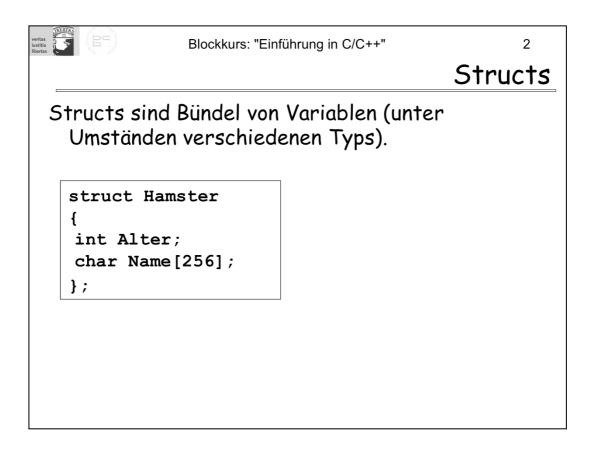


# Programmierkurs C/C++

"Donnerstag"

Sandro Andreotti andreott@inf.fu-berlin.de WS 2012/13

(Slides: Andreas Döring)



Man beachte das ; am Ende der Definition des Structs.

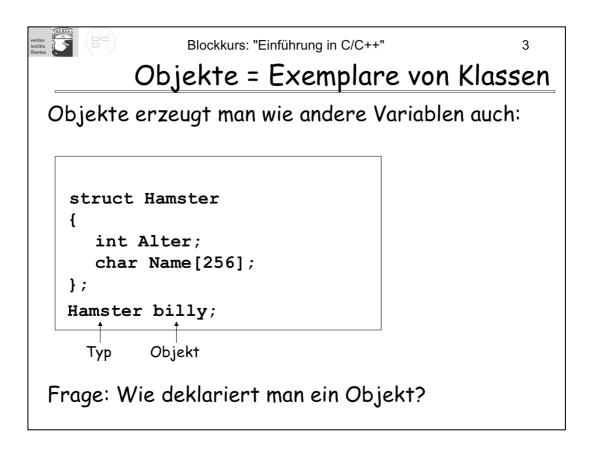
Die Variablen innerhalb eines Structs werden auch die "Datenmember" des Structs genannt. Programmierer anderer objektorientierter Programmiersprachen kennen Datenmember auch unter der Bezeichnung "Properties".

Anmerkung: In C++ wird das Struct-Konzept zum Klassen-Konzept ausgebaut (siehe unten).

In C++ definiert der Code oben bereits einen Typ namens "Hamster". In C muss man dafür wiederum auf typedef zurückgreifen:

```
typedef struct Irgendeinname
{
    int Alter;
    char Name[256];
} Hamster;
```

Statt "Irgendeinname" wäre hier z.B. \_Hamster (mit \_ am Anfang) oder eine andere Variation des späteren Typnamens üblich.



Durch die Definition des Objekts wird Speicher für alle Member-Variablen angelegt.

Ein Objekt deklariert man (ohne es dabei zu definieren) wie jede andere Variable: Mit **extern** (nur bei static-Objekten).



4

### member selection: Der . -Operator

Auf die Member eines Objektes kann man mit dem *member-selection*-Operator . zugreifen:

```
struct Hamster
{
   int Alter;
   char Name[256];
};

Hamster billy;
billy.Alter = 4;
```

Bei der Definition einer Klasse werden noch keine Variablen angelegt. Dies geschieht erst, sobald ein Objekt der Klasse erzeugt wird. In diesem Fall wird ein kompletter Satz aller Datenmember der Klasse erzeugt und kann anschließend über den member selection operator aufgerufen werden.

```
Blockkurs: "Einführung in C/C++"

Der -> -Operator

Hat man einen Pointer auf ein Objekt, so kann man auch den Pfeiloperator -> benutzen.

Hamster billy;
Hamster * mein_hamster = & billy;

mein_hamster->Alter = 3;

ist das gleiche wie:

(*mein_hamster).Alter = 3;
```

