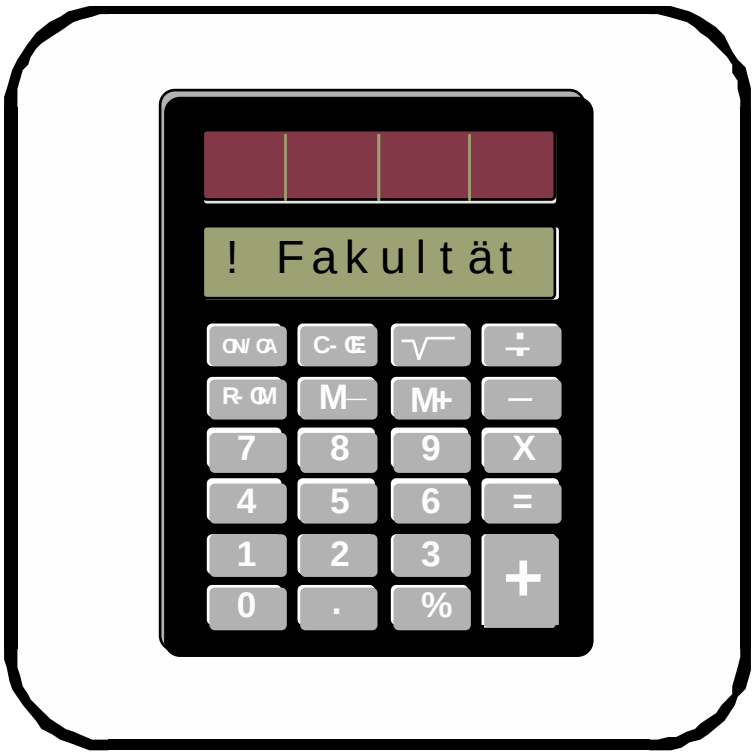
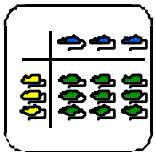
Fakultätsmigrenen Zahlen





Werkstatt Multiplikation
Posten: **Fakultät**

Informationsblatt für die Lehrkraft

Thema: Berechnung der Fakultätsfunktion

Schultyp: Mittelschule, technische Berufsschule, Fachhochschule

Vorkenntnisse: Grundlagen der Programmierung: ARRAY, INTEGER, REAL, FOR, INT(), einfache I/O-Operationen auf Tastatur/Bildschirm.
Keine Kenntnis von Unterprogrammen oder Prozeduren nötig.
Keine speziellen mathematischen Vorkenntnisse nötig.

Bearbeitungsdauer: 60 - 90 Minuten

Fassung vom: 15.9.95

Schulerprobung: nein

Übersicht

Der Posten besteht aus dem Auftrag ein Programm zur **Berechnung der Fakultätsfunktion** zu erstellen. Das Programm kann auf einem **Taschenrechner** oder auf einem **Computer**[1] erstellt werden. Die Fakultätsfunktion soll bis 99! auf alle Stellen exakt berechnet werden. Dabei ist aber die **Anzahl Stellen des Resultats grösser** als sie der Taschenrechner standardmässig bietet. So hat 99! **156 Dezimalstellen**. Ein **Rahmenprogramm wird vorgegeben**. Dadurch reduziert sich der Programmieraufwand auf die Multiplikation einer grossen Zahl[2] mit einer zweistelligen Zahl. Das Rahmenprogramm liegt als Pseudo-Code vor. Es muss vor Abgabe des Postens in eine **konkrete Programmiersprache** umgesetzt werden. Der Postenabsolvent muss **kein Spezialist** sein. Der Posten ist so konstruiert, dass er von **Programmieranfängern** erfolgreich gemeistert werden kann. **Motivierend** für den Postenabsolventen ist, dass das erstellte Programm die Fähigkeiten des eigenen Taschenrechners erweitert.

Lernziele

· Der Postenabsolvent sieht, wie man dem Taschenrechner den Umgang mit grossen Zahlen beibringen kann. Wenn er später wieder einmal mit grossen Zahlen rechnen wird, erinnert er sich an die hier verwendete Methode und Datenstruktur und passt sie an das neue Problem an.

Material

- *Theorie*: keine
- *Material:* **Programmierbarer Taschenrechner** mit Handbuch[3]. Die Aufgaben können aber auch auf einem **Computer** gelöst werden.

