Deuersprech - Benutzerdokumentation

github.com/deuersprechkompilierer

16. Dezember 2019

Inhaltsverzeichnis

1 Verwendung von Deuersprech **3**
1.1 Datentypen . . . 3
1.1.1 ganzzahl . . . 3
1.1.2 kettte . . . 3
1.2 Bezeichner . . . 3
1.3 Literale . . . 3
1.3.1 ganzzahl-Literale . . . 3
1.3.2 kette-Literale . . . 4
1.4 Besondere Formatierungszeichen . . . 4
1.5 Aussagen . . . 4
1.6 Variablen . . . 4
1.6.1 Deklaration . . . 4
1.6.2 Definition . . . 5
1.6.3 Aufruf . . . 5
1.7 Konstanten . . . 6
1.7.1 Deklaration . . . 6
1.7.2 Definition . . . 6
1.7.3 Ergebnisausgabe . . . 6
1.8 Funktionen . . . 6
1.8.1 Deklaration . . . 6
1.8.2 Aufruf . . . 7
1.8.3 Gültigkeitsbereiche . . . 7
1.8.4 Überladen . . . 8
1.9 Arithmetik . . . 8
1.9.1 Operationen . . . 8
1.9.2 Operanden . . . 9
1.9.3 Klammerung . . . 9
1.10 Aussagenlogik . . . 9
1.11 Bedingte Verzweigungen . . . 10
1.12 Schleifen . . . 10
1.13 Reservierte Schlüsselwörter und Symbole . . . 11

2 Aufruf des Deuersprech-Kompilierers **12**

1 Verwendung von Deuersprech

Ein gültiges Programm in Deuersprech besteht aus beliebig vielen Aussagen und Funktionsdefinitionen.

1.1 Datentypen

Deuersprech verfügt über zwei Datentypen.

1.1.1 ganzzahl

Der Datentyp 'ganzzahl' repräsentiert eine ganze Zahl mit einer Größe von 32 Binärziffern. Mögliche Werte liegen zwischen -2147483648 und 2147483647 (-2^{31} bis $2^{31} - 1$).

1.1.2 kettte

Der Datentyp 'kette' repräsentiert eine Kette von Zeichen. Eine kette darf 0 bis beliebig viele Zeichen enthalten. Ein Zeichen muss ein gültiges UTF-8-Symbol sein.

1.2 Bezeichner

Bezeichner werden genutzt, um Variablen, Konstanten sowie Funktionen zu benennen. Bezeichner müssen eindeutig sein, d.h. ein Bezeichner darf im entsprechenden Gültigkeitsbereich in nur einer Deklaration des gleichen Typ vorkommen. Gültige Bezeichner beginnen mit einem Klein- oder Großbuchstaben und dürfen von beliebig vielen weiteren Klein- bzw. Großbuchstaben sowie den Ziffern von 0 bis 9 gefolgt werden. Ein Bezeichner darf kein reserviertes Schlüsselwort sein (vgl. Liste der reservierten Schlüsselwörter). Desweiteren darf ein Bezeichner, der aus mehr als zwei Zeichen besteht, kein englisches Wort sein und muss mindestens einen Umlaut enthalten.

Beispiel:

```
ganzzahl gultigerDeuersprechBezeichner;
konstante ganzzahl EINANDERERGULTIGERBEZEICHNER;
kette undNochEinar;
ganzzahl x;
```

1.3 Literale

Literale sind konstante Ausdrücke, die im Quelltext verwendet werden. Der Wert eines Literals kann ab der Kompilierzeit nicht mehr verändert werden.

1.3.1 ganzzahl-Literale

ganzzahl-Literale bestehen aus einer beliebig langen Folge der Ziffern von 0 bis 9. Ein ganzzahl-Literal muss aus mindestens einer Ziffer bestehen.

Beispiel:

```
x ISTGLEICH 42
wenn(1) {
  druckzeile(839);
} sonst {
  druckzeile(3);
}
```

1.3.2 kette-Literale

kette-Literale bestehen aus einem doppelten Anführungszeichen, werden von beliebig vielen (auch 0) Zeichen gefolgt und mit einem weiteren doppelten Anführungszeichen wieder abgeschlossen.

Beispiel:

```
x ISTGLEICH "Hallo Welt";
y ISTGLEICH "";
druckzeile("Tja, einfach smart, deshalb haben wir es auch so genannt: das 1 u
```

1.4 Besondere Formatierungszeichen

Leerzeichen, Tabulatoren sowie Zeilenumbrüche dürfen in einer Quelltext-Datei vorkommen, werden jedoch vom Kompilierer ignoriert.