Man beachte, dass bei der zweiten schreibweise der Ausdruck (\*mein\_hamster) in Klammern stehen muss, da der member-selection-Operator . stärker bindet als der indirection-Operator \*. Darum bringt der Pfeiloperator -> eine deutliche Verbesserung der Lesbarkeit des Programmes.

```
Blockkurs: "Einführung in C/C++"

Objektpointer als Member

Eine Klasse C kann Member vom Typ C* enthalten.

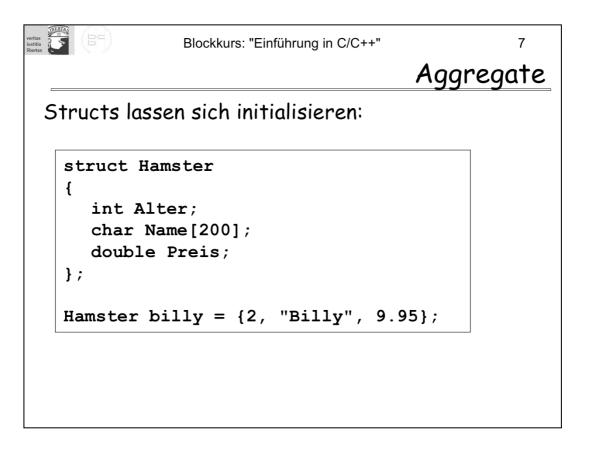
Bsp.:

struct Hamster

{
    Hamster * Sohn;
    Hamster * Mutter;
    };

Hamster nina;
Hamster billy;
billy.Mutter = & nina;
billy.Mutter->Sohn = & billy;
```

Anmerkung: Wie man hier klar erkennen kann, hat jedes Objekt seinen eigenen Satz an Membervariablen. In diesem Beispiel gibt es zwei Objekte **nina** und **billy**, und beide besitzen jeweils ein Member **Sohn** und ein Member **Mutter**.



Die Reihenfolge der Parameter im Aggregat (innerhalb der { }) muss der Reihenfolge der Member des **structs** entsprechen. Im Beispiel wird **Alter** also auf 2, **Name** auf "Billy" und **Preis** auf 9.95 initialisiert.

**Anmerkung:** Diese Art der Initialisierung ist nur bei "C-artigen" structs möglich, nicht aber bei C++-Klassen, die Konstruktoren bzw. Destruktoren besitzen. Für C ++-Klassen braucht man auch keine Aggregate, da es in C++ ja das viel mächtigere Konstruktor-Konzept gibt.



8

# Aufgabe: Verkettete Listen

Schreiben Sie eine Datenstruktur für eine einfach verkettete Liste, in der Alter und Namen von Hamstern gespeichert werden können!

```
struct ListElement
{
   int Alter;
   char Name [200];
   ListElement * Next;
};
```

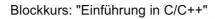


9

## Aufgabe: Element einfügen

Gegeben Pointer auf Listenanfang und Name und Alter eines Hamsters. Fügen Sie ein neues Listenelement an den Anfang der Liste ein!

```
ListElement * insertHamster(
  ListElement * anfang,
  int neues_alter,
  char * neuer_name)
{
  // ???
```





## Lösung: Element einfügen

```
ListElement * insertHamster(
  ListElement * anfang,
  int neues_alter,
  char * neuer_name)
{
    ListElement * neu = new ListElement;

    neu->Alter = neues_alter;
    strcpy(neu->Name, neuer_name);
    neu->Next = anfang;

    return neu;
}
```



11

### aus Structs werden Klassen

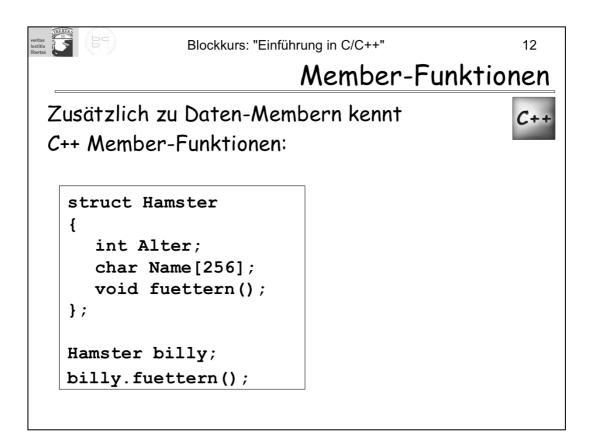
In C++ werden die Structs aus C um etliche Features erweitert:



