INF3.1

Zürcher Hochschule für Angewandte Wissenschaften

Arbeitsblatt: INF1

Name: Nathaniel Walser **Kurznamen:** walnat01

Variablen und einfache Datentypen

Ziele

- Aufbau einfacher C-Programme kennen und verstehen
- Funktionen *printf* und *scanf* anwenden können
- Variablen und Datentypen *int* und *double* einsetzen können

Aufgabe 1: Variablen

Einfach gesagt sind Variablen Namen für Speicherstellen, in denen Werte gespeichert werden können. Variablen in C haben einen bestimmten Datentyp und müssen entsprechend vereinbart werden. Hier einige Beispiele, Erklärungen dazu werden im Praktikum gegeben:

```
int a;
int wert = 12;
double x, y;
double z = 1.5;
```

Die gezeigten Variablenvereinbarungen enthalten zwei Aspekte: Zum einen wird hier angegeben, dass die Variable *a* vom Typ *int* ist (Deklaration). Zum anderen wird Platz im Speicher für die Variable reserviert (Definition). Variablenvereinbarungen müssen in C jeweils vor den anderen Anweisungen stehen (genau genommen gilt diese Einschränkung seit dem C99-Standard nicht mehr, es ist trotzdem empfehlenswert, sich daran zu halten).

Einer so vereinbarten Variablen können im weiteren Programmverlauf Werte zugewiesen werden. Auch kann die Variable in Ausdrücken oder als Argument von Funktionen verwendet werden:

```
a = 17;          /* a erhaelt den Wert 17 /
x = z + y;       /* z + y wird ausgerechnet und das Ergebnis der
                    Variablen x zugewiesen */
```

Aufgabe

- Passen Sie die *folgenden Anweisungen* so an, dass für die Zahlen in rot (Konstanten) Variablen verwendet werden.

```
printf("%d\n", 17);
printf("%d**%d%%\n", 17, 24);
printf("%x\n", 21*2);
printf("%d\n", 3/4);
```

```
printf("%f\n", 3/4);
printf("%f\n", 3.0/4.0);
```

Abgabe

Praktikum: INF3.2

Filename: testvars.c

Aufgabe 2: Werte einlesen

Mit *scanf* können Werte in Variablen eingelesen werden. Testen Sie dies mit dem folgenden C-Programm:

```
int main() {
    int zahl;
    printf("Bitte ganze Zahl eingeben: ");
    scanf("%d", &zahl);
    printf("Zahl dezimal: %d\n", zahl);
    printf("hexadezimal:  %x\n\n", zahl);
    return 0;
}
```

Hinweise

- Vergessen Sie nicht das & vor der Variablen in *scanf*. Warum dies nötig ist, wird erst im Zusammenhang mit der Behandlung von Zeigern in C verständlich.
- Der Kontrollstring in *scanf* enthält kein "\n".
- Die Funktion *scanf* ist nicht ganz unproblematisch. Wenn die Eingabe nicht genau im spezifizierten Format erfolgt, wird sich das Programm nicht wie erwartet verhalten. Wir werden noch andere Funktionen zum Einlesen von Werten kennenlernen.

Aufgaben

- Testen Sie das Programm mit folgenden Eingaben:
  2000
  2000000000 (2 Milliarden, 9 Nullen)
  3000000000
  –1
- Was beobachten Sie? Können Sie die Ergebnisse erklären?

a) Beobachtung/Erklärung

Bei 3 Milliarden gibt es einen Überlauf. Die Hexadezimale Zahl ist Negativ.
Bei der Zahl -1 kommt "ffffffff" heraus. Also die grösste Zahl die in einem int gespeichert werden kann.

- Können Sie aus dem Ergebnis schliessen, wie viele Bytes für ein *int* in Ihrer C-Installation reserviert werden?

b) Mein System verwendet 4 Bytes für einen Integer.

Aufgabe 3: Grundrechenarten mit int

Es soll ein Programm geschrieben werden, das zwei ganze Zahlen (Typ: int) einliest und diese beiden Zahlen addiert, subtrahiert, usw. entsprechend diesem Beispiel:

```
Bitte ganze Zahl eingeben:   15
Bitte weitere Zahl eingeben: 3
zahl1 + zahl2 = 18
zahl1 - zahl2 = 12
zahl1 * zahl2 = 45
zahl1 / zahl2 = 5
zahl1 % zahl2 = 0
```

Hinweis

Auch Ausdrücke, die ein geeignetes Ergebnis liefern können als Argumente bei Funktionsaufrufen verwendet werden. Die oben genannten Rechenoperationen können also problemlos direkt in der printf-Funktion angegeben werden, ohne die Ergebnisse zunächst in einer Variablen zu speichern. Beispiel: `printf("Ergebnis: %d\n", zahl1 + zahl2);`

Aufgaben

- Schreiben Sie das C-Programm gemäss den obigen Anforderungen. Achten Sie auf gute Lesbarkeit und versehen Sie das Programm mit Kommentaren wo nötig.
- Testen Sie das Programm mit verschiedenen Zahlenkombinationen.
- Was geschieht, wenn Sie als zweite Zahl 0 eingeben?

