情報処理 (10)

プログラミング

増原 英彦

1 プログラムの作成と実行

HWB17.4「プログラミングの基本 - Ocaml 編」に従ってプログラムの基本的な書き方を習得せよ。

- タイプミスなどがあると、同じ式を何度も入力し直さなければならなくなる。面倒を避けるために、(1) エディタ (Emacs) で、例えば`myprogram.ml` という名前のファイルを作り、(2) 入力する式は先にエディタに入力し、(3) それをコピーして Ocaml にペーストする、という手順を踏むと良いだろう。このようにしておけば、HWB17.4.14「原始プログラムの読み込み」で説明されている `#use` という指令によってプログラムをまとめて読み込ませることもできるようになる。
- すでに他のプログラミング言語を知っている場合でも、初めから一通り読んでみた方がよいだろう。

2 練習問題

以下の練習問題 10–1,2,3 を解く Ocaml の式、関数の定義を考え、解答を他の人の参考のために、この授業の掲示板に投稿せよ。同様に、練習問題を解く上での質問・他人の解答の誤りの指摘・コメント等も参考になるので積極的に掲示板に投稿せよ。

練習 10–1: (計算の練習) Ocaml を使って以下の計算を行ってみよ。

1. 自分が生まれてからの秒数 (n 歳なら「n 年間の秒数」で近似してよい)
2. 10,000,000 秒は何週間になるか
3. 方程式 $x^2 - 22x - 408 = 0$ の解
4. 半径 3.2 cm の円の周
5. 半径 3.2 cm の円の面積
6. セ氏 28 度は華氏で何度になるか (ヒント: セ氏 0 度が華氏 32 度、セ氏 100 度が華氏 212 度である。)
7. 華氏 50 度はセ氏で何度になるか

練習 10–2: (関数の定義) 次のような計算をする関数を定義せよ。

1. `fahrenheiht(x)`：セ氏`x` 度を華氏に変換する関数。これを使ってセ氏 $-10, 25, 36$ 度を華氏に変換せよ。
2. `celcius(x)`：華氏`x` 度をセ氏に変換する関数。これを使って華氏 100, 451, 911 度をセ氏に変換せよ。
3. `circle_round(r)`：半径`r` の円の周を求める関数。
4. `circle_area(r)`：半径`r` の円の面積を求める関数。
5. `sign(x)`：`x` が奇数なら 1, 偶数なら -1 となる関数。
6. `leibniz(x)`：次のような数列の`x` 番目の要素: $\frac{1}{1}, -\frac{1}{3}, \frac{1}{5}, -\frac{1}{7}, \frac{1}{9}, -\frac{1}{11}, \ldots$ (先頭が 1 番目とする。結果は実数になることに注意せよ。整数`x` を実数に変換する関数`float(x)` を使ってもよい。)

7. `wallis(x)`：次のような数列の`x` 番目の要素: $(\frac{2}{1}\cdot\frac{2}{3}),(\frac{4}{3}\cdot\frac{4}{5}),(\frac{6}{5}\cdot\frac{6}{7}),(\frac{8}{7}\cdot\frac{8}{9}),\ldots$ (同上。)

8. `collatz(x)`：ある数 `x` が与えらたとき、`x` が偶数の場合 `x/2` となり、`x` が奇数の場合 $3\mathtt{x}+1$ となるような関数。例えば `collatz(10)` $=5$, `collatz(5)` $=16$ となる。

練習 10–3: (繰り返しを伴う計算の定義)　次のような計算をする関数を定義せよ。

1. `factorial(n)`：`n` の階乗、つまり、$1\times 2\times\cdots\times\mathtt{n}$ を求める関数。(ヒント: `sum(n)`)

2. `power(x,n)`：`x` の`n` 乗を求める関数。

3. `gcd(x,y)`：`x` と`y` の最大公約数を求める関数。

4. `leibniz_pi(x)`：$4\times\sum_{\mathtt{i}=1}^{\mathtt{x}}\mathtt{leibniz(i)}$ を求める関数。(ライプニッツの公式)

5. `wallice_pi(x)`：$2\times\prod_{\mathtt{i}=1}^{\mathtt{x}}\mathtt{wallice(i)}$ を求める関数。(ウォリスの公式)

6. `combination(x,y)`： `x` 個の要素から`y` 個を選ぶ選び方の組み合わせの総数を求める関数。(ヒント 1: ABCDEFGHIJ という 10 個の要素から 5 個の要素を選ぶ組み合わせは、(1) A を選んで BCDEFGHIJ という 9 個の要素から 4 個の要素を選ぶ場合と、(2) A を選ばないで BCDEFGHIJ という 9 個の要素から 5 個の要素を選ぶ場合の合計になる。)

