Programmieren in C++

SS 2011

Vorlesung 4, Mittwoch 25. Mai 2011

(Felder, Strings, Zeiger, const, gdb)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Keine Fragestunde am Freitag, nur am Dienstag
  - Kommentare von Ihrem Tutor
  - Erfahrungen mit dem 3. Übungsblatt
- Themen heute
  - Felder, Strings, Zeiger ... und das das alles dasselbe ist
  - Die Operatoren [] und *
  - const bei Deklarationen
  - Debugging mit gdb
  - Diesmal gibt es zwei Übungsaufgaben zur Auswahl, 1a und 1b
    - vom Codeumfang her vergleichbar
    - aber 1b ist kniffliger als 1a

Kommentare von Ihrem Tutor

UNI FREIBURG

- Läuft auch über das SVN
  - Ihre Tutoren geben Ihnen Feedback zu Ihren Abgaben
  - Sie bekommen es mit svn update
    - in Ihrer Arbeitskopie von Ihrem SVN Ordner
  - Dann bekommen Sie pro Übungsblatt
    - eine Datei feedback-tutor.txt
    - enthält allgemeine Kommentare zu Ihrer Abgabe
    - mit evtl. Verweisen auf Kommentare in Ihrem Code

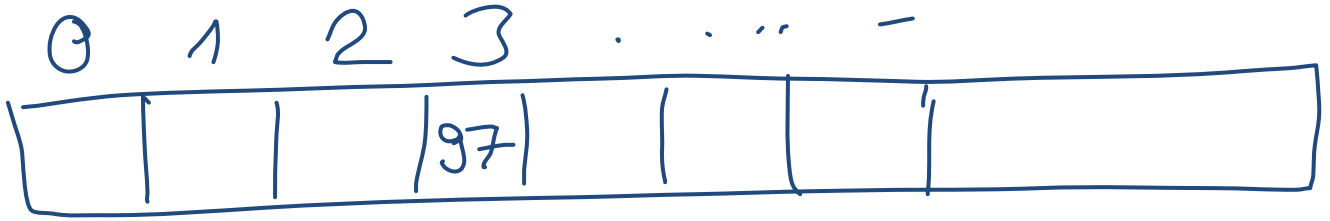
Erfahrungen mit dem Ü3 (Mandelbrot)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 25.5 15:00
  - Übungsblatt schwieriger / aufwändiger als bisher
  - Aufgabe sah bedrohlich aus ... dann aber doch ok
  - Einigen wenigen war es viel zu schwer
  - Mandelbrotmännchen ist cool
  - Infos in der Vorlesung nicht ganz ausreichend
  - Linter immer noch nervig ... aber nicht mehr so
  - Vim oder Emacs?
  - Warum durften wir nicht <complex> benutzen?
  - Generisches Makefile ist cool ... aber auch schwierig
  - Wo bleibt die Objektorientierung?

Speicher

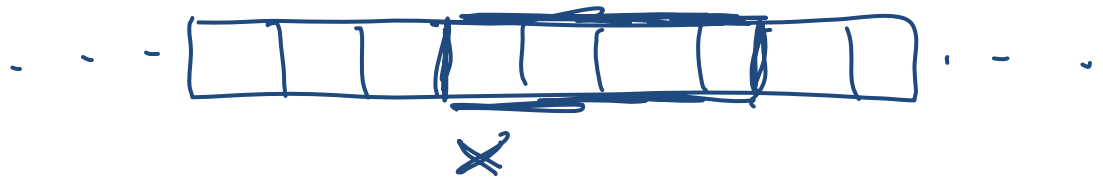
- Der Hauptspeicher von einem Rechner
  - Ist (konzeptuell) einfach eine Menge von Speicherzellen
  - Jede Speicherzelle fasst 1 Byte = 8 Bits
    - also eine Zahl zwischen 0 und 255 (einschließlich)
  - Die Speicherzellen sind fortlaufend durchnumeriert
  - Die Nummer einer Speicherzelle nennen wir ihre Adresse



