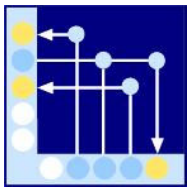
Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik



Compilerbau-Praktikum

Lehrveranstaltung im Sommersemester 2021
Prof. Dr. habil. Christian Heinlein

4. Aufgabenblatt (3. Mai 2021)

Aufgabe 5: Auswertung von Ausdrücken

Teilaufgabe 5.a)

Implementieren Sie die Auswertung von Ausdrücken der bis jetzt entwickelten Minisprache, indem Sie jedem Operator eine geeignet implementierte Auswertungsfunktion zuordnen! Beachten Sie hierzu Kapitel 4 der Dokumentation und berücksichtigen Sie insbesondere auch unnatürliche Werte!

Momentan besitzen alle Ausdrücke implizit den Typ `int`.

Die Operanden eines Ausdrucks sollen immer von links nach rechts ausgewertet werden.

Division durch 0 und Fakultät einer negativen Zahl liefern nil.

Die Potenz `x ^ y` ist für `y > 0` wie üblich definiert. Für `y = 0` ist das Ergebnis immer 1 (auch für `x = 0`). Für `y < 0` ist `x ^ y` gleichbedeutend mit `1 / x ^ -y` (was für `x = 0` nil und für die meisten anderen Werte von `x` 0 liefert).

`print` *`x`* `width` *`w`* gibt den Wert *`x`* dezimal mit Mindestbreite *`w`* auf der Standardausgabe aus, d. h. wenn die Dezimaldarstellung von *`x`* (inklusive eines eventuellen Vorzeichens) aus weniger als *`w`* Zeichen besteht, werden am Anfang entsprechend viele Leerzeichen hinzugefügt.
Wenn *`x`* ein unnatürlicher Wert ist, ist seine Dezimaldarstellung leer.
Wenn `width` *`w`* fehlt oder *`w`* ein unnatürlicher Wert ist, wird Mindestbreite 0 verwendet.
In jedem Fall wird ein abschließender Zeilentrenner ausgegeben.

Die Nacheinanderausführung (Semikolon) wertet ihre Operanden von links nach rechts aus und liefert den Wert des zweiten Operanden.

Vergleiche liefern anstelle des Booleschen Werts `true` bzw. `false` einen eindeutigen synthetischen Wert (d. h. immer denselben) bzw. nil.
Ein verketteter Vergleich $x_0\ op_1\ x_1\ op_2\ x_2 \ldots x_{n-1}\ op_n\ x_n$ wie z. B. `a < b > c` ist gleichbedeutend mit einer Konjunktion (siehe Teilaufgabe 5.b) $x_0\ op_1\ x_1$ `&` $x_1\ op_2\ x_2$ `&` $\ldots$ `&` $x_{n-1}\ op_n\ x_n$, deren Auswertung abgebrochen wird, sobald das Ergebnis feststeht. Allerdings wird jeder Operand höchstens einmal ausgewertet.

Teilaufgabe 5.b)

Erweitern Sie die bis jetzt entwickelte Minisprache dann um folgende Operatoren mit zugehörigen Auswertungsfunktionen:

- Logische Verknüpfungen:
  Negation (Präfixoperator `~`), Konjunktion (Infixoperator `&`) und Disjunktion (Infixoperator `|`).
  Disjunktion bindet schwächer als Konjunktion, Konjunktion bindet schwächer als Negation, Negation bindet schwächer als Vergleiche.
  Der zweite Operand einer Konjunktion oder Disjunktion wird nur ausgewertet, wenn dies zur Ermittlung des Ergebnisses notwendig ist.
  Auch diese Operatoren liefern anstelle Boolescher Werte entweder denselben synthetischen Wert wie Vergleiche oder nil. Umgekehrt wird jeder echte Wert eines Operanden (einschließlich 0) als `true` und der Wert nil als `false` interpretiert.

- Mehrfache Fallunterscheidung mit Signatur

  ```
  if ... then ... { elseif ... then ... } [ else ... ] end
  ```

  Auch hier werden nur diejenigen Operanden ausgewertet, für die dies notwendig ist.

  Als Ergebnis erhält man:

  - entweder den Wert des ersten `then`-Teils, dessen zugehörige `if`- bzw. `elseif`-Bedingung erfüllt ist (d. h. einen echten Wert liefert)
  - oder (wenn keine dieser Bedingungen erfüllt ist) den Wert des `else`-Teils
  - oder (wenn es keinen `else`-Teil gibt) nil.

- Allgemeine Schleife mit Signatur

  ```
  (while|until|do) ... { (while|until|do) ... } end
  ```

  die bei jeder Iteration ihre Operanden nacheinander auswertet, bis die Auswertung eines `while`-Teils nil oder die Auswertung eines `until`-Teils einen echten Wert liefert.
  Als Ergebnis wird die Anzahl der vollständig ausgeführten Iterationen geliefert.

  Zum Beispiel:

`while ... do ... end`	Kopfgesteuert mit Fortsetzungsbedingung
`do ... until ... end`	Fußgesteuert mit Abbruchbedingung
`do ... while ... do ... until ... do ... end`	Mehrere Forts.- und Abbr.-Bedingungen
`do ... end`	Endlosschleife

- Eingabe `read int` (zwei getrennte Namen!), die eine Zeile von der Standardeingabe liest, diese in eine ganze Zahl umwandelt und zurückliefert.
  Wenn keine korrekte ganze Zahl (eine oder mehrere Dezimalziffern mit optionalem Minuszeichen am Anfang, umgeben von optionalem Zwischenraum) eingegeben wird, ist das Ergebnis nil.

- Zuweisung und Abfrage von Variablen bzw. „Variablenscharen“:
  `var` `!` *`val`* weist der Variablen *`var`* den Wert *`val`* zu und liefert ihn zurück.
  `?`*`var`* liefert den aktuellen Wert der Variablen *`var`*. (Das Fragezeichen ist optional.)
  `var` muss eine Folge von Buchstaben und Ziffern sein, die mit einem Buchstaben beginnt, optional gefolgt von eckigen Klammern, die einen oder mehrere Indizes (getrennt durch Kommas) enthalten, z. B. `x`, `y1[1]` oder `ABC[2, 3, 4]`.
  Der Wert *`val`* und die Indizes können beliebige Ausdrücke sein, allerdings soll die Zuweisung stärker binden als die Nacheinanderausführung.

Variablen werden momentan nicht explizit deklariert, sondern entstehen automatisch bei ihrer ersten Verwendung.
Jede Variable kann gleichzeitig sowohl einen einzelnen Wert speichern (z. B. `x`) als auch die Werte von Feldern beliebiger Dimension (z. B. `x[1]` und `x[2, 3, 4]`) mit prinzipiell beliebiger, unbekannter Größe in jeder Dimension, d. h. die Werte der Indizes können beliebige ganze Zahlen sein.
Wenn einer Variablen *`var`* noch kein Wert zugewiesen wurde, liefert `?`*`var`* nil.
Wenn einer der Indizes ein unnatürlicher Wert ist, liefert `?`*`var`* ebenfalls nil und *`var`* `!` *`val`* wertet seine Operanden aus und liefert den Wert *`val`* zurück, führt aber keine Zuweisung aus.

Teilaufgabe 5.c)

Schreiben und testen Sie mit der entwickelten Minisprache folgende Anwendungsprogramme:

- Einlesen zweier positiver `int`-Werte und Ausgabe ihres größten gemeinsamen Teilers.
- Einlesen beliebig vieler `int`-Werte und sortierte Ausgabe der Werte.