Programmierung 1

Bit-Operationen und Dateien

Monika Schak

Woche 5

22. November 2023

Hochschule Fulda
University of Applied Sciences

Wiederholung

- 7 primitive Datentypen in C:
  - `char` für ASCII-Zeichen
  - `short`, `int`, `long` für Ganzzahlen
  - `float`, `double` für Gleitkommazahlen, Darstellung über Vorzeichen, Mantisse und Exponent (Achtung Rundungsfehler!)
  - `bool` für Wahrheitswerte
- Konstanten mit `#define` oder `const`
- Definition neuer Typnamen über `typedef`
- Auswertungsreihenfolge emtsprechend **Assoziativität** und **Präzedenz**

Hochschule Fulda
University of Applied Sciences

Bitoperationen

- Logische Operationen auf einzelnen Bits in einem Speicherwort: gearbeitet wird auf der Zahldarstellung im Binärsystem
- Anwendungen:
  - Zur Code-Beschleunigung in der Graphik, z.B. Flags setzen
  - Für Encoding und Decoding ($\rightarrow$ XOR)
  - Zum Ansprechen bzw. Steuern von Geräten, z.B. Gerät an/aus, Fehlermeldungen, etc.
- Nur anzuwenden auf ganzzahlige (meist vorzeichenlose) Werte

Hochschule Fulda
University of Applied Sciences

Bitoperationen

- Bitweises UND: a & b
- Bitweises ODER: a | b
- Bitweises XOR: a ˆ b (Exklusives ODER)
- Komplement: $\sim$a (Einerkomplement)
- Linksshift: a $<<$ n (*n*: Anzahl zu verschiebender Bits)
- Rechtsshift: a $>>$ n (*n*: Anzahl zu verschiebender Bits)

Hochschule Fulda
University of Applied Sciences

Beispiel: Bitweises UND und ODER

- UND: 59947 & 21845
- ODER: 59947 | 21845

Hochschule Fulda
University of Applied Sciences

Beispiel: Bitweises XOR und NOT

- XOR: 59947 ^ 21845
- NOT: ∼59947

Hochschule Fulda
University of Applied Sciences

Schiebeoperation (Shift)

- Linksshift <<
  - Verschiebt Daten um angegebene Anzahl von Bits nach links
  - Bits, die links aus der Zahl „herausfallen“ (Überlauf), verschwinden
  - Rechts wird mit Nullen aufgefüllt
  - Beispiel: `i <<= 3;`

Shift	Dezimal	Binär
`7 << 0`	7	`0000 0000 0000 0111`
`7 << 1`	14	`0000 0000 0000 1110`
`7 << 2`	28	`0000 0000 0001 1100`
`7 << 3`	56	`0000 0000 0011 1000`

Hochschule Fulda
University of Applied Sciences

Schiebeoperation (Shift)

- Rechtsshift `>>`
  - Verschiebt Daten um angegebene Anzahl von Bits nach rechts
  - Bits, die rechts aus der Zahl „herausfallen“ (Überlauf), verschwinden
  - Links wird mit **Nullen** oder Einsen aufgefüllt, je nach Vorzeichen, daher: unsigned verwenden!
  - Beispiel: `i >>= 3;`

Shift	Dezimal	Binär
`7 >> 0`	7	`0000 0000 0000 0111`
`7 >> 1`	3	`0000 0000 0000 0011`
`7 >> 2`	1	`0000 0000 0000 0001`
`7 >> 3`	0	`0000 0000 0000 0000`

Hochschule Fulda
University of Applied Sciences

Schiebeoperation (Shift)

- Um n Bits nach rechts verschieben, entspricht Division durch 2^n
- Um n Bits nach links verschieben, entspricht Multiplikation mit 2^n
- Zusammengesetzte Operatoren:
  - `a = a & b;` entspricht `a &= b;`
  - `a = a | b;` entspricht `a |= b;`
  - `a = a ^ b;` entspricht `a ^= b;`
  - `a = a >> n;` entspricht `a >>= n;`
  - `a = a << n;` entspricht `a <<= n;`