Einfache Variablen

- Zum Beispiel

```
int x;
x = 12;
```



  - Haben wir jetzt die ganze Zeit schon benutzt
  - Sie stehen an einer festen Stelle im Speicher
    - je nach Typ umfasst die Variable eine oder mehrere Speicherzellen; ein int hat zum Beispiel 4 bytes
    - die Anzahl Bytes bekommt man mit sizeof

      ```
      printf("%d\n", sizeof(x));
      ```
  - Der Variablenname steht für den **Wert** dieser Speicherzellen (nicht für die Adresse)
    - die Adresse bekommt man mit &x → **spätere VL**

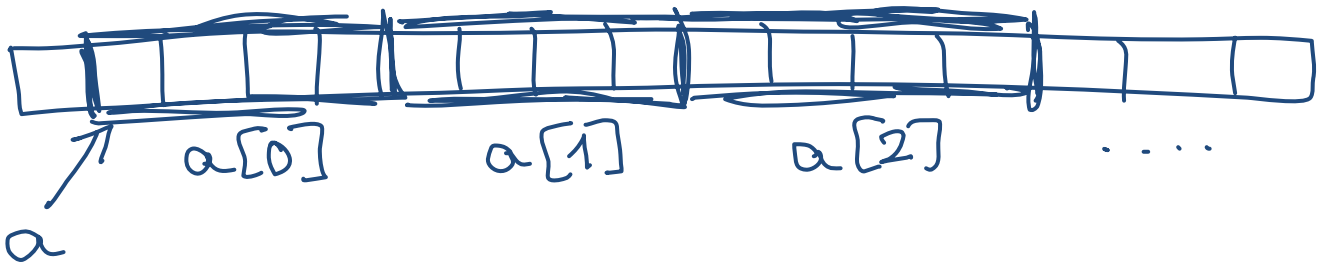
Felder, engl. Arrays 1/2

UNI FREIBURG

- Folge von Variablen vom selben Typ
  - auf die man alle mit demselben Namen und einem sogenannten Index zugreifen kann
  - zum Beispiel die Zahlenlisten vom 4. Übungsblatt

    ```
    int a[5];
    ```
  - Zugriff auf ein Element des Feldes mit dem [] Operator, wobei das erste Element Index 0 hat, das zweite 1, usw.

    ```
    printf("%d\n", a[2]);  // Print the *third* element.
    ```
  - Die Elemente stehen **hintereinander** im Speicher, und der Variablenname steht für die **Adresse** des ersten Elementes



Felder, engl. Arrays 2/2

UNI FREIBURG

- Initialisierung
  - Wie bei einfachen Variablen, kann man den Feldelementen schon bei der Deklaration Werte zuweisen

```
int a[3] = {1, 2, 3};
int a[3] = {1, 2, 3, 4};  // Too many values -> compiler error.
int a[3] = {1, 2};  // No compiler error for too few values.
```

Zeiger 1/2

UNI FREIBURG

- Zeiger sind Variablen, deren Wert eine **Adresse** ist
  - Bei der Deklaration gibt man an, wie der Inhalt des Speichers an dieser Adresse zu interpretieren ist

    ```
    int* p;  // Pointer to an integer (usually 4 bytes).
    ```

  - Zugriff auf diesen Wert mit dem * **vor** der Variablen

    ```
    printf("%d\n", *p);  // Print the (int) value pointed to.
    ```

  - Man kann aber auch den [] Operator benutzen

    ```
    printf("%d\n", p[0]);  // p[0] and *p are synonymous.
    printf("%d\n", p[2]);  // Works, but probably undesried.
    ```

Zeiger 2/2

UNI FREIBURG

- Zeiger und Felder sind **exakt** dasselbe in C / C++
  - Insbesondere könnte man schreiben

```
int a[5] = {1, 2, 3, 4, 5};
int* p = a;
printf("%d\n", *p);  // Will print 1.
```

- Arithmetik auf Zeigern
  - Man kann eine Zahl zu einem Zeiger dazuaddieren
  - Die wird dann mit der Größe des Typs multipliziert

```
int *p = NULL;  // Pointer to address 0.
p += 2;  // Now points to address 8 (if int has 4 bytes).
```