Quellen

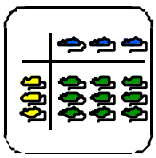
keine

---

1 Im Rest des Textes schreibe ich immer nur vom Taschenrechner. Für den Computer gilt natürlich alles sinngemäss.

2 Im ganzen Text verstehe ich als ***grosse Zahl*** eine ganze Zahl, die mehr gültige Stellen hat als der Taschenrechner. Ein üblicher Taschenrechner hat 11-12 gültige Stellen. 15! hat 13 Stellen.

3 Um die Befehle und deren Syntax nachschlagen zu können.



Hinweise, Lösungen

Lösung Auftrag 1

Aufgabenstellung:

Hier sollst Du nun Deinen Taschenrechner erweitern, damit er Dir auch bei grossen Zahlen das **exakte** Resultat für die Fakultätsfunktion liefert. Den Rahmen für das Programm gebe ich Dir vor. Du schreibst noch den **Kern des Programms:** Die Multiplikation einer grossen Zahl mit einer zweistelligen Zahl.

Lösung:

Für jedes Arrayelement muss

- das Arrayelement mit dem zweistelligen **Faktor multipliziert** werden
- ein allfälliger **übertrag** von der vorhergehenden Stelle **addiert** werden
- das Arrayelement **normalisiert** werden, d.h. auf 9 (Dezimal-) Stellen reduziert werden und der übertrag für die nächste Stelle gebildet werden.

In Pseudo-Code formuliert:

```
{ ----- Die Multiplikation GrosseZahl:= GrosseZahl * Faktor }
{ Initialisierung }
Uebertrag:= 0;
{ Jedes Arrayelement multiplizieren }
FOR i:= 1 TO 18 DO
    Tmp:= GrosseZahl[i] * Faktor + Uebertrag;
    { ----- Zahl normalisieren und Uebertrag bilden }
    { Die 10. und 11. Stelle gehören zum nächsten Arrayelement }
    Uebertrag:= INT(Tmp / 1000000000);
    { Die untersten 9 Stellen bleiben beim aktuellen Arrayelement }
    GrosseZahl[i]:= Tmp - Uebertrag * 1000000000;
END;
```

Die zusätzlich verwendeten Variablen müssen im Rahmenprogramm noch deklariert werden:

```
Tmp:        REAL;                  { Zwischenresultat Multiplikation }
Uebertrag:  REAL;                  { Uebertrag Multiplikation }
```

Einige Resultate:

10!= 3628800

15!= 1307674368000

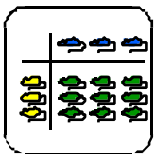
19!= 121645100408832000

30!= 265252859812191058636308480000000

55!= 12696403353658275925965100847566516959580321051449436762275840000000000000

80!= 71569457046263802294811533723186532165584657342365752577109445058227039255480148842668944867280814080000000000000000000000

99!= 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000



Lehrer-Lernkontrolle / Test

Aufgabe 1 (schriftlich)

Was musst Du an Deinem Programm ändern, damit es die Fakultäten bis und mit 999**!** berechnet? 999! ist eine Zahl mit 2'565 (Dezimal-) Stellen.

Es sind zwei Änderungen, die Du mir nennen musst. Du musst das neue Programm nicht hinschreiben. Mir reicht das Aufzählen der beiden Änderungen.

Aufgabe 2 (schriftlich)

Nachdem Du nun einige Erfahrung mit der Multiplikation gesammelt hast, sollst Du beim Aufbau einer Bibliothek von Mathematikroutinen mithelfen. Du darfst die **Multiplikationsroutine** schreiben. Doch jetzt sind beide Argumente (Multiplikand und Multiplikator) grosse Zahlen. Die grösste zugelassene Zahl hat dabei **100** (Dezimal-) Stellen. Beschreibe **in Worten** die Grundidee zu einer solchen Multiplikationsroutine für die Multiplikation von zwei 100stelligen Zahlen. Beachte, dass die Multiplikation von zwei n-stelligen Zahlen als Resultat eine 2n-stellige Zahl liefert!

Lösungen und Taxierung

Aufgabe 1 (schriftlich)

Lösung: Neu muss mit einer **dreistelligen** statt mit einer zweistelligen Zahl multipliziert werden. Da die Rechengenauigkeit immer noch nur 11 Stellen beträgt, können pro Arrayelement nur noch **8 Stellen** gespeichert werden. Da 999! 2'565 (Dezimal-) Stellen hat, braucht `GrosseZahl` jetzt neu **321 Arrayelemente**.

