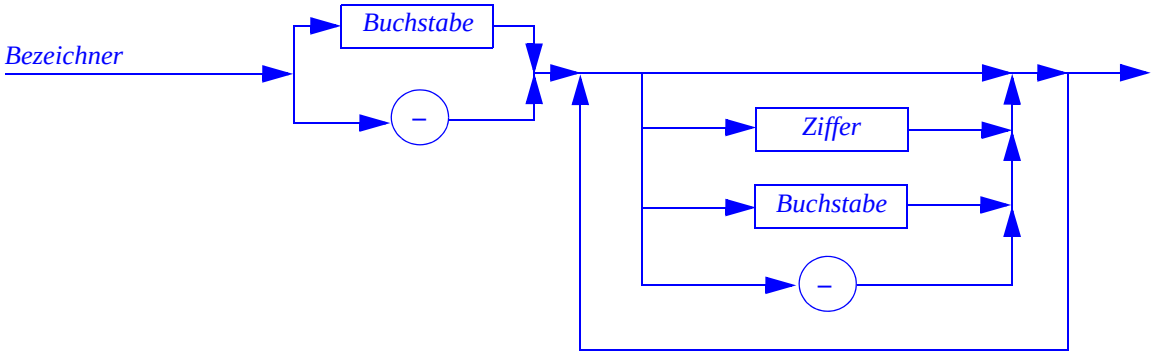
Übungsblatt 1 – Lösungen

Aufgabe 1 - Syntaxdiagramm

Erstelle ein Syntaxdiagramm, das den Aufbau eines *Bezeichners* definiert:
– Ein Bezeichner muss mindestens ein Zeichen lang sein.
– Ein Bezeichner muss mit einem *Buchstaben* oder dem Zeichen `_` beginnen.
– Ab der zweiten Position können beliebige *Buchstaben*, *Ziffern* oder das Zeichen `_` folgen.

Gehe davon aus, dass für die beiden Nichtterminalsymbole *Buchstabe* und *Ziffer* bereits Syntaxdiagramme vorliegen.



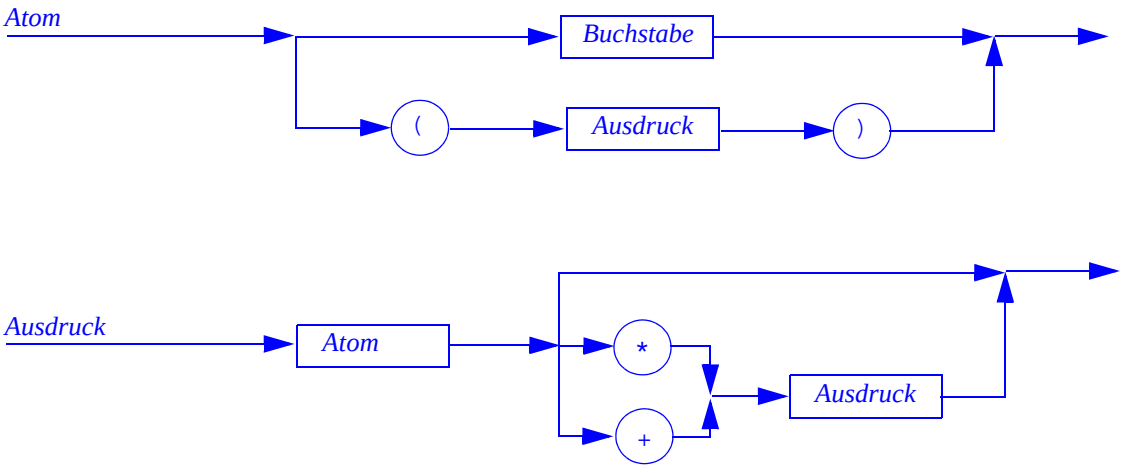
Aufgabe 2 - Syntaxdiagramm

Erstelle ein Syntaxdigramm, das den Aufbau von (vereinfachten) arithmetischen Ausdrücken definiert. Ein Ausdruck kann aus den beiden Operatoren + und *, aus *Buchstaben* und aus Klammerungen mit `(` ... `)` bestehen.
– Ein Ausdruck beginnt mit einem *Buchstaben* oder `(`.
– Vor einem Operator muss ein *Buchstabe* oder `)` stehen.
– Hinter einem Operator muss ein *Buchstabe* oder `(` stehen.
– Zwei *Buchstaben* dürfen nicht unmittelbar aufeinander folgen.
– Vor einem *Buchstaben* darf kein `)` stehen.
– Hinter einem *Buchstaben* darf kein `(` stehen.
– In keinem beliebigen Anfangsstück des Ausdrucks darf es weniger `(` als `)` geben.
– In einem Ausdruck muss es die gleiche Anzahl von `(` und `)` geben.
– Nicht erlaubt ist die Folge `()`.

Gehe davon aus, dass für das Nichtterminalsymbol *Buchstabe* bereits ein Syntaxdiagramm vorliegt.

Hinweis: Diese Regeln beschreiben letztlich das, was man unter einem Ausdruck mit Klammerung erwarten würde.

`(a)` `(B)+(c*(x+e))` `u+v+w` `u+v*w`



Dieses Beispiel zeigt, dass das Diagramm letztlich eindeutiger und leichter verständlich als die textuelle Beschreibung ist. Dieses Diagramm führt zu eindeutigen Ableitungen auch von `u+v+w`.

