Inhaltsverzeichnis:

1. Ausdrucksbaum

1.1. Lösungsidee:

Die Strukturen zum Aufbau des Ausdrucksbaums wurden Großteils während der Übung implementiert. Ein Ausdrucksbaum besteht aus einem Wurzelknoten, welcher entweder ein Operator-Knoten, ein Variablen-Knoten oder ein Wert-Knoten sein kann. Ausdrucksbäume werden evaluiert indem für den Wurzelknoten die Knoten-Evaluierungsmethode (siehe nächste Seite) aufgerufen wird, das Ergebnis wird zurückgegeben. Ausdrucksbäume können entweder über den überladenen „<<“ Operator oder über eine der Print-Methoden ausgegeben werden. Der überladene Operator gibt den Baum in in-order Notation aus. Die weiteren Print-Methoden sind:

- Baum in **pre-order** Notation ausgeben über Knoten-Methode
- Baum in **post-order** Notation ausgeben über Knoten-Methode
- **print_2d**: Der Baum wird seitlich ausgegeben, ganz links der Hauptknoten, oben die rechten Knoten und unten die linken Knoten. Alle Werte werden nur einmal ausgegeben. Die Knoten sind nach Tiefe versetzt. Diese Funktion ruft eine rekursive Hilfsfunktion auf. Wenn der aktuelle Knoten (anfangs der Hauptknoten) existiert, dann wird die rekursive Funktion zuerst für die rechten Knoten (oben) aufgerufen. Bei jedem Aufruf steigt der Wert, der die Tiefe misst, um 1. Dann wird der Inhalt des Knotens ausgegeben und vor dem Inhalt wird ein Abstand ausgegeben, der mit der Tiefe multipliziert ist, damit korrekte Ebenen entstehen. Danach wird die Funktion für die linken Knoten aufgerufen.

- **print_2d_upright**: Der Baum wird „normal“ ausgegeben. Alle Werte werden nur einmal ausgegeben. Zuerst wird durch eine rekursive Funktion die maximale Tiefe, also die Anzahl der Ebenen, des Baums ermittelt (nach der Vorgehensweise bei print_2d). Danach wird eine leere Matrix erstellt. Die Matrix hat so viele Zeilen wie es Ebenen in dem Baum gibt. Die Matrix wird gefüllt mit dem Knoten des Baumes nach dem Schema, dass in jeder Zeile alle Knoten (auch leere Knoten bzw. an Orten, wo potenziell Knoten sein könnten) erfasst werden. Dies geschieht über eine rekursive Hilfsfunktion. Der Rekursionsboden ist, dass die aktuelle Tiefe kleiner als die maximale Tiefe sein muss. Schema: 1. Zeile: Hauptknoten, 2. Zeile: Beide Knoten des Hauptknotens (auch wenn sie nicht existieren), 3. Zeile: insgesamt 4 Knoten, usw. Diese Matrix dient als Basis für das Ausgeben eines aufrechten Baums. Nun gilt es noch die Abstände, die vor den Knoten ausgegeben werden, zu berechnen. Es gibt 2 Arten von Abständen: Der Abstand zwischen den Knoten einer Zeile und der Abstand vor dem ersten Knoten einer Zeile und. Nach einer zeichnerischen Analyse ließ sich feststellen, dass sich der Abstand zwischen den Knoten einer Zeile von unten nach oben immer verdoppelt bei einem Zeilensprung. Der Startwert, der für einen besseren optischen Abstand sorgt (4), wurde durch Trial-and-Error gewählt.
  Beispiel: Abstand zwischen Knoten letzter Zeile also: 4 Einheiten, darüber 8 Einheiten, darüber 16 Einheiten, …
  Der zweite Abstand, also der Abstand des ersten Knotens jeder Zeile zum Rand, stellte sich nach zeichnerischen und rechnerischen Versuchen als der Abstand zwischen den Knoten aus der darunterliegenden Zeile heraus. Beispiel: Letzte Zeile: 2 Einheiten, darüber 4 Einheiten,…
  Nun muss also die Matrix mit den korrekten Abständen ausgegeben werden. Dafür wird durch die komplette Matrix und die erstellte Liste mit den Abständen iteriert und der Wert der Knoten ausgegeben. Anstatt von leeren Knoten wird einfach ein leerer Abstand ausgegeben, damit die Struktur des Baums stimmt.

