**ЗВІТ ПРО ВИКОНАНЕ ЗАВДАННЯ**

з курсу “Теорія програмування”

студента групи ПК-12-2

Бекленищева Владислава Ігоровича

ЗАВДАННЯ № 1

кафедра компютерних технологій, ДНУ

2014/2015 навч. р.

# 

# Постановка задачи

Написати програми на мові OCAML для роз'язання вказаних нижче завдань.

Продемонстрирувати їх роботу на тестових прикладах в середовищі OCAML:

1) в режимі набору тексту з командного рядка OCAML;

2) шляхом вводу тексту програми з текстового файлу.

Підготувати звіт згідно до форми (див. файл Вимоги.TXT)

Варіант - 1

Реализовать две функции:

а) вычисления суммы двух целых чисел

б) которая на основании заданного списка пар натуральных чисел формирует новый список так,

что для каждого элемента (N M) заданного списка элемент нового списка - это comb(N,M) - число сочетаний из N по M.

Примечание: данную лабораторную работу я выполнял в операционной системе Linux Ubuntu 14.04 в версии OCaml 4.01.0.

# Описание решения

## Задача 1

Для решения первой задачи нужно было задать функцию с помощью ключевого слова let:

# let sum a b =

(a + b);;

В ocaml функция задаётся может задаваться несколькими способами и это один из них. В данном случае мы связываем нашу переменную sum с самой функцией передавая ей два аргумента - a и b.

Данная функция работает только для целых чисел. Для того чтобы функция работала для дробных после оператора “+” нужно добавить “.” - “+.”, а сами числа a и b должны быть дробными.

Если хотя бы одно из чисел (рассматриваем исходный код задачи) окажется дробным, то будет ошибка.

Для чтения программы из текстового файла я воспользовался массивом Sys.argv, который содержит аргументы командной строки в виде строк - string. Для вывода такого результата на экран терминала я подключил пакет Printf, в котором определена функция известная функция printf с языка С. Должен отметить, что для перевода полученных строк из массива Sys.argv я воспользовался функцией int\_of\_string, указав ей через пробел в качестве параметра элемент массива (начиная с первого, так как нулевой - это имя программы).

При работе вручную пакет Printf я не подключал, так как интепретатор OCaml автоматически после ввода выражений либо выводит тип либо выводит полученный результат - после работы функции.

## Задача 2

Вторая задача оказалась сложнее первой. Для её решения мне понадобилось определить 3 функции (выражения) и несколько дополнительных функций для чтения кода программы их файла. Обо всём этом и пойдёт речь в этом разделе.

На вход программе должен подаваться список с записями, каждая из которых представляет собой пару натуральных чисел (N,M). Например, список такого вида: [(3,1) ; (1,4); (2,6)].

На выходе должен получиться обычный список состоящий с одиночных элементов (синглтонов), каждый из которых должен являться числом сочетаний из N по M - comb(N,M). Например, такого вида: [3 ; 4; 15]. Должен отметить, что не важно как пользователь будет записывать пары чисел (1,3) или (3,1). Программа позволяет делать и так и так, разница будет только в том, что большее из двух чисел станет N, а меньшее станет M. А если числа равны (N==M), то в новый список пойдет “1”. При решении такой задачи я старался делать так, чтобы новые элементы соответствовали предыдущим парам, из-за которых и получились. То есть, чтобы за элементами сохранялись места. В виду некоторых ограничений, по типу добавления элемента списка в голову, пришлось делать *reverse* массива. Об этом речь пойдёт ниже. В начале рассмотрим сами функции.

Функция combine позволяет генерировать новый список исходя из элементов предыдущего, выполняя над ними операцию comb:

#let rec combine xlist =

match xlist with

| [] -> []

| (c1,c2)::xtail ->

begin let c\_res = (comb c1 c2) in

c\_res::(combine xtail);

end

;;

Здесь я использовал очень удобную особенность языка OCaml, которую не видел еще не в одном языке программирования. Речь идёт о сопоставлении с образцом (pattern matching). В данном случае, функции combine подаётся на вход исходный список. В самом выражении match xlist with означает “сравниваем наш список с” и далее идут так называемые образцы (патерны). Стрелочка “->” означает “тогда” (then): если xlist - это пустой список [], то тогда результатом будет пустой список. Да, это является неким аналогом конструкции принятия решений if boolean-condition then expression. В целом, общую форму сравнения с образцом можно подать в таком виде:

match object with  
| pattern -> result  
| pattern -> result  
 ...

