Programmieren in C++

SS 2011

Vorlesung 3, Mittwoch 18. Mai 2011

(Grundlegende Konstrukte, mehr zu make, Vim)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - ECTS Punkte / Arbeitsaufwand / Note für die Veranstaltung
  - Erfahrungen mit dem 2. Übungsblatt
- Grundlegende Konstrukte
  - Variablen, Ausdrücke, while, for, if, else
  - Damit sollen sie dann ein Apfelmännchen malen
  - In der Vorlesung malen mir zusammen einen Kreis
- Außerdem
  - machen wir unser Makefile generischer
  - Compileroption -Wall
  - kurze Vim Einführung, zweiter Versuch

ECTS Punkte

- Für die Informatiker (= fast alle dieses Mal)
  - 4 ECTS Punkte = 120 Arbeitsstunden für das Semester
  - Davon ca. 30 Stunden für das Projekt am Ende
  - Bleiben ca. 7 Stunden / Vorlesungswoche (13 Termine)
  - Also ca. 5 Stunden pro Übungsblatt
- Für die Nicht-Informatiker
  - 6 ECTS Punkte

Erfahrungen mit dem 2. Übungsblatt

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 18.5. 15:18
  - Übungsblatt war bei den meisten schnell erledigt
  - Erklärungen zum Hintergrund (make, Bibliotheken, etc) waren interessant
  - 10 mal weniger Lint-Fehler als bei Blatt 1
  - Danke für das https statt http
  - Probleme mit dem SVN, insbesondere mit dem "relocate"
  - nm steht für "name"
  - Einige machen parallel zur Vorlesung das Übungsblatt
    - ... oder parallel zur Vorlesungsaufzeichnung

Variablen

UNI FREIBURG

- Die elementaren Datentypen
  - int = ganze Zahl, 4 bytes ($-2^{31}..2^{31}-1$) oder mehr
  - char = ein einzelnes ASCII Zeichen, 1 byte
  - float = Fließkommazahl, 4 bytes oder mehr
  - double = Fließkommazahl, 8 bytes oder mehr
  - bool = true (wahr) oder false (falsch)
- Deklaration
  - Variablen müssen vor der Benutzung deklariert werden
  - Können dabei auch gleich initialisiert werden, sonst ist der Wert bis zur ersten Zuweisung zufällig

```
int x;
int y = 10;
printf("%d %d\n, x, y);  // Wert für x ist zufällig
```

Zuweisung / Ausdrücke 1/2

UNI FREIBURG

- Zuweisung

  i = 2 * j; // Left side must be a variable.

- Ausdrücke
  - Im Wesentlichen beliebige geklammerte Ausdrücke mit den Operatoren +, -, *, /, % (modulo)

    17 * (x – y / 2) + 32 * x * y / (5 – numValues)

  - Für Ausdrücke vom Typ bool gibt es
    - die Operatoren && (und), || (oder), ! (nicht)
    - die Vergleichsoperatoren <, >, <=, >=, ==, !=
  - Dann gibt es noch die bitweisen Operatoren | und & und die Bitschiebeoperatoren << und >> siehe Referenzen!

Zuweisung / Ausdrücke 2/2

UNI FREIBURG

- Abkürzungen für häufige Konstrukte

```
i++; // Identical to i = i + 1.
i--; // Identical to i = i – 1;
i +=3; // Identical to i = i + 3. Also works for *, -, and /.
```

- Der 3-Wege Operator (3-way operator)

```
min = (x < y ? x : y);  // The minimum of x and y.
```

  – Allgemeine Form ist

    condition ? expression1 : expression2

    Der Wert des Ausdrucks ist expression1 wenn condition wahr ist und sonst expression2

    expression1 und expression2 müssen vom selben Typ sein!

