN EIN OBJEKT ZU ERZEUGEN

```
// classes and uniform initialization
#include <iostream>
using namespace std;
class Circle {
    double radius;
  public:
    Circle(double r) { radius = r; }
    double circum() {return 2*radius*3.14159265;}
};
int main () {
  Circle foo (10.0);   // functional form
  Circle bar = 20.0;   // assignment init.
  Circle baz {30.0};   // uniform init.
  Circle qux = {40.0}; // POD-like
  cout << "foo's circumference: " << foo.circum() << '\n';
  return 0;
}
```

DESTRUKTOREN

- Methode die bei Objektvernichtung GARANTIERT aufgerufen wird (!= JAVA)
  - Variable geht Out Of Scope
  - Explizit durch delete
- Deklaration durch ~class_name( )

DESTRUCTOR BEISPIEL

```
typedef struct my_struct
{
    int x;
    int y;

    ~my_struct()
    {
        printf("Oh no my values have been x=%d,y=%d", x, y);
    }
} my_struct_t;

int main()
{
    my_struct_t instance;
    instance.x = 187;
    instance.y = 4711;
    return 0x00;
}
```

MEMBER INITIALISIEREN

```
class Rectangle {
    int width,height;
  public:
    Rectangle(int,int);
    int area() {return width*height;}
};

Rectangle::Rectangle (int x, int y) { width=x; height=y; }

Rectangle::Rectangle (int x, int y) : width(x)
{ height=y; }

Rectangle::Rectangle (int x, int y) : width(x), height(y) {
}
```

ZUGRIFF AUF OBJEKTINHALTE

```
// pointer to classes example
#include <iostream>
using namespace std;
class Rectangle {
  int width, height;
public:
  Rectangle(int x, int y) : width(x), height(y) {}
  int area(void) { return width * height; }
};
int main() {
  Rectangle obj (3, 4);
  Rectangle * foo, * bar, * baz;
  foo = &obj;
  bar = new Rectangle (5, 6);
  baz = new Rectangle[2] { {2,5}, {3,6} };
  cout << "obj's area: " << obj.area() << '\n';
  cout << "*foo's area: " << foo->area() << '\n';
  cout << "*bar's area: " << bar->area() << '\n';
  cout << "baz[0]'s area:" << baz[0].area() << '\n';
  cout << "baz[1]'s area:" << baz[1].area() << '\n';
  delete bar;
  delete[] baz;
  return 0;
}
```

POINTER AUF OBJEKTE

expression	can be read as
`*x`	pointed to by `x`
`&x`	address of `x`
`x.y`	member `y` of object `x`
`x->y`	member `y` of object pointed to by `x`
`(*x).y`	member `y` of object pointed to by `x` (equivalent to the previous one)
`x[0]`	first object pointed to by `x`
`x[1]`	second object pointed to by `x`
`x[n]`	(`n+1`)th object pointed to by `x`

THIS POINTER

```
void CustomClass::doSomethigSmart(){
    CustomClass *this
    art!");
    this;
}

int main(){
```

GIBT ES PROBLEME BEI DIESEM CODE?

```
#include <iostream>
class fucking_big_class
{
  private:
    char buf[1024];
    char buf_2[2048];
    int64_t big_number_of_numbers[4096];
 public:
    char *getBuf2() { return buf_2; };
};
void do_something_smart(fucking_big_class big_class_p)
{
    printf("Yeah yeah very Big! %s", big_class_p.getBuf2());
}
int main()
{
    fucking_big_class biggi;
    do_something_smart(biggi);
    return 0x00;
}
```

RAII

resource acquisition is initialization

RAII (BEISPIEL)

```
typedef struct my_struct
{
    int x;
    int y;
    char * buffer;
```

```
    my_struct(int p_x,int p_y){
        x = p_x;
        y = p_y;
        buffer = new char[128];
    }

    ~my_struct()
    {
        printf("Oh no my values have been x=%d,y=%d", x, y);
        delete[] buffer;
    }
} my_struct_t;
```

•

VERERBUNG

```
class A { /* ... */ };
class B : public A    { /* ... */ };
class C : protected A { /* ... */ };
class D : private A   { /* ... */ }; // Standard (:A)
```

Ist ein Element in A	public	protected	private
... wird es in B	public	protected	nicht übergeben
... wird es in C	protected	protected	nicht übergeben
... wird es in D	private	private	nicht übergeben