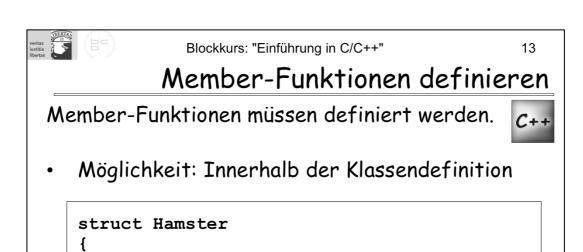
- Member-Funktionen
- Konstruktoren, Destruktoren
- eigene Operatoren
- Vererbung
- ...

Außerdem kann man in C++ auch class Statt struct schreiben.

Der einzige Unterschied zwischen **class** und **struct** in C++: Bei **class** ist vom Default her alles **private**, bei **struct** hingegen **public**, und zwar sowohl bei den Membern als auch bei der Verwerbung (zum Thema private Vererbung siehe unten).



Member-Funktionen kennt man bei anderen Programmiersprachen auch unter dem Namen "Methoden".



//jetzt bekommt der Hamster Futter!

void fuettern()

};

Daten-Member müssen nicht extra Definiert werden: Sie werden bei der Erzeugung eines Objektes automatisch miterzeugt. Dies gilt jedoch nicht für statische Daten-Member (siehe unten).



14

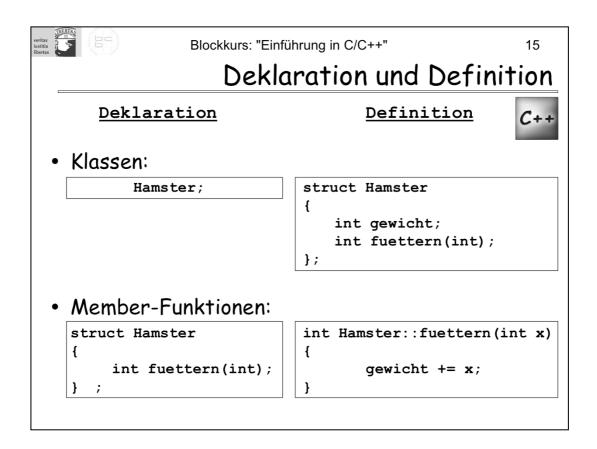
## Member-Funktionen definieren (2)

 Möglichkeit: Außerhalb der Klassendefinition



```
struct Hamster
{
    void fuettern();
};

void Hamster::fuettern()
{
    //jetzt bekommt der Hamster Futter!
}
```



Mehrfache Klassendefinitionen stören zwar den Compiler (bei jedem Compile-Durchgang), jedoch nicht den Linker, da bei einer Klassendefinition kein Code erzeugt wird, der verlinkt werden müsste. Darum ist es auch statthaft, Klassen in Header-Dateien zu definieren, und diese Header-Dateien dann in verschiedene cpp-Dateien einzubinden.

Member-Funktionen können natürlich auch während der Klassendefinition mitdefiniert werden und sind dann automatisch **inline**.

Bei extern definierten Memberfunktionen sieht es hingegen wieder anders aus: Hier würde sich wiederum bei Mehrfachdefinition auch der Linker beschweren – es sei denn, die Memberfunktion ist **inline** definiert.



16

### Klassen programmieren

### Typisches Vorgehen:



- In eine h-Datei: Klassendefinition
- In eine .cpp-Datei: Definition der Member-Funktionen.

Anmerkung: Klassen dürfen also in einer Headerdatei definiert werden.

(Frage: Wieso?)

Verstößt man beim Definieren von Klassen in Headern nicht gegen die "one definition rule", sobald man die Headerdatei in mehr als zwei .cpp-Dateien inkludiert?