while Schleife

- Das grundlegende Schleifenkonstrukt in C / C++

```
int i = 0;
while (i < 10)
{
   printf("%d\n", i);
   i++;
}
```

for Schleife

- Abkürzung für eine bestimte Art while Schleife

```
for (int i = 0; i < 10; i++)
{
   printf("%d\n", i);
}
```

```
int i = 0;
int j = 0;
int x;
x = i++; // x=0
x = ++j; // x=1
```

  – Konvention: for nur bei **einer** Schleifenvariablen, und relativ **einfacher** Abbruchbedingung, sonst while

```
for (int i = 0, int j = 10; i < j; i++, j--)  // Too complex,
{                                              //  better use while!
   printf("%d %d\n", i, j);
}
```

if … else

- Konditionale Ausführung

```
if (condition)
{
  <code block 1>;
}
else
{
  <code block 2>;
}
```

  - Falls condition wahr ist, wird <code block 1> ausgeführt, sonst <code block 2>
  - Der else Teil kann auch fehlen, dann wird bei falscher condition an dieser Stelle gar kein Code ausgeführt

Makefile 1/2

- Pattern-Regeln

  %Test.o: %Test.cpp

  <list of commands>

  – Wird angewendet für jedes target, das zu %Test.o passt
    - zum Beispiel bei make CircleTest.o
  – Das % auf der rechten Seite wird dann entsprechend ersetzt
    - in dem Beispiel durch Circle
- Automatische Variablen (beim Match einer Pattern-Regel)

  $@ ist das konkrete target

  $* ist dieses target ohne Suffix (z.B. CircleTest.cpp → CircleTest)

  $< ist die erste dependency nach dem target

  $^ sind alle dependencies nach dem target (mit Leerz. getrennt)

Makefile 2/2

- Variablen

  MAIN_BINARIES = CircleMain.cpp

  Wie in einem C++ Programm, nur ohne Variablentyp

- Funktionen

  $(wildcard *.cpp)

  Alle Dateien (im aktuellen Ordner) die auf *.cpp matchen

  $(basename Circle.o CircleMain.cpp CircleTest.o)

  Die Dateinamen ohne Ihre Suffixe (Circle, CircleMain, CircleTest)

  $(filter %Main.cpp, <list of strings>)

  $(filter-out %Main.cpp, <list of strings>)

  $(addsuffix .o, <list of strings>)

  Siehe Referenzen

g++ ... -Wall

- Mit –Wall ...
  - ... zeigt g++ auch Warnungen zu Konstrukten, die unter bestimmten Umständen korrekt sein können
  - ... die aber in der Regel (und gerade bei Anfängern) auf einen Fehler andeuten
  - ... und die man so oder so leicht vermeiden kann indem man den Code eindeutiger schreibt
  - Beispiele

    ... warning: 'xyz' may be used uninitialized in this function

    ... warning: unused variable 'xyz'

    ... warning: no return statement in function returning non-void

Kommentare in der .h und .cpp Datei

UNI FREIBURG

- Den Kopf einer Funktion schreiben wir zweimal
  - einmal in der .h Datei, zur Deklaration
  - einmal in der .cpp Datei, vor der Implementierung
- Konvention
  - Die Dokumentation zur Funktion **nur** in der .h Datei
  - In der .cpp Datei gar nichts oder // _____
  - Grund: Sonst derselbe Text an zwei Stellen, und es ist nur eine Frage der Zeit, bis die beiden Texte nicht mehr synchron sind
  - Und die .h Datei ist sozusagen die Übersicht über alle Funktionen, von daher passt da auch die Doku gut hin

Literatur / Links

- if, else, while, for, switch, break, ...
  - http://www.cplusplus.com/doc/tutorial/control
- Ausdrücke und Operatoren
  - http://www.cplusplus.com/doc/tutorial/operators/
- Variablen und Datentypen
  - http://www.cplusplus.com/doc/tutorial/variables/
- Makefile patterns / Variablen / Funktionen
  - http://www.gnu.org/software/make/manual/make.html#Pattern-Rules
  - .../make.html#Automatic-Variables
  - .../make.html#Functions