Tut 6 (Tutor) 2nd Ed.

Monday, 26. November 2018 19:16

O. WIEDERHOLUNG

- Es gibt kein #throwback

I.V.
- Der Präprozessor
- Von C zu C++

1. PRÄPROZESSOR

- Was macht ein Präprozessor ?

→ Vor dem Kompilieren werden Präprozessoranweisungen abgearbeitet !

PRÄPROZESSOR

... Anweisungen mit #

1.1 Verteilter Programmcode

- Wir kennen #include.

→ #include <stdio.h> fügt den Inhalt von der Header-Datei stdio mit Copy & Paste ein

```
#include { <fixyou.h>    Standardbibliothek
         { "fixyou.h"    Kompilierverzeichnis
```

- Unterschied ?

→ < > für allg. Standardbibliotheken
" " für 'selbstgeschriebene' Header-Files

- Wozu eigene Libraries schreiben in .h ?

→ Folien 'Million Lines of Code'
→ Motivation: Übersicht durch modulare Gestaltung

- Was in .h und was in .c ?

→ Deklarationen in .h : 'Bedienungsanleitung'
→ Actual Code (Implementierungen) in .c

- .c-Dateien einzeln kompilieren

→ Kompilieren kann Tage dauern. Alternativ: Einzeln kompilieren & anschließend zusammenlinken

```
gcc scientist.c -std=c11 -c
```

nur kompilieren. Produziert Object-File 'scientist.o'

- Nun Linken mit

```
gcc main.c scientist.o -std=c11 -o main
```

'Object Linking'

1.2 Makros

- Weitere Präprozessorbefehle: #define
→ Makros mit #define

Makros mit #define

```
# define   PI    3.14159265358979
# define   D_PI  (2 * PI)
```

Zeilenumbruch mit \

```
# define   ISNEG(a)    if ((a)<0) printf ("a<0"); \
                       else printf (" a>=0")
```

- Copy & Paste vom Compiler
- Klammer setzen sehr wichtig !!
→ HA-Vorschau

2. DIE PROGRAMMIERSPRACHE C++

- C++ erweitert C u.a. mit Objektorierung
  → Abstraktere Programme
- Was ändert sich ?

  → Alles & nichts.
  → Das 'nichts' arbeite ich schnell ab; der Rest des Tuts dreht sich um das 'alles'

- Nichts: Man kann ein C++ Prog. schreiben ohne C++.

2.1 Kompilieren, Dateinamen

Von C zu C++ .c → .cpp
.h → .hpp

```
g++ paradise.cpp -std=C++11 -Wall -o paradise
```

2.2 Wörterbuch C ↔ C++ ↔ Deutsch

URBAN GEEKTIONARY

C	C++	Deutsch
`#include <stdio.h>`	`#include <iostream>` ohne .h/.hpp Endung	Standardbibliothek einbinden
`printf ("x = %d\n", x);` mit printf-Fkt.	`std::cout << "x = " << x << std::endl;` im 'Stream' mit std::cout	Konsolenausgabe
`scanf ("%d", &y);` mit scanf-Fkt.	`std :: cin >> y;` im 'Stream' mit std::cin. ⚠ Richtung !	Eingabel Einlesen einer Variable
`char hi[3] = "hi";` sind '\0'-terminierte char-Arrays	`std :: string hi = "hi";` neuer, eigener Datentyp String !	Darst. v. Zeichenketten
`int var = 1;` jeder Datentyp ≠0 ist wahr	`bool var = true;` neuer DT mit true == 1, false == 0	Wahr oder falsch ? Binärer Datentyp
`NULL`	`nullptr`	Der Nullpointer
`int *arr = malloc (5* sizeof(int));` mit malloc-Funktion	`int *arr = new int[5];` mit Schlüsselwort 'new'	Reservieren eines dynamischen Arrays
`free (arr);` mit free-Funktion	`delete [] arr;` mit delete-Operator. [ ] kennzeichnet, dass arr ein Array ist.	Speicherfreigabe

- String: In C ist ein String ein Char-Array. Ein C-String bestand aus chars und einem Terminator, '\0' bzw. numerisch 0

- Boolean: In C gab es keinen Typ mit nur zwei Zuständen
  → Stattdessen war alles != 0 wahr (true) und = 0 falsch (false)
  → C++: Datentyp Boolean 'bool' nativ !

2.3 Namespaces & Referenzen

- C++ Exclusives im Doppelpack.

Namespaces

Zugriff per "Scope-resolution" ::

```
namespace sky {            |  main.cpp :
                           |
    int stars;             |  using namespace sky;
                           |  ~~sky::~~ stars = 5000;
    void moreStars(){      |  ~~sky::~~ moreStars();
        ++ stars;          |
    }                      |
}                          |
```

- Wozu Namespaces ?
→ Mehrfaches Verwenden von Variablen & Fkt.-Namen ohne Konflikte

- Kein Bock auf :: ? → using namespace
→ z.B. using namespace std;

Referenzen

```
int x = 5;
int &a = x;
```

" a ist ein weiterer Name von x "

→ Inhalt & Adresse identisch !

- Wozu Referenzen ?
→ Call by Reference (Adresse durchgeben)
"Ändere genau diese Variable, nicht eine Kopie davon"
→ Pointer-Alternative mit einfacherer Schreibweise
- ns-ref.cpp - Beispiel

2.3 Strings & Stringstreams

- Wie haben wir in C Strings zugewiesen ?
  → Elementweise im Array mit for ... != 0
- C++-String erlaubt direkte Zuweisung

C++ Strings

```
string str = "hi";
str =  "hej";
     ↑ direkt zuweisen !
```

- Interessant: Stringstreams

Stringstreams

```
stringstream sStr;
sStr << "x = " << x;        ← in den Fluss schieben
cout << sStr.str() << endl;   // Ausgabe
double y = 0.0f;
sStr >> y;                  → aus dem Fluss 'fischen'
```

2.4 Eingabe in C++

- Wie werden Strings in C++ eingelesen ?

i Strings einlesen

```
cout << "Eingabe : ";
string zeile;
getline (cin, zeile);
```

- Achtung: Bei (abwechselnder) Verwendung von cin und getline können Zeilenumbrüche ("Enter") fälschlicherweise als Line interpretiert werden

→ cin >> ws

entfernt mögliche whitespaces

i Whitespaces ignorieren

```
cin >> ws;
```

happy(cplusplus.exploring);
cplusplus.adventure(begin);

@frauki #lettheadventurebegin
#wishyouwerenorthwest