Hochschule Fulda
University of Applied Sciences

Übung

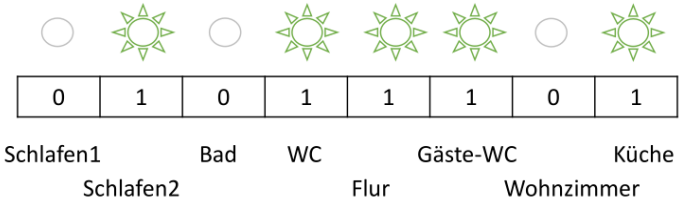
- Zahl `x` sei ein Integer
  - Woran sieht man anhand des Bitmusters, ob `x` gerade oder ungerade ist?
  - Welche Bitoperation kann dazu verwendet werden?
- Gegeben sei eine Variable `v` vom Typ `unsigned char` mit dem Wert 85.
  - Was ergibt `~v`?
  - Was ergäbe `~v`, wäre `v` stattdessen vom Typ `unsigned short`?

Hochschule Fulda
University of Applied Sciences

Beispiel: Gerätesteuerung

- Lichter im Haus steuern über Bitmuster

```
unsigned char licht = 93;
```



- Wie können alle Lichter, außer dem Flurlicht (soll den Wert nicht verändern) mit einem Befehl ausgeschaltet werden?
- Wie kann man das Licht nur in Flur und Küche einschalten? Alle anderen Lichter sollen aus sein.

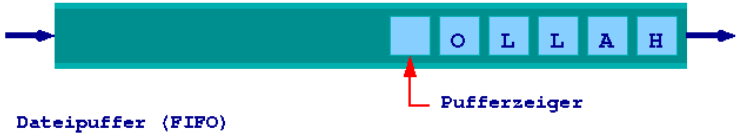
Hochschule Fulda
University of Applied Sciences

Datenverarbeitung

- Softwaresysteme sind meist auf Ein-/Ausgabe von Daten angewiesen
- Programme sollen nichts „vergessen“, sondern beim nächsten Start mit den Daten persistent weiterarbeiten können
- Je nach Anwendung unterschiedliche Dateiformate gebräuchlich, z.B. Konfigurationsdateien, Tabellen im CSV-Format, Bilddateien (PNG, JPG, ...)
- Dateizugriff ist betriebssystemabhängig
  - Dabei können diverse Probleme auftreten, z.B. Datei wurde nicht gefunden, keine Zugriffsrechte, Festplatte voll, usw.
  - Dateien sind Systemressourcen, daher sollten sie nach der Verwendung unbedingt geschlossen werden. Sonst sind Probleme möglich, z.B. kann eine Datei nicht zum Schreiben geöffnet werden, wenn sie noch zum Lesen geöffnet ist, oder das Betriebssystem-Limit an gleichzeitig geöffneten Dateien ist erreicht

Dateizugriff in C

- Zugrundeliegendes Modell: Datenstrom (Stream)
  - Bytes werden sequentiell gepuffert gelesen oder geschrieben
  - Beispiel: `printf("HALLO WELT\n");`



  - Konzept schon bekannt vom Einlesen eines Zeichens via `scanf("%c");`
- Grundlegender Datentyp: `FILE`
- Vordefinierte Variablen
  - Tastatur: `stdin`
  - Bildschirm: `stdout`, `stderr`

Hochschule Fulda
University of Applied Sciences

Dateien öffnen und schließen

- Dateien öffnen und schließen kann man über einen sogenannten File Pointer
  Zunächst Variable vereinbaren und initialiseren: `FILE *fp = NULL;`
- Dann Datei öffnen (hier Textdatei)
  Öffnen der Datei *file.txt*: `fp = fopen("file.txt", mode);` Rückgabewert bei Fehlern (z.B. Datei existiert nicht oder falscher Pfad): `NULL`
- Zugriffsmodi (mögliche Werte für mode):
  - `"r"`: zum Lesen öffnen (read), Beginn am Dateianfang, Datei muss existieren
  - `"w"`: zum Schreiben öffnen (write), Beginn am Dateianfang, erzeugt Datei bzw. überschreibt Datei falls diese schon vorhanden war
  - `"a"`: zum Schreiben öffnen (append): Anhängen neuer Daten am Dateiende
- Wichtig: Am Ende alle (vom OS angeforderten) Ressourcen wieder freigeben
  Nach Dateiverarbeitung erfolgreich geöffnete Datei wieder schließen: `fclose(fp);`

