Three Simple Exercises

Lab #1 (CS1010 AY2013/4 Semester 1)

Date of release: 24 August 2013, Saturday, 9am.

Submission deadline: 7 September 2013, Saturday, 9am.

School of Computing, National University of Singapore

0 Introduction

Important: Please read Lab Guidelines before you continue.

This lab requires you to do 3 simple exercises.
You may assume that the input data are according to specification, and hence there is no need to do input data validation, unless otherwise stated.
The maximum number of submissions for each exercise is **15**.

If you have any questions on the task statements, you may post your queries **on the relevant IVLE discussion forum**. However, do **not** post your programs (partial or complete) on the forum before the deadline!

1 Exercise 1: Investment

1.1 Learning objectives

- Reading input (scanf) and writing output (printf).
- Using data types: `int` and `float`.
- Using format specifier in output.
- Simple arithmetic computation.
- Using math function.

1.2 Task statement

If you invest *principal* amount of money (in dollars) at *rate* percent interest rate compounded annually, in *numYears* years, your investment will grow to

$$\frac{principal \times \left(1 - (rate/100)^{numYears+1}\right)}{1 - rate/100}$$

dollars.

Write a program **`invest.c`** that accepts positive integers *principal*, *rate* and *numYears* and computes the amount of money (of type `float`) earned after *numYears* years, presented in two decimal places.

You may assume that the interest rate is always smaller than 100.

1.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**).

Sample run #1:

```
Enter principal amount: 100
Enter interest rate   : 10
Enter number of years : 5
Amount = $111.11
```

Sample run #2:

```
Enter principal amount: 20000
Enter interest rate   : 5
Enter number of years : 10
Amount = $21052.63
```

1.4 Skeleton program and Test data

- The skeleton program is provided here: invest.c
- Test data: Input files | Output files

1.5 Important notes

- Do you remember to include the correct header files?
- If your program cannot be compiled, have you forgotten a certain compiler option?
- Print out the computed amount correct to **2 decimal places**.
- CodeCrunch awards mark for correctness ONLY if your output adheres to the given format. Hence, do not add any other characters (even blanks) that are not asked for in your output, or change the spelling of words in your output. The following outputs are all considered incorrect:
    - `amount = $111.11` (reason: "Amount" misspelt as "amount")
    - `Amount=$111.11` (reason: spaces around the = sign are missing)
    - `  Amount = $111.11` (reason: additional spaces before "Amount")
    - `Amount  =  $111.11` (reason: too many spaces around the = sign)
    - `Amount = 111.11` (reason: missing $ sign)
    - `Amount = $111.11.` (reason: additional dot at end of line)
- The skeleton program provided contains a few printf() statements, which you should not change, or your output will not be the same as the require one.
- Also remember to have a newline character (\n) in the last line of output of your program.

1.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 5 minutes
- Translating pseudo-code into code: 5 minutes
- Typing in the code: 5 minutes
- Testing and debugging: 15 minutes
- Total: 30 minutes

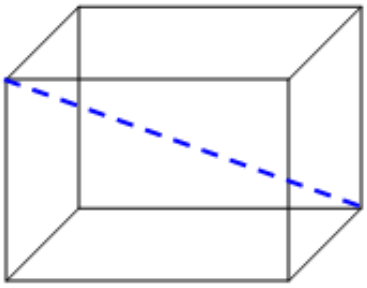
2 Exercise 2: Box Surface Area and Diagonal

2.1 Learning objectives

- Reading input (scanf) and writing output (printf).
- Using data types: `int` and `double`.
- Using format specifier in output.
- Simple arithmetic computation.
- Using math function.
- Writing your own user-defined functions.

2.2 Task statement

Write a program **`box.c`** that reads three positive integers representing the length, width and height of a box, and computes (1) its **surface area**, and (2) the length of the **diagonal** between two vertices of the box that are furthest apart.



Dotted blue line shows the diagonal connecting two vertices that are furthest apart.

You may assume that the surface area of the box does not exceed the maximum value representable in the `int` data type.

Hint: Do you know the formula for computing the length of the diagonal? If you are unable to get the formula, we will release it on the IVLE forum on 2 September 2013.

2.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**).
Sample run #1:

```
Enter length: 12
Enter width : 3
Enter height: 10
Surface area = 372
Diagonal = 15.91
```

Sample run #2:

```
Enter length: 10
Enter width : 20
Enter height: 30
Surface area = 2200
Diagonal = 37.42
```

2.4 Skeleton program and Test data

- The skeleton program is provided here: box.c
- Test data: Input files | Output files

2.5 Important notes

- Write two functions: **`compute_surface_area()`** and

`compute_diagonal()` to compute the surface area and length of diagonal of the box respectively. You are to determine what parameters to include for the functions.

- Do **not** use global variables. Global variables are variables that are declared outside all functions. Use of global variables will incur a big penalty.
- In writing functions, we would like you to include function prototypes before the main() function, and the function definitions after the main() function.
- The diagonal should be of `double` type.
- If your program cannot be compiled, have you forgotten a certain compiler option?
- Did you give descriptive names to your variables? Giving single-letter variable names (such as l, w, h) will incur some penalty.

