Three Simple Exercises

Lab #1 (CS1010 AY2011/2 Semester 4)

Date of release: 26 June 2012, Tuesday, 14:00hr.

Submission deadline: 30 June 2012, Saturday, 23:59hr.

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.

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!

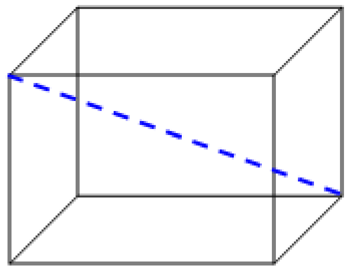
1 Exercise 1: Box Surface Area and Diagonal

1.1 Learning objectives

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

1.2 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, please email Song Yi.

1.3 Sample run

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall -lm box.c -o box
$ box
Enter length: 12
Enter width : 3
Enter height: 10
Surface area = 372
Diagonal = 15.91
```

1.4 Number of submission

For this exercise, the number of submission is **10**.

1.5 Important notes

- The skeleton program is provided here: box.c
- Write two functions: **`compute_surface_area`** and **`compute_diagonal`** to compute the surface area and length of diagonal of the box respectively.
- 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.
- Print out length of the diagonal 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:
    - `surface area = 372` (reason: "Surface" misspelt as "surface")
    - `Surface area=372` (reason: spaces around the = sign are missing)
    - `   Surface area = 372` (reason: additional spaces before "Surface")
    - `Surface area  =  372` (reason: too many spaces around the = sign)
    - `Surface area = 372.` (reason: additional dot at end of line)
- If your program cannot be compiled, have you forgotten a certain compiler option?

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 (most of the code is already given in the skeleton program)
- Testing and debugging: 10 minutes
- Total: 25 minutes

2 Exercise 2: Semester Average Point (SAP)

2.1 Learning objectives

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

- Using selection statement.

2.2 Task statement

You want to write a program **`sap.c`** to help fellow NUS undergraduates find out how well they should do in the current semester to meet their target Cumulative Average Point (CAP).

Given an undergraduate's current CAP, called `capCurrent`, the number of MCs he has already attempted, `mcAttempted`, the number of MCs he is attempting this semester, `mcAttempting`, and his target CAP at the end of this semester, `capTarget`, you are to help him compute his **Semester Average Point (SAP)**, `sap`, for the current semester, given this formula:

$$capTarget = \frac{(capCurrent \times mcAttempted) + (sap \times mcAttempting)}{mcAttempted + mcAttempting}$$

`mcAttempted` and `mcAttempting` are of `int` type; `capCurrent`, `capTarget` and `sap` are of `float` type. All are positve. You may assume that `capTarget` is not smaller than `capCurrent`.

2.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall sap.c -o sap
$ sap
Enter MCs attempted and current CAP: 100 1.90
Enter MCs attempting and target CAP: 20 2.00
SAP >= 2.50
```

Second sample run:

```
Enter MCs attempted and current CAP: 100 3.20
Enter MCs attempting and target CAP: 20 3.50
SAP >= 5.00
```

The third sample run below shows what the output should be if the SAP exceeds 5.00:

```
Enter MCs attempted and current CAP: 32 3.25
Enter MCs attempting and target CAP: 16 4.00
Mission impossible!
```

2.4 Number of submission

For this exercise, the number of submission is **15**.

2.5 Important notes

- The skeleton program is provided here: sap.c
- Write a function **`compute_sap`** to compute the SAP of an undergraduate.
- In writing functions, we would like you to include function prototypes before the main function, and the function definitions after the main function.
- Print out the computed SAP correct to **2 decimal places**.

