AUFGABE 1: STIMMT DIE CHEMIE?

Lösungsidee:

Als ich die Aufgabe das erste Mal las, dachte ich die Gleichungsinformationen in einer Baumstruktur zu speichern. Auf dieser Grundlage habe ich auch zu programmieren begonnen. Eine Gleichung war aufgeteilt in Produkt und Edukt-Seite, dann in Moleküle, welche wiederum eine beliebige Anzahl von Atomen aufnehmen konnten. Nachdem ich die Ein- und Ausgabe solcher Gleichungen fertig hatte, kam dann das Problem der Fragezeichen dran. Das lies mich vom (richtigeren) Weg abkommen und ich lies die Datenstruktur linear. Jetzt sieht sie so aus: Eine Gleichung ist eine Abfolge von Gleichungsbuchstaben. Unter den Gleichungsbuchstaben befinden sich alle Zahlen von 0 bis 255, alle dem Programm bekannten Elemente, und die Zeichen Plus, Pfeil, Fragezeichen und Doppelfragezeichen. Fragezeichen unterscheiden sich im Gegensatz zu Doppelfragezeichen darin, dass sie keine Trennung in 2 Moleküle zulassen. Also ein Fragezeichen kann nur ein Molekül werden, Doppelfragezeichen dagegen können sich in unendlich viele Moleküle zerlegen.

Eingegeben wird die Gleichung als einfacher Text. Dabei gibt es keine spezielle Tiefstellfunktion. Außerdem sind keine Klammern zugelassen. Anstatt $Ba(OH)_2$ muss der Benutzer BaO2H2 eingeben. Als Pfeilzeichen reicht es ein > zu schreiben. Um eine bessere Leserlichkeit zu gewährleisten, kann aber ruhig --> geschrieben werden, da Bindestriche nicht beachtet werden. Elemente beginnen immer mit einem Grossbuchstaben und haben als zweiten keinen oder einen Kleinbuchstaben.

Die Ausgabe soll auf 2 Varianten funktionieren. Erst einmal werden zur Laufzeit die Gleichungen die soeben als richtig ermittelt wurden auf den Bildschirm ausgegeben. Hierbei habe ich natürlich nicht die Möglichkeit Zahlen tiefzustellen bzw. einen schönen Pfeil zu zeichnen. Dafür kommt aber, wenn das Programm fertig ist, und der Benutzer seine eigenen Gleichungen eingeben hat, eine Ausgabe als HTML Datei. So können die Gleichungen leserlicher gemacht werden.

Nach der Eingabe sollen die Buchstaben in Elemente übersetzt werden, wobei ein Element immer mit einem Grossbuchstaben beginnt und wenn überhaupt einen zweiten Kleinbuchstaben besitzt. Im weiteren Verlauf des Programms sollen nur noch die Indizes der Elemente verwendet werden. Zahlen müssen auch vom Zeichensatz in „echte“ Zahlen übersetzt werden. Dabei ist zu beachten, das Moleküle auch mal mehr als 9fach vorhandene Elemente besitzen können. Pfeil, Plus und Fragezeichen benötigen dagegen keine weiteren Behandlungen. Zu jedem dieser Gleichungselemente kommt noch ein Wert hinzu, ob eine Editierung erlaubt ist. Dieser Wert ist nur wahr, wenn der Teil der Gleichung vom Programm selbst bestimmt wurde, und deshalb nicht 100% gleich bleiben muss.

Die Suche nach einer richtigen Lösung soll dann so ablaufen:

Zuerst wird getestet ob sich auf der Produkt und Eduktseite die gleiche Anzahl von Elementen befindet. Ist dies nicht der Fall wird ein Ausgleichsversuch getätigt und rekursiv von vorne gestartet. Ist kein Fehler aufgetreten, so werden alle Moleküle durchgecheckt, bis eins gefunden wird, welches nicht existieren kann. Bei diesem wird dann versucht, mit schon im Molekül vorhandenen Elementen auszugleichen (Fremdelemente können dabei nicht in das Molekül kommen, es müssen ja bei irrsinnigen Angaben auch keine Lösungen gefunden werden). Wurde hingegen kein einziges falsches Molekül gefunden, so ist die Gleichung chemisch richtig und kann ausgegeben werden.

Das Ausgleichen funktioniert so, dass das Programm an allen Stellen mit einem Fragezeichen das Element, welches zu wenig war, einfügt, dann die Suche fortsetzt und irgendwann wieder zurückkommt um an einer anderen Stelle einzusetzen. Bei der Wahl der richtigen Einsetzpunke sind natürlich auch Einschränkungen möglich. Ist ein Element in einem Molekül schon vorhanden, und kein Fragezeichen hinter dem Molekül postiert, oder eine Zahl dahinter, welche nicht editierbar ist, so wird an keiner Stelle des Moleküls das Element eingefügt.

Beim Feststellen, ob ein Molekül möglich ist, hatte ich Anfangs eine rekursive Methode. Da ich aber mit dieser nicht richtig die Ausgleichmöglichkeiten bestimmen konnte, hab ich sie verworfen und so umgebaut: Die Oxidationszahlen werden für die einzelnen Elemente wie in einer großen Zahl hintereinander gestellt, und die Zahl hoch gezählt. Ist ein Element ein Geber so müssen auch die Zahlen vor ihm Geber sein. Die OH Gruppe hab ich dabei natürlich auch berücksichtigt. Das System schränkt natürlich auch etwas ein. Elemente die in 2 verschiedenen Oxidationsstufen in einem Molekül vorkommen, werden nicht als richtig anerkannt, aber wem störte das im 19. Jahrhundert (mich jedenfalls nicht ☺).

Die Vorgehensweise, die für eine chemische Gleichung die Plausibilität errechnen, ist bei mir ein System, welches testet, ob die Lücken auch exakt so ausgefüllt wurden wie erlaubt und kein noch lesbarer Text überschrieben wurde. Die Gleichungen müssen ja so und so chemisch exakt sein um in die Lösungen aufgenommen zu werden.

Sonst gibt es eigentlich nur noch das Periodensystem, über welches ich noch nichts geschrieben habe. Um es zu generieren existiert ein gesondertes Programm, welches eine Liste von Molekülen speichert. Dabei werden Name, die möglichen Oxidationsstufen, ob das Molekül ein Gas ist und ob es eher Donnator oder Akzeptor ist gespeichert. Wobei die Donnator Akzeptor Informationen an keiner Stelle meines Programms zur Verwendung kommen. (Steht ja auch im FAQ, dass die vielleicht etwas zu stark hervorgehoben wurden).

Programmdokumentation:

Periodensystem:

Um die verschieden Spezifikationen der einzelnen Elemente zu speichern, habe ich den Typen `atom` deklariert.

```
  atom = RECORD
    Name: String[2];
    oxiz: SET OF Byte;
    gas, akz, don: Boolean;
  END;
```

Wie man sieht, werden für jedes Atom 2Buchstaben, die gas-Eigenschaft und die Vorlieben Akzeptor oder Donnator gespeichert. Hinzu kommt noch eine Zahlenmenge, welche ich für die Oxidationsstufen verwendet habe. Subtrahiert man von einer dieser Zahlen sieben, so erhält man die Oxidationszahl. (8 ist in der Menge enthalten → Oxidationsstufe 1 ist möglich).

Mit Hilfe des Programms periodensystem1.exe habe ich die Datei persys.dat erstellt. Diese wird beim Programmstart von Project1.exe eingelesen und in der Variable `persys` vom Typ **`array of`** `atom` gespeichert.

Datenstruktur der Reaktionsgleichungen:

Die 5 Verschiedenen Zeichenarten, die in Reaktionsgleichungen vorkommen können habe ich in `Tgltypen = (Frage, Plus, Pfeil, Zahl, Element);` festgelegt. Damit könnte man aber eine Gleichung noch nicht ausreichend exakt beschreiben. Darum habe ich den Typen so definiert:

```
Tgleichung = RECORD
    typ: Tgltypen;
    nr: Byte;
    edit: Boolean;
  END;
```

Die Variable `nr` speichert, wenn `typ=Zahl` den exakten Wert der Zahl, wenn `typ=Element` die Nummer des Elements im Periodensystem, und wenn `typ=Frage` ob es ein einfaches Fragezeichen oder ein Doppelfragezeichen(`nr=2`) ist. Im Programm wird mit diesem Typen der dynamische Array `gleichung` definiert. Dieser ist global und wird so lange modifiziert bis richtige Gleichungen entstehen.

stringtogleichung:

Da der Benutzer aber schlecht die `gleichung` direkt in dieser Struktur eingeben kann, habe ich die Prozedur `stringtogleichung` geschrieben. Beim Start des Programms wird die Eingabe der Variable `eingabe` verlangt. Da sie global ist, kann auch `stringtogleichung` auf sie zugreifen. Bei der Umwandlung wird so vorgegangen:

```
Wiederholen
  Wenn das Eingegebe Zeichen an der Stelle i
    ?              : Wenn nächstes Zeichen auch ? dann
                       Fragezeichen an die Gleichung anhängen nr = 2
                     sonst
                       Fragezeichen an die Gleichung anhängen nr = 1
    +              : Plus anhängen
    >              : Pfeil anhängen
    Grossbuchstabe : Wenn Folgelement ein Kleinbuchstabe dann
                       Element mit dem passenden 2 Buchstaben aus dem Periodensystem anfügen
                     sonst
                       Element mit dem passenden Buchstaben aus dem Periodensystem anfügen
    Zahl           : k := i; Solange k erhöhen bis Eingabe[k+1] keine Zahl mehr ist
                     Zahl mit dem Wert von eingabe[i bis k] anfügen
i erhöhen
bis i = Länge der Eingabe
```

Im Programm wird nach dieser Umwandlung die Gleichung noch einmal auf dem Bildschirm geschrieben (der Text, welcher sich in der Inhalt der Variable `gleichung` ergibt). Der Benutzer kann entscheiden, ob die Eingabe so korrekt war, und wenn nötig so lange verbessern, bis er seine Sache richtig erledigt hat.