ÜBERSCHREIBEN

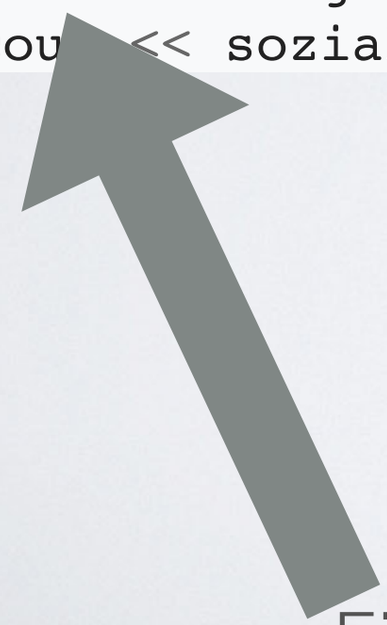
```
class Person {
    // ...
  public:
    void ausgeben() const; // gibt Personendaten auf stdout aus
};

class Mitarbeiter : public Person {
    // ...
  public:
    void ausgeben() const; // überschreibt Person::ausgeben
};
```

MÖGLICHE IMPLEMENTIERUNG

```
void Mitarbeiter::ausgeben() const {
  Person::ausgeben();                          // ruft Methode der Basisklasse auf
  cout << sozialversicherungsNr << endl; // zusätzliche Funktionalität
}
```

MÖGLICHE IMPLEMENTIERUNG

```
void Mitarbeiter::ausgeben() const {
  Person::ausgeben();                          // ruft Methode der Basisklasse auf
  cou  << sozialversicherungsNr << endl; // zusätzliche Funktionalität
}
```

