Programmieren in C++

SS 2016

Vorlesung 3, Dienstag 3. Mai 2016

(Grundlegende Konstrukte, noch mehr zu Make)

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

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Erfahrungen mit dem Ü2 — .h Dateien und Makefile
  - Es gab 193 Abgaben zum Ü1 — im SS 2014 waren es 145
- Inhalt
  - Grundlegende Konstrukte — while, for, if, else, switch, ...
  - Hinweise zum Ü3 — Beispielprogramm, ncurses
  - Globale Variablen — Deklaration mit "extern"
  - Mehr zu Make — Patterns, Variablen, Funktionen
  - **Übungsblatt 3:** Teilfunktionalität von Tetris implementieren und das Makefile generisch machen

Erfahrungen mit dem Ü2 1/3

- Zusammenfassung / Auszüge
  - Übungsblatt relativ einfach und schnell gemacht
  - Einige Diskussion wegen Punktabzügen

    z.B. 32. Mai statt 1. Juni ... dafür jetzt kein Punktabzug
  - Kommentare im Quelltext auf Deutsch oder Englisch

    Grundsätzlich auf Englisch bitte ... das gilt auch für die Benennung von Variablen, Funktionen, etc.
  - "Programmieren mit der Vorlesungsaufzeichnung nebenher geht super"

Erfahrungen mit dem Ü2 2/3

- Warum Graviton noch nicht entdeckt
  - Graviton ist sehr schüchtern aufgrund seines Gewichts
  - Der Vatikan erlaubt es nicht
  - Gravi-was?
  - Musste den Scheiß erstmal googeln, bleibt mir bitte fern mit Physik!
  - Wer früher Lego gespielt hat, weiß wie schwer es ist, kleine Teilchen zu finden

Erfahrungen mit dem Ü2 3/3

- Warum Graviton wirklich noch nicht entdeckt
  - Grundannahme 1: das Universum besteht aus Feldern

    elektro-magnetische, Gravitation, starke und schwache Ww
  - Grundannahme 2: die Felder sind nicht beliebig unterteilbar, sondern es gibt Grundeinheiten (Quanten)

    z.B. Quantum des EM Feldes = Photon, Quantum des Gravitationfeldes = Graviton
  - Problem 1: auf der Ebene eines Quants ist die Gravitation extrem schwach im Vergleich zu den anderen Kräften
  - Problem 2: mathematisch komplizierter als für EM etc.

Grundlegende Konstrukte 1/10

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)

Grundlegende Konstrukte 2/10

UNI FREIBURG

- Variablen
  - **Benennung** in camelCase mit erstem Buchstaben klein

    In der Regel Wörter in Variablennamen **nicht** abkürzen

    Ausnahme: Variable wird in einem lokalen Kontext häufig benutzt (z.B. Laufvariable einer Schleife), dann ist auch ein kurzer Name ok oder sogar besser (z.B. i oder j oder c)
  - **Deklaration** vor der Benutzung ist Pflicht
  - **Initialisierung** bei der Deklaration ist optional, sonst beliebiger unbekannter Wert:

```
int x;  // Has an unknown value after this.

int y = 10; // Value 10 after this.
```

Grundlegende Konstrukte 3/10

UNI FREIBURG

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

    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 >>

    Die brauchen wir jetzt noch nicht und werden in einer späteren Vorlesung erklärt

Grundlegende Konstrukte 4/10

- Zuweisungen

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

Abkürzungen für häufige Typen von Zuweisungen

```
i++;        // Identical to i = i + 1.
i--;        // Identical to i = i – 1.

x +=3;      //  Identical to x = x + 3.
x -=3;      //  Identical to x = x - 3.
x *= 3;     // Identical to x = x * 3.
x /= 3;     // Identical to x = x / 3.
x %= 3;     // Identical to x = x % 3.
```

Grundlegende Konstrukte 5/10

UNI FREIBURG

- Konditionale Ausführung von Code

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

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

Grundlegende Konstrukte 6/10

UNI FREIBURG

- Konditionale Ausführung mit **switch**
  - Bei vielen einfachen Gleichheitsbedingungen, z.B.

```
// Action depending on key pressed.
int key = getch();
switch (key) {
   case 65: rowIndex--;   break;   // Arrow up.
   case 66: rowIndex++;   break;   // Arrow down.
   case 67: colIndex++;   break;   // Arrow right.
   case 68: colIndex--;   break;   // Arrow left.
   default: ...;  // If none of the "case"s match
}
```