Das Programm endet bei der Division durch 0 ab.

- Was geschieht, wenn Sie für beide Zahlen 100’000 (ohne das ’-Zeichen) eingeben? Ein weiterer Punkt, der in der Programmbeschreibung zu dokumentieren ist.

Bei der Multiplikation gibt es einen Überlauf.

- Was geschieht, wenn Sie für beide Zahlen Fliesskommazahlen eingeben, e.g. 3.14

Die Eingabe für die zweite Zahl wird übersprungen. (Default von 0 wird genommen)

- Gibt es auch Zahlenkombinationen, bei denen die Addition falsche Ergebnisse liefert? Testen Sie es. Beschreiben Sie Ihre Beobachtungen?

Sobald die beiden Zahlen Addiert grösser als die Grösstmögliche darstellbare Zahl eines Integers ist, gibt es einen Überlauf und somit das falsche Ergebniss.

- Welche Schlussfolgerungen müssen daher für das Rechnen mit ganzen Zahlen in C gezogen werden?

Bei rechnen mit Zahlen in C müssen Bereichsüberprüfungen der Eingegebenen Werte erfolgen.

Aufgabe 4: Grundrechenarten mit *double*

Nun möchten wir noch wissen, wie die Rechenoperationen mit Fliesskommazahlen funktionieren, ob auch in diesem Fall fehlerhafte Ergebnisse ausgegeben werden.

Aufgaben

- Machen Sie eine Kopie ihres letzten Programms. Modifizieren es so, dass es mit *double*-Zahlen arbeitet (%lf beim Einlesen und Ausgeben) und entfernen Sie die letzte Zeile mit dem %-Operator.
- Testen Sie das Programm mit den Zahlen 9.6 und 5.7. Sind die Ergebnisse korrekt? Warum oder warum nicht?
- Testen Sie das Programm mit den Zahlen 1'000'000'000 (ohne die ’-Zeichen) und 2. Sind die Ergebnisse korrekt? Versuchen Sie Ihre Beobachtungen zu begründen?

Mit den Zahlen 9.6 und 5.7 funktioniert es.
Auch mit beiden Zahlen als 1000000000 funktioniert es.

Abgabe

Praktikum: INF3.1 (dieses Arbeitsblatt)

Filename: INF3.1.pdf

Aufgabe 5: Dreiecksfläche berechnen

Schreiben Sie ein Programm, das drei Seitenlängen als *double* einliest und daraus die Fläche des Dreiecks berechnet (siehe Auszug aus Wikipedia unten)

Alle drei Seitenlängen gegeben

Sind alle drei Seitenlängen eines Dreiecks bekannt, so lässt sich der Satz des Heron anwenden:

$$F = \sqrt{s(s-a)(s-b)(s-c)}$$

Dabei ist $s = \frac{a+b+c}{2}$ der halbe Umfang des Dreiecks.

Abgabe

Praktikum: INF3.5

Filename: triangle.c

Erkenntnisse

Aus den obigen Aufgaben können ein paar Schlüsse zum Rechnen mit *int*- und *double*-Zahlen gezogen werden. Es ist wichtig, diese Erkenntnisse beim Entwickeln von C-Programmen zu berücksichtigen.

- Ganze Zahlen vom Typ *int* liefern beim Anwenden von Rechenoperatoren falsche Ergebnisse, wenn der erlaubte Zahlenbereich verlassen wird (Integer-Überlauf). Es erfolgt in diesem Fall keine automatische Fehlermeldung. Das Abfangen solcher Fehler ist Aufgabe des Programmierers.
- Das Problem ist allerdings weniger gravierend, als es zunächst den Anschein hat. Fehlerhafte Ergebnisse sind nur beim Umgang mit sehr grossen Zahlen zu erwarten. Man sollte sich aber des Problems immer bewusst sein.
- Zahlen mit Nachkommastellen sind selten exakt als Fliesskommazahlen vom Typ *double* darstellbar. Daher ist auch bei Rechenoperationen immer mit Ungenauigkeiten zu rechnen. Wenn man das berücksichtigt und keine exakten Ergebnisse erwartet, ist es normalerweise kein grosses Problem.
- Das hier Gesagte gilt auch für die anderen Zahlentypen *double*, *long*, usw.