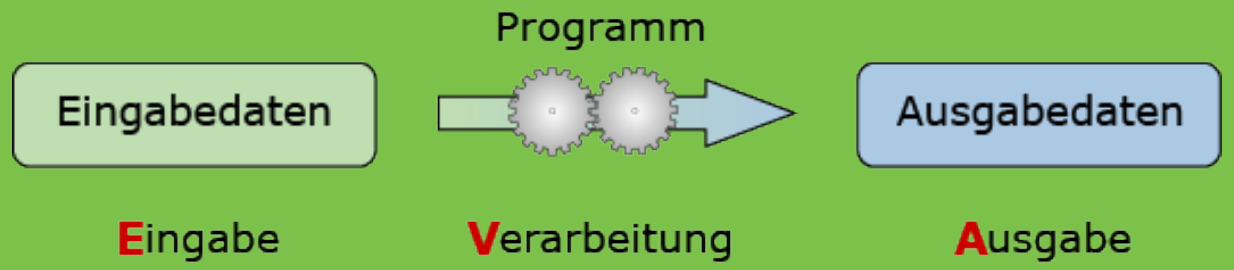
Hochschule Fulda
University of Applied Sciences

Programmierung 1 – Bitoperationen & Dateien



Yvonne Jung

Typnamen definieren

Hochschule Fulda
University of Applied Sciences

- Neue Typnamen lassen sich mit Schlüsselwort `typedef` definieren
  - Hier wird kein neuer Datentyp geschaffen, sondern nur ein Name festgelegt
  - Typdefinitionen steht meist am Anfang des Codes außerhalb von Funktionen
- Beispiel: `typedef double Real;`
  - Nutzt man nun überall im Code Typ *`Real`* statt `double`, kann man z.B. durch einfaches Ändern der Typdefinition rasch von `double` auf `float` wechseln
- Vorteile
  - Einfaches und durchgängiges Ändern des Typs von Variablen möglich
  - Informativere, problemangepasste oder ggfs. besser lesbare Typnamen
  - Zusammengesetzte Datentypen (→ später) wie Standardtypen verwendbar

Hochschule Fulda
University of Applied Sciences

Bitmanipulationen

Bitoperationen

Hochschule Fulda
University of Applied Sciences

- Logische Operationen auf einzelnen Bits in einem Speicherwort
  - Arbeiten auf Zahldarstellung im Binärsystem
    - https://www.arndt-bruenner.de/mathe/scripts/Zahlensysteme.htm
- Anwendungen
  - Zur Code-Beschleunigung in der Graphik (z.B. Flags setzen oder für Blitting)
  - Für En- u. Decoding sowie Verschlüsselung (→ XOR)
  - Zum Ansprechen bzw. Steuern von Geräten
    - Bsp.: Gerät ein oder aus, Drucker hat kein Papier oder Papierstau etc.
- Nur anzuwenden auf ganzzahlige (meist vorzeichenlose) Werte
  - Oft werden Zahlen dabei in hexadezimaler Form angegeben
    - Bsp.: `int a = 0xea2b, b = 0x5555;  // a=59947, b=21845`

Bitoperationen

Hochschule Fulda
University of Applied Sciences

- Bitweises UND: a & b
- Bitweises ODER: a | b
- Bitweises XOR: a ^ b (Exklusiv-ODER)

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

- Komplement: ~a (Einerkomplement)
- Linksshift: a << *n* (*n* Anzahl zu verschiebender Bits)
- Rechtsshift: a >> *n* (*n* ")

Bsp.: bitweises & und |

Hochschule Fulda
University of Applied Sciences

- AND: 59947 & 21845

```
   1110 1010 0010 1011
&  0101 0101 0101 0101   ergibt
-------------------------
   0100 0000 0000 0001
```

- OR: 59947 | 21845

```
   1110 1010 0010 1011
|  0101 0101 0101 0101   ergibt
-------------------------
   1111 1111 0111 1111
```

Bsp.: bitweises ^ und ~

Hochschule Fulda
University of Applied Sciences

- XOR: 59947 ^ 21845

```
  1110 1010 0010 1011
^ 0101 0101 0101 0101   ergibt
-------------------
  1011 1111 0111 1110
```

  - Zweimal XOR stellt alten Wert wieder her: (a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a

