# PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

# AGENDA

- Einen Schritt zurück gehen
- Das C++ Basis Quiz
- Smartpointer (wenn die Zeit reicht)

# WO KOMMEN WIR HER?

- Verschiedene Konstrukte der C++ Sprache kennengelernt

- Erster komplexerer Anwendungsfall TCP Verbindungen und Sockets

# WO WOLLEN WIR AM ENDE DES KURSES STEHEN

- Grundsätzlich Verständnis für C++ Entwicklung
- Handwerkszeug, um selbstständig weiter zu kommen
- Praktische Erfahrung sammeln

# DAS C++ BASIS QUIZ

- C++ Syntax Basics
- C++ Pointer
- C++ Strukturen
- C++ Klassen
- C++ Vererbung
- C++ Speichermanagement (lite)

# 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 BEDEUTET VIRTUAL?

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
int main()
{
    return 0x00;
}
```

# WAS BEDEUTET DAS =0

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
int main()
{
    return 0x00;
}
```

# WIE HEIßT SO EIN CONSTRUCT IN JAVA?

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
int main()
{
    return 0x00;
}
```

# WAS BEDEUTET DAS & BEIM RÜCKGABEWERT?

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
};
int main()
{
    OldMan oldMan;
    printf("Having a young old man here: %s, %c, %d\n",
        oldMan.getName().c_str(), //
        oldMan.getGender(), //
        oldMan.getAge()); //
    return 0x00;
}
```

# ZUGRIFFSMODIFIKATOR VON NAME?

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
};
int main()
{
    OldMan oldMan;
    printf("Having a young old man here: %s, %c, %d\n",
        oldMan.getName().c_str(), //
        oldMan.getGender(), //
        oldMan.getAge()); //
    return 0x00;
}
```

# PUBLIC BEI DER VERERBUNG?

```
#include <iostream>
class Profile
{
  public:
    virtual std::string &getName() = 0;
    virtual char getGender() = 0;
    virtual int getAge() = 0;
};
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
};
int main()
{
    OldMan oldMan;
    printf("Having a young old man here: %s, %c, %d\n",
        oldMan.getName().c_str(), //
        oldMan.getGender(), //
        oldMan.getAge()); //
    return 0x00;
}
```

# WHAT'S THE ~STUFF?

```
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
        ~OldMan(){};
};
int main()
{
    Profile *profile = new OldMan();
    printf("Having a young old man here: %s, %c, %d\n",
        profile->getName().c_str(), //
        profile->getGender(), //
        profile->getAge()); //
    delete profile;
    return 0x00;
}
```

# WHAT DOES „NEW“ MEAN?

```
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
        ~OldMan(){};
};
int main()
{
    Profile *profile = new OldMan();
    printf("Having a young old man here: %s, %c, %d\n",
        profile->getName().c_str(), //
        profile->getGender(), //
        profile->getAge()); //
    delete profile;
    return 0x00;
}
```

# WHAT DOES ,,DELETE“ MEAN?

```
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
        ~OldMan(){};
};
int main()
{
    Profile *profile = new OldMan();
    printf("Having a young old man here: %s, %c, %d\n",
        profile->getName().c_str(), //
        profile->getGender(), //
        profile->getAge()); //
    delete profile;
    return 0x00;
}
```

# WAS SAGT DER PFEIL AUS?

```
class OldMan : public Profile
{
    std::string name = "Klaus";
    public:
        virtual std::string &getName() override { return this->name; };
        virtual char getGender() override { return 'M'; };
        virtual int getAge() override { return 65; };
        ~OldMan(){};
};
int main()
{
    Profile *profile = new OldMan();
    printf("Having a young old man here: %s, %c, %d\n",
        profile->getName().c_str(), //
        profile->getGender(), //
        profile->getAge()); //
    delete profile;
    return 0x00;
}
```

# WAS IST EIN INCLUDE GUARD?

# WIE IST DIE GRÖßE DIESES STRUCTS?

```
typedef struct MY_STRUCT
{
    int x, y, z;
    char buf[255];
    int16_t shorty;

} my_struct_t;
```

# WIE GROß IST DIESER UNION?

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

# IST ETWAS AN DIESEM CODE PROBLEMATISCH?