Nein, denn: Es spielt keine Rolle, dass die Klasse für jede .cpp-Datei, die den Header einbindet, erneut definiert wird, da eine Klassendefinition keinen Speicherplatz reserviert und auch keinen Code erzeugt. Trotzdem darf eine Klasse nicht zweimal in der gleichen .cpp definiert werden, d.h. die Headerdatei darf von einer .cpp nicht doppelt "includet" werden.

Anmerkung: Im Gegensatz zu gewissen anderen "Kaffee-Programmiersprachen" gibt es in C++ keinen Zwang, die Dateien so zu nennen, wie die Klassen, die sie definieren. Man darf auch ruhig mehrere Klassen in einer Datei definieren, auch wenn dies nicht unbedingt empfohlen wird.



17

## Member-Funktion kennt Objekt

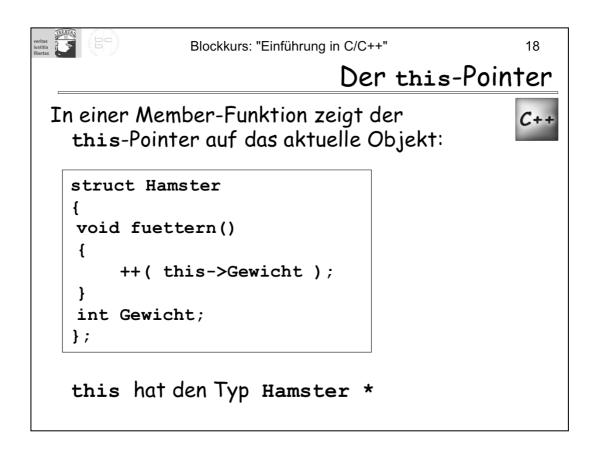
Eine Member-Funktion weiß, zu welchem Objekt sie aufgerufen wurde:



```
struct Hamster
{
  void fuettern()
  {
         ++Gewicht;
  }
  int Gewicht;
};

Hamster billy;
billy.fuettern();
cout << billy.Gewicht;</pre>

cerhöhe
```



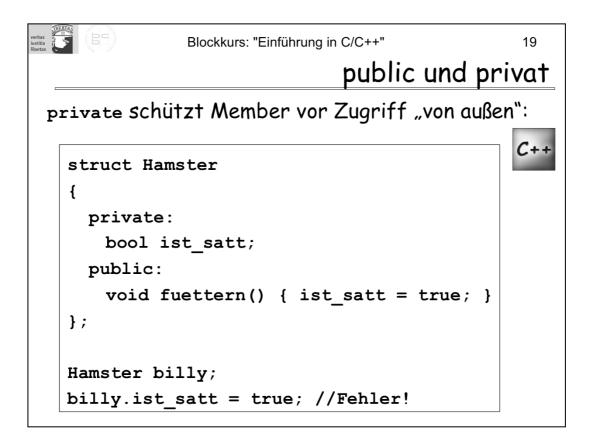
Man hätte in **fuettern** auch einfach schreiben können:

#### ++ Gewicht;

Der **this**-Pointer ist nur innerhalb einer Memberfunktion bekannt, dort aber ohne weitere Deklaration.

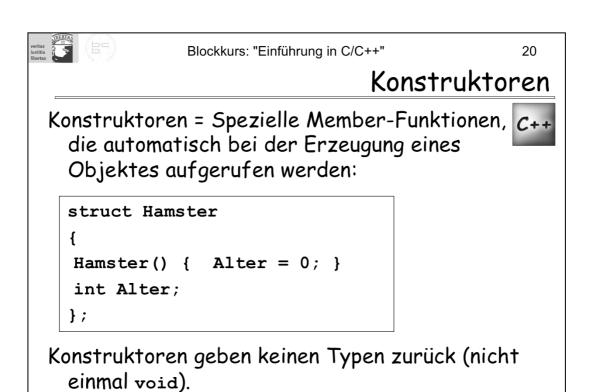
Tatsächlich wird der **this**-Pointer beim Aufruf der Funktion **fuettern()** der Funktion als verstecktes erstes Argument übergeben.

Auch Konstruktoren und der Destruktor (siehe unten) kennen den **this**-pointer.