Eine Namensliste wird dazu benötigt Ausdrucksbäume mit einem String als Schlüsselwort abzulegen. Wenn noch kein Eintrag unter einem bestimmten Namen besteht, wird der Ausdrucksbaum unter diesem Namen dort hinterlegt. Falls schon ein Eintrag unter einem bestimmten Namen besteht, wird dieser gelöscht und ein neuer Eintrag mit dem gewünschten Ausdrucksbaum angelegt. Wenn bei der Namensliste eine Anfrage über einen bestimmten Variablennamen eingeht, wird der Ausdrucksbaum, der für diesen Namen gespeichert ist, zurückgegeben. Bei nicht definierten Variablennamen handelt es sich dabei um einen Nullptr. Namenslisten können ausgegeben werden: Es wird der Variablenname gefolgt von dem zugehörigen Ausdrucksbaum in in-order Notation ausgegeben.

Es gibt drei verschiedene Arten von Knoten: Wert-Knoten, Operator-Knoten und Variablen-Knoten. Alle besitzen Methoden, um ihren linken und rechten Knoten zu setzen, beziehungsweise diese zurückzugeben. Von einem Knoten aus als Startpunkt können Ausdrucksbäume in den drei erwähnten Notationen ausgegeben werden: Bei der pre-order Notation wird der Inhalt des Knotens ausgegeben, bevor die Methode rekursiv für den rechten und linken Knoten aufgerufen wird, bei der in-order Notation passiert dies zwischen dem rekursiven Aufruf für den linken und rechten Knoten, bei der post-order Notation danach.

Alle dieser drei Knotenarten können evaluiert und ausgegeben werden:

- **Wert-Knoten**: Bei der Evaluierung wird der Inhalt zurückgegeben, bei der Ausgabe wird der Inhalt auf einem Stream ausgegeben.
- **Operator-Knoten**: Der Inhalt kann entweder ein Plus-, Minus-, Multiplikations-, Divisions- oder Potenz-Operator. Bei der Evaluierung werden der linke und der rechte Knoten evaluiert und die entsprechende Operation auf die Ergebnisse angewandt. Bei dem Divisions-Operator wird ein Fehler geworfen, wenn der rechte Knoten zu 0 evaluiert. Wenn es sich beim Inhalt des Knotens um keinen dieser fünf Operatoren handelt, wird auch ein Fehler ausgelöst. Bei der Ausgabe des Knotens wird der entsprechende Operator ausgegeben.
- **Variablen-Knoten**: Der Inhalt ist ein String. Bei der Evaluierung wird überprüft, ob ein Ausdrucksbaum unter diesem Namen in der Namensliste existiert. Dieser wird dann evaluiert. Falls kein Baum unter diesem Namen existiert, wird ein Fehler geworfen.

Mittels Scanner und Parser sollen Ausdrucksbäume zur Evaluierung der Ausdrücke aufgebaut werden.

