PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

AUFGABE UND HAUSAUFGABE VOM LETZTEN MAL

- Erweitern der Hero App!
    - Der Hero ist ab jetzt nicht mehr nur ein String sondern ein (C) struct bestehend aus char(20) für den Namen, und einer Zufallszahl zwischen 10-30 für die Lebenspunkte, sowie einem Pointer auf einen Struct Waffe
    - Die Waffe erhält einen von - bis Schaden von rand(3) bis Ergebnis+3
    - Wenn man einen Helden aus der Liste der erstellten oder vorgefertigten Charaktere wählt, wird ein Kampf gegen einen zufälligen Gegner aus der Liste gestartet
    - Beim Kampf schlagen beide Charaktere abwechselnd mit den Waffen aufeinander ein bis einer keine Lebenspunkte mehr hat, dieser Charakter ist dann tot
    - Es beginnt der Charakter mit dem längerem Namen

AGENDA FÜR DIE HEUTIGE VORLESUNG

- Header Dateien
- Heap und Stack
- Konzepte der Objektorientierung in C++

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

```cpp
// 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 */
```

STACK UND HEAP

STACK

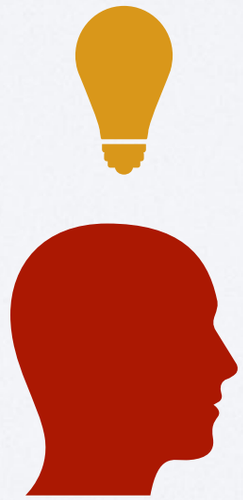
- Begrenzte Größe
- LIFO Datenstruktur (die zuletzt angelegten Daten werden als erstes wieder freigegeben, deshalb auch "Stapel")
- Wächst und schrumpft mit dem Programmverlauf
- Wird verwendet für lokale Variablen und Funktionsparameter
- Kein explizites Freigeben des Speichers nötig
- Das Ablegen und Entfernen von Elementen ist sehr effizient

HEAP

- Der Heap kann innerhalb der Prozessgrenze beliebig groß werden
- Anlegen und freigeben von Objekten ist vergleichsweise langsam
- Auf dem Heap angelegte Objekte können global verfügbar gemacht werden
- In Programmiersprachen ohne Garbage Collector muss der Speicher manuell freigegeben werden, wenn er nicht mehr benötigt wird

WANN LEGE ICH EIN OBJEKT / EINE DATENSTRUKTUR AUF DEM HEAP UND WANN AUF DEM STACK AN?

WANN LEGE ICH EIN OBJEKT / EINE DATENSTRUKTUR AUF DEM HEAP UND WANN AUF DEM STACK AN?



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

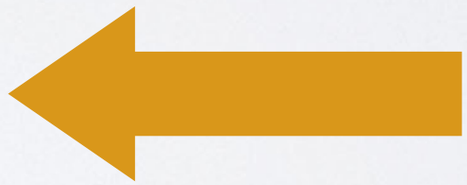
KLASSE IN C++ BEISPIEL

```
// 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

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( )

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`

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