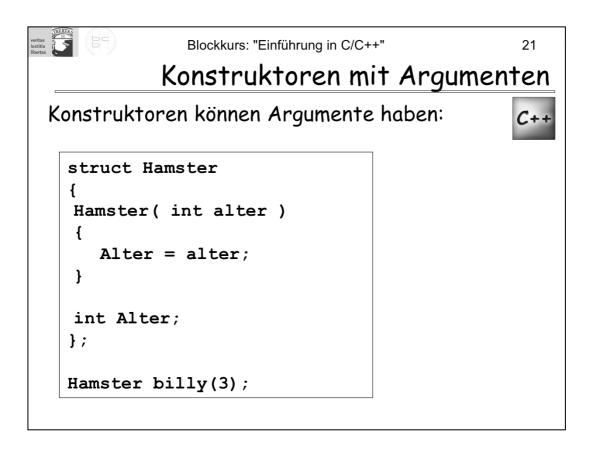
Bei Klassen, die mit dem Schlüsselwort **class** statt **struct** erzeugt werden, sind die Member ohne Angaben des Schlüsselwortes **public** automatisch **private**.

Es gibt außerdem das Schlüsselwort **protected**. Member, die **protected** deklariert worden sind, verhalten sich so, als wären sie **private** deklariert, ausgenommen für abgeleitete Klassen: Eine abgeleitete Klasse kann auf die **protected** Member ihrer Basisklasse (oder der Basisklasse ihrer Basisklasse, usw.) zugreifen.



Konstruktoren werden vor allem dazu verwendet, ein Objekt zu initialisieren.

Ein Konstruktor, der ohne Argument aufgerufen werden kann, wird "Default-Konstruktor" genannt.

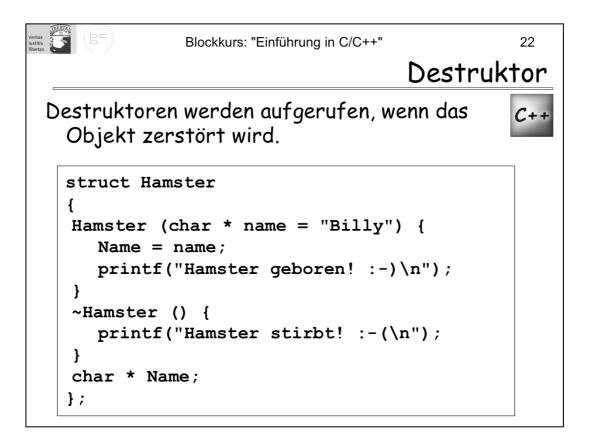


**Wichtig:** Den Default-Konstruktor (ohne Argument) ruft man *ohne Klammern* auf! Wenn es für die Klasse Hamster also einen Default-Konstruktor gibt (wie auf der vorangegangenen Folie), dann schreibt man einfach:

Hamster billy;

nicht aber

Hamster billy(); //Fehler!



(Das Programm geht auf der nächsten Folie weiter)

Bei Destruktoren gibt es keine Möglichkeit der Überladung: Jede Klasse kann nur einen Destruktor haben.



24

## mehrere Konstruktoren

### Konstruktoren sind oft überladen:



```
struct Hamster
{
  Hamster () { Gewicht = 0; }
  Hamster (int X) { Gewicht = X; }
  int Gewicht;
};

Hamster billy(20);
std::cout << billy.Gewicht; //20</pre>
```



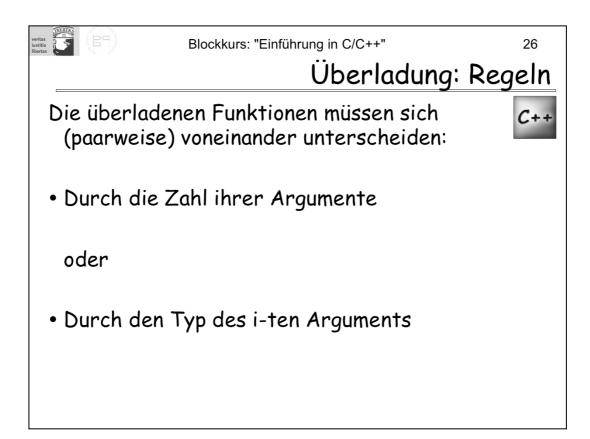
25

## Überladung bei "normalen" Funktionen

Verschiedene Funktionen mit gleichem Namen: C++



```
int max(int a, int b)
  if (a > b) return a;
  return b;
int max(int a, int b, int c)
  return max( max(a, b), c );
```