2.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 5 minutes
- Translating pseudo-code into code: 10 minutes
- Typing in the code: 10 minutes (most of the code is already given in the skeleton program)
- Testing and debugging: 20 minutes
- Total: 45 minutes

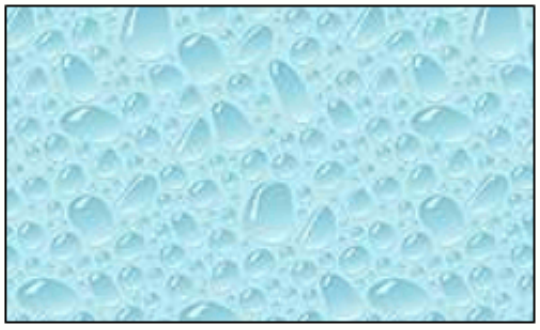
3 Exercise 3: Packing

3.1 Learning objectives

- Reading input (scanf) and writing output (printf).
- Using data type: `int`.
- Simple arithmetic computation.
- Writing function.
- Using selection statement.
- Simple problem solving.

3.2 Task statement

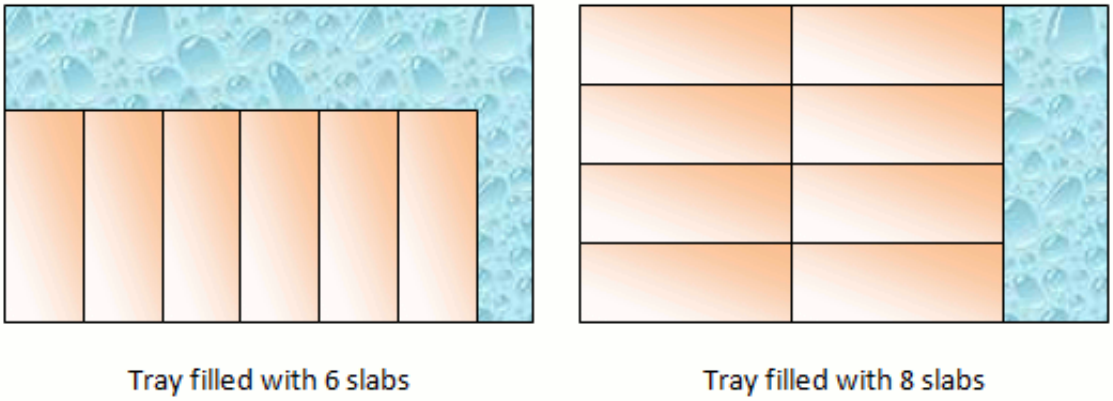
You are given a rectangle tray and an unlimited supply of slabs. An example of a 12×20 tray and a 8×3 slab are shown below.



A 12×20 tray

A 8×3 slab

You are to find the maximum number of slabs that can be packed into the tray. The slabs may be packed in either one of the two orientations, as shown below, but not in a mix of orientations.



The figures above show that the tray may be filled with 6 slabs arranged in one orientation, or 8 slabs arranged in the other orientation. Hence, the answer is 8.

You are to write a program **`packing.c`** to read in the dimensions of a tray and a slab, and to compute the **maximum possible number of slabs** that could be packed onto the tray.

You may assume that all inputs are positive integers.

3.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**).
Sample run #1:

```
Enter dimension of tray: 12 20
Enter dimension of slab: 8 3
Maximum number of slabs = 8
```

Sample run #2:

```
Enter dimension of tray: 60 35
Enter dimension of slab: 6 8
Maximum number of slabs = 40
```

3.4 Skeleton program and Test data

- The skeleton program is provided here: packing.c
- Test data: Input files | Output files

3.5 Important notes

- Write a function **`compute_max_slabs()`** to compute the maximum number of slabs that can be packed onto the tray. You may write other supporting functions if necessary.
- In writing functions, we would like you to include function prototypes before the main() function, and the function definitions after the main() function.

3.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 40 minutes

- Translating pseudo-code into code: 15 minutes
- Typing in the code: 15 minutes
- Testing and debugging: 30 minutes
- Total: 100 minutes

3.7 Exploration

Here are some possible extensions of the problem for you to try on your own. You do NOT need to submit them.

- If we allow mixing of orientations for the slabs, we can pack more slabs onto the tray. For the above example, we can pack 9 slabs.
- If we extend the problem to 3 dimensions, where the tray becomes a box, and the slabs become blocks, the problem would become more challenging (and beyond the scope of this module). Try it out nonetheless, to appreciate how much more complicated it becomes compared to the original problem.

4 Deadline

The deadline for submitting all programs is **7 September 2013, Saturday, 9am**. Late submission will NOT be accepted.

- 0 Introduction
- 1 Exercise 1: Investment
    - 1.1 Learning objectives
    - 1.2 Task statement
    - 1.3 Sample run
    - 1.4 Skeleton program and Test data
    - 1.5 Important notes
    - 1.6 Estimated development time
- 2 Exercise 2: Volume of a Box
    - 2.1 Learning objectives
    - 2.2 Task statement
    - 2.3 Sample runs
    - 2.4 Skeleton program and Test data
    - 2.5 Important notes
    - 2.6 Estimated development time
- 3 Exercise 3: Packing
    - 3.1 Learning objectives
    - 3.2 Task statement
    - 3.3 Sample runs
    - 3.4 Skeleton program and Test data
    - 3.5 Important notes
    - 3.6 Estimated development time
    - 3.7 Exploration
- 4 Deadline

Aaron
Monday, August 19, 2013 08:25:07 PM SGT