Taxierung: Die Bedingungen wurden geändert. Die Änderung geht über das reine Einsetzen von anderen Zahlen hinaus, da die Datenstruktur angepasst werden muss. Die Aufgabe ist deshalb **K3**.

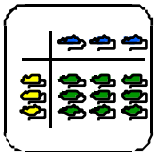
Aufgabe 2 (schriftlich)

Lösung: Statt nur mit einer (zweistelligen) Zahl muss mit einer grossen Zahl multipliziert werden. Dazu wird (wie bisher) das gesamte Array des Multiplikanden durchlaufen. Jedoch nicht nur einmal, sondern für jedes Arrayelement des Multiplikators. Da der Multiplikand während der ganzen Operation zur Verfügung stehen muss, wird das Resultat in einem neuen Array aufsummiert.

Um bei der Multiplikation von zwei Arrayelementen keinen überlauf zu erhalten, können pro Arrayelement nur noch 11 DIV 2= **5 Stellen** gespeichert werden. Die **Arrays** für den Multiplikanden und den Multiplikator müssen 100 Stellen fassen, also 100/5= **20 Elemente** gross sein. Das **Resultatarray** muss doppelt so gross sein: 200/5= **40 Elemente**.

Statt die Anzahl Stellen pro Arrayelement zu reduzieren, kann auch die Multiplikation von zwei Arrayelementen in mehreren Schritten erfolgen. Die zwei Zahlen können in eine untere und obere Hälfte aufgeteilt werden und dann miteinander multipliziert werden (Methode von Karatsuba). Dann können pro Arrayelement 11 Stellen gespeichert werden. Die **Arrays** für den Multiplikanden und den Multiplikator umfassen dann 100/11=9.1, also **10 Elemente**. Das **Resultatarray** muss doppelt so gross sein: 200/11=18.2, also **19 Elemente**.

Taxierung: Die Aufteilung einer Zahl in eine Summe von Zahlen muss auch für den Multiplikator angewendet werden (K3). Dieses Element (die Datenstruktur) muss mit dem Algorithmus synthetisiert werden (K5). Die Aufgabe ist deshalb **K5**.



Werkstatt Multiplikation Auftragsblatt
Posten: **Fakultät**

Was soll ich hier tun?

Auf wieviele Arten kannst Du Deine **Bücher ins Büchergestell** stellen? Die Mathematiker haben sich darum gekümmert. **Die Lösung** ist einfach: Um **20 Bücher** ins Bücherregal zu stellen, hast Du **20!** Möglichkeiten[4]. Leider können viele Taschenrechner das Resultat nicht direkt ausrechnen. Ein Grund ist, dass sie die **Funktion Fakultät nicht kennen**. Aber es gibt noch ein zweites Problem: Die **Zahlen** werden schnell **viel zu gross**. So ist 70! eine Zahl mit über 100 Stellen!

An diesem Posten kannst Du Deinem programmierbaren Taschenrechner diese Funktion auch für grosse Zahlen beibringen.

Der Posten besteht aus **einem Auftrag:**

Der Auftrag

Die meisten Taschenrechner rechnen nur auf etwa **11 Stellen genau**. Wenn Du auf Deinem Taschenrechner 15! ausrechnest, wird er Dir als Resultat zum Beispiel 1.307674368E12 liefern. Du weisst dann zwar, dass es etwa 1 Billion Möglichkeiten gibt, Deine 15 Bücher ins Büchergestell zu stellen. Wenn Du das Resultat aber exakt brauchst, lässt Dich der Taschenrechner im Stich. Er gibt Dir nur die ersten 10 Stellen des Resultats.

Hier sollst Du nun Deinen Taschenrechner erweitern, damit er Dir auch bei grossen Zahlen das **exakte** Resultat für die Fakultätsfunktion liefert. Den Rahmen für das Programm gebe ich Dir vor. Du schreibst noch den **Kern des Programms**: Die Multiplikation einer grossen Zahl mit einer zweistelligen Zahl.