7. `fibonacci(x)`：フィボナッチ数列の`x` 番目の要素を求める関数。ただし、フィボナッチ数列は 0 番目と 1 番目が 1 で、`x` 番目はその前 2 つの要素の和になっているような数である。

8. `coins(x)`： `x` 円を支払うのに必要な硬貨の最小枚数。ただし、支払いには 500 円、100 円、50 円、10 円、5 円、1 円玉のみを使うとして、何枚でも使ってよいとする。例えば、449 円を支払う場合は 100 円 $\times 4+$ 10 円 $\times 4+$ 5 円 $\times 1+$ 1 円 $\times 4=$ 449 円 なので、`coins(449)=13` となる。

9. `collatz_number(x)`：ある数 `x` を上で定義した`collatz` に従って変化させていったとき、`x` が 1 になるまでに要した回数を求める関数。例えば、`collatz_number(3)` は 3 を 上の`collatz` によって変化させてゆくと $3\to 10\to 5\to 16\to 8\to 4\to 2\to 1$ のようになるので、`collatz_number(3)` $=7$ となる。(ヒント: `collatz_number(3)` は、3 の次の数つまり`collatz(3)` が 1 になるまでの回数 `collatz_number(collatz(3))` より 1 大きい。)

10. `max_collatz_number_to(x)` : 1 から`x` までの整数のうち最大の `collatz_number`。例えば、`collatz_number(1)` $=0$, `collatz_number(2)` $=1$, `collatz_number(3)` $=7$, `collatz_number(4)` $=2$, `collatz_number(5)` $=5$ なので `max_collatz_number_to(5)` $=7$ となる。

    これを使って `max_collatz_number_to(10000)` を求めてみよ。さらに、どのくらい大きな数に対する `max_collatz_number_to` を求めることができるか。(整数の桁あふれに気を付けよ。)

11. `hanoi(x)`：`x` 個の円盤からなるハノイの塔問題で、`x` 個の円盤をすべて棒 A から棒 B に移動させるのに要する回数。ただし、ハノイの塔問題とは、3 本の棒 A,B,C があり、最初棒 A に大きさのすべて異なる`x` 個の円盤が差さっており、それらを以下の条件を守りながら全てを棒 B に移動させるというものである:

    - 条件 1: 小さい円盤の上に大き円盤を重ねてはいけない。
    - 条件 2: 一度に移動できる円盤は、棒 A,B,C の 1 番上にある円盤のどれか 1 つである。

    例えば、`x` $=3$ で 棒 A に大きさ 3,2,1 の円盤が下から重ねられ、棒 B,C には円盤がない状態を $[A=321, B=\cdot, C=\cdot]$ と書くことにする。この状態で動かすことができるのは円盤 1 だけであり、これを棒 B に移動させると、$[A=32, B=1, C=\cdot]$ となる。さらに円盤 2 を棒 C に移動させると $[A=3, B=1, C=2]$、さらに円盤 1 を棒 C に移動させると $[A=3, B=\cdot, C=21]$ さらに円盤 3 を棒 B に移動させると $[A=\cdot, B=3, C=21]$、さらに円盤 1 を棒 A に移動させると $[A=1, B=3, C=2]$、さらに円盤 2 を棒

B に移動させると $[A = 1, B = 32, C = \cdot]$、さらに円盤 1 を棒 C に移動させると $[A = \cdot, B = 321, C = \cdot]$ となり、問題が解けたことになる。よって要した手順 `hanoi(3)` $= 6$ となる。

ヒント: 棒 A にある円盤 1,2,3 を棒 B にすべて移動するのに要する回数`hanoi(3)` は、

- 棒 A にある円盤 1,2 を棒 C にすべて移動する手順 (つまり`hanoi(2)`)
- 棒 A にある円盤 3 を棒 B に移動する手順 (つまり 1 回)
- 棒 C にある円盤 1,2 を棒 B にすべて移動する手順 (つまり`hanoi(2)`)

の合計となる。

3 課題

課題 10–1: (簡単なプログラムの作成) 以下の計算を行う関数を定義し、計算せよ。

1. `divisible(x,y)` : `x` が`y` で割り切れるなら true, 割り切れないなら false となる関数。この関数を使って、自分の学生証番号が 2, 3, 5 で割り切れるかどうかを計算せよ。

2. `indivisible_from(x,y)` : `x` が 2 以上`y` 以下の整数で割り切れるなら false、そうでないなら true となる関数。(ヒント: `y` が 3 以上のとき、`x` が`y` で割り切れないなら、`indivisible_from(x,y)` は `indivisible_from(x,y-1)` と同じ結果になる。)

3. `prime(x)` : `x` が素数なら true, 素数でないら false となる関数。この関数を使って、自分の学生証番号, 生年月日をつなげた数, 電話番号の下 1 桁を除いた数が素数かどうかを計算せよ。

