Programmieren in C++

SS 2018

Vorlesung 3, Dienstag 8. Mai 2018

(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

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Ü2 — .h Dateien und Makefile
  - Thema Punktabzüge — ein paar Richtlinien
- 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:** ein Hüpfball mit Schwerkraft, gemalt in einfacher ASCII Grafik (benutzen wir auch für Ü4 und Ü5)

Erfahrungen mit dem Ü2 1/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Für die meisten einfach und schnell gemacht

    Das war Absicht, weil es in den ersten beiden Wochen erfahrungemäß noch viele Probleme mit dem Drumherum gibt; ab dem Ü3 jetzt etwas anspruchsvoller
  - Einige haben es einfach der Vorlesung nachgemacht

    Ok für den Anfang, aber irgendwann sollte man natürlich auch verstehen, was man da macht
  - Statt Wechsel zwischen Tabs, besser Bildschirm splitten?

    Habe ich schon mal versucht, hat sich nicht bewährt, ich habe aber vergessen warum, also heute nochmal

Erfahrungen mit dem Ü2 2/3

UNI FREIBURG

- VR-Brille … Ihre Kommentare
  - "Eine unheilige Kombination aus beiden Bildern, da die Sehfelder der beiden Augen sich teilweise überlappen"
  - "Weihnachtshasen und Ostermann"
  - "Weihnachtsmann mit Hasenohren"
  - "Ich habe lange gedacht, das Christkind wäre ein Hase"
  - "Osterhase wegen Sehschwäche auf dem linken Auge"
  - "Ich sehe eine kaputte VR-Brille"
  - "Bei voneinander stark abweichenden Retinabildern als separiert angebotene Sehreize kommt es zu keiner überlappenden Fusion, sondern eher zu Wechseleindrücken"

Erfahrungen mit dem Ü2 3/3

UNI FREIBURG

- Binokulare Rivalität (zwei inkompatible Bilder)
  - Grober Aufbau unseres visuellen Systems:

    Primärer visueller Kortex (Mustererkennung) V1
    Sekundärer visueller Kortex (Objekterkennung) V2-3

    Parietaler Pfad (Bewegung und Position) PP
    Temporaler Pfad (Farben, Muster, Formen) TP

  - Verhältnisse bei binokularer Rivalität:

    In V1-3 werden noch beide Bilder codiert

    In PP und TP im Wesentlichen nur noch das Bild, das bewusst wahrgenommen wird (auch schon bei Affen)

    Ohne Aufmerksamkeit, auch in PP und TP beide Bilder (messbar z.B. durch "frequency tagging" der Bilder)

Punktabzug

UNI FREIBURG

- Richtlinien
  - Mit Punktabzug müssen Sie immer dann rechnen, wenn:

    Etwas explizit auf dem Übungsblatt verlangt war

    Oder es auf Seite 3 des Ü1 steht (die 10 Gebote)

    Oder es im Unterforum "Ankündigungen" verlangt wurde

    Oder ihr Tutor es explizit angekündigt hat
  - Sie müssen nicht alle Posts auf dem Forum verfolgen

