Programmieren in C++

SS 2012

Vorlesung 3, Dienstag 15. Mai 2012

(Grundlegende Konstrukte, mehr zu make)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem 2. Übungsblatt
  - Nachtrag zu: header guards, Linker Standardpfade, ...
  - Wann machen wir endlich C++?
- Grundlegende Konstrukte
  - Variablen, Ausdrücke, while, for, break, continue, if, else, switch
  - Zur Übung sollen Sie eine kleine Animation implementieren
  - Ich mache Ihnen in der Vorlesung ein einfaches Beispiel vor
- Außerdem
  - ... machen wir unser Makefile generischer

Erfahrungen mit dem Ü2 (.h Dateien / make)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 15.5. 12:55
  - Übungsblatt relativ einfach und schnell gemacht
  - Insbesondere durch Beispiel aus der Vorlesung
  - Übungsblatt schon während der Vorlesung gemacht
  - Interessant / einige seit Urzeiten angestaute Fragen geklärt
  - SVN, gtest, checkstyle … klappt alles schon viel besser
  - Insbesondere: gute gtest Erklärung auf dem Wiki
  - Danke für das Feedback von Tutor / Tutorin
  - checkstyle will andere header guard lokal wie auf Jenkins!?
  - Wie bekomme ich die Gewitterwolke in Jenkins weg?
  - Linker kennt /usr/local/lib nicht (zur Laufzeit des Programms)
  - cpplint.py läuft nicht mit python3

Erfahrungen mit dem Ü2 (.h Dateien / make)

UNI FREIBURG

- Fortsetzung …
  - vimtutor hilfreich zum Einstieg in Vim
  - Sehr gut, dass "ganz unten" angefangen wird, so versteht man mal was wirklich im Hintergrund abläuft
  - Wann kommen wir endlich zum Programmieren in C++?
  - Zahlreiche Kommentare zum Tempo der Vorlesung
    - **passend** / in Ordnung / zügig aber nicht zu schnell
    - etwas **zu schnell** … aber Aufzeichnung hilft dann
    - **zu langsam** … schwer dabei aufmerksam zu bleiben
    - Konzentration lässt irgendwann nach, vielleicht kurze Pause?

Header guards lokal vs. auf Jenkins

UNI FREIBURG

- Die Variablennamen von den header guards
  - … müssen eindeutig für jede Datei sein
  - Per default nimmt checkstyle deswegen einfach den **absoluten** (und oft *sehr* langen) Pfad, z.B.

    HOME_BAST_TEACHING_CPPSS2012_VORLESUNGEN_VORLESUNG3_LEAPYEAR_H_
  - In einer SVN Arbeitskopie reicht aber der **relative** Pfad ab dem Oberverzeichnis der Arbeitskopie, z.B.

    VORLESUNGEN_VORLESUNG3_LEAPYEAR_H_
  - Das funktioniert aber nur, wenn Sie auch in einer SVN Arbeitskopie sind (und der aktuelle Ordner auch)
    - cpplint.py erkennt das an den (versteckten) .svn Ordnern, die im jeden Verzeichnis einer Arbeitskopie stehen

Linker — Standardpfade

- Die Standardpfade variieren von System zu System
  - Es gibt: /lib, /usr/lib, /usr/local/lib, usw.
  - Falls der Linker Bibliotheken zur Laufzeit nicht findet ...

    error while loading shared libraries: libgtest.so.0: cannot open shared object file

    ... wie in VL2 erklärt den LD_LIBRARY_PATH setzen, z.B.

    export LD_LIBRARY_PATH=/usr/local/lib
  - Man kann dem Linker aber auch ein für alle Mal einen neuen Standardpfad mitteilen

    sudo echo "/usr/local/lib" >> /etc/ld.so.conf.d/local.conf
    sudo ldconfig

SVN — Troubleshooting 1/2

- Wenn alles klappt wie es soll, ist es sehr einfach
  - Zum Anlegen eines neuen Unterverzeichnisses

    svn mkdir uebungsblatt-3
    svn commit uebungsblatt-3
  - Beim erstmaligen Hochladen neuer Dateien

    svn add Makefile *.h *.cpp
    svn commit Makefile *.h *.cpp
  - Beim Hochladen von Änderungen in bestehenden Dateien

    svn commit <Datei 1> <Datei 2> ...
- Wenn es Probleme gibt, wird es schnell schwierig
  - Man muss sich schon ziemlich gut mit SVN und seiner Funktionsweise auskennen um die dann im Einzelfall zu lösen