Der Parser enthält Funktionen, welche prüfen, ob es sich um den Beginn einen gültigen Ausdruck folgender in EBNF gegebenen Grammatik handelt:

```
Programm  = { Ausgabe | Zuweisung } .
Ausgabe   = „print“ „(“ Ausdruck „)“ „;“ .
Zuweisung = „set“ „(“ Identifier „,“ Ausdruck „)“ „;“ .

Ausdruck  = Term { AddOp Term } .
Term      = Faktor { MultOp Faktor } .
Faktor    = [ AddOp ] UFaktor .
```

```
UFaktor   = Monom | KAusdruck .
Monom     = WMonom [ Exponent ] .
KAusdruck = „(“ Ausdruck „)“ .
WMonom    = Identifier | Real .
Exponent  = „^“ [ AddOp ] Real .
AddOp     = „+“ | „-“ .
MultOp    = „*“ | „/“ .
```

Ob ein bestimmter Ausdruck folgt, wird überprüft, indem das erste Zeichen/der erste Ausdruck eines Ausdrucks mit den Grammatikregeln der einzelnen Ausdrücke verglichen wird:

- **Programm**: Muss mit Output oder Assignment beginnen
- **Output**: Muss mit Keyword print beginnen
- **Assignment**: Muss mit Keyword set beginnen
- **Expression:** Muss Mit Term beginnen
- **Term:** Muss mit Faktor beginnen
- **Faktor:** Muss mit AddOp oder UFaktor beginnen
- **UFaktor:** Muss mit Monom oder PExpression beginnen
- **Monom:** Muss mit WMonom beginnen
- **PExpression:** Muss mit öffnender Klammer beginnen
- **WMonom:** Muss mit Identifier (String) oder Zahl beginnen
- **Exponent:** Muss mit ^ beginnen
- **AddOp:** Muss mit + oder – beginnen
- **MultOp:** Muss mit * oder / beginnen

Im Konstruktor werden das „set“ und „print“-Keyword registriert und eine neue Namensliste angelegt. Im Destruktor werden der aktuelle Baum und die Namensliste gelöscht. Es sind Methoden vorhanden, um die gesamte Namenliste oder einen Baum aus der Namensliste (in beliebiger Notation) auszugeben.

Die Funktionen zum Parsen der Ausdrücke sind folgendermaßen aufgebaut:

- **Generelle Parse-Funktion**: Kann entweder mit einem Eingabestream oder einem String aufgerufen werden, welche dem Scanner „übergeben“ werden. Danach wird die entsprechende Aktion (set/print) durch die „Programm parsen“-Funktion ausgeführt und der Ausdruck aus dem Scanner „entfernt“. Wenn der Ausdruck noch nicht zu Ende ist wird ein Fehler geworfen.
- **Programm parsen**: Wenn der Ausdruck nicht wie ein Programm beginnt, wird ein Fehler geworfen (wird für jede Parse-Funktion durchgeführt), ansonsten wird entsprechend des Keywords entweder die „Output parsen“ oder „Assignment parsen“ Funktion aufgerufen.
- **Output parsen**: Nach dem Keyword muss eine öffnende Klammer folgen, danach wird ein Ausdrucksbaum durch die Funktion „Expression parsen“ erstellt. Das nächste Symbol muss eine schließende Klammer und danach ein Strichpunkt sein. Anschließend wird der Ausdrucksbaum evaluiert und das Ergebnis in der Konsole ausgegeben. Danach wird der aktuelle Ausdrucksbaum zurückgesetzt.