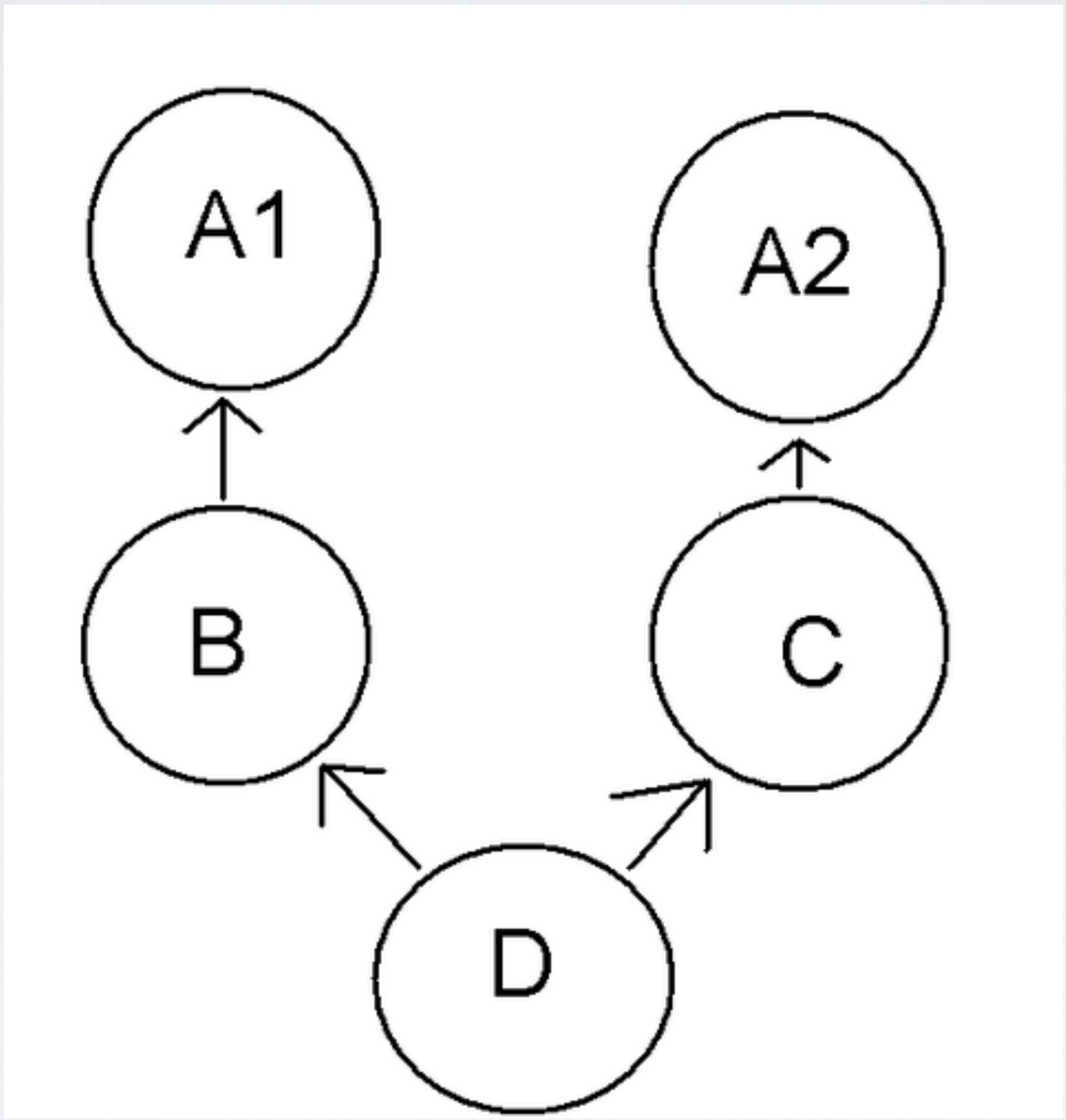


Ein bisschen wie „super“ in Java

NICHT VIRTUELLE VERERBUNG

```
class Datei {
  unsigned int position;
  /* ... */
};

class DateiZumLesen      : public Datei { /* ... */ };
class DateiZumSchreiben  : public Datei { /* ... */ };

class DateiZumLesenUndSchreiben: public DateiZumLesen, public DateiZumSchreiben { /* ... */ };
```

NICHT VIRTUELLE VERERBUNG



EFFEKT

- In unserem Beispiel erlaubt es zwei Zeiger zu haben: Einen zum Lesen und einen zum Schreiben

VIRTUELLE VERERBUNG

```
class Datei {
  unsigned int position;
  /* ... */
};

class DateiZumLesen      : public virtual Datei { /* ... */ };
class DateiZumSchreiben : public virtual Datei { /* ... */ };

class DateiZumLesenUndSchreiben: public DateiZumLesen, public DateiZumSchreiben { /* ... */ };
```

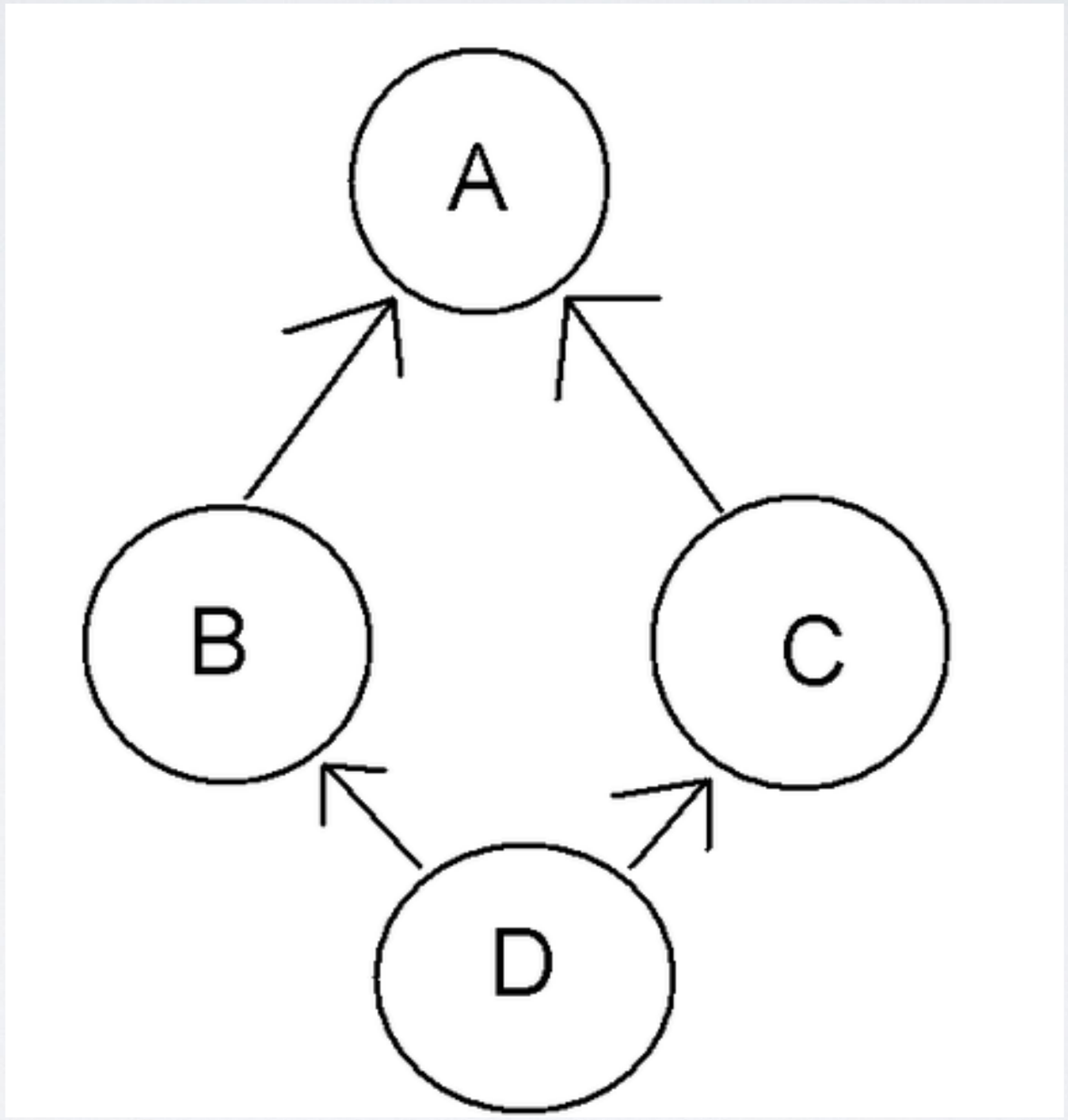
VIRTUELLE VERERBUNG

```
class Datei {
  unsigned int position;
  /* ... */
};
```

```
class DateiZumLesen     : public virtual Datei { /* ... */ };
class DateiZumSchreiben : public virtual Datei { /* ... */ };
```

```
class DateiZumLesenUndSchreiben: public DateiZumLesen, public DateiZumSchreiben { /* ... */ };
```