Die Prozedur `stringtogleichung` wird ein zweites Mal beim Eingeben von fremden Lösungsvorschlägen verwendet.

brute:

Die Prozedur `brute` ist die Hauptprozedur beim finden der möglichen Ergebnisse. Sie wird nach jeder Veränderung der Originalgleichung ausgeführt und entscheidet dann, ob die Gleichung noch verändert werden muss, oder bereits richtig ist und zu den möglichen Ergebnissen hinzugefügt werden kann. Damit die Suche nicht zu lange dauert, habe ich die Rekursion und die Gleichungslänge an dieser Stelle etwas eingeschränkt. Die Entscheidung ob richtige oder Falsche chemische Gleichung wird nur ausgeführt, wenn die Rekursionstiefe `deep` unter einem Wert `maxdeep` ist. Außerdem wird nach gefundenen Fehlern beim Molekülanzahlvergleich, nur dann ein Molekül bei einer Seite addiert, wenn die Länge der Gleichung unter `maxlen` ist. Beim Molekülcheck mache ich diesen Vergleich nicht, da ich hier nicht das Problem habe, dass maximal 4 Zeichen eingefügt werden können, wodurch die Gleichung sehr schnell sehr lang wird. Halbformal läuft die Funktion so:

```
Wenn Rekursionstiefe < maxdeep dann
  Erhöhe Rekursionstiefe
  Wenn sich auf rechter und linker Seite unterschiedlich viele Atome befinden (anzahlvergleich) dann
    Wenn Gleichung nicht länger als maxlen für bei der passenden Seite ein Atom ein
  Sonst
    Wenn beim Molekülcheck kein Fehler auftritt ist die entstandene Gleichung Richtig
  Verringere Rekursionstiefe
```

molekuelcheck:

Molekülcheck hört sich ziemlich einfach an, ist er aber nicht ganz. Bei einem einatomigen „Molekül“ ist der Test sehr einfach. Die Anzahl der Gasatome muss durch zwei teilbar sein. Befinden sich aber mehrere Moleküle in einem Atom, so muss man testen, ob eine Oxidationszahlenkombination der Elemente existiert, deren Summe Null ergibt. Meine erste Variante der Problemlösung war auch rekursiv, dabei erwies sich aber die Suche nach möglichen Ausgleichsmolekülen als sehr aufwändig. Deshalb habe ich die Rekursion aufgelöst und die auftretenden Daten der einzelnen Elemente in einem Array vom Typ `Toxicheck` gespeichert.

```
Toxicheck = RECORD
    mol, oxi, anz: Byte;
    edit: Boolean;
  END;
```

Um alle Oxidationsmöglichkeiten durchzutesten, werden die einzelnen Oxidationszahlen der Elemente im Array quasi von -7;-7;... auf 7;7;... hoch gezählt. Wenn ein weiter hinten stehendes Element von 7 auf -7 zurückwechselt, dann steigt die Oxidationszahl des vorherigen Elements zur nächsten Oxidationszahl. So können alle Kombinationen durchgetestet werden, und der Array bietet sich perfekt an bei fehlerhaften Oxidationszahlsummen die Ausgleichsatome zu finden.

Zum besseren Verständnis hier die ganze Abfolge der Prozedur:

```
Annahme: kein Fehler
Wiederhole
  Wenn i auf ein Element zeigt dann erhöhe Atomanzahl
  Wenn i am Ende eines Moleküls angekommen dann
    Wenn Atomanzahl = 0 dann --> Fehler zurückliefern
    Wenn Atomanzahl = 1 dann
      Wenn Gaselement und Anzahl nicht durch 2 Teilbar --> Fehler zurückliefern
      Wenn Fehler bei Gaselement und editierbar --> Element hinzufügen + Rekursion
    Wenn Atomanzahl > 1 dann
      Array mit den Daten (Elementnummer, Anzahl, editierbar) für jedes Element feststellen
      Annahme: Molekül gibt es nicht
      Wiederhole
        v an das Ende des Elementarray setzen
        Wiederhole
          Wenn Überlauf dann Oxidationszahl für Element v erhöhen
          Wenn weiter hinteres Element ein Geber dann Oxidationszahl auf Geber erhöhen
          Wenn die zwei Elemente O und H sind dann nicht
          Erhöhen der Oxidationszahl des Elements v, bis Oxidationszahl möglich
            Wenn Oxidationszahl dabei > 7 dann Überlauf := true; Oxidationszahl := -1;
          v erniedrigen
        Bis v das erste Element unterschreitet
        Wenn kein Überlauf (tritt nur beim letzten Versuch ein)
          Die Summe der Oxidationszahlen Bilden
          Wenn Summe = 0 dann Molekül kann existieren
          Sonst
            Alle Moleküle testen ob die momentane Oxidationszahl anderes Vorzeichen als die Summe
              besitzt; Wenn ja dann Molekül zu den Ausgleichsmöglichkeiten hinzufügen
      bis überlauf;
      Wenn Molekül nicht existiert --> Fehler zurückliefern
      Wenn OH Gruppe im Molekül sich nicht am Ende es Moleküls befindet --> Fehler zurückliefern
      Wenn Molekül nicht existiert oder immer Ausgleichsmöglichkeiten getestet werden sollen
        Dem Molekül alle Elemente im Ausgleichsmöglichkeiten ARRAY hinzufügen und brute starten
Bis i auf oder über das Ende der Gleichung hinaus zeigt
```

Durch die globale Boolean Konstante immertesten kann festgelegt werden, ob die Suche nach anderen Molekülkonstellationen auch fortgesetzt wird, wenn das Molekül schon existiert. So kann aus AlCl auch AlCl3 werden. An dieser Stelle des Programms könnte man zum Oxidationszahlenausgleich auch Fremdelemente einfügen. Dies habe ich zwar Anfangs vorgesehen (`elementhinzu` Variable), nach dem lesen der FAQ aber nicht mehr für nötig gehalten.

anzahlvergleich:

Die Funktion `anzahlvergleich`, durchsucht die ganze Gleichung um festzustellen, ob sich auf Produkt- und Eduktseite die gleiche Anzahl von Atomen befinden. Da hierzu immer Zahlenpaare bestehend aus Elementnummer und Anzahl benötigt werden, habe ich den Type `Tatome` definiert.

```
  Tatome = RECORD
    nr: Byte;
    anz: Integer
  END;
```

Dieser wird einerseits als Funktionsinterner Speicher für die verschiedenen Atome verwendet, aber auch die Rückgabe der Funktion verwendet diesen Datentyp. Die Funktion wird von der Prozedur `brute` aufgerufen, um die passenden Ausgleichsaktionen in die Wege zu leiten. Der Ablauf der Funktion geschieht wie nachfolgend beschrieben:

```
neu := true;
seite := 1;
Schleife i für alle Gleichungszeichen
  Wenn Gleichungszeichen[i]
  Plus    : neu := true; Molekülanzahl := 1;
  Pfeil   : neu := true; Molekülanzahl := 1; seite := -1
  Zahl    : Wenn neu dann Molekülanzahl := Zahl
  Element : neu := false;
            Wenn Element noch nicht im Atome array vorkommt, hinzufügen
            Wenn Folgezeichen eine Zahl dann
              Anzahl für dieses Element um Molekülanzahl * Seite * Zahl
            Sonst
              Anzahl für diesen um Molekülanzahl * Seite Erhöhen
Wenn alle Elemente die Anzahl 0 am Ende besitzen dann
```

```
  Atomnummer 255 zurückliefern
Sonst
  Eines der Atome aus dem Array ohne Anzahl 0 zurückliefern
```

atomeinfuegen:

Diese Funktion wird einerseits zum Einfügen von Atomen nur auf einer gewissen Gleichungsseite und zum Einfügen in ein Molekül verwendet. Die Variante bei der die Einfügepunkte auf eine Seite beschränkt, wird der Startpunkt der Fragezeichendurchsuchung auf 0 oder die Position des Pfeils gesetzt. Wenn für `pos` ein Wert übergeben wird, so ist dies bereits der Startpunkt eines gewissen Moleküls.

Das Durchsuchen der Gleichung wird Molekül für Molekül vorgenommen, da nicht jedes Fragezeichen für ein Einsetzversuch relevant ist. Befindet sich nämlich schon das gleiche Element im Molekül an einer Stelle, so ist das Einsetzen an einer anderen nicht sinnvoll. In diesem Fall muss man die Anzahl des schon vorkommenden Moleküls erhöhen. Um dies zu ermöglichen, habe ich den edit Parameter bei jedem Gleichungselement gespeichert. Er gibt an, ob das Zeichen veränderbar ist, was eigentlich nur eintrifft wenn es vom Programm selbst eingefügt wurde. Die verschiedenen Ersetzungsvarianten sehen so aus (zwingend editierbare Elemente sind fett gedruckt):

```
Wenn das Element vorhanden:   El2 → El3   El? → El2?   ?El → 2?El   2El → 3El
Wenn nicht:          ...?2... → ...?El2...   ...?... → ...?El?...   ...?... → ...+?El?
```

Im Programm hat das ganze dann diese Struktur:

```
Parameter: Seite, Elementposition, Elementnummer

Wenn Seite <> 0 dann
  Wenn Seite > 0 dann i auf die Position des Pfeils erhöhen
sonst
  i auf den Wert der Elementposition setzen
  falls i auf einen Pfeil oder ein Plus deutet i erhöhen

Wiederholen
  l := i;
  l so lange erhöhen bis Gleichungselement[l] ein Plus oder Pfeil, oder l größer als Gleichungslänge
    währenddessen: wenn Gleichungselement[l] ist Elementnummer dann enthalten := true; k := l;
  wenn enthalten dann
    wenn Folgeelement von k eine editierbare Zahl dann
      diese Zahl um eins erhöhen; brute;
      Veränderung rückgängig machen;
    sonst wenn Folgeelement ein Fragezeichen
      ab Fragezeichen alles um eins nach hinten schieben; Zahl mit Nummer 2 einfügen; brute;
      Veränderungen rückgängig machen;
    sonst
      k an den Start des Moleküls fahren;
      wenn sich dort ein Fragezeichen befindet dann
        ab Fragezeichen alles um eins nach hinten schieben; Zahl mit Nummer 2 einfügen; brute;
        Änderungen rückgängig machen;
      wenn sich dort eine Zahl befindet
        Zahl um eins erhöhen; brute;
        Änderungen rückgängig machen;
  sonst
    Wiederhole wenn Gleichungselement i ein Fragezeichen ist dann
      wenn Folgeelement eine Zahl ist dann
        Element an Position einfügen und Rest nach hinten verschieben (Bsp: ?Na2); brute;
        Veränderungen rückgängig machen;
      Element + Fragezeichen einfügen und Rest nach hinten verschieben; brute;
      Veränderungen rückgängig machen
      wenn Doppelfragezeichen dann
        "+?Element?" einfügen und Rest nach hinten verschieben; brute;
        Veränderungen rückgängig machen;
      erhöhe i;
    bis Gleichungselement Pfeil oder Plus;

  erhöhe i;
bis Gleichungselement ein Pfeil ist, i größer als die Gleichungselementanzahl, nur in einem Element
  ersetzt werden soll
```

reduce:

Nachdem das Programm eine Gleichung gefunden hat, welche zufällig mal richtig ist, muss das Ergebnis noch etwas frisiert werden, damit es wie eine „echte“ chemische Gleichung aussieht. Dazu müssen doppelte Elemente auf einer Seite zusammengefasst werden, und die Elementanzahl in den Molekülen möglichst klein gehalten werden. Das Aufgabenbeispiel liefert ohne diese Optimierung $H_4 + O_2 \rightarrow H_4O_2$. Es soll aber $2H_2 + O_2 \rightarrow 2H_2O$ rauskommen. Nachdem diese Arbeit erledigt ist, erstellt das Programm noch eine Stringgleichung und eine HTML Variante aus dem Ergebnis. Diese werden dann im `ergebnisse` Array vom Typ `Tergebnisse` gespeichert.

```
  Tergebnisse = RECORD
    glstring, glausgabe: String;
    fehler: Byte;
  END;
```

Im Programm wurde das Ausklammern so realisiert:

```
Momentane Gleichung in Gleichung2 übertragen;

Wiederhole
  Wenn Gleichung2 i = Element dann
    erhöhe Elementanzahl;
    Speichere ob Element ein Gas
```

```
    Wenn nachfolgendes Element eine Zahl dann
      Teiler = ggt von Elementanzahl und altem Teiler bzw. Teiler = Elementanzahl
    sonst
      Teiler = 1
  Wenn Molekül zu Ende dann
    Wenn Molekül ist ein Einatomiges Gas dann Teiler durch 2 teilen
    Wenn der Teiler > 1 ist dann
      Wenn Molekül mit Zahl beginnt dann
        Zahl  = Zahl * Teiler
      sonst
        Zahl an dieser Position einfügen und Wert des Teilers setzen
      Schleife für alle Zeichen des Moleküls
        Wenn Zeichen eine Zahl dann diese durch Teiler teilen
        Wenn dadurch eine 1 entsteht, diese Stelle löschen
  i vergrößern;
bis i am Ende der Gleichung2
```

Nachdem die einzelnem Moleküle nun ihre optimale Form besitzen nimmt sich das Programm noch die Produkt und Eduktseite vor. Hier der Ablauft für die Eduktseite:

```
Neustartpunkt der Suche;

Startpunkt := 0;
Endpunkt := Position des Pfeils;
Wiederholen
  k an den Startpunkt des Moleküls bringen
  l := k +1;
  l an den Endpunkt des Moleküls
  Wenn l < Endpunkt dann Wiederholen
    l zum Start des nächsten Moleküls vorrücken;
    Solange sich der Inhalt von Gleichung2[k+i] und Gleichung2[l+i] entspricht und kein Plus oder
      Pfeil ist i erhöhen
    Wenn Schleife wegen Plus oder Pfeil beendet dann
      k an den Start des zweiten Moleküls fahren
      wenn sich dort eine Zahl befindet diese speichern
      das zweite Molekül aus der Gleichung entfernen
      wenn sich am Start von Molekül eins eine Zahl befindet dann
        Zahl um Anzahl des zweiten Moleküls erhöhen
      wenn nicht dann
        Zahl einfügen und Wert auf 1+ Anzahl setzen
      zum Neustart springen
    l solange erhöhen bis nächstes Vergleichselement beginnt;
  bis l >= Endpunkt
  k solange erhöhen bis nächstes zu vergleichendes Molekül erreicht;
bis k >= Endpunkt
```

Beim eliminieren der doppelten Moleküle auf der Produktseite, wird die gleiche Abfolge verwand, nur mit anderen Start und Endpunkten. Wenn auch diese Seite erledigt ist, dann wird eine Stringgleichung erstellt und getestet, ob diese Lösung nicht schon einmal erreicht wurde. Bei einer neu entdeckten Lösungsmöglichkeit wird dieser String im `ergebnisse` Array abgespeichert und eine HTML Version dieser Gleichung erstellt. Dabei verwende ich zum Tiefsellen den Tag <sub> und zum Darstellen eines etwas schöneren Pfeils die Chars ─ (langer Bindestrich —) ► (Pfeil ►) aus Arial. Eie dadurch erstellte Gleichung sieht im Quelltext typischerweise so aus `2H<sub>2</sub> + O<sub>2</sub> &#9472;&#9472;&#9658; 2H<sub>2</sub>O` und ergibt dann diese Ausgabe im Browser: $2H_2 + O_2$ ——► $2H_2O$.

Benutzerdefinierte Lösungen:

Nachdem die rekursive Suche wieder zu ihrem Ausgangspunkt zurückgekommen ist, wird dem Benutzer die Eingabe von eigenen Lösungsmöglichkeiten gestattet. Damit er aber nicht einfach irgendeinen chemisch Unsinn eintippen kann, wird dabei immer überprüft, ob es sich um chemisch richtige Gleichungen handelt. Dies geschieht so:

```
Wiederholen
  Gleichung eingeben;
  Wenn eingebe <> '' dann
    String in Gleichung umwandeln;
    Wenn Anzahlvergleich OK und molekuelcheck OK dann
      mit reduce procedure Gleichung in den ergebnisse ARRAY aufnehmen
    sonst
      Gleichung Fehlerhaft ausgeben
bis nichts eingegeben wurde;
```

Dabei werden ausschließlich die Prozeduren verwendet, die auch schon während der Rekursion ihre Arbeit verrichtet haben.

endkontrolle:

Diese Prozedur erfüllt die letzte Aufgabenstellung, nämlich die Plausibilitätskontrolle. Diese macht eigentlich nur bei vom Benutzer hinzugefügten Ergebnissen Sinn, da die vom Programm errechneten Gleichungen chemisch richtig sein sollten und auch das Einsetzen an den Stellen mit Fragezeichen richtig durchgeführt wird. Wenn mein Programm mehrere Ergebnisse liefert, dann kann es ja nicht sagen: Gleichung 1 ist richtiger als Gleichung 2. Beim Plausibilitätscheck wird dann nur noch getestet ob die Gleichung zur Frage passt. Ist ein Fragezeichen in der Fragegleichung enthalten, so kann der Ergebnisstring an dieser Stelle Veränderungen aufweisen. Anderen Stellen wird dies sofort mit der Erhöhung eines Fehlercounters geahndet. Ist für alle Gleichungen die Fehleranzahl berechnet, so werden sie Sortiert und nach der Reihenfolge ausgegeben.

Probleme:

Ich denke, mein größtes Problem ist meine Datenstruktur. Sie mag vielleicht für die Eingabe und Ausgabe und die Exaktheit der Lösungen von Vorteil sein. Leider sind aber die Prozesse die molekuelcheck, anzahlvergleich, atomeinfuegen und reduce durchführen, dann doch etwas chaotisch. In einem sauberen Baum, der durch Parsing aus der Gleichung entstanden ist, wäre das wohl nicht passiert. In meinem ersten Ansatz bin ich auch so vorgegangen. Nur beim exakten einfügen von Elementen an den Fragezeichenstellen kamen dann Probleme auf, welche mich zu einem kompletten Umbau bewegten (nur das Periodensystem hab ich übernommen). Im Nachhinein denke ich, meine erste Datenstruktur hätte sich wohl im Große und Ganzen bessernd für die Ergebnisfindung ausgewirkt.

Meine Chemiekenntnisse aus dem 19. Jahrhundert haben sich auch als sehr schlecht erwiesen. Die meisten Reaktionen die ich noch aus dem Unterricht im Gedächtnis hatte, waren weit außerhalb dem begrenzten Chemiewissen der Programmen. Deshalb war ich über die Gleichungen die ich in der Newsgroup (die zwar auch nicht gerade so Besonders waren) fand sehr glücklich, und noch mehr als diese mein Programm auch lösen konnte. Weiter darüber hinaus konnte ich nicht recht viel testeten.

Programmablaufprotokoll:

Die Eingaben, die der Benutzer durchführen muss sind Fett gedruckt und mit Enter beendet.

```
Gleichung nach diesem Beispiel eingeben:  ?H2 + O2 --> ?

2Na + ? --> ?NaOH + ??↵

Eingelesen entspricht das:
2Na + ? --> ?NaOH + ?

Sind sie mit ihrer Eingabe einverstanden? Mit ¿ gekennzeichnete Molekuele kennt
das Programm nicht! (j oder n)n↵


Gleichung nach diesem Beispiel eingeben:  ?H2 + O2 --> ?

2Na + ? --> ?NaOH + ?↵

Eingelesen entspricht das:
2Na + ? --> ?NaOH + ?

Sind sie mit ihrer Eingabe einverstanden? Mit ¿ gekennzeichnete Molekuele kennt
das Programm nicht! (j oder n)j↵


Die verschiedenen Moeglichkeiten lauten:
2Na + 2NaH3O2 --> 4NaOH + H2
2Na + NaH5O3 --> 3NaOH + H2
2Na + 2H3NaO2 --> 4NaOH + H2
2Na + H5NaO3 --> 3NaOH + H2
2Na + 2H2O --> 2NaOH + H2

Mehr Ergebnisse konnte ich leider nicht finden.

Sie koennen jetzt auch noch eigene Ergebnisse anf³gen. (Leere Eingabe beendet)
2H2O --> 2H2 + O2↵
Gleichung wurde so in die Liste aufgenommen: 2H2O --> 2H2 + O2
↵

Nun werden die Gleichungen auf ihre Plausibilität beurteilt und als ausgabe.htm
gespeichert.
↵
```

Beispiele:

Fragestellung:
?H_2 + O_2 ⟶ **?**

Mögliche Ergebnisse in der Reihenfolge ihrer Qualität:
$2H_2$ + O_2 ⟶ $2H_2O$

Fragestellung:
2HCl + **?** ⟶ H_2O + **?**Na**?**

Mögliche Ergebnisse in der Reihenfolge ihrer Qualität:
2HCl + Na_2O ⟶ H_2O + 2NaCl

Fragestellung:
2Na + **?** ⟶ **?**NaOH + **?**

Mögliche Ergebnisse in der Reihenfolge ihrer Qualität:
2Na + $2NaH_3O_2$ ⟶ 4NaOH + H_2
2Na + NaH_5O_3 ⟶ 3NaOH + H_2

$2Na + 2H_3NaO_2 \longrightarrow 4NaOH + H_2$
$2Na + H_5NaO_3 \longrightarrow 3NaOH + H_2$
$2Na + 2H_2O \longrightarrow 2NaOH + H_2$

Quelltexte:

Project1.dpr

```
PROGRAM aufgabe1;
{$APPTYPE CONSOLE}


USES
  SysUtils, brute_attack In 'brute.pas';                          //rekursive Programmteile einladen

VAR
  eingabe, janein, originalgleichung: String;                       //Rest der benötigten Variabeln

PROCEDURE persys_einlesen;                  //Lest den Inhalt des Periodensystems aus persys.dat ein
VAR
  datei: FILE OF atom;
BEGIN
  setlength(persys, 1);                    //das Periodensystem soll mit einem Fake Element beginnen
  persys[0].oxiz := [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14];
  persys[0].Name := chr(168);
  assignfile(datei, 'persys.dat');                    //persys.dat öffnen und alle Elemente auslesen
  reset(datei);
  IF Not (EOF(datei)) THEN
    REPEAT
      setlength(persys, length(persys) + 1);
      Read(datei, persys[length(persys) - 1]);
    UNTIL EOF(datei);
  closefile(datei);
END;


PROCEDURE addgl(nr: Byte; typ: Tgltypen);                     //zur Gleichung ein Element hinzufügen
BEGIN
  setlength(gleichung, length(gleichung) + 1);
  gleichung[length(gleichung) - 1].nr := nr;
  gleichung[length(gleichung) - 1].typ := typ;
  gleichung[length(gleichung) - 1].edit := False;
END;

FUNCTION welcheselement(inp: String): Byte;      //Errechnet aus der Buchstabenfolge eines Elementes
VAR                                                          //dessen Nummer im Periodensystem
  i: Byte;
BEGIN
  i := 0;
  WHILE (i <> length(persys)) And (inp <> persys[i].Name) DO inc(i);
  IF (i = length(persys)) THEN welcheselement := 0
  ELSE
    welcheselement := i;
END;


PROCEDURE stringtogleichung;              //wandelt den eingelesenen String in die intern verwendete
VAR                                                                 //Gleichungstruktur um
  i, k: Byte;
BEGIN
  eingabe := eingabe + ' ';
  setlength(gleichung, 0);
  i := 1;
  REPEAT
    CASE eingabe[i] OF
      '?': BEGIN                                                  //Buchstabe ist ein Fragezeichen
          IF eingabe[i + 1] = '?' THEN                  //bei 2 Fragezeichen --> + Zeichen erlauben
          BEGIN
            addgl(2, frage);
            inc(i)
          END ELSE                                                //bei einem nur Element editierbar
            addgl(1, frage)
        END;
      '+': addgl(0, plus);                                                    //Pluszeichen
      '>': addgl(0, Pfeil);                                                  //Reaktionspfeil
      'A'..'Z': IF eingabe[i + 1] In ['a'..'z'] THEN                        //Buchstaben umwandeln
        BEGIN                                                              //Element aus 2 Zeichen
          addgl(welcheselement(eingabe[i] + eingabe[i + 1]), Element);
          inc(i);
        END ELSE                                                    //Element aus einem Zeichen
          addgl(welcheselement(eingabe[i]), Element);
        '0'..'9':                                                            //Zahlen einlesen
        BEGIN
          k := i;
          WHILE eingabe[k + 1] In ['0'..'9'] DO inc(k);        //Mehrstelligkeit von Zahlen erkennen
          addgl(StrToInt(copy(eingabe, i, k - i + 1)), Zahl);
          i := k;
        END;
    END;
    inc(i);
  UNTIL i > length(eingabe);                        //Schleife für nach letztem Buchstaben abbrechen
```

```
END;


PROCEDURE endkontrolle;     //Endkontrolle = check ob nur an stellen mit Fragezeichen editiert wurde
VAR                                  // + Ausgabe einer HTML Datei mit den formatierten Ergebnissen
  i, x, y: Byte;
  vorhanden: Boolean;
  h: Tergebnisse;
  f: textfile;
BEGIN
  IF length(ergebnisse) <> 0 THEN
    FOR i := 0 TO length(ergebnisse) - 1 DO                          //Schleife für alle Ergebnisse
    BEGIN
      ergebnisse[i].fehler := 0;
      y := 0;
      FOR x := 0 TO length(originalgleichung) - 1 DO            //Schleife für die Originalgleichung
      BEGIN
        IF originalgleichung[x] = '?' THEN                  //? --> den eingefügten Teil überspringen
        BEGIN
          WHILE (originalgleichung[x + 1] <> ergebnisse[i].glstring[y]) And
            (y < length(ergebnisse[i].glstring)) DO inc(y);
        END ELSE
        BEGIN
          IF originalgleichung[x] <> ergebnisse[i].glstring[y] THEN inc(ergebnisse[i].fehler);
          inc(y);                    //Änderung der Gleichung an Positionen ohne ? --> Fehler erhöhen
        END;
      END;
    END;

  FOR x := length(ergebnisse) - 1 DOWNTO 0 DO          //Bubblesort der Ergebnisse nach Fehleranzahl
    FOR y := 2 TO x DO
      IF ergebnisse[y - 1].fehler < ergebnisse[y].fehler THEN
      BEGIN
        h := ergebnisse[y - 1];
        ergebnisse[y - 1] := ergebnisse[y];
        ergebnisse[y] := h
      END;


  assignfile(f, 'ausgabe.htm');                                      //Ausgabe des HTML Kopfes
  rewrite(f);
  writeln(f, '<html><title>Ergebnisse zur Aufgabe1</title><body bgcolor="#FFFFFF" text="#000000">',
    '<font face="Arial">');
  writeln(f, 'Fragestellung:<br>');
  eingabe := originalgleichung;
  stringtogleichung;

  vorhanden := True;

  IF length(gleichung) <> 0 THEN                  //Ausgabe der HTML formatierten Originalgleichung
    FOR i := 0 TO length(gleichung) - 1 DO
      CASE gleichung[i].typ OF
        Plus: BEGIN                                                                          //Plus
                Write(f, ' + ');
                vorhanden := True;
              END;
        Pfeil: BEGIN
                 Write(f, ' ⟶ ');                                  //Reaktionspfeil
                 vorhanden := True;
               END;
        Frage: BEGIN
                 Write(f, '<b>?</b>');                                          //Fragezeichen
                 vorhanden := True;
               END;
        Zahl: IF vorhanden THEN Write (f, IntToStr(gleichung[i].nr))              //Normale Zahlen
              ELSE
                Write(f, '<sub>' + IntToStr(gleichung[i].nr) + '</sub>');     //Tiefgestellte Zahlen
        Element: BEGIN
                   Write(f, persys[gleichung[i].nr].Name);                              //Elemente
                   vorhanden := False;
                 END;
      END;

  writeln(f, '<br>');
  writeln(f, '<br>');
  writeln(f, 'Mögliche Ergebnisse in der Reihenfolge ihrer Qualität:<br>');

  IF length(ergebnisse) <> 0 THEN                                  //Ausgabe der möglichen Gleichungen
    FOR i := 0 TO length(ergebnisse) - 1 DO
      writeln(f, ergebnisse[i].glausgabe, '<br>');

  Close(f);

END;


BEGIN                                                                                //Hauptteil

  persys_einlesen;                                                 //Periodensystem einlesen

  REPEAT
    Writeln('Gleichung nach diesem Beispiel eingeben:  ?H2 + O2 --> ?');
```

```
    Writeln;

    setlength(gleichung, 0);                                          //Gleichung einlesen
    readln(eingabe);

    stringtogleichung;                              //in Programminternes Gleichungsystem umrechnen

    writeln;
    writeln('Eingelesen entspricht das:');

    gleichungausgeben;                                    //umgerechnete Gleichung wieder ausgeben

    writeln;                                          //Richtigkeit der Gleichung bestätigen lassen
    Write('Sind sie mit ihrer Eingabe einverstanden? Mit ', chr(168),
      ' gekennzeichnete Molekuele kennt das Programm nicht! (j oder n)');
    readln(janein);
    writeln;
    writeln;
  UNTIL (janein[1] In ['J', 'j']);                //Schleife bis Gleichung richtig eingegeben wurde

  writeln('Die verschiedenen Moeglichkeiten lauten:');

  deep := 0;
  brute;                                                          //Rekursive Suche starten


  writeln;
  writeln('Mehr Ergebnisse konnte ich leider nicht finden.');
  writeln;
  writeln('Sie koennen jetzt auch noch eigene Ergebnisse anfügen. (Leere Eingabe beendet)');
  originalgleichung := eingabe;                                   //Originalgleichung sichern
  REPEAT
    readln(eingabe);                                        //Andere Gleichungen einlesen lassen

    IF eingabe <> '' THEN
    BEGIN
      stringtogleichung;
      deep := maxdeep;                                      //Test ob diese Chemisch mögich sind
      IF (anzahlvergleich.nr = 255) And molekuelcheck THEN
      BEGIN
        Write ('Gleichung wurde so in die Liste aufgenommen: ');
        reduce;                                                 //In den Ergebnisarray aufnehmen
      END ELSE
        Writeln('Die Gleichung enthaelt Fehler!!!');
    END;

  UNTIL eingabe = '';                       //Solange Wiederholen bis keine Gleichung eingegeben wurde

  writeln;
  writeln('Nun werden die Gleichungen auf ihre Plausibilität beurteilt und als ausgabe.htm
gespeichert.');

  endkontrolle;                                             //Plausibilitätstest und HTML Ausgabe

  readln;

END.
```