SVN — Troubleshooting 2/2

- Die beste Strategie **ohne** tieferes Verständnis:
  - Die ganze Arbeitskopie irgendwo anders hinschieben

    mv hb42 hb42.OLD
  - Eine neue Arbeitskopie auschecken

    svn checkout https://daphne.../svn/Progr...SS2012/hb42
  - In der Arbeitskopie gewünschte neue Unterordner anlegen, und committen wie auf der Folie vorher erklärt
  - Gewünschte Dateien aus der verschobenen Arbeitskopie in die neue Arbeitskopie kopieren

    cp hb42.OLD/uebungsblatt-3/Doof.h hb42/uebungsblatt-3
  - Die Dateien in der neuen AK adden und committen wie gehabt

    svn add Doof.h gefolgt von svn commit Doof.h

Wann lernen wir C++?

UNI FREIBURG

- In erster Linie sollen Sie hier Programmieren lernen
  - Und in zweiter Linie C++
  - Die Erfahrung zeigt (mir), dass die allermeisten Probleme bei den absoluten Grundlagen liegen
    - nicht nur bei Anfängern, sondern auch bei älteren Semestern / Leuten, die schon länger programmieren
  - Auch in der Praxis (egal ob Uni oder Firma) zählt vor allem das sichere Beherrschen der Grundlagen
    - was nutzt ein tolles Design Pattern, wenn man nicht mal einen 10-Zeiler auf die einfachste, effizienteste und natürlichste Art programmieren kann
    - wenn man umgekehrt Letzteres beherrscht, kann man fast schon alles was man braucht

Variablen

- 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

- 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

- 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!

Schleifen: while und for 1/2

- while = grundlegendes Schleifenkonstrukt in C / C+

```
// Print the numbers from 1 to 10.
int i = 1;
while (i <= 10) {
  printf("%d\n", i);
  i++;
}
```

- Äquivalent, übersichtlicher und kürzer als for Schleife

```
// Print the numbers from 1 to 10.
for (int i = 1; i <= 10; i++) {
  printf("%d\n", i);
}
```

Schleifen: while und for 2/2

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

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

// Equivalent while loop, longer but easier to understand.
int i = 0;
int j = 10;
while (i < j) {
  printf("%d %d\n", i, j);
  i++;
  j--;
}
```

Schleifen: break und continue

- Sehr nützlich ...
  - um eine Schleife vorzeitig abzubrechen (break)
  - oder einzelne Iterationen zu überspringen (continue)

```
// Let the user enter years, and for each year >= 1582 say
// whether it's a leap year or not. Stop when -1 is entered.
char line[1024];  // Space for 1024 characters.
while (true) {
  fgets(line, 1024, stdin);  // Read line from console.
  int n = atoi(line);  // Convert to int.
  if (n == -1) break;  // Leave loop when user entered -1.
  if (n < 1582) continue;  // Gregorian calendar starts 1582.
  printf("%d is %s leap year\n", n, isLeapYear(n) ? "a" : "no");
}
```

Konditionale Ausführung: if ... else

- Je nach aktueller Bedingung, diesen oder jenen Code:

```
if (condition) {
  // Code block 1.
  ...
} else {
  // Code block 2.
  ...
}
```

Falls condition wahr ist, wird Code unter Code block 1 ausgeführt, sonst der Code unter Code block 2

– Der else Teil kann auch fehlen, dann wird bei falscher condition an dieser Stelle gar kein Code ausgeführt

Konditionale Ausführung: switch

- Bei mehr als zwei Möglichkeiten