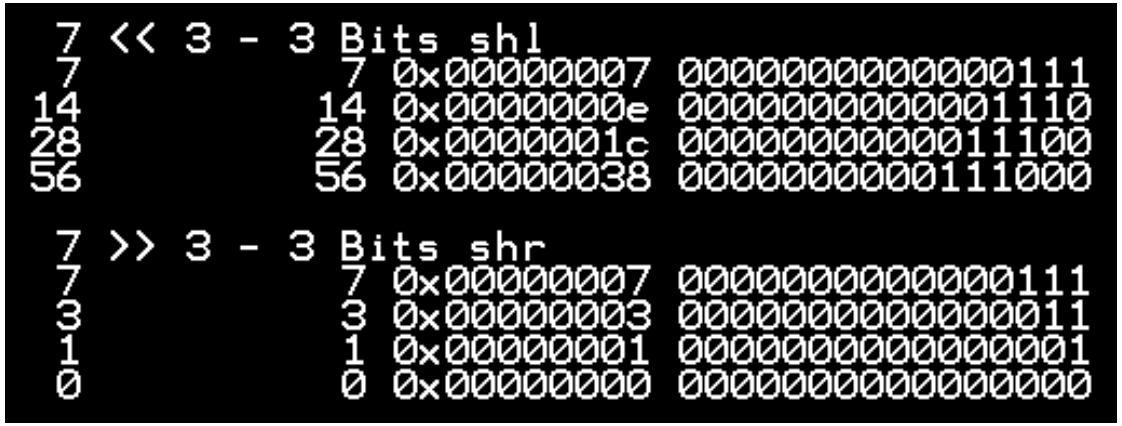
- NOT: ~ 59947

```
~ 1110 1010 0010 1011   ergibt
-------------------
  0001 0101 1101 0100
```

Schiebeoperationen (Shift)

Hochschule Fulda
University of Applied Sciences

- Linksshift<<
  - Verschiebt Daten um angegebene Anzahl von Bits nach links
  - Bits, die links aus Zahl “herausfallen“ (Überlauf), verschwinden
  - Rechts wird mit Nullen aufgefüllt
  - Bsp.: `i <<= 3;  // identisch zu i *= 8`



Schiebeoperationen (Shift)

Hochschule Fulda
University of Applied Sciences

- Um *n* Bit nach rechts verschieben entspricht Division durch 2^n
- Um *n* Bit nach links verschieben entspricht Multiplikation mit 2^n

>>	`1110 1010 0010 1011 >> 5` ergibt
(rechts schieben)	`1111 1111 0101 0001` oder
	`0000 0111 0101 0001`
	(Ergebnis abhängig von Vorzeichen, daher **unsigned** nutzen)
<<	`1110 1010 0010 1011 << 5` ergibt
(links schieben)	`0100 0101 0110 0000`

Übung

Hochschule Fulda
University of Applied Sciences

- Zahl **`x`** sei ein Integer
  - Woran sieht man anhand des Bitmusters, ob **`x`** gerade oder ungerade?

    Ist niedrigstes Bit 0 → gerade, 1 → ungerade
  - Welche Bitoperation kann dazu verwendet werden?

    x & 1
- Geg. sei Variable **`v`** vom Typ *unsigned char* mit Wert 85
  - Was ergibt **`~v`**?

    170 (8 Bit, daher bei Binärdarstellung 0 voranstellen)
  - Was ergäbe **`~v`**, wäre **`v`** stattdessen ein *unsigned short*?

    65450

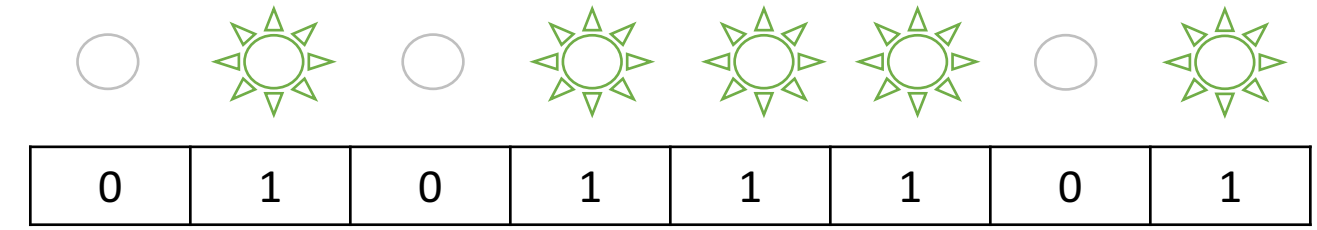
```
85 : 2 = 42 R 1
42 : 2 = 21 R 0
21 : 2 = 10 R 1
10 : 2 =  5 R 0
 5 : 2 =  2 R 1
 2 : 2 =  1 R 0
 1 : 2 =  0 R 1
```