1.5 Aussagen

Alle Aussagen außer wenn-dann-Anweisungen und Schleifen werden mit einem Semikolon ";" beendet.

Beispiel:

```
druckzeile(39);
druck(42);
konstante ganzzahl PI;
ganzzahl x;
x ISTGLEICH 57;
```

1.6 Variablen

1.6.1 Deklaration

Eine Variablendeklaration besteht aus einem Datentyp sowie dem gewünschtem Namen (Bezeichner).

Beispiel:

```
ganzzahl a;
kette b;
```

1.6.2 Definition

Nach dem eine Variable deklariert wurde, kann ihr ein Wert zugewiesen werden. Eine Zuweisung erfolgt durch das Zuweisungsoperator "ISTGLEICH". Die linke Seite der Zuwesiung ist dabei der Name der Variable, die rechte Seite kann ein Literal, ein arithmetischer Ausdruck, der Aufruf einer Funktion, eine Variable oder eine Konstante sein. Wertzuweisungen von Variablen dürfen beliebig oft erfolgen. Die Zuweisung muss getrennt von der Deklaration erfolgen.

Bemerkung: Die Deklaration **muss** unbedingt von der Definition getrennt sein, d.h. bei der Deklaration kann keine Wertzuweisung erfolgen.

Beispiel:

```
beispiel ISTGLEICH 42;
beispiel ISTGLEICH eineFunktion();
beispiel ISTGLEICH 1 PLUS 2 PLUS 3;
beispiel ISTGLEICH eineAndereVariable;
beispiel ISTGLEICH EINEKONSTANTE;
```

1.6.3 Aufruf

Variablen dürfen auch in der rechten Seite einer Zuweisung vorkommen.

Beispiel:

```
ganzzahl beispiel1;
ganzzahl beispiel2;

beispiel1 ISTGLEICH 42;
beispiel2 ISTGLEICH beispiel1 MINUS 1;
```

1.7 Konstanten

Konstanten werden ähnlich wie Variablen verwendet. Nur `ganzzahl`en können Konstanten sein.

1.7.1 Deklaration

Im Gegensatz zu einer Variablen wird dem Namen das Schlüsselwort *konstante ganzzahl* anstatt nur *ganzzahl* vorgestellt.

Beispiel:

```
konstante ganzzahl BEISPIEL;
```

1.7.2 Definition

Die Wertzuweisung erfolgt wie bei einer Variable, darf jedoch nur ein mal pro Konstante erfolgen.

1.7.3 Ergebnisausgabe

Mit Hilfe der Funktion `druckzeile(<argument>)` können Literale, arithmetische Ausdrücke, logische Ausdrücke, Vergleiche, Rückgabewerte von Funktionen, Variablen und Konstanten auf die Konsole ausgegeben werden.
Die Funktion `druck(<argument>)` funktionert analog, gibt jedoch nach dem ausgegebenen Ausdruck keinen Zeilenumbruch aus.

Beispiele:

```
ganzzahl beispiel;
konstante ganzzahl BEISPIEL;

beispiel ISTGLEICH 42;
BEISPIEL ISTGLEICH 123;

druckzeile(42);
druckzeile(42 MAL 5);
druckzeile(42 KLEINER 5);
druckzeile(42 KLEINER 5 UND beispiel GLEICH BEISPIEL);
druckzeile(testFunktion(42));
druck(beispiel);
druck(BEISPIEL);
```

1.8 Funktionen

1.8.1 Deklaration

Funktionen könnnen nahezu an einer beliebigen Stelle in der Quellkodierung deklariert werden. Die Deklaration darf jedoch nur unmittelbar vor oder unmittelbar nach einer abgeschlossenen Aussage stattfinden. Ein Aufruf ist auch vor der Deklaration möglich.

Eine Funktionsdeklaration setzt sich folgendermaßen zusammen: Schlüsselwort "ganzzahl" oder "kette" für den Typ des Rückgabewerts, Bezeichner als Name der Funktion, öffnende runde Klammer, beliebig viele Variablendeklarationen (keine Übergabeparameter sind auch möglich), schließende runde Klammer und schließlich ein Funktionsrumpf. Der Funktionsrumpf beginnt mit einer öffnenden geschweiften Klammer gefolgt von beliebig vielen Aussagen. Eine besondere Aussage ist die Rückgabe. Beim Auruf von *gebzueruck* wird die Funktion verlassen. Wenn hinter *gebzueruck* ein Literal, eine Variable, eine Konstante, ein arithmetischer Ausdruck, ein logischer Ausdruck oder der Aufruf einer weiteren Funktion folgt, so stellt dies den Rückgabewert der Funktion dar. Der Funktionsrumpf wird mit einer schließenden geschweiften Klammer abgschlossen.