Вторым образцом является элемент головы, который определяется так: (c1,c2)::tail. (c1,c2) однозначно указывает на то, что в исходном списке элементы принадлежат типу int \* int, то есть являются кортежами (tuple). Возможно, по условию можно было воспользоваться именно структурой даных запись (record), но последняя определяется более строго чем кортеж. Тем более tuple намного удобнее в использовании.

Итак, если у нас в списке есть первый элемент типа int \* int (а тип в списке может быть только один), то мы выполняем последовательность выражений, ограниченных конструкцией begin … end. Вместо этой конструкции можно было бы воспользоваться обычными скобками “(“ и “)”. В некоторых туториалах так и сказано, что begin … end является “синтаксическим сахаром” для обычных круглых скобок. В этой конструкции мы объявляем переменную c\_res, которая будет своеобразным сокращением к выражению comb c1 c2, и будет существовать до самого конца выражения - до “;;”. comb c1 c2 возвращает число сочетаний из (N, M). Что из них с1, а что с2 выясняется в самой функции, которую мы рассмотрим ниже.

Выражение типа let name = expression in рассматривается так: name заменяет выражение следуемое после знака “=” дальше по коду, вплоть до “;;”. Стоит отметить, что после in нельзя ставить “;” - separator выражений локального блока кода.

Выражение c\_res::(combine xtail) означает, что c\_res (результат комбинации двух натуральных чисел) будет записываться в голову нового списка. Этот список будет формироваться в последствии рекурсивных вызовов функции combine, которая, в свою очередь, будет получать “хвост” списка в виде аргумента xtail. Для справки: в OCaml у списка есть head (голова) и tail (хвост). head - это первый элемент списка, а tail - это остальная часть списка, то есть всё, что не включает head. Таким образом, получается, что первый раз выполнение выражение c\_res::(combine xtail) добавит элемент c\_res самым последнем, то есть он станет головой нашего списка. В данном случае рекурсия позволяется благодаря тому, что в определении переменной функции combine я указал *rec.*

Дальше, рассмотрим функцию comb:

#let comb n m =

if n == m then 1

else if n > m then (fact n) / ((fact m) \* (fact (n - m)))

else (fact m) / ((fact n) \* (fact (m - n)))

;;

Эта функция не является рекурсивной, она просто считает число сочетаний двух чисел и попутно вызывает рекурсивную функцию fact, которая считает факториал числа:

#let rec fact x =

if x <= 1 then 1

else x \* fact (x-1)

;;

Обе функции работают только с целыми числами. Для работы с числами с плавающей точкой нужно подавать им на вход соответствующие числа и также применять соответствующие операции, то есть вместо, скажем, операции “/” ставить “/.”, или операции “-” - “-.”

При работе непосредственно с консолью Ocaml функции нужно объявлять в такой последовательности:

1. fact x
2. comb n m
3. combine xlist

Так как одна функция может вызывать другую функцию, то вторая функция должна быть уже определена выше - иначе будет ошибка.

Определять исходный массив можно либо перед выше указанными функциями либо после:

#let l = [(3,1);(1,1);(2,6)];;

Стоить учесть, что числа в массиве должны быть целыми. Так как в условии сказано, что числа должны быть натуральными, я ограничился только множеством целых чисел, которое является ОДЗ типа int. Тип int в OCaml тоже не является простым. Дело в том, что число такого типа будет занимать в памяти 31 бит на 32-битном процессоре, а на 64 битном - 63 бита.