brute.pas

```
UNIT brute_attack;

INTERFACE

USES sysutils;

TYPE
  atom = RECORD                                 //Typ um einzelne Atome im Periodensystem darzustellen
    Name: String[2];                                                                        //Name
    oxiz: SET OF Byte;                                                  //Oxidationsmöglichkeiten
    gas, akz, don: Boolean;                                 //Gas, Donnator und Akzeptoreigenschaften
  END;

  Tatome = RECORD                               //Variablentyp den die Funktion anzahlvergleich ausgiebt
    nr: Byte;                                                                   //Nummer des Elements
    anz: Integer                                                                          //Anzahl
  END;

  Toxicheck = RECORD          //Typ der die einzelnen Atome eines Molekuel in molekuelcheck darstellt
    mol, oxi, anz: Byte;                                          //ElementNummer, Oxidationszahl, Anzahl
    edit: Boolean;                          //true wenn die Anzahl des Molekuels verändert werden kann
  END;

  Tergebnisse = RECORD                                      //Speichert die entstandenen Ergebnisse
    glstring, glausgabe: String;
    fehler: Byte;
  END;

  Tgltypen = (Frage, Plus, Pfeil, Zahl, Element);     //Die einzelnen Zeichentypen in einer Gleichung

  Tgleichung = RECORD                   //ein ARRAY dieses Types stellt eine chemische Gleichung dar
    typ: Tgltypen;                                                      //Art des Teils der Gleichung
    nr: Byte;                                           //Zahl oder Elementnummer oder Fragezeichenart
```

```
    edit: Boolean;                                  //True wenn an diesem Zeichen editiert werden darf
  END;

CONST
  elementehinzu: Boolean = False;     //Falls ich mein Programm noch für elementhinzufügen erweitere
  immertesten: Boolean = True;//false Programm versucht nicht möglichen Molekuele weiter umzuwandeln
  maxdeep: Byte = 20;                                                  //Maximale Tiefe des Suchbaumes
  maxlen: Byte = 30;                                                    //Maximale Länge der Gleichung
VAR
  persys: ARRAY OF atom;                                                                //das Periodensystem
  gleichung: ARRAY OF Tgleichung;                                                 //die chemische Gleichung
  deep: Word;                                                    //die momentane Suchtiefe des Programmes
  ergebnisse: ARRAY OF Tergebnisse;                             //die bei der Suche gefundenen Ergebnisse


PROCEDURE brute;
FUNCTION molekuelcheck: Boolean;
PROCEDURE atomeinfuegen(seite: Integer; pos: Byte; nr: Byte);
FUNCTION anzahlvergleich: Tatome;
PROCEDURE gleichungausgeben;
PROCEDURE reduce;

IMPLEMENTATION


PROCEDURE brute;                                //die Prozedur, welche die Rekursion in die Tiefe treibt
VAR
  inp: Tatome;
BEGIN
  IF deep < maxdeep THEN                                    //wenn die Tiefe noch nicht überschritten ist
  BEGIN
    inc(deep);
    inp := anzahlvergleich;                  //Test ob die beiden Gleichungsseiten ausgeglichen sind
    IF (inp.nr <> 255) THEN                                                                 //wenn nein dann
    BEGIN       //wenn maximale Länge nicht überschritten --> an allen möglichen Stellen ausgleichen
      IF length(gleichung) < maxlen THEN atomeinfuegen(inp.anz, 0, inp.nr);
    END ELSE
    BEGIN                                                 //sonst Testen ob alle Molekuele möglich sind
      IF molekuelcheck THEN reduce;                                          //wenn ja Gleichung speichern
    END;
    dec(deep);
  END;
END;



FUNCTION ggt(a, b: Word): Word;                                                 //Bildet den GGT zweier Zahlen
VAR
  d: Word;
BEGIN
  REPEAT
    IF b <> 0 THEN d := a Mod b;
    a := b;
    b := d;
  UNTIL b = 0;
  ggt := a;
END;



PROCEDURE reduce;                    //fasst die einzelnen Molekuele zusammen, gibt die Gleichung aus
LABEL                                                                   //und erstellt HTML Gleichungen
  restart1, restart2;
VAR
  i, k, l, m, t, anz, start, ende: Byte;                                          //massig Zählvervariablen
  gl2: ARRAY OF Tgleichung;                                  //Nimmt die Gleichung für das Kürzen auf
  gas, vorhanden: Boolean;                                //Molekuel ist Gas, Ergebnis schon vorhanden
  ergebnis: String;                                                  //die entstandene StringGleichung
BEGIN

  setlength(gl2, length(gleichung));                                                      //gl2 := gleichung
  IF length(gl2) <> 0 THEN
    FOR i := 0 TO length(gl2) - 1 DO
    BEGIN
      gl2[i] := gleichung[i];
    END;

  anz := 0;
  i := 0;
  t := 0;
  k := 0;
  gas := False;
  REPEAT                                                  //Schleife für i bis zum Gleichungsende
    IF gl2[i].typ = element THEN                                  //Zeigt i gerade auf ein Element
    BEGIN
      inc(anz);                                                                 //Elementanzahl erhöhen
      gas := persys[gl2[i].nr].gas;                                  //Gas Eingenschaften speichern
      IF gl2[i + 1].typ = zahl THEN                                //Wenn Folgeobjekt eine Zahl ist
      BEGIN
        IF t = 0 THEN t := gl2[i + 1].nr          //t=0 ensp. erstes ELement --> t := Elementanzahl
        ELSE
          t := ggt(t, gl2[i + 1].nr);                              //sonst t := ggt(Elementanzahl, t)
      END ELSE
        t := 1;                                                 //wenn Folgeobjekt keine Zahl t := 1
    END;
```

```
                                                          //Wenn i an den Rand eines Molekuels stößt
  IF (gl2[i].typ In [Plus, Pfeil]) Or (i = length(gl2)) THEN
  BEGIN
    IF (anz = 1) And gas THEN                                  //Molekuel ist ein Gas (H2, N2, ...)
    BEGIN
      t := t Div 2;                                                    //damit der 2er stehenbleibt
    END;
    IF (t > 1) THEN    //wenn ein ggT > 1 für alle Indizes der Atome existiert (z.B. H4O2 -> 2H2O)
    BEGIN
      IF gl2[k].typ = zahl THEN                     //wenn das Molekül schon mit einer Zahl beginnt
      BEGIN                                                          //wird diese einfach erhöht
        gl2[k].nr := gl2[k].nr * t;
      END ELSE
      BEGIN                                       //sonst wird die Gleichung um eine Stelle erweitert und
        setlength(gl2, length(gl2) + 1);                          //die Zahl vor dem Moleül eingeschoben
        FOR l := length(gl2) - 1 DOWNTO k DO gl2[l] := gl2[l - 1];
        gl2[k].typ := zahl;
        gl2[k].nr := t;
        inc(i);
      END;
      inc(k);
      REPEAT                    //nun wird die Gleichung durchlaufen und alle Indizes durch k geteilt
        IF gl2[k].typ = zahl THEN
        BEGIN
          gl2[k].nr := gl2[k].nr Div t;
          IF gl2[k].nr = 1 THEN              //wenn dadurch ein Index 1 entsteht, wird er entfernt
          BEGIN
            IF k <> length(gl2) THEN FOR l := k TO length(gl2) - 2 DO gl2[l] := gl2[l + 1];
            setlength(gl2, length(gl2) - 1);
          END;
        END;
        inc(k);
      UNTIL k = i;                        //Wird solange durchgeführt bis k ans Molekülende i kommt
    END;
    k := i + 1;                                        //Variablen für das nächste Molekül vorbereiten
    t := 0;
    anz := 0;
    gas := False;
  END;
  inc(i);
UNTIL i >= length(gl2) + 1;

restart1:                                                    //Restart beim Suchen gleicher Moleküle
                              //Aufgabe des Programmabschnittes: z.B H2O + H2O in 2H2O umzuwandeln
start := 0;
ende := 0;                                                //Start und Ende der Linken Seite feststellen
WHILE gl2[ende].typ <> Pfeil DO inc(ende);
k := 0;
REPEAT                                                      //Schleife die wenn k ende erreicht endet
  WHILE gl2[k].typ <> Element DO inc(k);                  //Startelement eines Molekuels herausfinden
  l := k + 1;
  WHILE (gl2[l].typ In [Element, Zahl, Frage]) And (l < ende) DO inc(l);                //Endpunkt

  IF l < ende THEN REPEAT
      WHILE gl2[l].typ <> Element DO inc(l);                //l bis zum Start des nächsten vorrücken
      i := 0;                                 //die zwei markierten Elemente Stück für Stück vergleichen
      WHILE Not ((gl2[l + i].typ In [Plus, Pfeil])) And (gl2[l + i].typ = gl2[k + i].typ) And
        (gl2[l + i].nr = gl2[k + i].nr) DO inc(i);
                                                          //wenn sich die Elemente entsprochen haben
      IF (((gl2[l + i].typ In [Plus, Pfeil])) And (gl2[k + i].typ In [Plus, Pfeil])) THEN
      BEGIN

        t := l;                                      //ermittlen welche Anzahl das 2te Molekül besitzt
        WHILE Not (gl2[t].typ In [Plus, zahl]) And (t >= start + 1) DO dec(t);
        IF gl2[t].typ = Zahl THEN anz := gl2[t].nr
        ELSE
          anz := 1;
        WHILE gl2[t].typ <> Plus DO dec(t);
                                                                  //das 2te Molekül entfernen
        FOR m := t TO length(gl2) - 1 DO gl2[m] := gl2[m + l + i - t];
        setlength(gl2, length(gl2) + t - l - i);

        t := k;                                                  //zum Start des Moleküls wandern
        WHILE Not (gl2[t - 1].typ In [Plus, zahl]) And (t > start) DO dec(t);
        IF gl2[t - 1].typ = Zahl THEN    //wenn sich dort schon eine Zahl befindet einfach erhöhen
        BEGIN
          inc(gl2[t - 1].nr, anz);
        END ELSE
        BEGIN                                       //sonst Platz schaffen und die passende Zahl einfügen
          setlength(gl2, length(gl2) + 1);
          FOR m := length(gl2) - 1 DOWNTO t + 1 DO gl2[m] := gl2[m - 1];
          gl2[t].typ := zahl;
          gl2[t].nr := anz + 1;
        END;

        GOTO restart1;
                //zu Restart springen (weiteres einrücken wollte ich keinem Hirn nicht mehr antun)
      END;
                                                          //zum nächsten Vergleichelement wandern
      WHILE (gl2[l].typ In [Element, Zahl, Frage]) And (l < ende) DO inc(l);
    UNTIL l >= ende;
                                                 //zum nächsten zu vergleichenden Element wandern
  WHILE (gl2[k].typ <> Plus) And (k < ende + 1) DO inc(k);
UNTIL k >= ende;
```

```
  start := ende + 1;                                  //start und ende der Produktseite herausfinden

  restart2:

  ende := length(gl2) - 1;
  k := start;


  REPEAT                                                          //Schleife die wenn k ende erreicht endet
    WHILE (gl2[k].typ <> Element) And (k < ende) DO inc(k);//Startelement eines Molekuels herausfind
    l := k + 1;