```
void SomeMethod()
{
  ClassA *a = new ClassA;
  SomeOtherMethod();
  delete a;
}
```

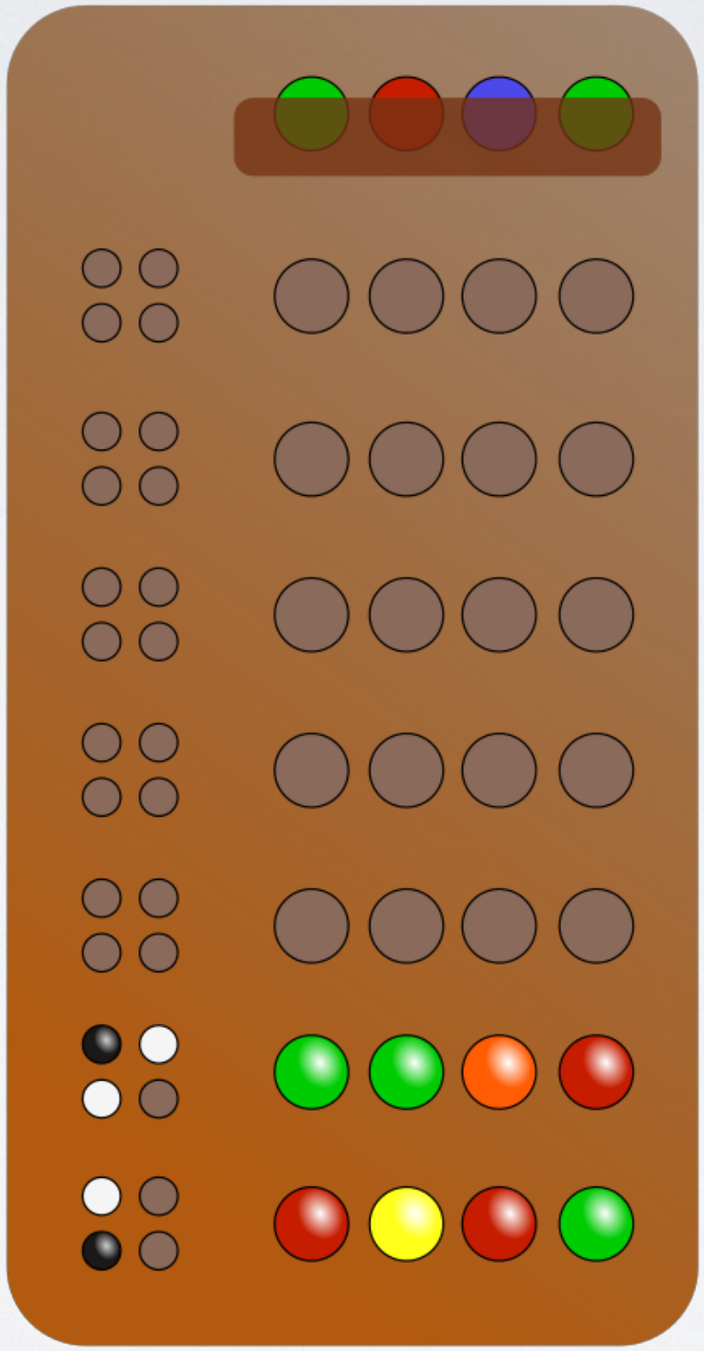
# GIBT ES PROBLEME BEI DIESEM CODE?