Die **Datenstruktur für die grosse Zahl** ist ein Array `GrosseZahl[1..18]`. In jedem Arrayelement sind 9 (Dezimal-) Ziffern der Zahl gespeichert. In `GrosseZahl[1]` sind die niederwertigsten 9 Stellen.
Ein Beispiel: `GrosseZahl[1]= 333'444'555`
`GrosseZahl[2]= 111'222`
`GrosseZahl[3..18]= 0`
steht für die Zahl 111'222'333'444'555
Mit dieser Datenstruktur kannst Du Zahlen mit bis zu 18*9= 162 Stellen darstellen. 99! hat 156 Stellen.

Das **Rahmenprogramm** findest Du am Ende dieses Kapitels.

Füge in diesem Rahmenprogramm an der angegebenen Stelle Dein Multiplikationsprogramm ein.

Vor Deinem Programmteil gilt:

- `GrosseZahl[1..18]` enthält den **Multiplikanden**.
- Der **Multiplikator** ist in `Faktor` gespeichert.

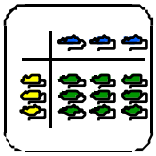
Nach Deinem Programmteil muss gelten:

- In `GrosseZahl[1..18]` ist das **Resultat** der Multiplikation `Faktor * GrosseZahl[1..18]` gespeichert.

Ein kleiner **Tip**: Wahrscheinlich brauchst Du die INT-Funktion. Die INT-Funktion liefert den ganzzahligen Teil des Arguments zurück. Also z.B. INT(3.4)= 3, INT(5.9)= 5. Manchmal heisst die Funktion auch TRUNC oder IP.

---

4 ! ist der Fakultätsoperator. n! ist definiert als n!= n*(n-1)*(n-2)*...*2*1. Also 5!= 5*4*3*2*1= 120. Ein Spezialfall ist: 0!= 1.



Werkstatt Multiplikation Auftragsblatt
Posten: **Fakultät**

Beachte bitte:

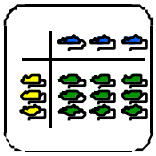
- Pro Arrayelement in `GrosseZahl` sind genau 9 (Dezimal-) Stellen gespeichert.
- Vom Rahmenprogramm darfst Du die Variablen `GrosseZahl` und `i` verändern.
- Du kannst davon ausgehen, dass bei der Multiplikation **kein überlauf** (Overflow) auftritt.

Ich erwarte das Programm mit Kommentaren auf Papier und auf Deinem Taschenrechner implementiert. Ich werde Deine Lösung mit Dir besprechen. Dabei ist mir **wichtig**:

- Deine Lösung ist klar und einfach nachvollziehbar.
- Deine Lösung berechnet die Fakultäten 1! bis 99! korrekt.

Wenn Du diese beiden Punkte einhälst, ist der Auftrag **erfüllt**. Ich werde einige Fakultäten mit Deinem Taschenrechner berechnen.

Diesen Auftrag sollte **jeder für sich** bearbeiten.



Das Rahmenprogramm in Pseudo-Code

```
{ Programm zur Berechnung der Fakultätsfunktion }

{ ----- Deklaration der Variablen }
VAR i:          INTEGER;                { Laufvariable }
    GrosseZahl: ARRAY [1..18] OF REAL;  { Resultat der Funktion }
    Argument:   INTEGER;                { Argument der Funktion }
    Faktor:     INTEGER;                { Aktueller Multiplikator }

{ ----- Initialisierung: GrosseZahl:= 1 }
GrosseZahl[1]:= 1;
FOR i:= 2 TO 18 DO
    GrosseZahl[i]:= 0
END;

{ ----- Funktionsargument vom Benutzer eingeben lassen }
WriteLn "Fakultaet von ";
INPUT Argument;

{ Fakultaet nur von 0! bis 99! }
IF Argument > 99 THEN HALT END;
IF Argument <  0 THEN HALT END;

{ ----- Berechnung der Fakultaet (Argument)! }
FOR Faktor:= 1 TO Argument DO
    { ----- Die Multiplikation GrosseZahl:= GrosseZahl * Faktor }

    Hier kommt Dein Code


END; { FOR Faktor }

{ ----- Ausgabe des Resultats }
{       Es werden einfach alle Arrayelemente von GrosseZahl ausgegeben. }
{       Zuerst das hochwertigste Element }
WriteLn Argument, "!=";
FOR i:= 18 DOWNTO 1 DO
    WriteLn GrosseZahl[i];
END;
```