Gerätesteuerung

Hochschule Fulda
University of Applied Sciences

- Lichter im Haus steuern

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

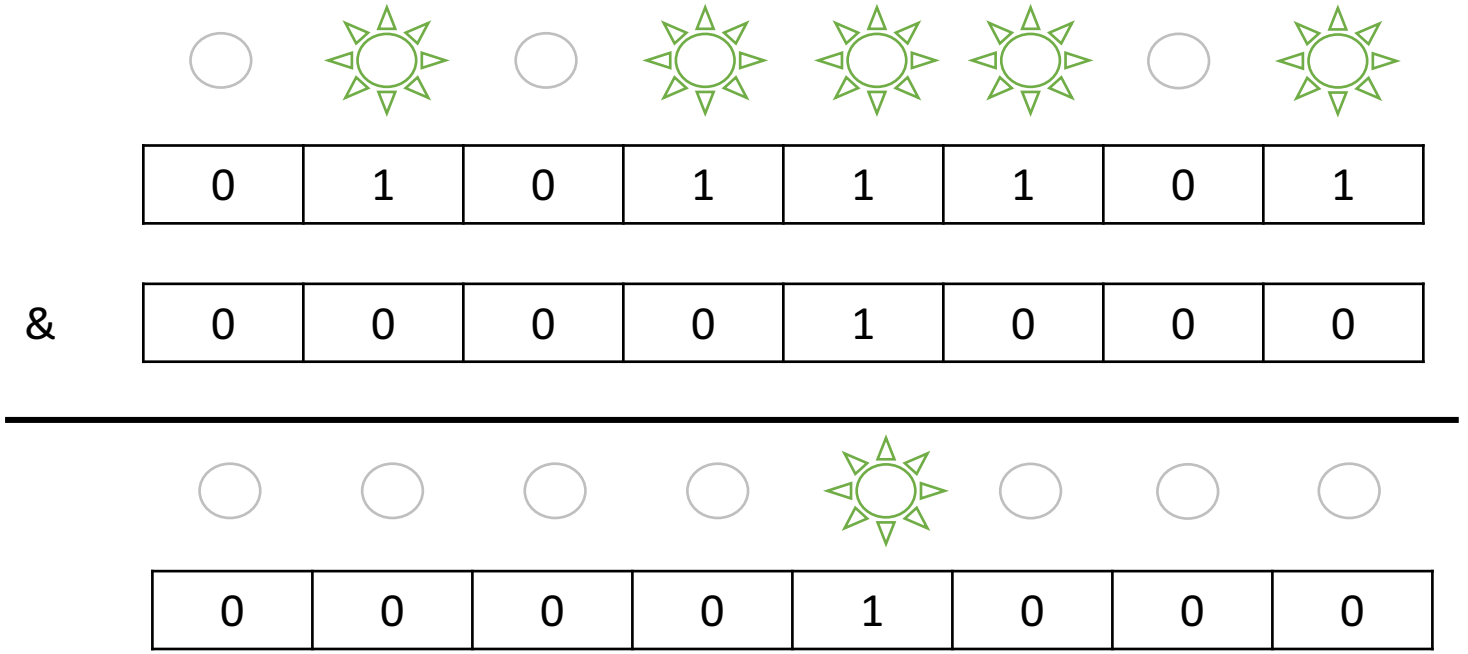


- Wie können alle Lichter außer dem Flurlicht (soll seinen Wert nicht verändern) mit einem Befehl ausgeschaltet werden?
- Wie kann man das Licht nur in Flur und Küche einschalten?

Gerätesteuerung

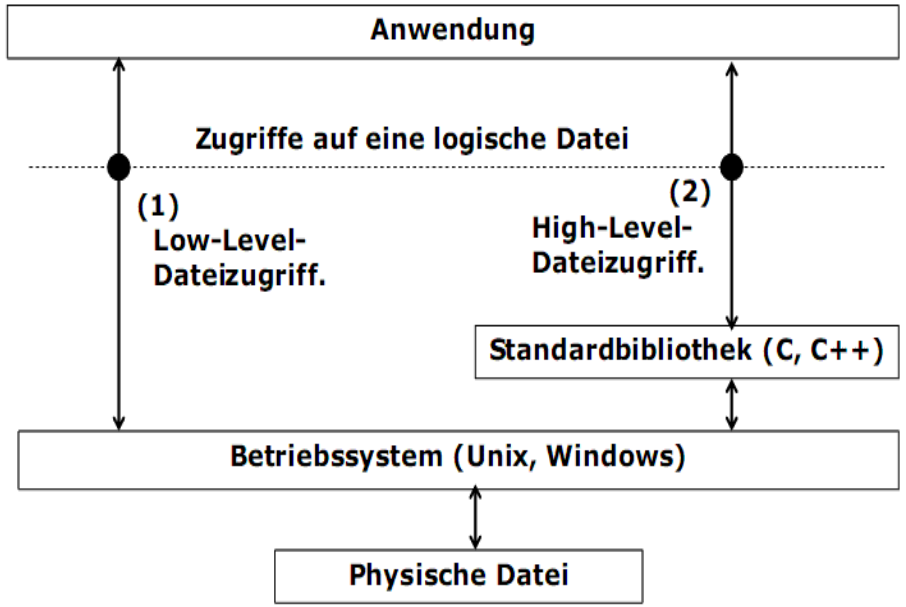
Hochschule Fulda
University of Applied Sciences