При создании текстового файла (а точнее файла с расширением .ml), из которого будет загружаться в дальнейшем код программы, пришлось добавлять еще несколько функций. Их мы и рассмотрим.

let rec get\_data x i =

if i < (Array.length Sys.argv - 1)

then get\_data (((int\_of\_string Sys.argv.(i)),(int\_of\_string Sys.argv.(i + 1)))::x) (i+2)

else x

;;

let print\_input\_data =

let i = ref 1 in

printf "input: [";

while !i<(Array.length Sys.argv - 1) do

printf "(%i, %i);" (int\_of\_string Sys.argv.(!i)) (int\_of\_string Sys.argv.(!i+1));

i := !i + 2

done;

printf "]\n";

;;

let rev list =

let rec aux acc = function

| [] -> acc

| h::t -> aux (h::acc) t in

aux [] list

;;

(\* reverse for get original view (as in input) \*)

let list\_of\_tuples = rev (get\_data [] 1);;

(\* …………………….. main solution ………………...\*)

(\* ………………………………………………………………..\*)

let print\_results =

printf "output: [";

List.iter (printf "%i;") result\_l;

printf "]\n";

;;

1. Функция get\_data позволяет рекурсивно сформировать новый список из элементов полученных из массива Sys.argv. Элементы добавляются попарно начиная с 1-го. И тут есть удобная особенность: если пользователь введёт непарное количество элементов через пробел в командной строке, то после число (без пары) не будет записано в исходный список - в списке будут только пары элементов.
2. Функция print\_input\_data выводит на экран входной список в виде:   
   input: [(item\_11,item\_12); (item\_21,item\_22);...;(item\_n1; item\_n2);].
3. Функция rev получает на вход массив и выдаёт в результате массив, элементы которого отображаются в обратной последовательности. Эту функцию я использую только при чтении кода программы с текстового файла. Дело в том, что функция get\_data формирует массив так, что элементы, которые идут первыми, добавляются в список первыми. То есть, после всех добавлений в список первый элемент становится последним - не так как хотел бы этого пользователь. Поэтому я решил, что будет удобно если будет формироваться новый массив с обратным отображением элементов - так как вводит пользователь:   
   (\* reverse for get original view (as in input) \*)  
   let list\_of\_tuples = rev (get\_data [] 1);;
4. Функция print\_results печатает массив полученный в результате комбинаций элементов. Здесь выражение List.iter, которое возвращает итератор на каждый элемент списка result\_l и к каждому элементу позволяет применить указанную функцию - printf. Другим словами, это выражение является аналогом функции for\_each из С++.

# Исходный текст программы решения задачи

## Задача 1

### Ввод текста программы в командной строке Ocaml

# let sum a b =

(a + b);;

### Ввод текста программы из текстового файла

open Printf

let sum a b =

(a + b);;

let () =

printf "%i\n" (sum (int\_of\_string Sys.argv.(1)) (int\_of\_string Sys.argv.(2)))

;;

## Задача 2

### Ввод текста программы в командной строке Ocaml

#let rec fact x =

if x <= 1 then 1

else x \* fact (x-1)

;;

#let comb n m =

if n == m then 1

else if n > m then (fact n) / ((fact m) \* (fact (n - m)))

else (fact m) / ((fact n) \* (fact (m - n)))

;;

#let rec combine xlist =

match xlist with

| [] -> []

| (c1,c2)::xtail ->

begin let c\_res = (comb c1 c2) in

c\_res::(combine xtail);

end

;;

#let list\_of\_tuples = [(3,1);(1,4);(2,6)];;

#let result\_l =

combine list\_of\_tuples

;;

### Ввод текста программы из текстового файла

open Printf

let rec get\_data x i =

if i < (Array.length Sys.argv - 1)

then get\_data (((int\_of\_string Sys.argv.(i)),(int\_of\_string Sys.argv.(i + 1)))::x) (i+2)

else x

;;

let print\_input\_data =

let i = ref 1 in

printf "input: [";

while !i<(Array.length Sys.argv - 1) do

printf "(%i, %i);" (int\_of\_string Sys.argv.(!i)) (int\_of\_string Sys.argv.(!i+1));

i := !i + 2

done;

printf "]\n";

;;

let rev list =

let rec aux acc = function

| [] -> acc

| h::t -> aux (h::acc) t in

aux [] list

;;

(\* reverse for get original view (as in input) \*)

let list\_of\_tuples = rev (get\_data [] 1);;

(\*.................. main solution ......................\*)

let rec fact x =

if x <= 1 then 1

else x \* fact (x-1)