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): 15 minutes
- Translating pseudo-code into code: 15 minutes
- Typing in the code: 10 minutes
- Testing and debugging: 20 minutes
- Total: 60 minutes

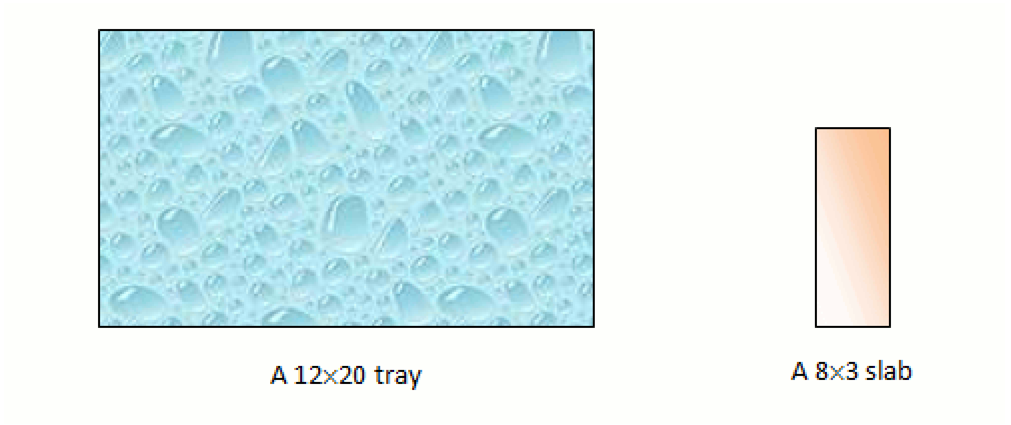
3 Exercise 3: Packing

3.1 Learning objectives

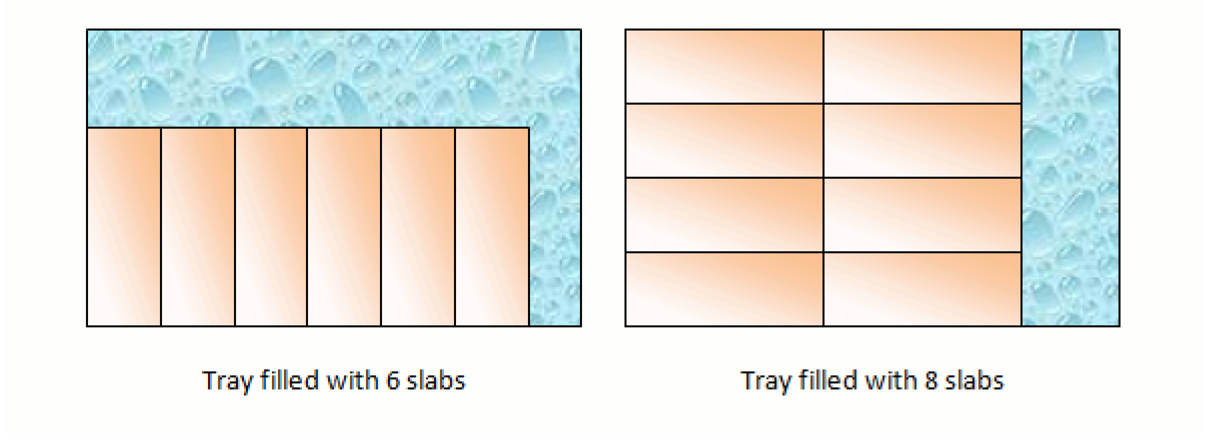
- Reading input (scanf) and writing output (printf).
- Using data type: `int`.
- Simple arithmetic computation.
- Writing function.
- Using selection statement.
- Simple problem solving.

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.



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**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall packing.c -o packing
$ packing
Enter dimension of tray: 12 20
Enter dimension of slab: 8 3
Maximum number of slabs = 8
```

Second sample run:

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

3.4 Number of submission

For this exercise, the number of submission is **15**.

3.5 Important notes

- The skeleton program is provided here: packing.c
- 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 **30 June 2012, Saturday, 23:59hr**. Late submission will NOT be accepted.

Go back to CS1010 Labs page.

Aaron Tan
Wednesday, August 31, 2011 06:46:32 PM SGT