- Lösung 1: `licht = licht & 8;`



- Lösung 2: `licht = 9;`

Dateioperationen



Datenverarbeitung

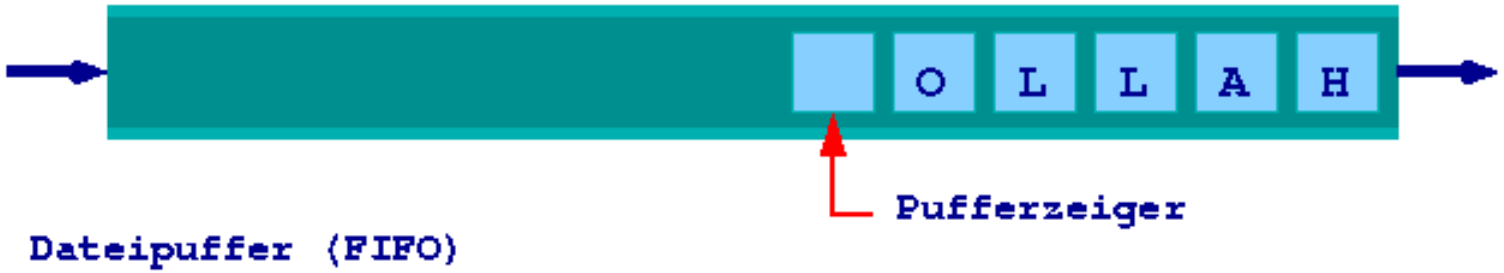
Hochschule Fulda
University of Applied Sciences

- 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 nach Verwendung schließen
    - Sonst Probleme möglich (z.B. Datei kann nicht zum Schreiben geöffnet werden, da noch zum Lesen offen, oder Betriebssystem-Limit an gleichzeitig offenen Dateien ist erreicht)

Dateizugriff in C

Hochschule Fulda
University of Applied Sciences

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



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

Dateien öffnen u. schließen

Hochschule Fulda
University of Applied Sciences

- Geschieht über sog. Dateizeiger (File Pointer)
  - Zunächst Variable vereinbaren (und initialisieren): `FILE *fp = NULL;`
- Dann Datei öffnen (hier Textdateien)
  - Beispiel (Öffnen von 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);`

Dateien lesen u. schreiben

Hochschule Fulda
University of Applied Sciences

- 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);
}
```

- Beispiel zum Einlesen von Werten (solange Dateiende noch nicht erreicht)

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

Zeichenweise Verarbeitung

Hochschule Fulda
University of Applied Sciences

- 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

Anhang

Testen und Debuggen

Gemeldete Fehlerarten

Hochschule Fulda
University of Applied Sciences

1. Compilerfehler
   - Syntax-Verletzungen
   - Beispielfehlermeldung:

   error C2065: 'ptr': nichtdeklarierter Bezeichner
2. 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"
3. Laufzeitfehler
   - Unzulässige Speicherzugriffe, keine vorherige Initialisierung, Teilen durch 0, …
4. (Warnungen)

Testen und Fehlerbeseitigung

Hochschule Fulda
University of Applied Sciences

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 Ergebnis) sind Fehler, sind nicht so leicht zu finden und beschäftigen auch erfahrene Programmierer.

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 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 vornherein vermieden werden:
Programmierstil, Kommentierung, Layout.

Programmierrichtlinien

Hochschule Fulda
University of Applied Sciences

- Fehlerentstehung soll erschwert werden
  - Fehlerauffindung soll erleichtert werden
- Code soll übersichtlich und leicht lesbar sein
  - Z.B. Einrückungen von Blöcken immer mit 4 Spaces (keine Tabs)
- Wartbar, d.h. auch von anderen leicht modifizierbar
  - Code nicht kopieren, mehrfach vorkommenden Code auslagern
    - In Funktionen (→ später)
  - Aussagekräftige, kurze Variablen- u. Funktionsnamen, CamelCase oder mit '_'
- Kommentieren nicht vergessen!
  - Insbes. wenn der Code umfangreicher oder komplizierter ist

Vielen Dank!

Noch Fragen?