VIRTUELLE VERERBUNG



EFFEKT

- Die Instanz hat nun nur noch einen Positionszeiger

BEISPIEL

```
class Auto
{
public:
    void gibGas()
    {
        printf("Wrumm Wrumm\n");
    }
};
class Sportwagen : public Auto
{
public:
    void gibGas()
    {
        printf("Sportlich Wrumm Wrumm\n");
    }
};
```

BEISPIEL

```
void soundTest(Auto *testeEs)
{
    testeEs->gibGas();
}
```

```
int main()
{
    Auto fiesta;
    Sportwagen panamera;

    soundTest(&fiesta);
    soundTest(&panamera);

    return 0x0;
}
```

AUSGABE

```
localhost:day4 mhptorsten$ ./car
Wrumm Wrumm
Wrumm Wrumm
localhost:day4 mhptorsten$
```

?

BEISPIEL

```
class Auto
{
public:
    void virtual gibGas()
    {
        printf("Wrumm Wrumm\n");
    }
};
class Sportwagen : public Auto
{
public:
    void virtual gibGas() override
    {
        printf("Sportlich Wrumm Wrumm\n");
    }
};
```

BEISPIEL

```
void soundTest(Auto *testeEs)
{
    testeEs->gibGas();
}
```

```
int main()
{
    Auto fiesta;
    Sportwagen panamera;

    soundTest(&fiesta);
    soundTest(&panamera);

    return 0x0;
}
```

BEISPIEL

```
localhost:day4 mhptorsten$ ./vcar
Wrumm Wrumm
Sportlich Wrumm Wrumm
localhost:day4 mhptorsten$
```

FRÜHE BZW. SPÄTE BINDUNG

- Virtual Keyword sagt, dass das Objekt entscheiden muss welche Funktion aufgerufen wird (späte Bindung)
- Ohne Virtual wird anhand des Pointers entschieden (Frühe Bindung)

NÄCHSTES MAL

- Smart Pointer
- Exceptions
- Noch mehr Aufgaben

GENERIERUNG EINES MINESWEEPER FELDS

- Generiere ein Feld mit 16x16 Feldgröße und 99 Minen

- Fülle die Felder drumherum aus, basierend darauf wie viele Mienen um Sie herum sind

- Gib das Ergebnis am Ende aus

- Bonus: Implementiere die Spiellogik :-)