4. `next_prime(x)` : `x` 以上の最小の素数。この関数を使って、自分の学生証番号, 生年月日をつなげた数, 電話番号の下 1 桁を除いた数の下 6 桁について、`next_prime(x)` の値を求めよ。(注意: `x` が素数なら `next_prime(x)` の値は`x` になる。)

5. `nth_prime_from(n,x)` : 素数`x` より`n` 番目に大きな素数。`n` が 0 のときは `nth_prime_from(n,x)` の値は x とする。この関数を使って、自分の学生証番号, 生年月日をつなげた数, 電話番号の下 1 桁を除いた数の下 6 桁の数`n` について、`nth_prime_from(n,2)` の値を求めよ。(ヒント: `nth_prime_from(3,2)` は素数 2 より 3 番目に大きな素数なので、`nth_prime_from(2,3)` (素数 3 より 2 番目に大きな素数) に等しく、`nth_prime_from(1,5)` (素数 5 より 1 番目に大きな素数) に等しく、`nth_prime_from(0,7)` (素数 7 より 1 番目に大きな素数) に等しくなる。)

6. `leap_year(y)` : 西暦`y` 年が閏年なら true、平年なら false となる関数。(ヒント: 閏年は 4 の倍数年である。ただし 100 の倍数年は平年である。ただし、100 の倍数年でも 400 の倍数年は閏年である。)

7. `days_of_month(y,m)` : 西暦`y` 年`m` 月の日数を求める関数。(ヒント: 1,3,5,7,8,10,12 月は 31, 4,6,9,11 月は 30, 閏年の 2 月は 29, 平年の 2 月は 28 となる。)

8. `year_of_next(y,m,d)`, `month_of_next(y,m,d)`, `date_of_next(y,m,d)` : `y` 年`m` 月`d` 日の翌日の年, 月, 日をそれぞれ求める関数。(例えば `year_of_next(2004,12,31)` $= 2005$, `moth_of_next(2004,12,31)` $= 1$, `date_of_next(2004,12,31)` $= 1$, `year_of_next(2005,7,6)` $= 2005$, `moth_of_next(2005,7,6)` $= 7$, `date_of_next(2005,7,6)` $= 7$ となる。)

9. `days_between(y1,m1,d1,y2,m1,d2)` : `y1` 年`m1` 月`d1` 日から`y2` 年`m2` 月`d2` 日までの日数を数える関数。この関数を使って、自分の誕生日から 2005 年 7 月 5 日までの日数を計算せよ。(ヒント: 「開始日から終了日までの日数」は、「開始日の翌日から終了日までの日数」より 1 大きい。)

10. `day_of_week(y,m,d)` : `y` 年`m` 月`d` 日の曜日を 0 から 6 の数で計算する関数。0=日曜日, 1=月曜日, 2=火曜日, ... とする。`y` は 1900 以上としてよい。(ヒント: 1900 年 1 月 1 日の曜日を`cal` コマンドで調べておけば、その日からの日数を 7 で割った余りになる。)

提出方法　上の問題を 4 問以上解き、それぞれの考え方・計算結果・関数の定義を書いたものをテキスト形式で 1 つのレポートにまとめよ。説明等は注釈にして、そのまま Ocaml 処理系に読み込めるものが望ましい (下の例を参考にせよ)。提出は電子メールで`masuhara-js-report@lecture.ecc.u-tokyo.ac.jp` と自分自身に送れ。ただし、

- レポートは本文中に直接テキスト形式で書き込め。添付ファイルや、リッチテキスト形式は用いないこと。
- レポートには、課題を完了するのに要した時間と授業に対する感想も書くこと
- メールは必ず教育用計算機システムから送ること
- 再提出する場合も全ての内容を送ること。(再提出があった場合は、前に提出したレポートを無視します。)

提出期限

- 2 年生 — 2005 年 7 月 29 日 (金) 23 時 59 分
- 1 年生 —
  - (1 次) 2005 年 7 月 29 日 (金) 23 時 59 分 (100%の得点を与える)
  - (2 次) 2005 年 8 月 5 日 (金) 23 時 59 分 (80%の得点を与える)
  - (3 次) 2005 年 8 月 19 日 (金) 23 時 59 分 (60%の得点を与える)

体裁の例:

```
(*                          情報処理レポート 課題 10-1
                                          学生証番号: 987654 東大太郎

1. divisible(x,y) : x が y で割り切れるなら true となる関数
● 考え方: …(略)…

● 実行結果:
# divisible(987654,2);;
- : bool = true
# divisible(987654,3);;
- : bool = true
# divisible(987654,5);;
- : bool = false

● 定義:
*)
let divisible(x,y) =
  ...(略)

(*

2. indivisible_from(x,y) : x が 2 以上 y 以下の数で割り切れるなら false となる
関数
…(略)
```