Programmieren in C

SS 2021

Vorlesung 3, Dienstag 4. Mai 2021

(Grundlegende Konstrukte, noch mehr zu Make)

Prof. Dr. Peter Thiemann
Lehrstuhl für Programmiersprachen
Institut für Informatik
Universität Freiburg
Folienvorlagen von Prof. Dr. Hannah Bast

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, ...
  - Globale Variablen — Deklaration mit "extern"
  - Mehr zu Feldern — Init., 2D, Strings!
  - Präprozessor — weitere Instruktionen

Erfahrungen mit dem Ü2

UNI FREIBURG

- 275 Abgaben, 234 Erfahrungen, 224 mit lesbarer Zeit
- Zeitstatistik (in Stunden)

Min	Q1	Med	Q3	Max	Avg	MD	Var
0.33	3.0	4.0	6.0	36.0	4.94	2.14	12.04

- Regexp-Stats

#	Schlüsselworte
61	schwer|schwierig|anspruch|aufwendig|fordernd|hart|viel zeit|lange gedauert
1	nicht schwer|nicht schwierig|nicht anspruch|nicht aufwendig|unaufwendig|nicht fordernd|nicht hart|nicht viel zeit|nicht lange gedauert
121	cool|nett|spaß|gut|schön|toll|super
1	nicht cool|uncool|nicht nett|keinen spaß|nicht gut|nicht schön|unschön|nicht toll|nicht super
12	unklar|verwirrend

Punktabzug

UNI FREIBURG

- Richtlinien (vgl Ü3)
  - Ab diesem Übungsblatt nehmen wir an, dass Sie Makefiles zum Kompilieren und Linken verwenden können.
  - Es gibt 0 Punkte falls make compile und make test nicht zumindest die von Ihnen bearbeiteten Aufgaben erfolgreich kompilieren und linken.
  - Dadurch können die Tutoren ihre Zeit sinnvoller nutzen und Ihnen ausführlicheres Feedback geben.
  - Stellen Sie also sicher, dass auf dem Buildserver alle relevanten Programme erfolgreich gebaut werden.
  - Bei Problemen melden Sie sich bitte rechtzeitig im Forum oder in den Tutoraten.

Tutorate

- Räume über Ilias erreichbar

  Do 08:00-10:00 (Jannik Söhnlein)

  Do 08:00-10:00 (Marco Kaiser)

  Do 10:00-12:00 (Florian Pollitt)

  Do 10:00-12:00 (Lena Funk)

  Do 10:00-12:00 (Lukas Kleinert)

  Do 12:00-14:00 (Gloria Dobreva)

  Do 12:00-14:00 (Simon Blauth)

  Do 14:00-16:00 (Christian Handschuh)

  Do 14:00-16:00 (Lisa Hofert)

  Do 16:00-18:00 (Darius Schönlein)

  Fr 12:00-14:00 (Pascal Hunkler)

  Mi 14:00-16:00 (Jascha Hettich)

Grundlegende Konstrukte 1/10

UNI FREIBURG

- Die elementaren Datentypen
  - char = einzelnes ASCII Zeichen, ganze Zahl, mind 8 Bit

    'A' == 65, '0' == 48, '0' != 0
  - int = ganze Zahl, mindestens $(-2^{15}+1..2^{15}-1)$
  - long = ganze Zahl, mindestens $(-2^{31}+1..2^{31}-1)$
  - float = Gleitkommazahl, 4 Bytes oder mehr

    3.14f // otherwise type double
  - double = Gleitkommazahl, 8 Bytes oder mehr
  - bool = true (wahr, 1) oder false (falsch, 0)

    --> #include "stddef.h" oder #include "stdbool.h"

Grundlegende Konstrukte 2/10

UNI FREIBURG

- Der Typ size_t
  - Definiert in stddef.h
  - Für Größen und Indizes von Arrays
- Das Präfix unsigned
  - Die ganzzahligen Typen können auch ohne Vorzeichen verwendet werden
  - Ein Extra-Bit zur Verfügung!
    - char (0 .. 2^7-1)
    - int (0, 2^{15}-1)
    - long (0 .. 2^{31}-1)

Grundlegende Konstrukte 3/10

UNI FREIBURG

- Promotion
  - Typen kleiner als int werden bei Verwendung zu int
  - Der Typ float wird bei Verwendung zu double
- Bedeutung für printf
  - Bool, char, int können mit %d gedruckt werden
  - Float und double können (zB) mit %g gedruckt werden

Grundlegende Konstrukte 4/10

UNI FREIBURG

- Variablen
  - **Benennung** in snake_case 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 5/10

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 >>

    → später

Grundlegende Konstrukte 6/10

UNI FREIBURG

- Zuweisungen
  - Normale Zuweisung

```
i = j + 2;  // Left side must be variable or array ref.
```

  - 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 7/10

UNI FREIBURG

- Konditionale Ausführung

```
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 gar kein Code ausgeführt

Grundlegende Konstrukte 8/10

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 folgende case ausgeführt, das ist in der Regel nicht erwünscht!

Grundlegende Konstrukte 9/10

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 10/10

UNI FREIBURG

- Der Komma-Operator
  - Wertet mehrere Ausdrücke sequentiell aus:

    ```
    answer = (x = 17, 2 * (x+4));  // Answer to everything.
    ```

  - Gleichwertig zu

    ```
    x = 17;
    answer = 2 * (x+4);
    ```

  - Der Vollständigkeit halber
    - Schlecht lesbar
    - Eher nicht verwenden

Grundlegende Konstrukte 11/10

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:

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

Grundlegende Konstrukte 12/10

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 13/10

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

Globale Variablen 1/3

UNI FREIBURG

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

