Einführung in C++ (Erinnerungen an C)

Carsten Gips (HSBI)

Warum?

- C++ erlaubt ressourcenschonende Programmierung
- Objektorientierter “Aufsatz” auf C
- Verbreitet bei hardwarenaher und/oder rechenintensiver Software

Sie werden C++ im Modul “Computergrafik” brauchen!

Warum?

- C++ erlaubt ressourcenschonende Programmierung
- Objektorientierter "Aufsatz" auf C
- Verbreitet bei hardwarenaher und/oder rechenintensiver Software

Sie werden C++ im Modul "Computergrafik" brauchen!

Geschichte

- 1971-73: Ritchie entwickelt die Sprache C
- Ab 1979: Entwicklung von C++ durch Bjarne Stroustrup bei AT&T
  - Erweiterung der prozeduralen Sprache C
  - Ursprünglich "C mit Klassen", später "C++" (Inkrement-Operator)
- Bis heute: Fortlaufende Erweiterungen: alle 3 Jahre neuer Standard (C++11, C++14, ...)

Hello World!

```cpp
/*
 * HelloWorld.cpp (g++ -Wall HelloWorld.cpp)
 */

#include <cstdio>
#include <iostream>
#include <cstdlib>

using namespace std;

int main() {
    printf("Hello World from C++  :-)\n");
    cout << "Hello World from C++  :-)" << endl;
    std::cout << "Hello World from C++  :-)" << std::endl;

    return EXIT_SUCCESS;
}
```

Konsole: HelloWorld.cpp

Im Wesentlichen wie von C und Java gewohnt ... :-)

- Wichtig(st)e Abweichung:

  Im booleschen Kontext wird `int` als Wahrheitswert interpretiert:
  Alle Werte ungleich 0 entsprechen `true` (!)

=> Vorsicht mit

```
int c;
if (c=4) { ... }
```

Ein- und Ausgabe mit *printf* und *cin/cout*

- `printf(formatstring, ...)`

```
string foo = "fluppie";
printf("hello world : %s\n", foo.c_str());
```

- Standardkanäle: `cin` (Standardeingabe), `cout` (Standardausgabe), `cerr` (Standardfehlerausgabe)

```
// Ausgabe, auch verkettet
string foo = "fluppie";
cout << "hello world : " << foo << endl;

// liest alle Ziffern bis zum ersten Nicht-Ziffernzeichen
// (fuehrende Whitespaces werden ignoriert!)
int zahl; cin >> zahl;
```

Beispiel: cin.cpp

Sichtbarkeit und Gültigkeit und Namespaces

- C++ enthält den Scope-Operator `::` => Zugriff auf global sichtbare Variablen

```
int a=1;
int main() {
    int a = 10;
    cout << "lokal: " << a << "global: " << ::a << endl;
}
```

- Alle Namen aus `XYZ` zugänglich machen: `using namespace XYZ;`

```
using namespace std;
cout << "Hello World" << endl;
```

- Alternativ gezielter Zugriff auf einzelne Namen: `XYZ::name`

```
std::cout << "Hello World" << std::endl;
```

Beispiel: cppScope.cpp

Arrays und Vektoren in C++

- Syntax: `Typ Name[AnzahlElemente];`

```
int myArray[100];
int myArray2[] = {1, 2, 3, 4};
```

- Vordefinierter Vektor-Datentyp `vector`

```
vector<int> v(10);
vector<double> meinVektor = {1.1, 2.2, 3.3, 4.4};
meinVektor.push_back(5.5);
cout << meinVektor.size() << endl;
```

```
cout << v[0] << endl;          // ohne Bereichspruefung!
cout << v.at(1000) << endl;    // mit interner Bereichspruefung
```

```
vector<double> andererVektor;
andererVektor = meinVektor;
```

Alias-Namen für Typen mit *typedef* und *using*

- Syntax: `typedef existTyp neuerName;` (C, C++)

```
typedef unsigned long uint32;
uint32 x, y, z;
```

- Syntax: `using neuerName = existTyp;` (C++)

```
typedef unsigned long uint32;        // C, C++
using uint32 = unsigned long;        // C++11

typedef std::vector<int> foo;        // C, C++
using foo = std::vector<int>;        // C++11

typedef void (*fp)(int,double);      // C, C++
using fp = void (*)(int,double);     // C++11
```

Wrap-Up

- C/C++ sind enge Verwandte: kompilierte Sprachen, C++ fügt OO hinzu
- Funktionsweise einfachster Make-Files

- Wichtigste Unterschiede zu Java
  - Kontrollfluss wie in Java
  - Basisdatentypen vorhanden
  - Typ-Modifikatoren zur Steuerung des Speicherbedarfs/Wertebereich
  - Integer können im booleschen Kontext ausgewertet werden
  - Operator `sizeof` zur Bestimmung des Speicherbedarfs
  - Alias-Namen für existierende Typen mit `typedef` definierbar
  - Funktionen mit Default-Parametern und Überladung

LICENSE