Fachhochschule Bingen

Programmieren 2
C++:
Schreibweisen, Konventionen, C-Erweiterungen ...

Prof. Dr. Maximilian Mengel,
Professur Programmiermethodik,
Grundlagen der Informatik und Multimedia
Gebäude 1, Raum 212
Tel.: 06721-409 152
E-Mail: mengel@fh-bingen.de

- Verschiedene Befehle / Vereinbarungen sind in C nicht möglich bzw. werden kaum genutzt.
  In C++ werden diese jedoch benötigt:
  - Übergabe per Referenz

```
void swap(int& a, int& b)
{
    int t;
    t = a;
    a = b;
    b = t;
}
```

    - Aufruf: `swap(x,y); // OHNE &-Operator!!!`
  - Vorteil: einfachere Schreibweise innerhalb der Funktion oder Methode
  - Nachteil: Verwechslungsgefahr mit Call by Value

- ...
  - Per Referenz übergebene Parameter als const vereinbart

```
class complex {
    ...
    public:
      void copy(const complex& z1);
    ...
}
```

    - Der Parameter wird zwar on Reference übergeben, darf aber nicht verändert werden
    - Vorteil: Es wird keine Kopie des übergebenen Objekts erzeugt und das Objekt kann dennoch nicht verändert werden => Speicherplatz sparend, schnell und sicher!

- Für Funktions- oder Methodenparameter können Default-Werte vorgegeben werden
- Die Default-Werte können beim Aufruf überschrieben werden
- Wenn der Default-Wert nicht überschrieben werden soll, wird der Parameter beim Aufruf nicht angegeben
- Wird ein Parameter nicht angegeben, so dürfen auch alle weiteren rechts davon stehenden Parameter nicht angegeben sein

Vorbesetzte Parameter: Beispiel Funktionen

```
int add(int a, int b, int c=0, int d=0)
{
   return a+b+c+d;
}

int mult(int a, int b, int c=1, int d=1)
{
   return a*b*c*d;
}

main()
{
   cout << add(2,3) << add(2,3,4) << add(2,3,4,5);
   cout << mult(2,3) << mult(2,3,4) << mult(2,3,4,5);
}
```


Vorbesetzte Parameter: Fehler

```
int add(int a=0, int b, int c=0, int d=0) // FEHLER
{
   return a+b+c+d;
}

int mult(int a, int b, int c=1, int d=1)
{
   return a*b*c*d;
}

main()
{
   cout << mult(2,3,,4); // FEHLER
   cout << mult(2);      // FEHLER
}
```


C++ Spracherweiterungen: Überladene Funktionen

- Mehrere Funktionen (oder Methoden) dürfen den gleichen Namen haben, solange Ihre Parameterdeklarationen verschieden sind
- Es müssen entweder...
  - die Parametertypen,
  - die Anzahl der Parameter,
  - oder Beides
- ... verschieden sein
- Es muss beachtet werden dass auch bei Defaultparametern keine Mehrdeutigkeiten entstehen
- **PS.: Bei verschiedenen Klassen dürfen Methoden sowieso gleich lauten!!!**

Beispiel: Überladene Funktionen

```
int add(int a, int b, int c=0, int d=0)
{
   return a+b+c+d;
}

double add(double a, double b, double c=1, double d=1)
{
   return a+b+c+d;
}

main()
{
   cout << add(2,3) << add(2,3,4) << add(2,3,4,5);
   cout << add(2.2,3.1) << add(2.0,3.1,4) << add(2,3,4,5.0);
}
```

Neue Schreibweise / Einbindung von Header-Dateien

- In ANSI C++ werden für die Standard-Bibliotheken keine Dateinamen sondern Standardbezeichner angegeben
  - Da die Bezeichner keine Dateien direkt benennen existieren auch keine Dateierweiterungen (insbesondere kein *.h)
- ANSI C++ nutzt Namensräume
  - Ein Namensraum ist ein Gültigkeitsbereich für Bezeichner wie Klassen, Variablen, Funktionen, ...
  - Alle in Standard-Bibliotheken enthaltenen Bezeichner befinden sich im Namensraum std
  - Um diese zu nutzen muss dieser Namensraum mit der using-Anweisung bekannt gemacht werden

Alte Header / Alte Compiler

- Alte C-Header
  - Um C-Headerdateien einzubinden nutzt man entweder die „alten“ Datei-Namen oder man nutzt – **besser** – den alten Namen ohne das .h und mit einem c vor dem Namen
    - math.h => cmath oder string.h => cstring
- Viele ältere (und auch noch einige aktuelle) Compiler unterstützen noch nicht die neuen Header und die Namensräume
  - Das Einbinden von Header-Dateien geschieht dann wie bei C (mit Endung .h und ohne using).

Beispiel/Gegenüberstellung

- ANSI C++

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int i = 5;
    cout << “ANSI C++\n“;
    cout << pow(2,i);
}
```

- Alte Compiler

```
#include <iostream.h>
#include <math.h>

int main()
{
    int i = 5;
    cout << “Altes C/C++\n“;
    cout << pow(2,i);
}
```

Auch wenn zur Zeit praktisch alle Compiler die „alte“ Schreibweise und nur einige neue die ANSI-Schreibweise verstehen, sollten Sie sich dennoch die neue Schreibweise angewöhnen, da diese die in Zukunft unterstützte Schreibweise sein wird !!!

Wiederholung Konstruktor

- Der Konstruktor einer Klasse heißt genauso wie die Klasse selbst
  - Der Konstruktor wird beim Instanzieren des Objektes ausgeführt
  - Im Konstruktor initialisiert man interne Variablen und fordert eventuell benötigten Speicherplatz an
  - Konstruktor besitzen keinen Rückgabewert (auch nicht void)
  - Der Konstruktor kann, wie jede Methode, Parameter übergeben bekommen
  - Der Konstruktor ohne Parameter heißt Standardkonstruktor
  - Wenn man keinen Konstruktor angibt wird vom System automatisch ein default-Standardkonstruktor erzeugt

Konstruktoren 2: Copy-Konstruktor

- Ein besonderer Konstruktor ist der sog. Copy-Konstruktor, der ein Objekt mit den Werten eines anderen Objektes der selben Klasse initialisiert
  - Wenn kein Copy-Konstruktor angegeben ist, wird einer bei Bedarf vom System bereitgestellt
    - Der vom System bereitgestellte Copy-Konstruktor erzeugt eine bitweise Kopie des on Reference übergebenen Originals
  - Wenn immer dynamisch Speicher angefordert wird sollte ein Copy-Konstruktor implementiert werden

Beispiel

```
class int stack{
        int top;
        int size;
        int* stck;
public:
        stack(int i = 16);
        stack(const stack& s);
        void push(int wert);
        int pop();
        ...
};

stack::stack(int i )
{
        top = 0;
        size = i;
        stck = new int[size];
}
```

```
stack::stack(const stack& s )
{
        top = s.top;
        size = s.size;
        stck = new int[size];
        for (int i=0; i<size; ++i)
                stck[i] = s.stck[i];
}

void stack::push(int wert)
{
...
}
int stack::pop()
{
...
}

...
```


Der Zeiger this

- Innerhalb jedes Objektes existiert ein Zeiger auf das eigene Objekt: **`this`**
- Somit kann man jedem Zugriff auf eine Member-Variable oder eine Methode **`this->`** voranstellen
- Damit lassen sich z.B. Mehrdeutigkeiten zwischen Übergabeparameter und Membervariablen vermeiden
- Beispiel:

```
class demo {
      int a;
public:
      seta(int a)
            { this->a = a; };
...
}
```