Es reicht nicht, wenn sich nur der Typ des Rückgabewertes unterschiedet. Bsp.:

```
int f() { ... }
char f() { ... } // Fehler! Beide f haben 0 Argumente.
```



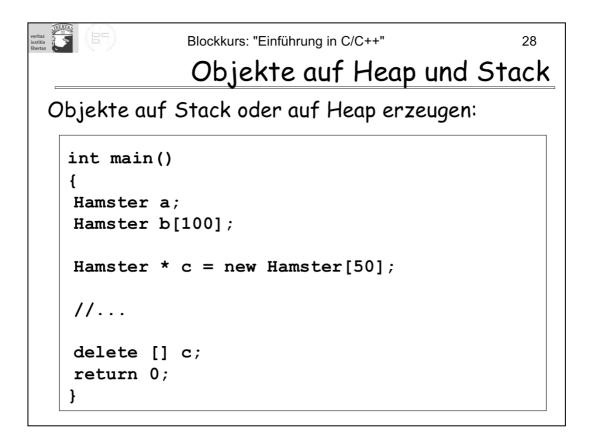
27

## Überladene Member-Funktionen

Memberfunktionen können überladen werden:



```
struct Hamster
{
  void fuettern(int X) { Gewicht += X; }
  void fuettern() { ++Gewicht; }
  int Gewicht;
};
```



Der new Operator ruft die Konstruktoren aller 50 Hamsterobjekte auf.

Man beachte, dass hier der delete []-Operator verwendet werden sollte, damit auch am Ende der Funktion die Destruktoren aller 50 Hamsterobjekte aufgerufen werden.

#### Aufgaben zum Donnerstag:

#### 9. Aufgabe:

Schreiben Sie verschiedene Klassen, wobei sie Anzahl, Typen und die Reihenfolge der Daten-Member variieren. Lassen Sie sich anschließend mit sizeof() die Größe des für die Objektinstanzen benötigten Speichers ausgeben. Probieren Sie ein wenig herum und stellen Sie anschließend eine Theorie darüber auf, wie sich die Größe einer Klasse aus den Membern ableiten lässt.

#### 10. Aufgabe:

Schreiben Sie eine Klasse, die eine doppelt verkettete Liste implementiert und Zahlen vom Typ int speichert. Implementieren Sie die Member-Funktionen insert und extract zum Einfügen eines neuen Elements in die Liste hinter dem aktuellen Element bzw. zum Löschen des aktuellen Elements aus der Liste

#### PRÜFUNGSAUFGABE:

Schreiben Sie eine Klasse "best3", die folgende Member-Funktionen exportiert:

- 1. Eine Member-Funktion "push(unsigned int x)", mit der man der Klasse eine Zahl x übergeben kann.
- 2. Eine Member-Funktion "print()", welche **die Werte der drei größten Zahlen**, die bisher mit "push" übergeben worden sind, auf dem Bildschirm ausgibt. Sollten bisher weniger als drei Zahlen mit "push" eingegeben worden sein, so soll die Funktion auch nur diese bisher eingegebenen Zahlen ausgeben.
- 3. Eine Member-Funktion "count()", die einem die Anzahl der push Operationen für ein Objekt der Klasse zurückgibt.
- 4. Eine Member-Funktion "count\_all()", die einem die Anzahl der ausgeführten push-Operationen für alle Objekte der Klasse zurückgibt.

Hinweis: In einem Objekt vom Typ "best3" müssen tatsächlich zu jedem Zeitpunkt nur drei der bislang eingegebenen Zahlen gespeichert werden, d.h. es ist nicht nötig, alle mit "push" eingegebenen Zahlen in "best3" abzuspeichern.

Denken Sie daran, dass sie bis morgen (Freitag) alle vier Prüfungsaufgaben fertig haben sollten.