    WHILE (gl2[l].typ In [Element, Zahl, Frage]) And (l < ende) DO inc(l);                //Endpunkt

    IF l < ende THEN REPEAT                                        //l bis zum Start des nächsten vorrücken
        WHILE (gl2[l].typ <> Element) And (l < ende) DO inc(l);
        i := 0;
                                              //die zwei markierten Elemente Stück für Stück vergleichen
        WHILE Not ((gl2[l + i].typ In [Plus, Pfeil])) And (gl2[l + i].typ = gl2[k + i].typ) And
          (gl2[l + i].nr = gl2[k + i].nr) DO inc(i);
                                                       //wenn sich die Elemente entsprochen haben
        IF (((gl2[k + i].typ In [Plus, Pfeil])) And ((gl2[l + i].typ In [Plus, Pfeil]) Or
          (l + i = ende + 1))) THEN
        BEGIN

          t := l;                                    //ermittlen welche Anzahl das 2te Molekül besitzt
          WHILE Not (gl2[t].typ In [Plus, zahl]) And (t >= start + 1) DO dec(t);
          IF gl2[t].typ = Zahl THEN anz := gl2[t].nr
          ELSE
            anz := 1;
          WHILE gl2[t].typ <> Plus DO dec(t);
                                                                                  //das 2te Molekül entfernen
          FOR m := t TO length(gl2) - 1 DO gl2[m] := gl2[m + l + i - t];
          setlength(gl2, length(gl2) + t - l - i);

          t := k;                                                    //zum Start des Moleküls wandern
          WHILE Not (gl2[t - 1].typ In [Plus, zahl]) And (t > start) DO dec(t);

          IF gl2[t - 1].typ = Zahl THEN    //wenn sich dort schon eine Zahl befindet einfach erhöhen
          BEGIN
            inc(gl2[t - 1].nr, anz);
          END ELSE
          BEGIN                                  //sonst Platz schaffen und die passende Zahl einfügen
            setlength(gl2, length(gl2) + 1);
            FOR m := length(gl2) - 1 DOWNTO t + 1 DO gl2[m] := gl2[m - 1];
            gl2[t].typ := zahl;
            gl2[t].nr := anz + 1;
          END;

          GOTO restart2;       //zu Restart springen (den Programmtiel hab ich hald einfach kopiert)
        END;
                                                             //zum nächsten Vergleichelement wandern
        WHILE (gl2[l].typ In [Element, Zahl, Frage]) And (l < ende) DO inc(l);
      UNTIL l >= ende;
                                                //zum nächsten zu vergleichenden Element wandern
    WHILE (gl2[k].typ <> Plus) And (k < ende + 1) DO inc(k);
  UNTIL k >= ende;

  ergebnis := '';                          //den ARRAY gl2 wieder in eine STRING-Gleichung umwandeln

  IF length(gl2) <> 0 THEN
    FOR i := 0 TO length(gl2) - 1 DO
      CASE gl2[i].typ OF
        Plus: ergebnis := ergebnis + ' + ';
        Pfeil: ergebnis := ergebnis + ' --> ';
        Zahl: ergebnis := ergebnis + IntToStr(gl2[i].nr);
        Element: ergebnis := ergebnis + persys[gl2[i].nr].Name;
      END;


  vorhanden := False;                          //testen ob dieser String schon einmal errechnet wurde
  IF length(ergebnisse) <> 0 THEN
    FOR i := 0 TO length(ergebnisse) - 1 DO IF ergebnisse[i].glstring = ergebnis THEN
        vorhanden := True;

  IF Not (vorhanden) THEN                       //wenn nicht den String zu den Ergebnissen hinzufügen
  BEGIN
    setlength(ergebnisse, length(ergebnisse) + 1);
    ergebnisse[length(ergebnisse) - 1].glstring := ergebnis;
    writeln(ergebnis);

    ergebnis := '';                                                        //HTML Ergebnis erstellen
    vorhanden := True;
    IF length(gl2) <> 0 THEN
      FOR i := 0 TO length(gl2) - 1 DO
        CASE gl2[i].typ OF
          Plus:
            BEGIN
              ergebnis := ergebnis + ' + ';
              vorhanden := True;
            END;
          Pfeil:
            BEGIN
              ergebnis := ergebnis + ' ⟶ ';
              vorhanden := True;
            END;
```

```
          Zahl: IF vorhanden THEN ergebnis := ergebnis + IntToStr(gl2[i].nr)
            ELSE
              ergebnis := ergebnis + ' + IntToStr(gl2[i].nr) + ';
            Element:
            BEGIN
              ergebnis := ergebnis + persys[gl2[i].nr].Name;
              vorhanden := False;
            END;
        END;
    ergebnisse[length(ergebnisse) - 1].glausgabe := ergebnis;              //HTML Ergebnis speichern
  END;

END;



PROCEDURE gleichungausgeben;                  //verwendete ich zum debugging ;-) und  immer noch - um
VAR                             //dem Benutzer am Start zu fragen ob er die Frage richtig gestellt hat
  i: Byte;
BEGIN                                          //schreibt einfach die Gleichung auf den Bildschirm
  IF length(gleichung) <> 0 THEN
    FOR i := 0 TO length(gleichung) - 1 DO
      CASE gleichung[i].typ OF
        Plus: Write(' + ');
        Pfeil: Write(' --> ');
        Frage: Write('?');
        Zahl: Write(gleichung[i].nr);
        Element: Write(persys[gleichung[i].nr].Name);
      END;
  writeln;
END;



FUNCTION molekuelcheck: Boolean;//testet ob Moleküle laut ihren Oxidationszahlen bestehen können und
VAR                                       //ertmittlent bei Fehlern mögliche Ausgleichsvarianten
  i, k, l, m, atome, startpos: Byte;                                        //Zählervariablen
  oxis: ARRAY OF Toxicheck;    //beschreibt ein Molekül mit seinen Elementen , Oxidationszahlen, ...
  ausgmoeg: ARRAY OF Byte;     //die verschiedenen ELementente mit denen ausgeglichen werden könnte
  ueberlauf, exist, vorhanden: Boolean;
  v, oxidat: Integer;
BEGIN

  molekuelcheck := True;
  i := 0;
  k := 0;
  atome := 0;

  REPEAT

    IF gleichung[i].typ = Element THEN inc(Atome);                     //zählt die enthaltenen Atome
                                                  //startet wenn das Ende eins Moleküls erreicht ist
    IF (gleichung[i].typ In [Plus, Pfeil]) Or (i >= length(gleichung) - 1) THEN
    BEGIN
      IF i = length(gleichung) - 1 THEN inc(i);
      IF atome = 0 THEN
      BEGIN
        molekuelcheck := False
      END ELSE IF atome = 1 THEN                                          //ein einatomiges Molekül
      BEGIN
        startpos := k;
        WHILE gleichung[k].typ <> Element DO inc(k);  //ist zu diesem Zeitpunkt das Molekül korrekt?
        IF persys[gleichung[k].nr].gas And ((gleichung[k + 1].typ = Frage) Or
          ((gleichung[k + 1].typ = Zahl) And (gleichung[k + 1].nr Mod 2 <> 0)) Or
          (gleichung[k + 1].typ In [Plus, Pfeil]) Or (k = length(gleichung) - 1)) THEN
          molekuelcheck := False;                          //wenn Änderungsmöglichkeit vorhanden dann
        IF persys[gleichung[k].nr].gas And ((gleichung[k + 1].typ = Frage) Or
          ((gleichung[k + 1].typ = Zahl) And gleichung[k + 1].edit And
          (gleichung[k + 1].nr Mod 2 <> 0))) THEN
        BEGIN
          atomeinfuegen(0, startpos, gleichung[k].nr);   //an der passenden Stelle ein Atom einfügen
        END;

      END ELSE                                                            //mehr als eine Atomsorte
      BEGIN
        startpos := k;

        IF i = length(gleichung) - 1 THEN inc(i);                        //den oxis Array "befüllen"
        setlength(ausgmoeg, 0);
        setlength(oxis, atome);
        atome := 0;
        REPEAT                                                            //neue Elemente hinzufügen
          IF gleichung[k].typ = Element THEN
          BEGIN
            oxis[atome].mol := gleichung[k].nr;
            oxis[atome].edit := False;
            IF gleichung[k + 1].typ = Zahl THEN             //die Anzahl deren Vorkommnis festhalten
            BEGIN
              oxis[atome].anz := gleichung[k + 1].nr;
              oxis[atome].edit := True;
            END ELSE
              oxis[atome].anz := 1;                       //feststellen ob sie vermehrt werden dürfen
            IF gleichung[k + 1].typ = Frage THEN oxis[atome].edit := True;
```

```
      oxis[atome].oxi := 0;                          //momentane oxidationszahl auf -7 setzen
      inc(atome);
    END;
    inc(k);
  UNTIL k = i;

  exist := False;
  REPEAT

    v := atome - 1;                                  //v auf das letzte Atom im oxis ARRAY setzen
                          //jetzt kommt eine lustige Schleife die alle Oxidationskombinationen
    ueberlauf := True;                    //durchtestet und mir eine Rekursion erspart hat!!!
    REPEAT
      IF ueberlauf THEN inc(oxis[v].oxi);      //wenn überlauf im hinteren Atom --> oxi erhöhen
{   if (v<>atome-1) and (oxis[v+1].oxi>7) and (oxis[v].oxi<7)  and not(
      (persys[oxis[v].mol].name = 'O') and (persys[oxis[v+1].mol].name = 'H')
       and (oxis[v].anz = oxis[v+1].anz))
      then oxis[v].oxi := 8;  }

                    //ist der Hintermann ein Geber muss das aktuelle Atom auch ein Geber sein!
      IF (v <> atome - 1) And (oxis[v + 1].oxi > 7) And (oxis[v].oxi < 7) THEN oxis[v].oxi:=8;

                   //tritt die OH Gruppe als Ausnahme auf wird die vorherige Regel überstimmt
      IF (v = atome - 2) And
        (persys[oxis[v].mol].Name = 'O') And (persys[oxis[v + 1].mol].Name = 'H') And
        (oxis[v].anz = oxis[v + 1].anz) THEN
      BEGIN
        oxis[v].oxi := 5;                    //Passende Oxidationszahlen für OH Gruppe festlegen
        oxis[v + 1].oxi := 8;
        ueberlauf := True;

      END ELSE
      BEGIN

        ueberlauf := False;
                                 //oxidationszahlen so lange erhöhen bis für das Atom möglich
        WHILE Not (oxis[v].oxi In persys[oxis[v].mol].oxiz) DO
        BEGIN
          inc(oxis[v].oxi);
          IF oxis[v].oxi > 14 THEN                                    //wenn oxi > 14 (+7) dann
          BEGIN
            oxis[v].oxi := 1;              //oxidationszahlen wieder von vorne beginnen lassen
            ueberlauf := True;                                              //Überlauf setzen
          END;
        END;

      END;

      dec(v);

    UNTIL (v = -1);                  //wenn die oxidationszahlen für alle Atome bestimmt wurden
    IF Not (ueberlauf) THEN  //und kein überlauf (überlauf tritt bei dem letzen durchlauf ein)
    BEGIN

      oxidat := 0;
      IF atome <> 0 THEN                //Die Summe der Oxidationszahlen für das Molekül bilden
        FOR v := 0 TO atome - 1 DO inc(oxidat, (oxis[v].oxi - 7) * oxis[v].anz);
      IF oxidat = 0 THEN exist := True;

      IF elementehinzu THEN          //hab ich bis jetzt noch nicht eingebaut und werd ich auch
      BEGIN                    //nicht mehr, da es mir mitlerweile als nicht sinnvoll erscheint

      END ELSE
      BEGIN                                        //Wenn die Oxidationszahlensumme nicht 0 ist dann
        IF (oxidat <> 0) And (atome <> 0) THEN
          FOR l := 0 TO atome - 1 DO IF oxis[l].edit And ((oxis[l].oxi - 7) * oxidat < 0) THEN
            BEGIN       //alle Elemente deren Oxidationszahl gerade ein anderes Vorzeichen als
              //die Summe hat, zum ausgmoeg ARRAY hinzufügen (wenn sie nicht schon drin sind)
              vorhanden := False;
              IF length(ausgmoeg) <> 0 THEN