    Ist allerdings nicht viel Arbeit und kann sich lohnen

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

UNI FREIBURG

- Zuweisungen
  - Normale Zuweisung

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

  - Abkürzungen für häufige Muster 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.

```
int key = getch();
switch (key) {
   case KEY_UP:        y--;     break;
   case KEY_DOWN:      y++;     break;
   case KEY_LEFT:      x--;     break;
   case KEY_RIGHT:     x++;     break;
   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

UNI FREIBURG

- 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

UNI FREIBURG

- 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

UNI FREIBURG

- 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 umschließenden Schleifen

Hinweise zum Ü3 1/5

UNI FREIBURG

- Hüpfball
  - Aufgabe des Ü3 ist es, einen Ball zu malen, der sich unter dem Einfluss von Schwerkraft bewegt
  - Dazu brauch man drei Paare von Variablen:

    Position: die aktuellen (x, y) Koordinaten des Balles

    Geschwindigkeit: die Änderung der Position pro Zeiteinheit (in x und in y Richtung)

    Beschleunigung: die Änderung der Geschwindigkeit pro Zeiteinheit (in x und in y Richtung)
  - Zur Unterstützung für das Übungsblatt malen wir in der Vorlesung zusammen eine Kreisscheibe

Hinweise zum Ü3 2/5

UNI FREIBURG

- Ncurses … Initialisierung
  - 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 presses 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 3/5

[Suite Code]
Kreis mit Radius r :
$dx^2 + dy^2 \leq r^2 \rightarrow \frac{dx^2}{r^2} + \frac{dy^2}{r^2} \leq 1$
Ellipse mit Radius rx, ry
$\frac{dx^2}{rx^2} + \frac{dy^2}{ry^2} \leq 1$
$\Leftrightarrow dx^2 ry^2 + dy^2 rx^2 \leq rx^2 ry^2$

- Ncurses … Funktionen
  - Schreiben an eine bestimmte Position

    mvprintw(y, x, "…");

    Achtung: die Koordinaten sind aus "Terminalsicht" = erst die Zeile (y-Koordinate), dann die Spalte (x-Koordinate)
  - Invers malen (Vorder- und Hintergrundfarbe vertauschen)

    attron(A_REVERSE); // Switch on reverse mode
    attroff(A_REVERSE); // Switch off reverse mode.
  - Code der letzten gedrückten Taste

    int key = getch();
    if (key == KEY_UP) { … } // Arrow up key pressed.

Man pages:
sudo apt-get install ncurses-doc
und dann z.B. man getch

Hinweise zum Ü3 4/5

UNI FREIBURG

- Ncurses … weitere Funktionen und Tipps
  - Damit die Änderungen auch wirklich erscheinen:

    ```
    refresh()
    ```

  - Zur Dosierung der Zeit zwischen den einzelnen "Frames"

    ```
    #include <unistd.h>

    usleep(100 * 1000);  // Do nothing for 100 milliseconds.
    ```

  - Um die Position nur um einen Bruchteil zu ändern

    ```
    float x = 1.7;
    float y = 1.2;
    x += 0.5;
    mvprintw(y, x, "Doof");  // Will round down (position 1, 2).
    ```

Hinweise zum Ü3 5/5

UNI FREIBURG

- Ncurses und Unit Tests
  - Achten Sie darauf, dass der Code für das **Bewegen** und für das **Malen** in verschiedenen Funktionen steht

    Für das Ü3 sind die Funktionen bereits so vorgegeben, dass das der Fall ist
  - Für die Funktionen, die nur malen, brauchen Sie keinen Unit Test zu schreiben (es wäre auch kompliziert)
  - Der Unit Test für die Initialisierung braucht auch nicht zu überprüfen, ob ncurses richtig initialisiert wurde (dito)

Globale Variablen 1/3

UNI FREIBURG

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

```
int x;

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

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

    Es ist dasselbe Prinzip wie bei unseren Funktionen bisher, siehe nächste Folien

Globale Variablen 2/3

UNI FREIBURG

- 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, inkludiert wird
  - Jede Funktion muss in genau einer Datei implementiert sein

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

Globale Variablen 3/3

UNI FREIBURG

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

```
extern int x;
extern int y;

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

  - Muss dann in einer anderen Datei implementiert sein:

```
int x;
int y;
```

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

    "undefined reference to ..."

Noch mehr zu Make 1/5

UNI FREIBURG

- Pattern-Regeln

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

  - Wird angewendet für jedes target, das zu %.o passt, z.B.

    make Ball.o

  - Das % auf der rechten Seite wird dann entsprechend ersetzt

    Im Beispiel durch "Ball"

Noch mehr zu Make 2/5

UNI FREIBURG

- 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 Ball.o ausgeführt:

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

Noch mehr zu Make 3/5

UNI FREIBURG

- 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

UNI FREIBURG

- Funktionen
  - Liste aller Dateien im aktuellen Ordner (durch Leerzeichen getrennt), die zu einem bestimmen Muster passen

    $(wildcard *.cpp)
  - Entfernen aller Suffixe einer Liste von Strings (Dateinamen)

    $(basename Ball.o BallMain.cpp BallTest.o)
  - Weitere Funktionen (Erklärung siehe Referenzen am Ende)

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

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

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

Noch mehr zu Make 5/5

UNI FREIBURG

- 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

UNI FREIBURG

- 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 pages
  - sudo apt-get install ncurses-doc
  - man ncurses oder man mvprintw oder man getch oder …