- **Assignment parsen:** Nach dem Keyword muss eine öffnende Klammer folgen, danach wird das Keyword für den Ausdrucksbaum in der Namensliste ermittelt (nächster Ausdruck), nach einem Beistrich wird dann ein Ausdrucksbaum durch die Funktion „Expression parsen“ erstellt. Das nächste Symbol muss eine schließende Klammer und danach ein Strichpunkt sein. Der Ausdrucksbaum wird unter dem Keyword in der Namensliste registriert und der aktuelle Ausdrucksbaum zurückgesetzt.
- **Expression parsen**: Der linke Knoten wird über die „Term parsen“ Funktion ermittelt. Wenn der nächste Ausdruck im Scanner eine AddOp ist, dann wird der Hauptknoten durch die „AddOp parsen“ Funktion ermittelt, der zuvor ermittelte Knoten als linker Knoten festgelegt und der rechte Knoten mittels der „Expression parsen“ Funktion ermittelt, da noch weitere Expressions folgen können. Es wird entweder nur der linke (keine folgende AddOp) oder der Hauptknoten zurückgegeben.
- **Term parsen**: Der linke Knoten wird über die „Faktor parsen“ Funktion ermittelt. Wenn der nächste Ausdruck im Scanner eine MultOp ist, dann wird der Hauptknoten durch die „MultOp parsen“ Funktion ermittelt, der zuvor ermittelte Knoten als linker Knoten festgelegt und der rechte Knoten mittels der „Term parsen“ Funktion ermittelt, da noch weitere Terme folgen können. Es wird entweder nur der linke (keine folgende MultOp) oder der Hauptknoten zurückgegeben.
- **Faktor parsen**: Falls eine AddOp der nächste Ausdruck ist, wird diese durch die „AddOp parsen“ Funktion als Hauptknoten und 0 als linker Knoten festgelegt. Die Funktion „UFaktor parsen“ wird aufgerufen und entweder nur dieser Knoten zurückgegeben (kein AddOp) oder als rechter Knoten festgelegt (AddOp) und der Hauptknoten zurückgegeben.
- **UFaktor parsen**: Wenn es sich beim nächsten Ausdruck um ein Monom handelt wird der zurückzugebende Knoten über die „Monom parsen“ Methode bestimmt, ansonsten wird wenn es sich um eine PExpression handelt der zurückzugebende Knoten über die „PExpression parsen“ Methode bestimmt, ansonsten wird ein Fehler geworfen.
- **Monom parsen**: Der linke Knoten wird mittels der „WMonom parsen“ Methode bestimmt und zurückgegeben falls kein Exponent folgt. Ansonsten wir der Hauptknoten (und rechte Knoten) über die „Exponent parsen“ Methode ermittelt, der zuvor ermittelte Knoten als linker Knoten festgelegt und zurückgegeben.
- **PExpression parsen**: Es ist wichtig, dass im Scanner die öffnende und schließende Klammer vor beziehungsweise nach dem Ausdruck vorkommen müssen. Der Knoten wird mittels der „Expression parsen“ Methode ermittelt und zurückgegeben.
- **WMonom parsen**: Wenn es sich beim nächsten Ausdruck um einen Indentifier handelt, wird der zurückzugebende Knoten bestimmt indem der Identifier vom Scanner geholt wird, ansonsten wird wenn es sich um eine Zahl handelt der zurückzugebende Knoten bestimmt indem die Zahl vom Scanner geholt wird, ansonsten wird ein Fehler geworfen.
- **Exponent parsen**: Wenn das nächste Symbol kein „^“ ist, dann wird ein Fehler geworfen, ansonsten wird der Hauptknoten als EXP-Operator-Knoten festgelegt. Wenn der nächste Ausdruck keine Zahl ist wird ein Fehler geworfen, ansonsten wird diese Zahl vom Scanner geholt und als rechter Knoten festgelegt.

- **AddOp parsen**: Wenn es sich beim nächsten Ausdruck um ein „+“ handelt, dann wird ein ADD-Operator-Knoten zurückgegeben, wenn es sich um ein „-“ handelt, dann wird ein SUB-Operator-Knoten zurückgegeben und ansonsten wird ein Fehler geworfen.
- **MultOp parsen**: Wenn es sich beim nächsten Ausdruck um ein „*“ handelt, dann wird ein MUL-Operator-Knoten zurückgegeben, wenn es sich um ein „/“ handelt, dann wird ein DIV-Operator-Knoten zurückgegeben und ansonsten wird ein Fehler geworfen.

Durch den Aufbau des Ausdrucksbaums gelten sowohl Klammerungsregeln als auch Punkt vor Strich-Rechnung.