```
int x;

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

  - Globale Variablen dürfen überall im Code benutzt werden, auch in anderen Dateien

Dasselbe Prinzip wie bei Funktionen,
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 .c 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 mit dem Schlüsselwort extern in .h Datei

```
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 beim Linken nicht gefunden wird, kommt auch

"undefined reference to ..."

Felder 1/9

UNI FREIBURG

- Felder können auch mehrdimensional sein

  ```
  float matrix[3][4];
  ```

- Ein Array of arrays:
  - matrix ist ein array mit drei Elementen
  - matrix[i] ist ein array mit vier Element (i=0,1,2)
- Initialisierung von Arrays entsprechend Index-Reihenfolge

  ```
  int im[2][3] = {{ 1, 0, -1}, { 2, 2, 0}};
  ```

- Mit [] wird die Größe aus der Initialisierung bestimmt

  ```
  int a[] = { 0, 1, -1 };  // 3 elements
  ```

- Geht nur beim äußersten Array

  ```
  int m[2][] = { {0, 1}, {-1,0} };  // error
  ```

  ```
  int m[][2] = {{ 0, 1}, {-1, 0}};  // accepted
  ```

Felder 2/9

UNI FREIBURG

- Arrays als Funktionsargumente
  - Von einem Array wird immer die (Start-) Adresse übergeben
  - Jede Änderung am Array innerhalb der Funktion ist für den Aufrufer sichtbar:

```
void f(int m[]) {
   m[0] = 42;
}
void g(void) {
   int a[1] = { 666 };
   f(a);
   printf("%d", a[0]);  // prints 42 (what else?)
}
```

Zeichenketten / Strings 1/9

UNI FREIBURG

- Eine **Zeichenkette** ist ein Feld von Elementen vom Typ **char**

  char a[5] = {'D', 'o', 'o', 'f', 0};

  char a[] = "Clever"; // Appends the 0 automatically

- In C ist der Typ für Strings **char*** (Erklärung nächste Woche)
- Damit geht auch

  const char* s = "very clever"; // s is the address of 'c'.

  Ohne das const gibt es eine Compiler-Warnung

- Strings in C/C++ sind **null-terminiert**, d.h. für "foo" wird Platz für **vier** Zeichen bereitgestellt, am Ende steht **0**

  Achtung Fehlerquelle: vergessene 0 am Stringende!!

Strings 2/9

UNI FREIBURG

- Felder von Zeichenketten
  - Der Typ der main() Function:

    int main(int argc, **char* argv[]**)
  - Mit anderen Worten: **argv** ist ein Feld von Strings
  - **argc** gibt an, wie viele Elemente das Feld hat

Strings 3/9

UNI FREIBURG

- Nützliche Funktionen auf Strings
- #include „string.h“ (hier vereinfachter Auszug)
  - size_t strlen(char str[]);

    assert (strlen(„foo“) == 3); // string length
  - int strcmp(char str1[], char str2[]);

    assert (strcmp(“Anton“, “Berta“) < 0); // lex. Comparison

    assert (strcmp (“Oskar“, “Oskar“) == 0);
  - int snprintf(char s[], size_t n, const char format[], …);

    wie printf(), aber Ausgabe in einen String

    char output[100];

    snprintf(output, 100, “the magic number is %d“, 42);
  - Achtung Fehlerquelle, wenn die Größe falsch ist!

Strings 4/9

UNI FREIBURG

- Stdlib.h
  - double atof(char str[])

    "ASCII to float": konvertiert einen String in Gleitkommazahl vom Typ double
  - int atoi(char str[])

    "ASCII to int": konvertiert nach int
  - int atol(char str[])

    "ASCII to long": konvertiert nach long

Präprozessor 101

UNI FREIBURG

- Der Präprozessor wird noch vor dem eigentlichen Compiler ausgeführt. Selbst eine kleine Programmiersprache … Dies sind die wichtigsten Anweisungen
  - #include - kennen wir schon
  - #define <name> <optionaler text>
  - #ifdef <name> - Konditional
  - #ifndef <name> - Konditional, kennen wir schon
  - #if <expr> - Konditional
  - #else
  - #endif
  - #error

Präprozessor 102

UNI FREIBURG

- #define definiert ein Makro

```
#define BUFSIZE 1024
/* every future occurrence of BUFSIZE is replaced by 1024 */
int buffer[BUFSIZE];
for(size_t i = 0; i < BUFSIZE; i++) {
   … buffer[i] …
}
```

- C verwendet das für Konstanten
- Meist in .h Dateien

Präprozessor 103

UNI FREIBURG

- Verwendung von #if und #error

```
#if __STDC_VERSION__ < 201112
#error “This program requires a C11 compliant compiler.“
#endif
```

Literatur / Links

UNI FREIBURG

- Grundlegende Konstrukte in C
  - https://en.cppreference.com/w/c/language/expressions
  - https://en.cppreference.com/w/c/language/statements
- Strings
  - https://en.cppreference.com/w/c/string/byte
- Präprozessor
  - https://en.cppreference.com/w/c/preprocessor