Bemerkung: Die gebzueruck-Aussage darf nicht weggelassen werden.

Beispiel:

```
ganzzahl addiere(ganzzahl a, ganzzahl b) {
        druckzeile(a);
        druckzeile(b);
        gebzueruck a PLUS b;
}
```

1.8.2 Aufruf

Funktionsaufrufe dürfen in der rechten Seite von Zuweisungen, als Argument für die druckzeile-Funktion und als Teil von logischen sowie arithmetischen Ausdrücken vorkommen. Funktionsaufrufe dürfen auch eigenständige Aussagen sein. Ein Funktionsaufruf setzt sich wie folgt zusammen:
`[nameDerFunktion]([Parameterliste])`
Die Funktion wird beim Verlassen mit dem zuvor definierten Rückgabewert substituiert.

Beispiel:

```
ganzzahl x; ganzzahl y; ganzzahl z;

x ISTGLEICH 40;
y ISTGLEICH 2;
z ISTGLEICH addiere(x, y);
```

1.8.3 Gültigkeitsbereiche

Funktionen besitzen einem vom Hauptprogramm verschiedenen Gültigkeitsbereich. Das bedeutet, dass Variablen, die innerhalb eines Funktionsrumpfs deklariert werden, außerhalb der Funktion nicht verwendet werden können.

Das folgende Beispiel verdeutlicht dieses Prinzip:

```
ganzzahl zufallsZahl() {
    ganzzahl i;
    i ISTGLEICH 42;
```

```
    gebzurueck i;
}

ganzzahl i;
i ISTGLEICH 2;

druckzeile(zufallsZahl());
druckzeile(i);
```

Dieses Programm erzeugt die Ausgabe

```
42
2
```

1.8.4 Überladen

In Deuersprech ist es möglich Funktionen zu überladen, d.h. dass in der gleichen Quelldatei mehrere Funktionen mit dem gleichen Bezeichner existieren dürfen, wenn die Funktionen sich in ihrer Parameterliste und/oder ihrem Rückgabewert unterscheiden.

Beispiel:

```
ganzzahl testFunc() {
  gebzurueck 42;
}

ganzzahl testFunc(ganzzahl a) {
  gebzurueck a;
}

druckzeile(testFunc());
druckzeile(testFunc(23));
```

Dieses Programm erzeugt die Ausgabe

```
42
23
```

1.9 Arithmetik

Arithmetische Ausdrücke dürfen folgende Komponenten besitzen:

1.9.1 Operationen

Jede Operation setzt sich aus einem linken Operanden, einem Operator sowie einem rechten Operanden zusammen.
Zulässige Operationen sind:

Operator	Operation
PLUS	Additionen
MINUS	Subtraktionen
MAL	Multiplikationen
DURCH	Divisionen

Die Operationen sind hier in aufsteigender Bindung aufgelistet, d.h., dass beispielsweise eine Division eine höhere Bindung als eine Addition besitzt. (Umgangssprachlich "Punkt vor Strich")

1.9.2 Operanden

Zulässige Operanden für aritmetische Operationen sind:

- ganzzahl-Literale
- Variablen vom Typ ganzzahl
- Konstanten
- Funktionen (bzw. deren Rückgabewerte) mit dem Rückgabetyp ganzzahl
- weitere Operationen

Aus dem letzten Eintrag dieser Liste ergibt sich eine Rekursion, die eine beliebige Länge von arithmetischen Ausdrücken ermöglicht.

1.9.3 Klammerung

Operationen dürfen Klammern mit beliebiger Verschachtelungstiefe enthalten. Geklammerte Terme besitzen die höchst mögliche Bindung, d.h. höher als eine Division.

1.10 Aussagenlogik

Logische Ausdrücke werden wie in C intern als "ganzzahl" gespeichert und besitzen keinen eigenen Datentyp. Dabei repräsentiert der Wert 0 den Wahrheitswert *falsch*, alle anderen Werte gelten als *wahr*. Wie auch arithmetische Ausdrücke, dürfen logische Ausdrücke eine beliebige Länge besitzen und eine beliebig tiefe Klammerschachtelung besitzen.
Logische Ausdrücke werden verwendet, um Bedingungen auszudrücken, also in wenn-sonst-Aussagen sowie in Schleifen.
Folgende Operationen stehen dafür in dieser Priorität zur Verfügung:

Operation	Operator	Ergebnis
Konjunktion	UND	Wahr, wenn beide Operanden wahr sind. Sonst falsch.
Disjunktion	ODER	Wahr, wenn ein oder beide Operanden wahr sind. Sonst falsch.
Negation	NICHT	Das Gegenteil das negierten Term.