;;

let comb n m =

if n == m then 1

else if n > m then (fact n) / ((fact m) \* (fact (n - m)))

else (fact m) / ((fact n) \* (fact (m - n)))

;;

let rec combine xlist =

match xlist with

| [] -> []

| (c1,c2)::xtail ->

begin let c\_res = (comb c1 c2) in

c\_res::(combine xtail);

end

;;

(\* combine our list with empty list \*)

let result\_l =

combine list\_of\_tuples

;;

(\*................. end of main solution ..................\*)

let print\_results =

printf "output: [";

List.iter (printf "%i;") result\_l;

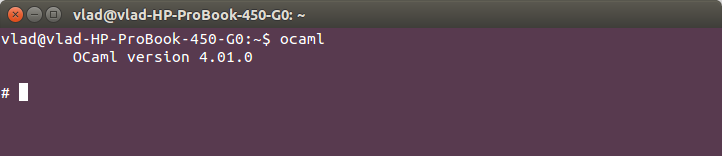
printf "]\n";

;;

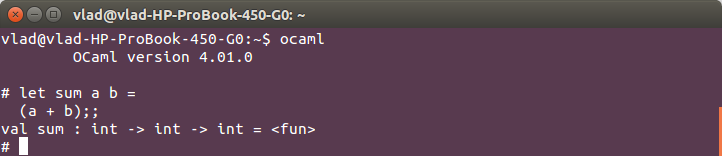
# Описание интерфейса (руководство пользователя)

## Задача 1

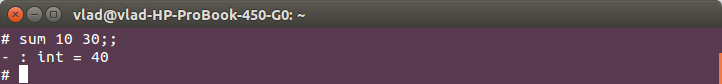
При вводе текста программы с командной строки пользователь должен запустить интерпретатор ocaml:



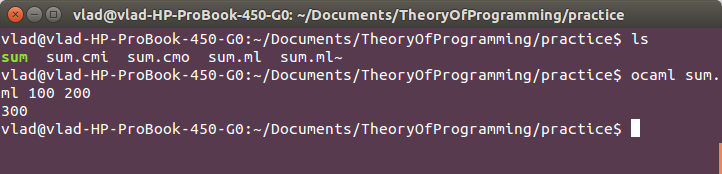
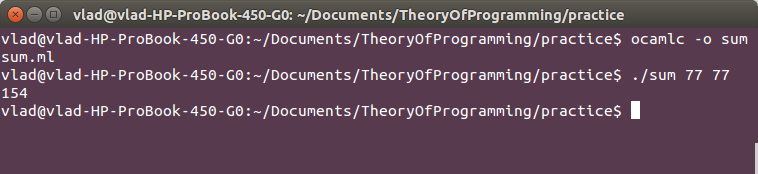
Задать функцию sum с двумя аргументами:



И вызвать её передав ей два параметра:

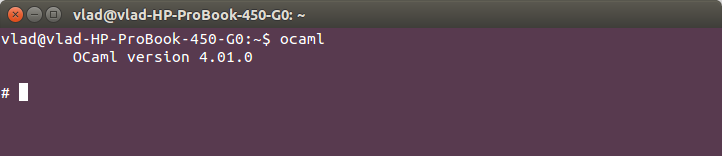


Для чтения текста программы из файла есть 2 способа:

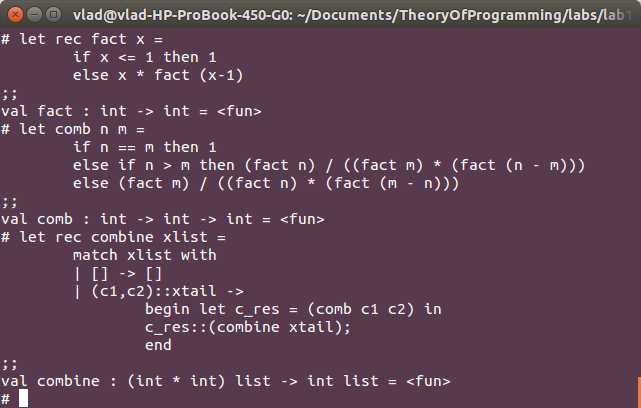
1. Ввести команду: ocaml sum.ml arg1 arg2:
2. Откомпилировав файл sum.ml командой ocamlc с параметром -o:  
   ocamlc -o sum sum.ml  
   После этого нужно запустить файл sum передав 2 аргумента командной строки:  
   ./sum arg1 arg2:

## Задача 2

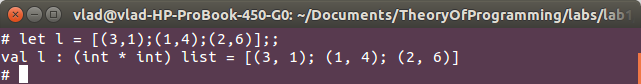
При вводе текста программы с командной строки пользователь должен запустить интерпретатор ocaml:



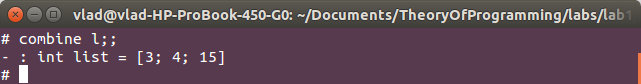
После этого нужно задать функции fact, comb и combination:



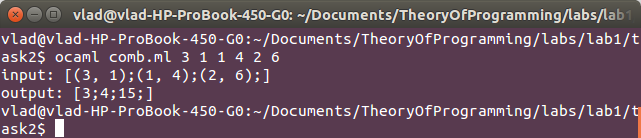
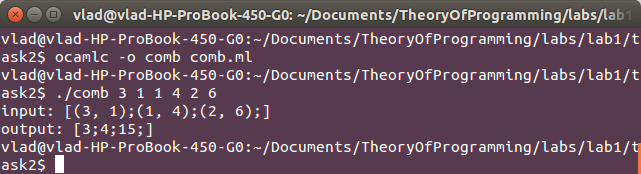
И задать list содержащий элементы в виде tuple - пар целых чисел:



После этого нужно вызвать функцию combine передав ей в качестве аргумента список:



Для чтения текста программы из файла есть 2 способа:

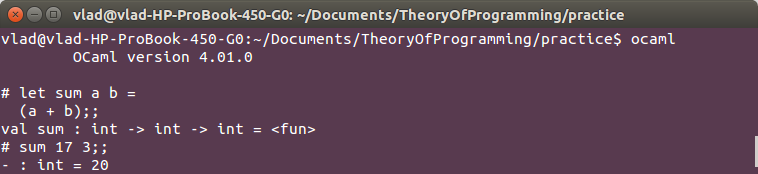
1. Ввести команду: ocaml comb.ml arg1 arg2 arg3 arg4 … argn-2 argn-1
2. Откомпилировав файл comb.ml командой ocamlc с параметром -o:  
   ocamlc -o comb comb.ml  
   После этого нужно запустить файл sum передав 2 аргумента командной строки:  
   ./comb arg1 arg2 arg3 arg4 … argn-2 argn-1

Обратите внимание на то, что все аргументы командной строки вводятся исключительно через пробел. Каждые 2 числа указанные после названия скрипта составляют пару. Если у числа нету пары, то такое число не входит в input.

# Описание тестовых примеров

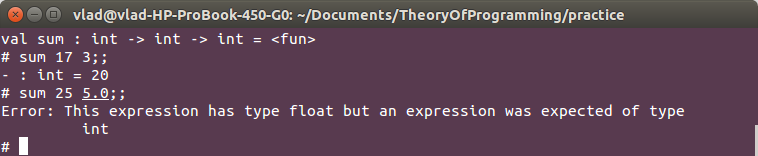
## Задача 1

Тест 1: “Целые числа”:



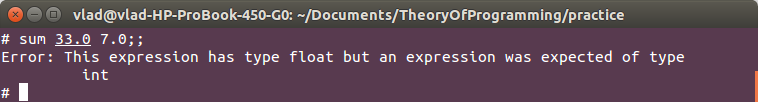
Как видим, на вход подаётся два целых числа 17 и 3, а на выходе результат операции является тоже целым числом - 20. Здесь соблюдены все типы, которые Ocaml смог вывести.

Тест 2: “Целое и дробное число”:



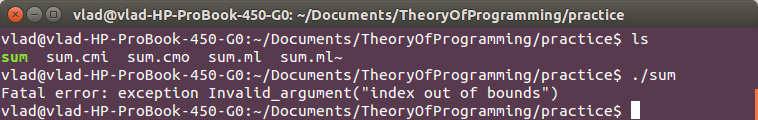
Здесь появляется ошибка при выводе типа, а именно - второй параметр является числом с плавающей запятой типа float, что противоречит заданной функции.