1.2. Testfälle:

Microsoft Visual Studio Debug Console

```
TESTFALL 1: Kein Strichpunkt
Evaluating expr 'print(5+4)' ...
Expected: EXCEPTION
Actual result/exception: Expected 'semicolon' but have {{end of file,ts,7}}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 2: Kein Strichpunkt
Evaluating expr 'set(a,5+4)' ...
Expected: EXCEPTION
Actual result/exception: Expected 'semicolon' but have {{end of file,ts,7}}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 3: Keine Klammern
Evaluating expr 'print 5+4' ...
Expected: EXCEPTION
Actual result/exception: Expected 'left parenthesis' but have {{integer,tc,2},'5'}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 4: Keine Klammern
Evaluating expr 'set a,5+4)' ...
Expected: EXCEPTION
Actual result/exception: Expected 'left parenthesis' but have {{identifier,tc,1},'a'}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 5: Keine Klammern
Evaluating expr 'set (a,5+4]' ...
Expected: EXCEPTION
Actual result/exception: Unknown character ']' encountered.
------- TEST ERFOLGREICH -------
```

```
TESTFALL 6: Keine Klammern
Evaluating expr 'set (a,5+4' ...
Expected: EXCEPTION
Actual result/exception: Expected 'right parenthesis' but have {{end of file,ts,7}}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 7: Keine Klammern
Evaluating expr 'set (a,5+4' ...
Expected: EXCEPTION
Actual result/exception: Expected 'right parenthesis' but have {{end of file,ts,7}}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 8: Keine Beistrich
Evaluating expr 'set (a 5+4);' ...
Expected: EXCEPTION
Actual result/exception: Expected 'comma' but have {{integer,tc,2},'5'}.
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 9: Kein Keyword
Evaluating expr '(a,5+4);' ...
Expected: EXCEPTION
Actual result/exception: ERROR parsing 'Programm'
------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------

TESTFALL 10: Kein Keyword
Evaluating expr '(5+4);' ...
Expected: EXCEPTION
Actual result/exception: ERROR parsing 'Programm'
------- TEST ERFOLGREICH -------
```

```
TESTFALL 11: Ungueltige Expression

Evaluating expr 'print(*5+4);' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Expression'

------- TEST ERFOLGREICH -------

-------------------------------------------------------------------------------

TESTFALL 12: Ungueltige Expression

Evaluating expr 'print(5+4+);' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Expression'

------- TEST ERFOLGREICH -------

-------------------------------------------------------------------------------

TESTFALL 13: Ungueltige Expression

Evaluating expr 'print(5*45+);' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Expression'

------- TEST ERFOLGREICH -------

-------------------------------------------------------------------------------

TESTFALL 14: Leerer Input

Evaluating expr '' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Programm'

------- TEST ERFOLGREICH -------

-------------------------------------------------------------------------------

TESTFALL 15: Leerer Input

Evaluating expr ' ' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Programm'

------- TEST ERFOLGREICH -------
```

```
TESTFALL 16: Ungueltiger Input

Evaluating expr 'print;' ...

Expected: EXCEPTION

Actual result/exception: Expected 'left parenthesis' but have {{semicolon,ts,15}}.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------


TESTFALL 17: Ungueltiger Input

Evaluating expr 'print();' ...

Expected: EXCEPTION

Actual result/exception: ERROR parsing 'Expression'

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------


TESTFALL 18: Ungueltiger Input

Evaluating expr 'set;' ...

Expected: EXCEPTION

Actual result/exception: Expected 'left parenthesis' but have {{semicolon,ts,15}}.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------


TESTFALL 19: Ungueltiger Input

Evaluating expr 'set();' ...

Expected: EXCEPTION

Actual result/exception: Expected 'comma' but have {{semicolon,ts,15}}.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------


TESTFALL 20: Undefinierte Variable

