Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Aufgabenblatt 1
12. Mai 2006

Praktikum zu Grundlagen der Programmierung

Aufgabe 1 Funktionen in OCaml

Für die Berechnung sich wiederholender Ausdrücke verwendet man in Programmiersprachen und damit auch in OCaml *Funktionen*.

a) **Definition von Funktionen:**

(i) Definieren Sie eine einstellige Funktion `square` die zu einer Zahl x das Quadrat x^2 dieser Zahl zurückgibt. Der Typ von x soll `float` sein. Geben Sie zusätzlich die Signatur der Funktion an.

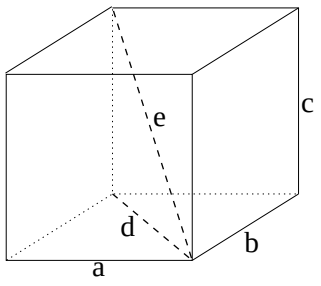
(ii) Definieren Sie eine einstellige Funktion `square_root` die zu einer Zahl x die Wurzel dieser Zahl zurückgibt. (**Hinweis:** In OCaml können sie die Wurzel mit sqrt berechnen.) Testen Sie ihre Funktion mit einigen Werten.

(iii) Definieren Sie eine einstellige Funktion `square_id` die zu einer Zahl x die Wurzel des Quadrates $\sqrt{x^2}$ zurückgibt. Verwenden sie hierzu die Funktionen `square` und `square_root`.

(iv) Definieren Sie eine einstellige Funktion `validate_id` die für eine Zahl x überprüft ob `square_id(x)` den Wert x zurückgibt.

b) **Zusammengesetzte Funktionen:**

(i) Definieren Sie nun mit Hilfe der obigen Funktionen eine neue Funktion `diag_r` die zu einem Rechteck mit den Seitenlängen a und b die Länge der Diagonale $d = \sqrt{a^2 + b^2}$ berechnet. Testen Sie diese Funktion mit einigen Werten.



(ii) Definieren Sie eine Funktion `diag_q` die zu einem Quader mit den Seitenlängen a, b und c die Raumdiagonale e berechnet. Testen Sie ihre Funktion mit einigen Werten.

(iii) Definieren Sie nun mit Hilfe der obigen Funktionen eine neue Funktion `diag_c` die zu einem Würfel mit der Seitenlänge a die Länge der Raumdiagonale berechnet.

c) **Partielle Funktionsanwendung:** Nimmt man eine n-stellige Funktion und wendet sie auf ein einzelnes Argument an, so erhält man eine $(n-1)$-stellige Funktion. Die Funktion 'schluckt' das erste Argument.

(i) Definieren Sie basierend auf der Funktion `diag_q` von oben eine zweistellige Funktion, die die Raumdiagonale eines Quaders mit der Breite $a = 1$ berechnet.

(ii) Vergleichen Sie das Ergebnis mit der ursprünglichen Funktion. Vergewissern Sie sich, daß die neue Funktion tatsächlich ein Argument 'geschluckt' hat.

Aufgabe 2 Osterberechnung

Carl Friedrich Gauß (1777 - 1855) entwickelte im Jahre 1800 die so genannte „Osterformel". Damit läßt sich für jedes Jahr von 1583 bis 8202 der Ostersonntag berechnen. Die Berechnung des Ostersonntags gestaltet sich deswegen kompliziert, da gemäß des 1. Kirchenkonzils im Jahr 325 Ostern stets am ersten Sonntag nach dem ersten Vollmond des Frühlings statt zu finden hat. Da jedoch die Dauer eines Erdjahres und die Dauer eines Mondumlaufes um die Erde in keinem ganzzahligen Verhältnis stehen, kommt man (nach einigem Kopfzerbrechen in Gauß'schem Stil) zu folgendem Algorithmus:

Zwischen der Jahreszahl J sowie den ganzen Zahlen m und n bestehe der durch nebenstehende Tabelle gegebene Zusammenhang. Bezeichnet man nun mit a, b, c, d und e die Reste

J	m	n
1583 - 1699:	22	2
1700 - 1799:	23	3
1800 - 1899:	23	4
1900 - 2099:	24	5
2100 - 2199:	24	6
2200 - 2299:	25	0

$$\begin{aligned} a &= J \bmod 19, \\ b &= J \bmod 4, \\ c &= J \bmod 7, \end{aligned}$$

$$\begin{aligned} d &= (19a+m) \bmod 30, \\ e &= (2b+4c+6d+n) \bmod 7 \end{aligned}$$

Dann folgt für den 1. Osterfeiertag:

$$\text{1. Ostersonntag} = \begin{cases} (22+d+e)\text{. März}, & \text{wenn } (22+d+e) < 32 \\ (d+e-9)\text{. April}, & \text{sonst} \end{cases}$$

Aufgabenstellung: Implementieren Sie ein geeignetes System von Funktionen und testen Sie den Algorithmus für das aktuelle Jahr. Verwenden Sie zur Darstellung des Datums, die aus der Vorlesung bekannten Datentypen, für Datumswerte.

