

PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

AUFGABE VOM LETZTEN MAL

- Baue ein Telefonbuch
- Verwalte die Telefonnummern mit Vor- und Nachname in einem (c)Struct
- Lasse den `std::vector` Zeiger auf den Eintrag halten
- Erlaube die Suche nach Nachnamen und gib alle Einträge mit passendem Nachnamen aus

AGENDA FÜR DIE HEUTIGE VORLESUNG

- Wiederholung und Übungen
- Header Dateien

QUIZ

```
typedef struct fun_struct{  
    int32_t x;  
    char magic[50];  
} fun_struct_t;
```

```
int main(){
```

```
    fun_struct_t an_instance;
```

```
}
```

•



QUIZ

Wo liegt diese Instanz?

```
fun_struct_t another_instance = new fun_struct_t;
```

•

QUIZ



```
fun_struct_t another_instance = new fun_struct_t;
```

-

QUIZ

```
fun_struct_t* another_instance = new fun_struct_t;
```

QUIZ

```
#include <iostream>
typedef struct fun_struct{
    int32_t x;
    char magic[50];
} fun_struct_t;
```

```
int main(){
```

```
    fun_struct_t an_instance;
```

```
    size_t size_of_an_instance = sizeof(an_instance);
```

```
    fun_struct_t* another_instance = new fun_struct_t;
```

```
    size_t size_of_another_instance = sizeof(another_instance);
```

```
    printf("Size of an_instance = %zu, size of another_instance = %zu \n"
        ,size_of_an_instance,size_of_another_instance);
```

```
}
```


QUIZ

- ```
localhost:day3 mhptorsten$ g++ -o struct Task1/PointerFunAdventure.cpp -std=c++11
localhost:day3 mhptorsten$./struct
Size of fun_struct_t = 56, size of an_instance = 56, size of another_instance = 8
localhost:day3 mhptorsten$
```

**WTF?**

# MEMORY ALIGNMENT

```
typedef struct fun_struct{
 int32_t x;
 char magic[50];
 // two padding bytes for CPU alignment
} fun_struct_t;
```

# QUIZ

```
#include <iostream>
typedef struct fun_struct{
 int32_t x;
 char magic[52];
} fun_struct_t;
```

```
int main(){
```

```
 fun_struct_t an_instance;
```

```
 size_t size_of_an_instance = sizeof(an_instance);
```

```
 fun_struct_t* another_instance = new fun_struct_t;
```

```
 size_t size_of_another_instance = sizeof(another_instance);
```

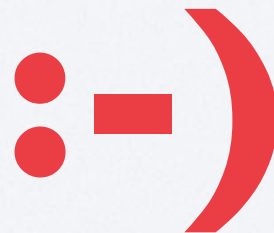
```
 printf("Size of an_instance = %zu, size of another_instance = %zu \n"
 ,size_of_an_instance,size_of_another_instance);
```

```
}
```



# QUIZ

- ```
localhost:day3 mhptorsten$ g++ -o struct Task1/PointerFunAdventure.cpp -std=c++11
localhost:day3 mhptorsten$ ./struct
Size of fun_struct_t = 56, size of an_instance = 56, size of another_instance = 8
localhost:day3 mhptorsten$
```



SIZEOF

- Deshalb immer mit sizeof arbeiten statt selbst großen zusammen zurechnen ;-)

QUIZ

```
int main(){
```

```
    fun_struct_t an_instance;
```

```
    size_t size_of_an_instance = sizeof(an_instance);
```

```
    fun_struct_t* another_instance = new fun_struct_t;
```

```
    size_t size_of_another_instance = sizeof(another_instance);
```

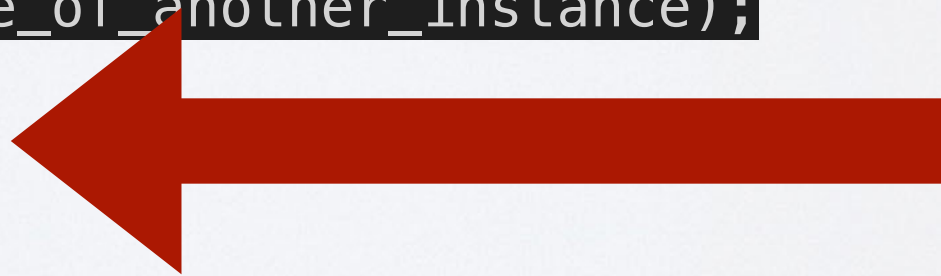
```
    printf("Size of an_instance = %zu, size of another_instance = %zu \n",  
           size_of_an_instance, size_of_another_instance);
```

```
}
```

Was fehlt?