Den **default** Teil kann man auch einfach weglassen

Achtung: ohne das **break** wird auch der anschließende Code ausgeführt, das ist in der Regel nicht das, was man möchte

Grundlegende Konstrukte 7/10

- Der 3-Wege Operator
  - Sehr nützlich, um bei einfachen Konditionalen ein if - else über mehrere Zeilen zu vermeiden:

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

  - Die 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

Grundlegende Konstrukte 8/10

- Schleifen: while und for

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

– Äquivalent dazu, aber kürzer und besser lesbar:

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

Grundlegende Konstrukte 9/10

UNI FREIBURG

- 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--;
}
```

Grundlegende Konstrukte 10/10

- Schleifen: break und continue
  - Schleife vorzeitig abbrechen: break
  - Eine Iteration überspringen: continue

```
// Read key, print if letter, stop when '!' pressed.
while (true) {
   int key = getch();
   if (key == '!') { break; }
   if (key < 'a' || key > 'z') { continue; }
   printf("You pressed the letter: %c\n", key);
}
```

Bei geschachtelten Schleifen: Abbruch aus der Schleife, in der das break steht, nicht auch aus den enthaltenden Schleifen

Hinweise zum Ü3 1/6

- Tetris
  - Aufgabe der nächsten Übungsblätter ist es, eine einfache Version von Tetris zu implementieren
  - Erst mal selber spielen, z.B. auf www.tetris24.com
  - Für das Ü3, erst mal folgende Teilfunktionalität:
    - Eine Sorte Tetromino ihrer Wahl (außer Quadrat)
    - Jedes Tetromino fällt ganz nach unten, egal welche Tetrominos von vorher da schon rumliegen
    - Optional: Drehen (sonst auf dem Ü4), weniger Flackern, besseres aspect ratio, ...

Hinweise zum Ü3 2/6

- Erweiterte Ausgabe auf dem Terminal
  - Normalerweise drucken printf etc. einfach in die aktuelle Zeile, und wenn man unten ankommt, wird ge"scrolled"
  - Man kann den Cursor aber auch ABSOLUT auf dem Bildschirm positionieren, farbig oder **fett** drucken, etc.
  - Das geht mit sogenannten ANSI escape codes, z.B.

    printf("\x1b[1mHello!\x1b[0m;"); // Print in **bold**.
  - Ausführliche Erklärung und Dokumentation z.B. unter

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

Hinweise zum Ü3 3/6

- Erweiterte Ausgabe auf dem Terminal
  - Ein paar nützliche Beispiele für das Ü3:

```
printf("\x1b[2J");          // Clears the screen.
printf("\x1b[5;7H");        // Put cursor at row 5, column 7.
printf("\x1b[1m");          // Print following stuff in bold.
printf("\x1b[31m");         // Print following stuff in red.
printf("\x1b[0m");          // Print normally again.
```

Hinweise zum Ü3 4/6

- Einlesen der aktuell gedrückten Taste
  - Das geht im Prinzip mit der Funktion getchar()

    Allerdings **wartet** die auf einen Tastendruck, und druckt alles auf den Bildschirm samt Cursor

    Das lässt sich zwar anders einstellen, ist aber in C/C++ sehr aufwändig und wenig erquicklich
  - Alternative: die Funktion getch() aus der Bib. ncurses

    Die wird auf der nächsten Folie erklärt

Hinweise zum Ü3 5/6

- Ncurses
  - Eine Bibliothek für erweiterte Ausgabe über die Konsole und Eingabe über die Tastatur / Maus