Aufgabe 3 (H) IF-THEN-ELSE Verzweigungen (5 Punkte)

Schreiben Sie zur Währungsumrechnung eine zweistellige Funktion, die als ersten Parameter einen Geldbetrag in Euro nimmt (*float* Variable) und als zweiten Parameter einen Selektionswert (*Integer* Variable), der die Zielwährung bestimmt. Dabei soll die entsprechende Umrechnung anhand folgender Selektionswerte erfolgen:

$$\begin{aligned} 0 &: Umrechnung\ \ 1\ Euro = 1.95583\ DM \\ 1 &: Umrechnung\ \ 1\ Euro = 1.27367\ US-Dollar \\ 2 &: Umrechnung\ \ 1\ Euro = 0.68439\ Pfund \\ 3 &: Umrechnung\ \ 1\ Euro = 142.150\ Yen \\ 4 &: Umrechnung\ \ 1\ Euro = 1.56039\ SchweizerFranken \end{aligned}$$

D.h. wird die Funktion z.B. mit den Parametern 100.0 und 1 aufgerufen, so soll die Ausgabe 127.367 (US-Dollar) lauten. Sie können beliebig viele weitere Währungen selbst hinzufügen. Falls als Selektionswert ein *Integer* eingegeben wird, für den keine Umrechnung definiert wurde, so soll einfach der eingegebene Eurobetrag unverändert wieder ausgegeben werden.

Aufgabe 4 (H) Primzahltest (10 Punkte)

Ziel dieser Aufgabe ist es eine einstellige Funktion zu schreiben, die als Eingabeparameter einen *Integer* Wert nimmt, von dem möglichst effizient festgestellt werden soll ob es sich um eine Primzahl handelt oder nicht. Dafür ist es zunächst sinnvoll, die Überprüfung in einzelne Teilschritte zu zerlegen:

a) Eine Primzahl > 2 muss in jedem Fall eine ungerade Zahl sein. Schreiben Sie daher eine einstellige Funktion, die für einen *Integer* Wert feststellt, ob es sich um eine gerade oder ungerade Zahl handelt (d.h. ob die Zahl restfrei durch 2 teilbar ist oder nicht). Ist die Zahl ungerade (d.h. es kann sich evtl. um eine Primzahl handeln), dann soll 1 zurückgegeben werden, sonst 0. Eine Überprüfung, ob die Zahl größer 2 ist, muss nicht vorgenommen werden.

b) Weiterhin lassen sich alle Primzahlen > 3 in der Form $Primzahl = 6n + 1$ oder $Primzahl = 6n - 1$ darstellen. Schreiben Sie daher zwei einstellige Funktionen die jeweils den eingegebenen *Integer* Wert auf eines der o.g. Kriterien überprüfen. Erfüllt die Zahl das Kriterium (d.h. es kann sich evtl. um eine Primzahl handeln) so soll 1 zurückgegeben werden, sonst 0. Eine Überprüfung, ob die Zahl größer 3 ist, muss nicht vorgenommen werden.

c) Nun soll der wesentliche Teil des Primzahltests erstellt werden. Schreiben Sie dazu eine Funktion, die überprüft ob der eingegebene *Integer* Wert n eine Primzahl ist, d.h. ob n nur durch 1 und sich selbst teilbar ist. Verwenden Sie dazu am besten eine rekursive Funktion, die schrittweise die Teilbarkeit von n durch $2..n-1$ prüft. Falls n durch eine Zahl aus dem Intervall $2..n-1$ restfrei geteilt werden kann, so handelt es sich nicht um eine Primzahl. Die Rekursion kann abgebrochen werden und gibt 0 aus. Falls die Teilertests erfolglos sind (d.h. es ist eine Primzahl) so wird 1 ausgegeben. Überlegen Sie wie der Test noch beschleunigt werden kann. **Hinweis:** Sie können Funktionen für *float* Werte auch auf *Integer* Werte anwenden, indem Sie Typecasts verwenden. So berechnet z.B. der Ausdruck
int_of_float (exp (float_of_int x)) die *Integer*-Variante für e^x. Da *exp* nur für *float* Werte definiert ist, muss der *Integer* Wert x zunächst mit *float_of_int* nach *float* konvertiert werden. Das Ergebnis der Operation (e^x) liegt dann ebenfalls wieder als *float* vor und muss daher wieder mit *int_of_float* zurück gecastet werden, um in einer *Integer*-Berechnung weiter verwendet werden zu können. Der Nachkommateil des *float*-Ergebnisses von e^x wird dabei abgeschnitten.

d) Schreiben Sie nun eine einstellige Funktion für einen schnellen Primzahltest, die auf die oben erstellten Teile zurückgreift und 1 ausgibt, falls es sich um eine Primzahl handelt, sonst 0. Gehen Sie dabei am besten wie folgt vor: der eingegebene *Integer* Wert soll zunächst darauf überprüft werden, ob er $\leqq 3$ ist. Falls ja, dann handelt es sich sicher um eine Primzahl und der Test kann mit der Ausgabe 1 abbrechen. Wenn nicht, so kann durch die in Teilaufgabe a und b erstellten Funktionen überprüft werden, ob die Eingabe überhaupt noch eine Primzahl sein kann. Ist dies der Fall so kann der eigentliche Test aus Teilaufgabe c durchgefürht werden, ansonsten wird 0 ausgegeben.