Const 1/2

UNI FREIBURG

- Const steht vor Deklarationen einer Variable

  ```
  const int Pi = 3;
  ```

  – Das const bedeutet, dass man den Wert dieser Variablen nicht mehr verändern darf

  – Wann immer Variablen nur zum Lesen gedacht sind, sollte man const davor schreiben, als Schutz

  – const **vor** einem Zeiger bedeutet, dass man den Speicher auf den der Zeiger zeigt **nicht** verändern darf, aber schon wohin der Zeiger zeigt

  ```
  const int* p = &x;  // Adress of variable x.
  *p = 1;  // Will produce a compiler error.
  p = &y;  // This is fine though.
  ```

Const 2/2

UNI FREIBURG

- Das Umgekehrte gibt es auch
  - const **nach** einem Zeiger bedeutet, dass man den Speicher verändern darf auf den der Zeiger zeigt, aber nicht wohin er zeigt

```
int* const p = &x;  // Adress of variable x.

*p = 1;  // This is fine now.

p = &y;  // This will give a compiler error now.
```

  - Aber das braucht man äußerst selten und ist hier nur der Vollständigkeit halber erwähnt

Zeichenkette, engl. Strings

UNI FREIBURG

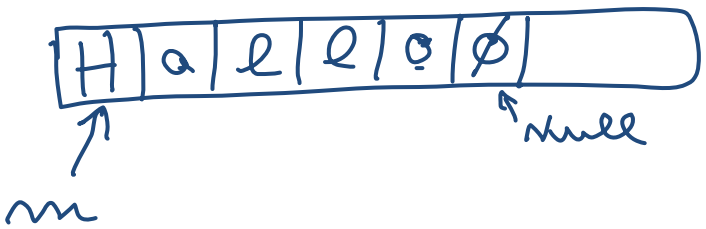
- Eine Zeichenkette ist auch nur ein Feld bzw. Zeiger
  - und zwar mit Typ char

    ```
    char a[5] = {'H', 'a', 'l', 'l', 'o'};
    char* p = a;  // Points to the cell containing the 'H'.
    ```

  - Kann man auch einfacher so initialisieren

    ```
    const char* m = "Hallo";  // m points to the cell with the 'H'.
    ```

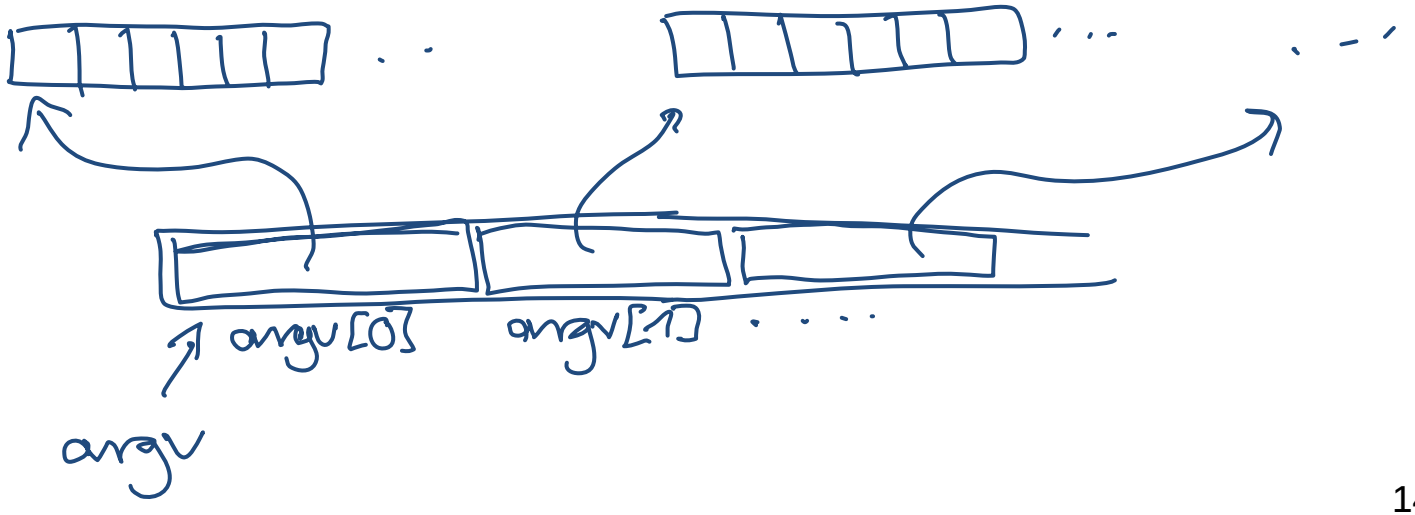
    Achtung: ohne das const gibt es eine Compiler-Warnung!
  - Strings in C / C++ sind null-terminated, d.h. bei "Hallo" wird Platz für **sechs** Zeichen gemacht, und am Ende steht eine 0
    - damit man weiß, wo die Zeichenkette aufhört