Desweiteren stehen folgende Vergleichsoperationen zur Verfügung:

Operator	Vergleich
KLEINER	kleiner als
KLEINERGLEICH	kleiner als oder gleich
GROESSER	größer als
GROESSERGLEICH	größer als oder gleich
GLEICH	gleich

Wenn ein Vergleich eine wahre Aussage ist, ist das Ergebnis des Vergleich *wahr*. Wenn ein Vergleich eine falsche Aussage ist, ist das Ergebnis *falsch*.

1.11 Bedingte Verzweigungen

Eine bedingte Verzweigung ist ein Programmabschnitt der abhängig von einer Bedingung ausgeführt oder ignoriert wird. Es ist ebenfalls möglich eine Alternative anzugeben, die nur ausgeführt wird, falls die Bedingung nicht erfüllt wird.
Eine bedingte Verzweigung beginnt mit dem Schlüsselwort *wenn* gefolgt von einer Bedingung umschlossen von runden Klammern. Dabei muss die Bedingung ein logischer Ausdruck sein. Wenn die Bedingung zu *wahr* evaluiert wird, werden die Aussagen, die umgeben von geschweiften Klammern auf die Bedingung folgen, ausgeführt. Anschlißend müssen das Schlüsselwort *sonst* sowie weitere Aussagen umschlossen von geschweiften Klammern folgen. Diese Anweisungen werden ausgeführt falls die Bedingung zu *falsch* evaluiert wird. Der sonst-Teil darf nicht ausgelassen werden.

Beispiel:

```
ganzzahl x;
ganzzahl y;

x ISTGLEICH 42;
y ISTGLEICH 3;

wenn(x KLEINER y) {
  druckzeile("Hurra!");
} sonst {
  druckzeile(":-(");
}
```

1.12 Schleifen

Eine Schleife beginnt mit dem Schlüsselwort *waehrend*, wird von einer Bedingung in runden Klammern gefolgt und schließt mit einem Schleifenrumpf ab. Der Schleifenrumpf wiederrum wird mit geschweiften Klammern geöffnet und abgeschlossen.
Sowohl beim ersten Aufruf der Schleife als nach jedem Schleifendurchlauf wird überprüft, ob die gegebenene Bedingung wahr oder falsch ist. Ist die Bedingung wahr, werden die Anweisungen im Schleifenrumpf ausgeführt. Falls die Bedingung falsch ist, werden die Anweisungen nicht ausgeührt.
Beispiel (gibt die Summe der Zahlen von 1 bis 10 aus):

```
ganzzahl i;
ganzzahl x;
```

```
i ISTGLEICH 0;
x ISTGLEICH 0;

wahrend(i KLEINERGLEICH 10) {
        i ISTGLEICH i PLUS 1;
        x ISTGLEICH x PLUS i;
}
druckzeile(x);
```

1.13 Reservierte Schlüsselwörter und Symbole

Schlüsselwort	Bedeutung
ganzzahl	Deklariert eine Variable vom Typ ganzzahl
kette	Deklariert eine Variable vom Typ kette
konstante ganzzahl	Deklariert eine Konstante
druckzeile	Aufruf der Ausgabefunktion
gebzueruck	Hinter "gebzueruck" folgt der Rückgabewert einer Funktion
wenn	Beginn einer wenn(-sonst)-Aussage
sonst	Beginn des sonst-Teil einer wenn-sonst-Aussage
waehrend	Beginn einer waehrend-Schleife

Symbol	Bedeutung
ISTGLEICH	Zuweisungsoperator
PLUS	Operator für Additionen
MINUS	Operator für Subtraktionen
MAL	Operator für Multiplikationen
DURCH	Operator für Divisionen
KLEINER	Vergleichsoperator für kleiner als
KLEINERGLEICH	Vergleichsoperator für kleiner gleich
GROESSER	Vergleichsoperator für größer als
GROESSERGLEICH	Vergleichsoperator für größer gleich
GLEICH	Operator um zwei Werte auf Gleichheit zu überprüfen
UND	logisches Und
ODER	logisches Oder (inklusiv)
NICHT	logisches Nicht

2 Aufruf des Deuersprech-Kompilierers

Der Deuersprech-Kompilierer benötigt zur Ausführung die Java Runtime Environment.
Um eine Textdatei mit Anweisungen in Deuersprech zu übersetzen, muss folgender Aufruf auf der Kommandozeilenebene erfolgen:

```
java -jar dsk.jar quellkodierung.ds
```

Die Ausgabedatei deuersprechAusstelle.class kann nun mit Hilfe der JRE ausgeführt werden.

```
java deuersprechAusstelle
```