Evaluating expr 'print(a);' ...

Expected: EXCEPTION

Actual result/exception: ERROR: Variable a was not defined.

------- TEST ERFOLGREICH -------
```

```
TESTFALL 21: Division by 0

Evaluating expr 'print(10/0);' ...

Expected: EXCEPTION

Actual result/exception: ERROR: Division by Zero.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------------

TESTFALL 22: Division by 0

Evaluating expr 'print(10/(5-5));' ...

Expected: EXCEPTION

Actual result/exception: ERROR: Division by Zero.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------------

TESTFALL 23: Division by 0

Evaluating expr 'print(10/((5-2)*0));' ...

Expected: EXCEPTION

Actual result/exception: ERROR: Division by Zero.

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------------

TESTFALL 24: Einzelne Zahl

Evaluating input 'print(5+4);' ...

Expected result: 9

Actual result: 9

------- TEST ERFOLGREICH -------


--------------------------------------------------------------------------------------------------

TESTFALL 25: Addition

Evaluating input 'print(5+4);' ...

Expected result: 9

Actual result: 9

------- TEST ERFOLGREICH -------
```

```
TESTFALL 26: Subtraktion
Evaluating input 'print(51.63-4.42);' ...
Expected result: 47.21
Actual result: 47.21

------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------------------------

TESTFALL 27: Multiplikation
Evaluating input 'print(5*2.2);' ...
Expected result: 11
Actual result: 11

------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------------------------

TESTFALL 28: Division
Evaluating input 'print(5/4);' ...
Expected result: 1.25
Actual result: 1.25

------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------------------------

TESTFALL 29: Exponent
Evaluating input 'print(2^2);' ...
Expected result: 4
Actual result: 4

------- TEST ERFOLGREICH -------

--------------------------------------------------------------------------------------------------

TESTFALL 30: Komplexer Ausdruck
Evaluating input 'print(5/4*5+(8-21.3)-5+2^2);' ...
Expected result: -22.05
Actual result: -22.05

------- TEST ERFOLGREICH -------
```

```
TESTFALL 31: Komplexer Ausdruck

Evaluating input 'print(5*(4+1));' ...

Expected result: 25

Actual result: 25


------- TEST ERFOLGREICH -------


------------------------------------------------------------------------


TESTFALL 32: Komplexer Ausdruck

Evaluating input 'print(2*4+5*6+7+9+10+1-1-4/2);' ...

Expected result: 66

Actual result: 66


------- TEST ERFOLGREICH -------
```

```
TESTFALL 33: Musterbeispiel

Evaluating input 'set(a,1);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'set(pi,3.1415);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'set(k,-(a+3)*pi^0.5);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'print(k);' ...

Expected result: -7.08971

Actual result: -7.08971

------- TEST ERFOLGREICH -------

Evaluating input 'print(3);' ...

Expected result: 3

Actual result: 3

------- TEST ERFOLGREICH -------

Evaluating input 'print(4/(2*(3-1)));' ...

Expected result: 1

Actual result: 1

------- TEST ERFOLGREICH -------

Evaluating input 'set(x,15);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'print(x*2);' ...

Expected result: 30

Actual result: 30

------- TEST ERFOLGREICH -------
```

```
Evaluating input 'set(x,y^2-4);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(y, 6);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(x);' ...
Expected result: 32
Actual result: 32

------- TEST ERFOLGREICH -------

Evaluating input 'print(y);' ...
Expected result: 6
Actual result: 6

------- TEST ERFOLGREICH -------

Evaluating input 'print(x / 2)' ...
Expected result: 16
Actual result: Exception: Expected 'semicolon' but have {{end of file,ts,7}}.
-X-X-X- TEST FEHLGESCHLAGEN -X-X-X-

Evaluating input 'set(y,5);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(x);' ...
Expected result: 21
Actual result: 21

------- TEST ERFOLGREICH -------

Evaluating input 'print(y);' ...
Expected result: 5
Actual result: 5

------- TEST ERFOLGREICH -------

Evaluating input 'print(x/2);' ...