QUIZ

```
int main(){  
    fun_struct_t an_instance;  
    size_t size_of_an_instance = sizeof(an_instance);  
    fun_struct_t* another_instance = new fun_struct_t;  
    size_t size_of_another_instance = sizeof(another_instance);  
    printf("Size of an_instance = %zu, size of another_instance = %zu \n"  
        ,size_of_an_instance,size_of_another_instance);  
    delete another_instance;  
}
```



FILE STREAMS

- Package `<fstreams>` (Standardlib)
 - `ofstream` —> Output-Operationen
 - `ifstream` —> Input-Operationen
 - `fstream` —> Beides :-)

OFSTREAM EXAMPLE

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;
```

```
int main () {
    ofstream myfile;
    myfile.open ("example.txt");
    myfile << "Writing this to a file.\n";
    myfile.close();
    return 0;
}
```


OFSTREAM

- Vergleichbar zum geschrieben in den Outputstream
- Gleiche Stream-Operatoren
- Ofstream wird automatisch mit schreibendem Modus geöffnet

LESEN VON EINEM (ZEICHEN- STREAM)

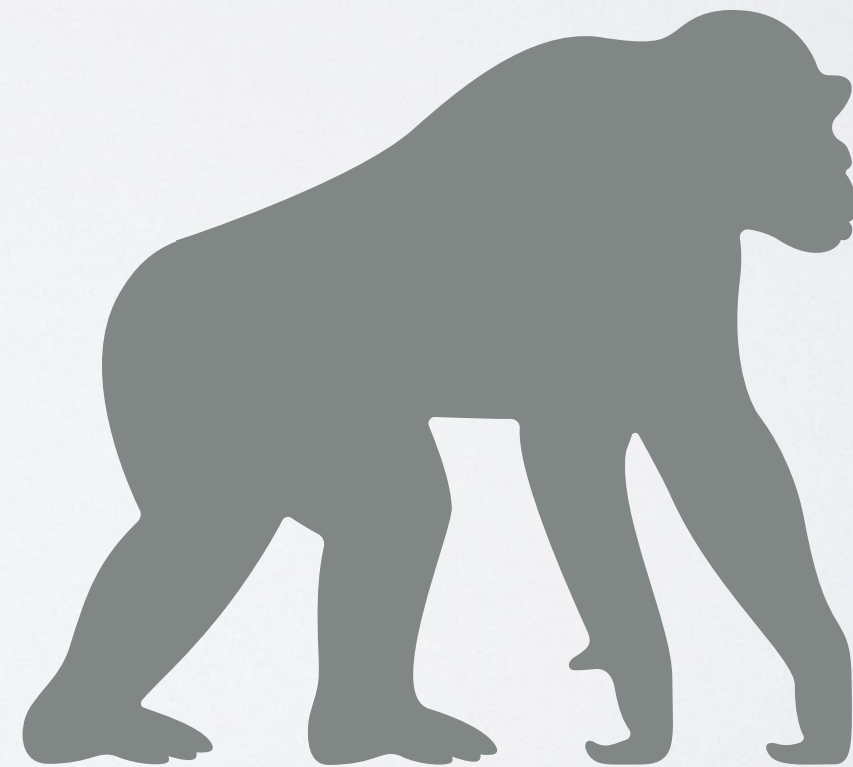
```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
```

```
int main () {
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            cout << line << '\n';
        }
        myfile.close();
    }
```

```
    else cout << "Unable to open file";
```

```
    return 0;
}
```

- Close nie vergessen!



MODI BEIM ÖFFNEN VON STREAMS

`ios::in`

→ Input mode

`ios::out`

→ Output mode

`ios::binary`

→ Binary Mode (write, read instead of getLine...)

`ios::ate`

→ at the end

`ios::app`

→ appending

`ios::trunc`

→ truncating

AUFGABE I

Aufgabenstellung:

1. Erweitere deine Struct-basierte Telefonbuch App so, dass du die angelegten Nummern in einer Datei speichern kannst.
2. Bei Start der Applikation sollen diese Nummern wieder geladen werden.

**** Achtet darauf alle Streams zu schließen und eventuellen Speicher wieder frei zu geben! ****

KONVERSION STRING ZU INT ODER FLOAT

```
std::stof(myString)  
std::stoi(myString)
```


Eine Header-Datei ist in der Programmierung, insbesondere in den Programmiersprachen C++ und C, eine Textdatei, die Deklarationen und andere Bestandteile des Quelltextes enthält. Quelltext, der sich in einer Header-Datei befindet, ist im Allgemeinen zur Verwendung in mehreren Programmen oder mehreren Teilen eines Programmes vorgesehen.

- Wikipedia

HEADER IN C++

- Sollen (i.d.R.) Deklaration von Definition trennen
 - Deklaration: Beschreibung der Funktionen / Schnittstellen
 - Definition: Implementierung dieser Schnittstellen
- Seltener: Header Only Bibliotheken

HEADER