```
#include <ncurses.h>

initscr();              // Initializiation.
cbreak();               // Don't wait for RETURN.
noecho();               // Don't echo key pressed on screen.
curs_set(false);        // Don't show the cursor.
nodelay(stdscr, true);  // Don't wait until key pressed.
keypad(stdscr, true);   // For KEY_LEFT, KEY_UP, etc.
```

  - Beim Linken brauch man dann noch: -lncurses
  - Falls nicht installiert: sudo apt-get install libncurses-dev

Hinweise zum Ü3 6/6

UNI FREIBURG

- Ncurses und Unit Tests
  - Achten Sie darauf, dass der Code für das **Bewegen** und für das **Malen** in verschiedenen Funktionen steht
  - Für die Funktionen, die nur malen, brauchen Sie keinen Unit Test zu schreiben (es wäre auch kompliziert)
  - Für die Funktion zur Initialisierung von ncurses brauchen Sie ebenfalls keinen Unit Test zu schreiben

Globale Variablen 1/3

- Was + warum
  - Variablen, die außerhalb einer Funktion definiert sind, nennt man **globale Variablen**

```
int rowIndex;

void someFunction() {
  // rowIndex can be used here.
  ...
}
```

  - Globale Variablen kann man im Prinzip überall im Code benutzen, auch in anderen Dateien

    Selbe Prinzip wie bei Funktionen, siehe nächste Folien

Globale Variablen 2/3

- Wiederholung: Linken von Funktionen
  - Jede Funktion muss:
    - vor der Benutzung deklariert werden

      Üblicherweise in einer .h Datei, die dann in jeder .cpp Datei, in der die Funktion benötigt wird, includiert wird
    - in genau einer Datei implementiert sein

      Die dazugehörige .o Datei bzw. Bibliothek muss dann beim Linken dabei sein

Globale Variablen 3/3

- Das gilt genauso für globalen Variablen
  - Die Deklaration geht dort mit dem Schlüsselwort extern

```
extern int rowIndex;
int main(int argc, char** argv) {
   ...
}
```

Muss dann in einer anderen Datei implementiert sein:

```
int rowIndex;   // Can be initialized here or in the code.
```

Wenn eine globale Variable mit extern deklariert wurde und dann beim Linken nicht gefunden wird, gibt es auch einfach

"undefined reference to ..."

Noch mehr zu Make 1/5

- Pattern-Regeln

```
%.o: %.cpp
        <command1>
        <command2>
        ...
```

  - Wird angewendet für jedes target, das zu %.o passt
    - zum Beispiel bei make Graviton.o
  - Das % auf der rechten Seite wird dann entsprechend ersetzt
    - in dem Beispiel durch Graviton

Noch mehr zu Make 2/5

- Automatische Variablen (beim Match einer Pattern-Regel)

  $@ ist das konkrete target

  $* ist dieses target ohne Suffix

  $< ist die erste dependency nach dem target

  $^ sind alle dependencies nach dem target

  Beispiel:

```
%.o: %.cpp
        g++ -o $@ -c $^
```

  Dann wird bei make Graviton.o ausgeführt:

```
g++ -o Graviton.o -c Graviton.cpp
```

Noch mehr zu Make 3/5

- Variablen

  CXX = g++

  Wie in einem C++ Programm, nur ohne Variablentyp, kann man dann an andere Stelle im Makefile so verwenden:

```
%.o: %.cpp
        $(CXX) -o $@ $^
```

Noch mehr zu Make 4/5

- Funktionen

  $(wildcard *.cpp)

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

  $(basename Graviton.o GravitonMain.cpp GravitonTest.o)

  Die Dateinamen ohne Ihre Suffixe: Graviton, GravitonMain, GravitonTest

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

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

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

  Für mehr Funktionen, siehe Referenzen (Link auf der letzten Folie)

Noch mehr zu Make 5/5

- Die main Funktion in der ...Test.cpp Datei
  - Brauchen wir, damit es überhaupt linkt und damit das Programm dann 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 die main Funktion in ...Test.cpp weglassen !

Literatur / Links

- Grundlegende Konstrukte in C++
  - http://www.cplusplus.com/doc/tutorial/variables/
  - http://www.cplusplus.com/doc/tutorial/operators/
  - http://www.cplusplus.com/doc/tutorial/control
- Makefile Patterns, Variablen, Funktionen
  - http://www.gnu.org/software/make/manual/make.html

    Kapitel: "Pattern Rules", "Automatic Variables", "Functions"
- Ncurses
  - man ncurses vorher evtl.: sudo apt-get install ncurses-doc