```
// Number of days for the given month and year.
int numberOfDays(int month, int year) {
  int result;
  switch (month) {
    case 1: result = 31; break;
    case 2: result = isLeapYear(year) ? 29 : 28; break;
    case 3: result = 31; break;
    ...
    case 12: result = 31; break;
    default: result = -1;  // not a valid month;
  }
  return result;
}
```

Variablendeklaration mit "extern"

- Aus der VL2 wissen wir, wie man Funktionen
  - … aus verschiedenen .cpp files zusammen"linkt"
  - Dazu musste jede Funktion vor Ihrer Benutzung deklariert werden (und das haben wir in der .h Datei gemacht)
  - Und jede Funktion musst in einer der .cpp Dateien (oder in einer Bibliothek) implementiert sein
- Das geht genauso mit (globalen) Variablen
  - Die Deklaration geht dort mit dem Schlüsselwort extern
  - Zum Beispiel, in der Datei Animation.h

    ```
    extern int posX;  // Used but not defined here.
    ```
  - Und in einer der .cpp Dateien muss sie dann wirklich stehen

    ```
    int posX = 1;  // Actual definition.
    ```

Hinweise zum 3. Übungsblatt 1/2

- Aufgabe: einfache Animation auf dem Terminal
  - Dazu muss man den Cursor absolut positionieren können, und Sachen von der Art
  - Das geht mit sog. ANSI escape codes, zum Beispiel:

```
printf("\x1b[2J");  // Clears the screen.
printf("\x1b[1;1H");  // Put cursor in upper left corner.
printf("\x1b[?25l");  // Hide the cursor.
printf("\x1b[?25h");  // Show the cursor.
printf("\x1b[1m");  // Print following stuff in bold.
printf("\x1b[31m");  // Print following stuff in red.
printf("\x1b[0m");  // Print normally again.
```

  - Ausführliche Erklärung und Dokumentation z.B. unter

    http://en.wikipedia.org/wiki/ANSI_escape_code

Hinweise zum 3. Übungsblatt 2/2

UNI FREIBURG

- Basisversion der Aufgabe (für alle)
  - Wie im Beispiel aus der Vorlesung, aber Ball an allen vier Rändern "reflektieren" ... und ohne "Spur"
- Zusatzaufgabe (ohne Punkte, just for fun)
  - Mit Schwerkraft, nicht-lineare Flugbahn, mehrere Bälle ... was immer Ihnen einfällt

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 AnimationTest.o
  - Das % auf der rechten Seite wird dann entsprechend ersetzt
    - in dem Beispiel durch Animation
- Automatische Variablen (beim Match einer Pattern-Regel)

  $@ ist das konkrete target

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

  $< ist die erste dependency nach dem target

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

Makefile 2/2

- Variablen

  MAIN_BINARIES = AnimationMain.cpp

  Wie in einem C++ Programm, nur ohne Variablentyp

- Funktionen

  $(wildcard *.cpp)

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

  $(basename Animation.o AnimationMain.cpp AnimationTest.o)

  Die Dateinamen ohne Ihre Suffixe: Animation, AnimationMain, AnimationTest

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

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

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

  Siehe Makefile Manual ... Links dazu auf der letzten Folie!

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

- 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 steht derselbe Text an zwei Stellen, und es ist nur eine Frage der Zeit, bis die beiden Texte nicht mehr übereinstimmen
  - Und die .h Datei ist sozusagen die Übersicht über alle Funktionen, von daher passt da auch die Doku gut hin

-lgtest_main

- Die main Funktion in der XyzTest.cpp Datei
  - Brauchen wir, damit es überhaupt linkt und damit ./XyzTest wie gewünscht alle Tests ausführt
  - Diese main Funktion ist aber immer **genau** dieselbe:

```
int main(int argc, char** argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

  - Deswegen gibt es eine extra Bibliothek, die nichts anderes enthält wie genau diese main Funktion
  - Die kann man einfach mit –lgtest_main dazulinken
  - Dann können (und müssen) wir die main Funktion in der XyzTest.cpp Datei weglassen

Literatur / Links

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