```
// Text wird per COPY and PASTE kopiert  
#include <stdio.h>
```

```
/* ... (weiterer Programmtext) */
```



HEADER IN C++

- Arbeiten mit Text-Ersetzung
- Werden zum Zeitpunkt der Kompilierung aufgelöst
- Funktionieren über Copy and Paste
- Pattern (i.d.R.):
 - CodeBlock.hpp —> CodeBlock.cpp
 - Code.h —> Code.c

INCLUDE GUARDS

- Makros die Verhindern sollen, dass der gleiche Header mehrfach eingefügt wird
- Makros ersetzen das Makro mit dem definierten Inhalt
- Können Conditionals darstellen und Abfragen
- `#define <MakroName> [Ersetzung]`
- `#ifdef #ifndef #endif`

INCLUDE GUARD

```
// A.h  
const int M = 123;
```

```
class A  
{ /* ... */ };  
•
```



INCLUDE GUARD

```
// B.h  
#include "A.h"
```

```
class B : public A  
{ /* ... */ };
```



INCLUDE GUARD

```
// program.cpp  
#include "A.h"  
#include "B.h"  
int main() { /* ... */ }
```



INCLUDE GUARD SOLUTION

```
// A.h  
#ifndef A_H  
#define A_H
```

```
class A  
{ /* ... */ };
```

```
#endif /* A_H */
```



#PRAGMA ONCE

Ersetzt einen klassischen Include Guard. Warum würdet ihr trotzdem klassische include guards verwenden?

```
#pragma once
```

```
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
```

```
template <class Container>
void split_by_comma(const std::string& str, Container& cont, char delim = ',')
{
    std::stringstream ss(str);
    std::string token;
    while (std::getline(ss, token, delim)) {
        cont.push_back(token);
    }
}
```

HAUSAUFGABE

Aufgabe 2

- * Erstelle ein Objekt, welches auf die Zeilen aus der beigefügten CSV und lese diese ein
- * Beantworte die folgenden Fragen:
 - * Wie viele Verbrechen gab es in Sacramento in 2006?
 - * Was war das häufigste Verbrechen?
 - * Welches ist der gefährlichste District? (gemessen an der Menge der Vorfälle?)
 - * Welche Adressen kamen mehr als einmal vor?

ENDE

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

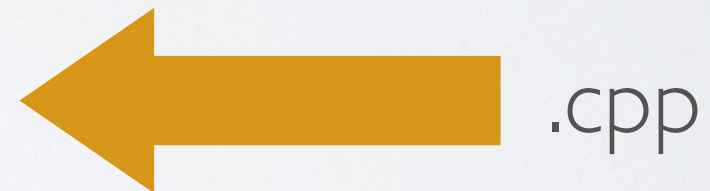
KLASSE IN C++ BEISPIEL

```
// 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;
}
```



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;
}
```



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 (\neq JAVA)
- Variable geht Out Of Scope
- Explizit durch delete
- Deklaration durch `~class_name()`

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
<code>*x</code>	pointed to by <code>x</code>
<code>&x</code>	address of <code>x</code>
<code>x.y</code>	member <code>y</code> of object <code>x</code>
<code>x->y</code>	member <code>y</code> of object pointed to by <code>x</code>
<code>(*x).y</code>	member <code>y</code> of object pointed to by <code>x</code> (equivalent to the previous one)
<code>x[0]</code>	first object pointed to by <code>x</code>
<code>x[1]</code>	second object pointed to by <code>x</code>
<code>x[n]</code>	(<code>n+1</code>)th object pointed to by <code>x</code>

VERERBUNG TEASER

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

VERERBUNG TEASER

- Public —> Vererbung alles so public wie möglich
- Protected beschränkt den Zugriff weiter (public Felder sind von hier an Protected)
- Private Vererbung ist beachten wir heute noch nicht

VERERBUNG BEISPIEL

```
class A  
{  
    int x;
```

```
    //...  
};
```

```
class B  
{  
    double y;
```

```
    //...  
};
```

```
class C : public A, public B  
{  
    char z;
```

```
    //...  
};
```

(HAUS)AUFGABE FÜR DIESE UND EVNTL ÜBERNÄCHSTE WOCHE

- Programmiert ein Schachspiel bei denen Abwechselnd beide Spieler ihre Züge machen
- Modelliert dabei die Figuren als Klassen, die ihr eigenes Move-Set haben
- Nutzt die bekannten Techniken zur Modellierung
- Eingabe der Züge erfolgt über die Kommandozeile (Beispiel: Erstes Kommando selektiert Figur, zweites gibt das Zielfeld an)
- Die Kommandozeile gibt dabei das Log für das Spiel an