Тест 3: “Дробные числа”:



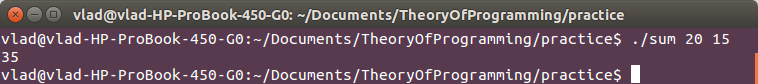
В данном примере мы пробуем задать функции sum два параметра типа float, но и это невозможно так как функция работает строго с целочисленными числами - int.

Тест 4: “Отсутствие аргументов командной строки”:



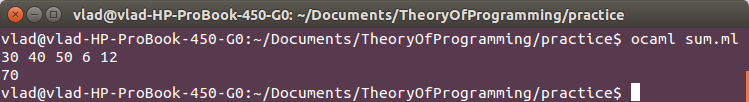
Если мы пропустим параметры для функции sum при передаче программе аргументов командной строки, то мы выйдем за пределы массива Sys.args, в котором на самом деле окажется всего один элемент - 0-й с названием программы в качестве значения.

Тест 5: “Два целых числа в качестве аргументов командной строки”:

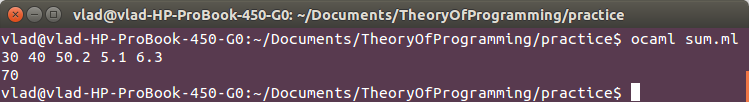


Здесь, мы вводим два целых числа 20 и 15. В результате получаем их сумму - 35, то есть всё верно.

Тест 6: “Больше двух целых чисел в списке аргументов”:

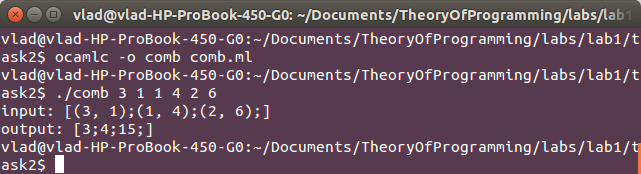


В данном тесте мы вводим сразу 5 целых чисел, а в результате программа считает сумму только первых двух. Даже если после двух первых чисел начать вводить дробные числа, то программа всё-равно будет считать только первые два:



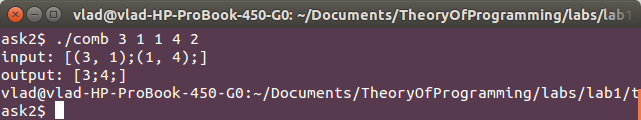
## Задача 2

Тест 1: “Ввод из файла с четным количеством параметров командной строки”:



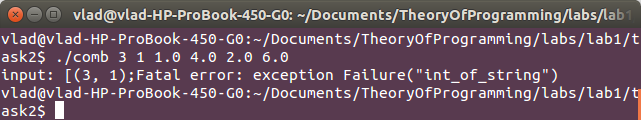
В примере показано, что при вводе четного числа параметров в input идут все числа по-парно. В результате работы программы мы получаем новый список, который состоит из комбинации (comb(m,n)) чисел, которые изначально входят в состав пар списка input.

Тест 2: “Ввод из файла с нечетным количеством параметров командной строки”:



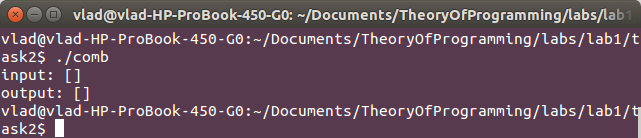
Предположим, что пользователь забыл ввести еще одно целое число для получения пары. В результате этого число, которое уже составляет часть пары, не попадёт в список input и не будет учитываться в результате - в списке output.

Тест 3: “Попытка ввода дробных чисел”:



При попытке ввода дробных чисел, эти числа не будут записываться в список input. При этом будет выдано исключение “ошибки приведения типа”. Напомню, что данная программа работает только с натуральными числами, поэтому приведение string к double здесь неуместно.

Тест 4: “Ввод без параметров командной строки”:



Если мы просто запустим скрипт на выполнение без указания аргументов командной строки, мы получим пустые списки input и output соответственно.

# Анализ ошибок (описание устранения замечаний)