Expected result: 10.5
Actual result: 10.5

------- TEST ERFOLGREICH -------
```

```
TESTFALL 34: Verschachtelte Variablen
Evaluating input 'set(myvar,5+3);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(myvar2,myvar+2);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(myvar3, myvar * myvar2);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar);' ...
Expected result: 8
Actual result: 8

------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar2);' ...
Expected result: 10
Actual result: 10

------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar3);' ...
Expected result: 80
Actual result: 80

------- TEST ERFOLGREICH -------
```

```
TESTFALL 35: Redefinition von Variablen
Evaluating input 'set(myvar,5+3);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(myvar2,myvar+2);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar);' ...
Expected result: 8
Actual result: 8

------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar2);' ...
Expected result: 10
Actual result: 10

------- TEST ERFOLGREICH -------

Evaluating input 'set(myvar,-17.3); ' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar); ' ...
Expected result: -17.3
Actual result: -17.3

------- TEST ERFOLGREICH -------

Evaluating input 'print(myvar2); ' ...
Expected result: -15.3
Actual result: -15.3

------- TEST ERFOLGREICH -------
```

```
TESTFALL 36: Undefinierte Variablen
Evaluating input 'set(a,5+3);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(b,c+2);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(b);' ...
Expected result: EXCEPTION
Actual result: Exception: ERROR: Variable c was not defined.
------- TEST ERFOLGREICH -------

-----------------------------------------------------------------------------------------------

TESTFALL 37: Undefinierte Variablen
Evaluating input 'set(a,5+3);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'set(b,a+2);' ...
Expected result:
Actual result:
------- TEST ERFOLGREICH -------

Evaluating input 'print(b);' ...
Expected result: 10
Actual result: 10

------- TEST ERFOLGREICH -------

Evaluating input 'print(c);' ...
Expected result: EXCEPTION
Actual result: Exception: ERROR: Variable c was not defined.
------- TEST ERFOLGREICH -------
```

```
TESTFALL 38: Namelist ausgeben

Evaluating input 'set(a,5+3);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'set(c,5-17+9*51^2-(8-7.4+8-9));' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'set(b,a + 2); ' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------

Evaluating input 'print(c); ' ...

Expected result: -23419.4

Actual result: -23419.4

------- TEST ERFOLGREICH -------

Evaluating input 'Get Namelist' ...

Expected result: PRINTNAMELIST

a: 5 + 3
b: a + 2
c: 5 - 17 + 9 * 51 ^ 2 - 8 - 7.4 + 8 - 9

------- TEST ERFOLGREICH -------
```

```
TESTFALL 39: Print-Methoden ausgeben

Evaluating input 'set(a,4*(5+3)+5+2+1*4);' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------


Evaluating input 'a' ...

Expected result: PRINTMETHODS

4 * 5 + 3 + 5 + 2 + 1 * 4
post-order: 4 5 3 + * 5 2 1 4 * + + +
pre-order: + * 4 + 5 3 + 5 + 2 * 1 4


                    4
               *
                    1
          +
               2
     +
          5
+
               3
          +
               5
     *
          4




                         +
             *                        +
       4             +             5             +
                  5       3                  2       *
                                                  1   4
------- TEST ERFOLGREICH -------
```

```
TESTFALL 40: Print-Methoden ausgeben

Evaluating input 'set(a,4*(5+3));' ...

Expected result:

Actual result:

------- TEST ERFOLGREICH -------


Evaluating input 'a' ...

Expected result: PRINTMETHODS

4 * 5 + 3
post-order: 4 5 3 + *
pre-order: * 4 + 5 3


          3
     +
          5
*
     4



       *
   4       +
         5   3
------- TEST ERFOLGREICH -------
```

Da die Strings in den Tests in Istreams umgewandelt werden, müssen Tests mit Istreams nicht explizit durchgeführt werden.