argv

- Jetzt können wir auch das char** argv verstehen
  - Das zweite Argument der main Funktion

    int main(int argc, char** argv)
  - char** ist ein Zeiger auf Werte vom Typ char*
  - Bzw. ein Feld von Elementen vom Typ char*, d.h. jedes Element zeigt auf eine Zeichenkette
  - Und argc sagt einfach wieviele Elemente es gibt



Debugging 1/4

UNI FREIBURG

- Fehler im Programm kommen vor
  - Und jetzt wo Sie Zeiger kennen, werden Sie gemeine segmentation faults produzieren
    - das passiert bei versuchtem Zugriff auf Speicher, der Ihrem Programm gar nicht gehört, zum Beispiel

```
char* p = NULL;  // Address 0 is written as NULL.
*p = 'x'; // Will produce a segmentation fault.
```

  - Die sind schwer zu debuggen, weil man nicht weiß, wo im Programm der Fehler aufgetreten ist, man bekommt einfach nur die Fehlermeldung segmentation fault
  - Manche Fehler sind zudem "nicht-deterministisch"

Debugging 2/4

UNI FREIBURG

- Methode 1 ("printf")
  - printf statements einbauen
    - an Stellen wo der Fehler vermutlich auftritt
    - von Variablen wo man denkt dass etwas falsch läuft
  - Haupt-Vorteil
    - Einfach
  - Haupt-Nachteil
    - Man muss jedesmal rekompilieren (das kann bei größeren Programmen lange dauern)

Debugging 3/4

UNI FREIBURG

- Methode 2 ("gdb")
  - Mit dem gdb = GNU debugger
  - Der kann so Sachen wie
    - Anweisung für Anweisung durch das Programm gehen
    - Sogenannte breakpoints im Programm setzen und zum nächsten breakpoint springen
    - Werte von allen möglichen Variablen ausgeben
  - Das wollen wir jetzt mal anhand eines Beispiels machen
  - Haupt-Vorteil:
    - Viel interaktiver als mit printf statements
  - Haupt-Nachteil:
    - Man muss sich ein paar gdb Kommandos merken

Debugging 4/4

UNI FREIBURG

- Ein paar grundlegende GDB Kommandos
  - **Wichtig:** Programm kompilieren mit der –g Option!
  - gdb aufrufen, z.B. gdb ./ListProcessingMain
  - Programm starten mit run <command line arguments>
  - stack trace (nach seg fault) mit backtrace oder bt
  - breakpoint setzen, z.B. break Number.cpp:47
  - breakpoints löschen mit delete oder d
  - Weiterlaufen lassen mit continue oder c
  - Nächste Programmzeile ausführen step oder s
  - Wert einer Variablen ausgeben, z.B. print x oder p i
  - Kommandoübersicht / Hilfe mit help oder help all
  - gbd verlassen mit quit oder q
  - Wie in der bash command history mit Pfeil hoch / runter

Literatur / Links

- Felder / Arrays
  - http://www.cplusplus.com/doc/tutorial/arrays
- Zeichenketten / Strings
  - http://www.cplusplus.com/doc/tutorial/ntcs
- Zeiger / Pointers
  - http://www.cplusplus.com/doc/tutorial/pointers
- Const
  - http://www.cplusplus.com/doc/tutorial/constants
- GNU debugger (gdb)
  - http://sourceware.org/gdb/current/onlinedocs/gdb