                FOR v := 0 TO length(ausgmoeg) - 1 DO IF ausgmoeg[v] = oxis[l].mol THEN
                  vorhanden := True;

              IF Not (vorhanden) THEN
              BEGIN
                setlength(ausgmoeg, length(ausgmoeg) + 1);
                ausgmoeg[length(ausgmoeg) - 1] := oxis[l].mol;

              END;
            END;
      END;
    END;
  UNTIL ueberlauf;                           //das durchtesten der Oxidationszahlen hat ein ENDE

  IF Not (exist) THEN        //wenn nicht wenigstens einmal die oxidationszahlsumme 0 entstand
  BEGIN
    molekuelcheck := False;                                                    //false rückgeben
  END ELSE
  BEGIN          //ebenso wenn sich im Molekül eine OH Gruppe befindet die nicht zusammensteht
    k := 255;
    l := 255;
    FOR m := 0 TO length(oxis) - 1 DO
    BEGIN
      IF persys[oxis[m].mol].Name = 'O' THEN k := m;
      IF persys[oxis[m].mol].Name = 'H' THEN l := m;
```

```
          END;

          IF (k <> 255) And (l <> 255) THEN
            IF (oxis[k].anz = oxis[l].anz) THEN IF k <> l - 1 THEN molekuelcheck := False;

        END;

        IF Not (exist) Or immertesten THEN   //wenn ein kein richtiges Molekül oder immertesten dann
        BEGIN
          IF length(ausgmoeg) <> 0 THEN           //alle Ausgleichsmöglichkeiten dem Molekül anfügen
            FOR l := 0 TO length(ausgmoeg) - 1 DO atomeinfuegen(0,startpos, ausgmoeg[l]);
        END;
      END;
      atome := 0;
      k := i + 1;
    END;                                          //jetzt kommen auch noch alle anderen Moleküle dran
    inc(i);
  UNTIL i >= length(gleichung);

END;


                                  //fügt Atome vom Typ nr an mit ? gekennzeichneten Stellen ein,
                          //wobei man zwischen Seitenbeschränkung und Molekülbeschränkung wählen kann
PROCEDURE atomeinfuegen(seite: Integer; pos: Byte; nr: Byte);
VAR
  i, k, l, anzahl: Byte;
  enthalten, geteilt: Boolean;
BEGIN
  geteilt := False;
  i := 0;

  IF seite <> 0 THEN                                                  //wenn Seitenbeschränkung aktiv
  BEGIN
    IF seite > 0 THEN
    BEGIN                                          //wenn seite positiv wird i auf den Pfeil vorgefahren
      WHILE gleichung[i].typ <> Pfeil DO inc(i);
      inc(i);
    END;
  END ELSE                                                            //wenn Molekülbeschränkung
  BEGIN
    i := pos;
    WHILE gleichung[i].typ In [Pfeil, Plus] DO inc(i);          //zum wirklichen Molekülstart fahren
  END;


  REPEAT    //Schleife die beim Pfeil, am Gleichungsende und wenn Molekülbeschränkung aktiv abbricht

    l := i;                                                                 //Startpunkt speichern
    enthalten := False;
    WHILE (length(gleichung) > l) And Not (gleichung[l].typ In [plus, pfeil]) DO
    BEGIN
      IF (gleichung[l].typ = Element) And (gleichung[l].nr = nr) THEN
      BEGIN                                          //Test ob das Element im Molekül enthalten ist
        enthalten := True;
        k := l;
      END;
      inc(l);

    END;

    //einfügen
    IF enthalten THEN                                          //wenn das Element schon im Molekül
    BEGIN
      i := l;                                                 //Folgezeichen ist Zahl und editierbar
      IF gleichung[k + 1].edit And (gleichung[k + 1].typ = Zahl) THEN
      BEGIN                                                              //-->Zahl um eins
erhöhen
        inc(gleichung[k + 1].nr);

        brute;
        dec(gleichung[k + 1].nr);
      END ELSE IF gleichung[k + 1].typ = Frage THEN              //Folgezeichen ist ein Fragezeichen
      BEGIN
        setlength(gleichung, length(gleichung) + 1);
        FOR l := length(gleichung) - 1 DOWNTO k + 2 DO gleichung[l] := gleichung[l - 1];
        gleichung[k + 1].typ := Zahl;
        gleichung[k + 1].nr := 2;                      //Gleichung um eins verlängern und 2 einfügen
        gleichung[k + 1].edit := True;

        brute;
        FOR l := k + 2 TO length(gleichung) - 1 DO gleichung[l - 1] := gleichung[l];
        setlength(gleichung, length(gleichung) - 1);
      END ELSE
      BEGIN

        WHILE Not ((k = 0) Or (gleichung[k - 1].typ In [Pfeil, Plus])) DO dec(k);

        IF gleichung[k].typ = frage THEN           //Wenn das Molekül mit einem Fragezeichen startet
        BEGIN
          setlength(gleichung, length(gleichung) + 1);
          FOR l := length(gleichung) - 1 DOWNTO k + 1 DO gleichung[l] := gleichung[l - 1];
          gleichung[k].typ := Zahl;
          gleichung[k].nr := 2;                  //Gleichung um eins verlängern und 2 vor ? einfügen
          gleichung[k].edit := True;
```

```
          brute;
          FOR l := k + 1 TO length(gleichung) - 1 DO gleichung[l - 1] := gleichung[l];
          setlength(gleichung, length(gleichung) - 1);

        END;
        IF (gleichung[k].typ = Zahl) And gleichung[k].edit THEN
        BEGIN                                        //editierbare Zahl --> diese um eins erhöhen
          inc(gleichung[k].nr);

          brute;
          dec(gleichung[k].nr);
        END;


      END;

    END ELSE
    BEGIN                                                 //Element noch nicht im Molekül vorhanden
      REPEAT
        IF gleichung[i].typ = Frage THEN                                  //ein einfaches Fragezeichen
        BEGIN
                    //darauf folgt eine Zahl die mit der gewünschten Ausgleichsanzahl übereinstimmt
          IF (i <> length(gleichung) - 1) And (gleichung[i + 1].typ = Zahl) THEN
          BEGIN
            setlength(gleichung, length(gleichung) + 1);
            FOR k := length(gleichung) - 1 DOWNTO i + 2 DO gleichung[k] := gleichung[k - 1];
            gleichung[i + 1].typ := Element;
            gleichung[i + 1].nr := nr;                    //Gleichung verlängern und Element einfügen
            gleichung[i + 1].edit := True;

            brute;
            FOR k := i + 2 TO length(gleichung) - 1 DO gleichung[k - 1] := gleichung[k];
            setlength(gleichung, length(gleichung) - 1);
            inc(i);

          END;

                                                                                  //immer getestet wird:
          setlength(gleichung, length(gleichung) + 2);
          FOR k := length(gleichung) - 1 DOWNTO i + 2 DO gleichung[k] := gleichung[k - 2];
          gleichung[i + 1].typ := Element;
          gleichung[i + 1].nr := nr;                                //Element + Fragezeichen einfügen
          gleichung[i + 1].edit := True;
          gleichung[i + 2].typ := Frage;
          gleichung[i + 2].nr := 1;
          gleichung[i + 2].edit := True;

          brute;
          FOR k := i + 3 TO length(gleichung) - 1 DO gleichung[k - 2] := gleichung[k];
          setlength(gleichung, length(gleichung) - 2);

                                                  //wenn ein Doppelfragezeichen eingegeben wurde:
          IF (gleichung[i].nr > 1) And (Not (gleichung[i + 1].typ In [Pfeil, Plus]) Or
            Not (geteilt)) THEN
          BEGIN
            IF gleichung[i + 1].typ In [Pfeil, Plus] THEN geteilt := True;
            setlength(gleichung, length(gleichung) + 4);
            FOR k := length(gleichung) - 1 DOWNTO i + 5 DO gleichung[k] := gleichung[k - 4];
            gleichung[i].nr := 1;
            gleichung[i + 4].typ := Frage;
            gleichung[i + 4].nr := 2;
            gleichung[i + 3].typ := Element;               //In die Gleichung "+?Element?" einfügen
            gleichung[i + 3].nr := nr;
            gleichung[i + 3].edit := True;
            gleichung[i + 2].typ := Frage;
            gleichung[i + 1].typ := Plus;

            brute;
            gleichung[i].nr := 2;
            FOR k := i + 5 TO length(gleichung) - 1 DO gleichung[k - 4] := gleichung[k];
            setlength(gleichung, length(gleichung) - 4);
          END;
        END;
        inc(i);                                                          //bis zum Molekülende
      UNTIL (gleichung[i].typ In [Plus, Pfeil]) Or (i > length(gleichung));
    END;
    inc(i);                                      //bis zum Gleichungsende oder Molekülende (seite = 0)
  UNTIL (gleichung[i - 1].typ = Pfeil) Or (i > length(gleichung)) Or ((seite = 0));

END;


FUNCTION anzahlvergleich: Tatome;   //ermittelt ob rechte und linke Gleichunsseite ausgeglichen sind
VAR
  atome: ARRAY OF Tatome;     //Speichert die Anzahl der Atome auf der rechen+linken Gleichungsseite
  i, k, mom, multi: Byte;
  neu: Boolean;
  seite: Integer;
BEGIN
  setlength(atome, 0);

  multi := 1;                                                    //Molekülanzahl auf 1 setzen
  neu := True;                                                    //--> neues Molekül beginnt
```

```
  seite := 1;                                                          //Linke Gleichungsseite
  IF length(gleichung) <> 0 THEN
    FOR i := 0 TO length(gleichung) - 1 DO                    //Schleife für alle Gleichungschars
      CASE gleichung[i].typ OF
        Plus:                                                //Plus --> neues Element, Molekülanzahl = 1
          BEGIN
            neu := True;
            multi := 1;
          END;
        Pfeil:                                                 //Pfeil --> Plus + rechte Gleichungsseite
          BEGIN
            neu := True;
            seite := -1;
            multi := 1
          END;
        Zahl: IF neu THEN multi := gleichung[i].nr;          //Zahl + neu --> Molekülanzahl := Zahl
        Element:
          BEGIN                                                                          //Element

            neu := False;
            mom := length(atome);            //Testen ob sich Molekül schon im atome ARRAY befindet
            IF length(atome) <> 0 THEN FOR k := 0 TO length(atome) - 1 DO
                IF atome[k].nr = gleichung[i].nr THEN mom := k;
            IF mom = length(atome) THEN                  //wenn noch nicht vorhanden Array erweitern
            BEGIN
               setlength(atome, mom + 1);
              atome[mom].nr := gleichung[i].nr;
              atome[mom].anz := 0;
            END;                  //wenn Folgechar eine Zahl ist zahl*Molekülanzahl addieren/subtrahieren
            IF (i < length(gleichung) - 1) And (gleichung[i + 1].typ = Zahl) THEN
              inc(atome[mom].anz, multi * gleichung[i + 1].nr * seite)
            ELSE                                             //sonst Molekülanzahl addieren/subtrahieren
              inc(atome[mom].anz, multi * seite)
          END;
      END;


  anzahlvergleich.nr := 255;
  IF length(atome) <> 0 THEN                          //suchen ob die Anzahl eines Moleküls <> 0 ist
    FOR i := 0 TO length(atome) - 1 DO
    BEGIN
                                          //wenn ja wird Elementnummer + Abweichung zurückgeliefert
      IF atome[i].anz <> 0 THEN anzahlvergleich := atome[i];
    END;
  setlength(atome, 0);
END;

END.                                                                 //Ende gut alles gut (Hoff ich mal)
```