Hochschule Fulda
University of Applied Sciences

Dateien lesen und schreiben

- Formatierte Dateiausgabe über `fprintf()`
  Beispiel (zwei Werte in Datei schreiben, falls Datei erfolgreich geöffnet wurde):

```
if (fp != NULL) {
  fprintf(fp, "%d %d\n", 23, 42);
}
```

- Formatiertes Einlesen von Werten (solange Dateiende noch nicht erreicht):

```
while (!feof(fp)) {
  int a, b;
  // Rueckgabe von [f]scanf: Anzahl korrekt eingelesener Vars
  int i = fscanf(fp, "%d %d", &a, &b);
  if (i < 2) {
    break;
  }
}
```

Hochschule Fulda
University of Applied Sciences

Zeichenweise Verarbeitung

- Zeichen von Tastatur einlesen (beide Varianten sind identisch)
  - Variante 1: `char c = getchar();`
  - Variante 2: `char c = fgetc(stdin);`
  - Ersetzt man bei `fgetc()` Standardeingabe `stdin` durch einen Dateizeiger auf eine zum Lesen geöffnete Datei, kann man aus der Datei lesen
  - Hat eingelesenes Zeichen den Wert `EOF`, dann wurde das Dateiende erreicht
- Zeichen auf Konsole schreiben (beide Varianten sind identisch)
  - Variante 1: `putchar(c);`
  - Variante 2: `fputc(c, stdout);`
  - Ersetzt man bei `fputc()` Standardausgabe `stdout` durch einen Dateizeiger auf eine zum Schreiben geöffnete Datei, kann man in die Datei schreiben

Hochschule Fulda
University of Applied Sciences

Testen & Debuggen

Gemeldete Fehlerarten

- Compilerfehler
  - Syntax-Verletzungen
  - Beispielfehlermeldung:
    error C2065: 'ptr': nichtdeklarierter Bezeichner
- Linkerfehler
  - Es werden nicht alle zu bindenden Teile gefunden
  - Beispielfehlermeldung:
    error LNK2019: Verweis auf nicht aufgelöstes externes Symbol "__RTC_CheckESP" in Funktion "_main"
    error LNK2001: Nicht aufgelöstes externes Symbol "__RTC_InitBase"
- Laufzeitfehler
  - Unzulässige Speicherzugriffe, keine vorherige Initialisierung, Teilen durch 0, ...
- (Warnungen)

Hochschule Fulda
University of Applied Sciences

Testen und Fehlerbeseitigung

- **Syntaxfehler** (Verstöße gegen die Grammatik einer Programmiersprache) sind typische Fehler, die man bekommt, wenn man eine neue Programmiersprache erlernt. Compiler melden Syntaxfehler.
- **Logische Fehler** (ein Programm ist syntaktisch in Ordnung, liefert aber ein falsches Ergebnise) sind Fehler, die nicht so leicht zu finden sind.
- Wenn die Aufgabenstellung nicht trivial ist, ist es in der Regel nicht leicht, ein **logisch korrektes Programm** zu **schreiben**. Der Nachweis, dass ein Programm logisch korrekt ist, lässt sich (fast) nicht automatisieren.
- Mit **Testen** lässt sich herausfinden, ob ein Stück Software Fehler enthält. Testen dient der Überprüfung, ob ein Stück Software das gewünschte Verhalten zeigt.
- Nach dem Testen kommt die **Fehlerbeseitigung (Debugging)**. Dies bezeichnet die Suche nach der Ursache eines Fehlers und seine Beseitigung.
- **Wartbare Quelltexte** helfen Programme so zu schreiben, dass Fehler von vorneherein vermieden werden: **Programmierstil, Kommentierung, Layout**

Programmierrichtlinien

- Fehlerentstehung soll erschwert werden - Fehlerauffindung soll erleichtert werden
- Code soll übersichtlich und leicht lesbar sein, z.B. durch Einrücken von Blöcken immer mit 4 Spaces (keine Tabs)
- Code soll wartbar sein, d.h. auch von anderen leicht modifizierbar. Mehrfach vorkommender Code sollte nicht kopiert, sondern ausgelager werden ($\rightarrow$ Funktionen - später!). Variablen- und Funktionsnamen sollten kurz und aussagekräftig sein (z.B. Verwendung von CamelCase oder mit _)
- Kommentieren nicht vergessen! Insbesondere dann, wenn der Code umfangreicher und/oder komplizierter ist.

Hochschule Fulda
University of Applied Sciences