```
#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;
}
```

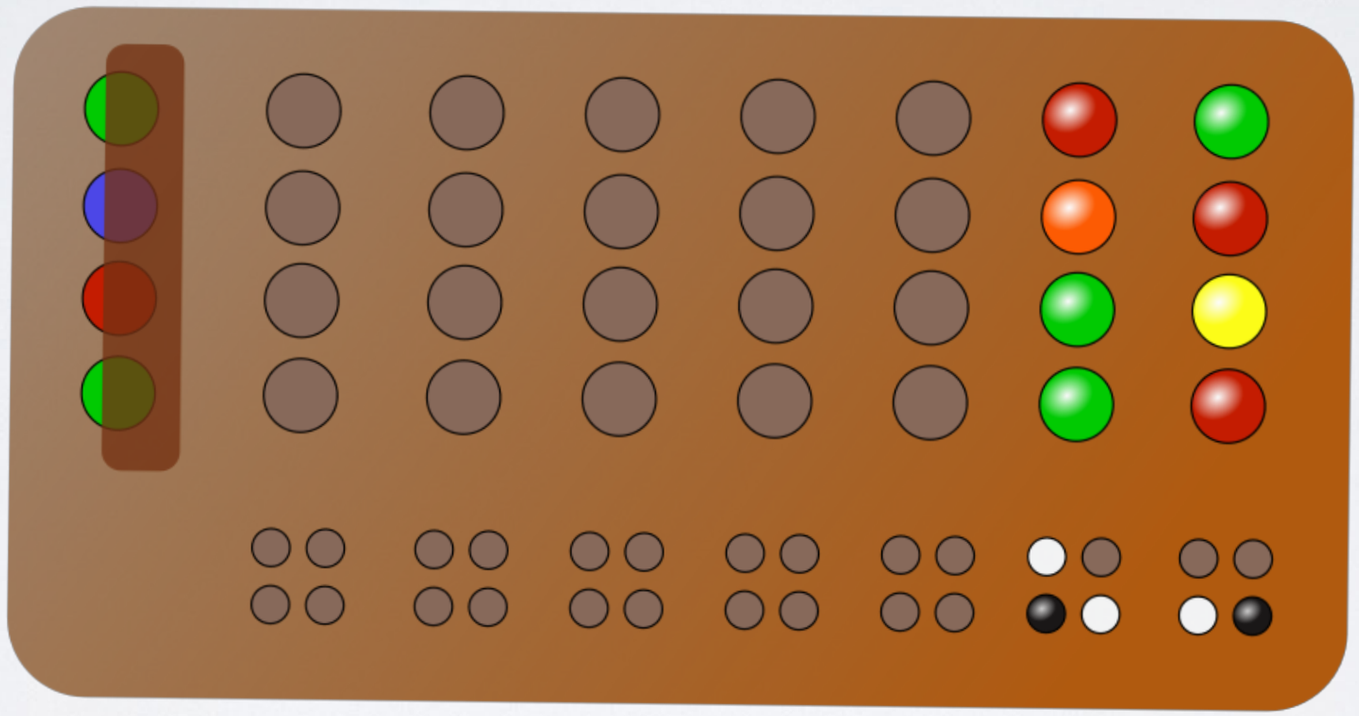
# PROGRAMMIERAUFGABE FÜR ZWISCHENDURCH

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



# SMART POINTER

```
void SomeMethod()
{
   ClassA *a = new ClassA;
   SomeOtherMethod();
   delete a;
}
```

# SMART POINTER ERKLÄRUNG

- Smart Pointer wrappen normale Pointer, um das Handling mit ihnen weniger Fehleranfällig zu machen

- Sie sollten in der Regel vor normalen Pointern bevorzugt werden

- Existieren in verschiedenen Arten

# SMART POINTER (ABSTRAKTES) BEISPIEL

```
SomeSmartPtr<MyObject> ptr(new MyObject());
ptr->DoSomething(); // Use the object in some way.
```

# SMART POINTER ARTEN

- Kommen ursprünglich aus Boost, sind aber mit C++11 in den Standard übergegangen
- Es gibt noch den bösen Bruder und nicht besonders smarten std::auto_ptr aus C++98
- Arten von Smart Pointern:
  - Unique-Pointer
  - Shared-Pointer
  - Weak-Pointer (Gefährlich)

# UNIQUE POINTER

```
void f()
{
    {
        std::unique_ptr<MyObject> ptr(new MyObject());
        ptr->DoSomethingUseful();
    } // boost::scopted_ptr goes out of scope --
      // the MyObject is automatically destroyed.
}
```

- unique_ptr kann nicht kopiert werden
- Nützlich wenn man ein Objekt nur in einem Scope braucht, oder seine Lebenszeit als Member eines anderen Objekts beschreibt

# SHARED_PTR

```
void f()
{
    typedef std::shared_ptr<MyObject> MyObjectPtr; // nice short alias
    MyObjectPtr p1; // Empty

    {
        MyObjectPtr p2(new MyObject());
        // There is now one "reference" to the created object
        p1 = p2; // Copy the pointer.
        // There are now two references to the object.
    } // p2 is destroyed, leaving one reference to the object.
} // p1 is destroyed, leaving a reference count of zero.
  // The object is deleted.
```

- Pointer mit Reference-Counting
- Nützlich, wenn die Lebensdauer des Objekts komplizierter ist
- Gefahren: Dangling Pointers und Circular References

# WEAK_PTR

- std::weak_ptr ist eine Referenz auf ein Objekt, welche nicht mit ins Reference Counting eingeht
- Wenn man ein Konstrukt hat, wo man diesen Pointer braucht, sollte man seinen Programmentwurf wirklich hinterfragen

# ERZEUGUNGSVARIANTEN

- std::make_shared<Class>(Construktur Args)
- std::make_unique<Class>(Construktor Args)
- std::shared_ptr<Class>(new Class(Constructor…))
- std::unique_ptr<Class>(new Class(Constructor…))

# AUTO_PTR

- Niemals benutzen
- Wie Unique_ptr nur dass er kopiert werden kann und dabei die Ownership auf das Objekt übergibt
- Der alte Unique_Ptr wird ungültig ohne es zu merken

# 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 :-)

# MINESWEEPER