Aufgabe 3 - Algorithmen konzipieren

Skizziere Algorithmen in Umgangssprache für die folgenden Abläufe. Überlege insbesondere, wie während der Ausführung deren Ende feststellt werden kann. Gehe bei allen Algorithmen davon aus, dass vor Dir zwei unsortierte Stapel mit je etwa 500 Klausuren liegen, auf deren Deckblatt die Matrikelnummer des bearbeitenden Studierenden steht.

- Gibt es mindestens einen Studierenden, der beide Klausuren mitgeschrieben hat?
  Annahme: Der erste Stapel S_1 besteht aus n_1 Klausuren, der zweite Stapel S_2 aus n_2 Klausuren.
  ohne Sortierung: Für jede Klausur k aus S_1 muss S_2 komplett durchgesehen werden. Die Ausführung endet beim ersten "Treffer" in S_2 oder nach Durchsehen aller möglichen Kombinationen. Im schlechtesten Fall müssen $n_1 * n_2$ Vergleiche vorgenommen werden.
  bei Sortierung: Für jede Klausur k aus S_1 muss S_2 nur für die Matrikelnummern durchgesehen werden, die kleiner als k sind. Das Suchen in S_2 kann immer am Punkt der letzten Unterbrechung in S_2 fortgesetzt werden. Im schlechtesten Fall müssen nur $n_1 + n_2$ Vergleiche vorgenommen werden.
- Gibt es keinen Studierenden, der beide Klausuren mitgeschrieben hat?
  Die Problemstellung entspricht der ersten Frage.
- Wie viele Studierende haben beide Klausuren mitgeschrieben?
  ohne Sortierung: Für jede Klausur k aus S_1 muss S_2 komplett durchgesehen werden. Bei jedem Treffer kann die Durchsicht von S_2 abgebrochen und der Zähler um 1 erhöht werden. Das Zählen endet nach allen Kombinationen.
  bei Sortierung: Für jede Klausur k aus S_1 muss S_2 nur für die Matrikelnummern durchgesehen werden, die kleiner als k sind. Das Suchen in S_2 kann immer am Punkt der letzten Unterbrechung in S_2 fortgesetzt werden.
- Gibt es im Stapel 1 mindestens eine Matrikelnummer, die größer ist als jede Matrikelnummer im Stapel 2?
  ohne Sortierung: Für S_1 und S_2 muss jeweils das Maximum bestimmt werden. Das kann jeweils in einem Durchlauf erfolgen. Anschließend müssen die beiden Maxima verglichen werden.
  bei Sortierung: Es muss nur die letzte Klausur (= größte Matrikelnummer) aus S_1 mit der letzten Klausur aus S_2 verglichen werden.
- Ist im Stapel 1 jede Matrikelnummer größer als jede Matrikelnummer im Stapel 2?
  ohne Sortierung: Für S_1 muss das Minimum und für S_2 das Maximum bestimmt werden. Das kann jeweils in einem Durchlauf erfolgen. Anschließend müssen die beiden Werte verglichen werden.
  bei Sortierung: Es muss nur die erste Klausur (= kleinste Matrikelnummer) aus S_1 mit der letzten Klausur aus S_2 verglichen werden.

Was ändert sich an den Algorithmen, wenn die beiden Klausurenstapel aufsteigend nach Matrikelnummern sortiert sind?

Aufgabe 4 - Ganzzahlige Berechnungen

Benutze in den Rümpfen der Methoden nur die `return`-Anweisung und die mathematischen Operatoren `+`, `-`, `/` oder `*` (Multiplikation). Beachte, dass `/` für Werte des Typs `int` die *ganzzahlige* Division durchführt.

Implementiere die folgenden Methoden. Gehe davon aus, dass nur positive Werte als Argumente übergeben werden.

- Die Methode `int remainder( int dividend, int divisor )` soll den Wert zurückgeben, der als Rest der Division von `dividend` durch `divisor` bleibt.

```
public static int remainder( int dividend, int divisor )
{
    return dividend - dividend / divisor * divisor;
}
```

- Die Methode `int isOdd( int value )` soll `1` zurückgeben, falls dem Parameter `value` ein ungerader Wert als Argument übergeben wird. Sie soll `0` zurückgeben, falls dem Parameter `value` ein gerader Wert übergeben wird.

```
public static int isOdd( int value )
{
    return remainder( value, 2 );
}
```

- Die Methode `int isEven( int value )` soll `1` zurückgeben, falls dem Parameter `value` ein gerader Wert als Argument übergeben wird. Sie soll `0` zurückgeben, falls dem Parameter `value` ein ungerader Wert übergeben wird.

```
public static int isEven( int value )
{
    return 1 - isOdd( value );
}
```

- Die Methode `int toEven( int value )` soll Folgendes leisten: Falls dem Parameter `value` ein ungerader Wert als Argument übergeben wird, soll der nächstgrößere gerade Wert zurückgegeben werden. Falls dem Parameter `value` ein gerader Wert übergeben wird, soll dieser (unverändert) zurückgegeben werden.

```
public static int toEven( int value )
{
    return value + isOdd( value );
}
```

- Die Methode `int isDivisible( int dividend, int divisor1, int divisor2 )` soll `0` zurückgeben, falls `dividend` sowohl durch `divisor1` als auch durch `divisor2` ganzzahlig – also ohne Rest – teilbar ist. Sonst soll ein beliebiger Wert ungleich `0` zurückgegeben werden.

```
public static int isDivisible( int dividend, int divisor1, int divisor2 )
{
    return remainder( dividend, divisor1 ) + remainder( dividend, divisor2 );
}
```