Periodensystem.pas

```
UNIT periodensystem;

INTERFACE

USES
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

TYPE
  TForm1 = CLASS(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    Edit1: TEdit;
    Label1: TLabel;
    CheckBox13: TCheckBox;
    Edit2: TEdit;
    Label2: TLabel;
    ComboBox1: TComboBox;
    Button2: TButton;
    PROCEDURE FormCreate(Sender: TObject);
    PROCEDURE ListBox1Click(Sender: TObject);
    PROCEDURE Button1Click(Sender: TObject);
    PROCEDURE FormDestroy(Sender: TObject);
    PROCEDURE Edit1KeyDown(Sender: TObject; VAR Key: Word;
      Shift: TShiftState);
    PROCEDURE Button2Click(Sender: TObject);
  Private
    { Private-Deklarationen }
  Public
    { Public-Deklarationen }
  END;

TYPE
  atom = RECORD                                      //Typ speichert die benötigten Atomeigenschaften
    Name: String[2];
    oxiz: SET OF Byte;
    gas, akz, don: Boolean;
  END;

VAR
  Form1: TForm1;
```

```
VAR
  persys: ARRAY OF atom;
  datei: FILE OF atom;

  ausgewaehlt: Integer = -1;

IMPLEMENTATION

{$R *.DFM}

PROCEDURE TForm1.FormCreate(Sender: TObject);       //lest beim Programmstart das Periodensystem ein
BEGIN
  assignfile(datei, 'persys.dat');
  reset(datei);
  IF Not (EOF(datei)) THEN
    REPEAT
      setlength(persys, length(persys) + 1);
      Read(datei, persys[length(persys) - 1]);
      listbox1.Items.Add(persys[length(persys) - 1].Name);
    UNTIL EOF(datei);
  closefile(datei);
END;


PROCEDURE TForm1.ListBox1Click(Sender: TObject);
VAR
  i: Word;
  s, s2: String;
BEGIN
  IF ausgewaehlt <> -1 THEN                 //wenn Element ausgewählt dann müssen die möglicherweise
    WITH persys[ausgewaehlt] DO                                   //veränderten Daten gespeichert werden
    BEGIN
      Name := edit1.Text;                                                              //Name speichern
      listbox1.Items.Strings[ausgewaehlt] := Name;
      oxiz := [];
      s := edit2.Text;
      IF combobox1.ItemIndex = 0 THEN                                  //Donator, Akzeptor eigenschaften
      BEGIN
        don := True;
        akz := False;
      END ELSE IF combobox1.ItemIndex = 1 THEN
      BEGIN
        don := False;
        akz := True;
      END ELSE
      BEGIN
        don := True;
        akz := True;
      END;
                                                                                     //Oxidationszahlen
      IF s <> '' THEN REPEAT
          i := pos(';', s);

          oxiz := oxiz + [StrToInt(copy(s, 0, i - 1)) + 7];
          s := copy(s, i + 1, 100);

        UNTIL s = '';

      gas := checkbox13.Checked;                                                     //Gas-Moleküle
    END;

  FOR i := 0 TO listbox1.Items.Count - 1 DO IF listbox1.Selected[i] THEN ausgewaehlt := i;
  IF ausgewaehlt <> -1 THEN
    WITH persys[ausgewaehlt] DO
    BEGIN
      edit1.Text := Name;                                                          //Name ausgeben
      edit2.Text := '';                                          //Oxidationszahlenstring erstellen
      FOR i := 0 TO 14 DO IF i In oxiz THEN edit2.Text := edit2.Text + IntToStr(i - 7) + ';';

      checkbox13.Checked := gas;                                                   //Gas-Molekül
      IF akz And don THEN combobox1.ItemIndex := 2               //Akzeptor, Donnator eigenschaften
      ELSE IF akz THEN combobox1.ItemIndex := 1
      ELSE IF don THEN combobox1.ItemIndex := 0;
    END;

END;



PROCEDURE TForm1.Button1Click(Sender: TObject);                          //klick auf anfügen Button
VAR
  i: Byte;
  s, s2: String;
BEGIN
  setlength(persys, length(persys) + 1);                                //Periodensystem verlängern
  listbox1.Items.Add(edit1.Text);                                                    //und Listbox

  ausgewaehlt := listbox1.Items.Count - 1;
  WITH persys[ausgewaehlt] DO
  BEGIN                                                                          //Namen übertragen
    Name := edit1.Text;
    listbox1.Items.Strings[ausgewaehlt] := Name;
    oxiz := [];
    s := edit2.Text;
```

```
    IF combobox1.ItemIndex = 0 THEN                    //Donator, Akzeptor eigenschaften speichern
    BEGIN
      don := True;
      akz := False;
    END ELSE IF combobox1.ItemIndex = 1 THEN
    BEGIN
      don := False;
      akz := True;
    END ELSE
    BEGIN
      don := True;
      akz := True;
    END;

    IF s <> '' THEN REPEAT                             //Oxidationszahlen aus String extrahieren
        i := pos(';', s);

        oxiz := oxiz + [StrToInt(copy(s, 0, i - 1)) + 7];
        s := copy(s, i + 1, 100);

      UNTIL s = '';

    gas := checkbox13.Checked;                         //Gaseigenschaften aus der Checkbox
  END;


END;

PROCEDURE TForm1.FormDestroy(Sender: TObject);//Speichert das veränderte Periodensystem in
persys.dat
VAR
  i: Word;
BEGIN

  assignfile(datei, 'persys.dat');
  rewrite(datei);
  FOR i := 0 TO length(persys) - 1 DO
  BEGIN
    Write(datei, persys[i]);
  END;

  closefile(datei);
END;

PROCEDURE TForm1.Edit1KeyDown(Sender: TObject; VAR Key: Word; Shift: TShiftState);
BEGIN                              //Wenn ich ein Element in die Textbox tippe und Enter drücke wird
  IF key = VK_RETURN THEN                              //das Element automatisch in die Liste aufgenommen
  BEGIN
    button1Click(Sender);
    edit1.SelectAll;
  END;
END;

PROCEDURE TForm1.Button2Click(Sender: TObject);      //löscht letztes Element aus dem Periodensystem
BEGIN
  setlength(persys, length(persys) - 1);
  listbox1.items.Delete(listbox1.items.Count - 1);
END;

END.
```