Department of Computing and Information Systems

COMP20005 Engineering Computation
Semester 1, 2016
Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of loops and `if` statements by writing a program that sequentially processes a file of text data. You are also expected to make use of functions; and in the final stage of the project, will need to make use of arrays (covered in Weeks 6 and 7, and in Chapter 7).

Spatial Data

Spatial data is very important for planning and optimization purposes, varying from services such as Google maps through to tools for computer-aided design. In this project you will work with (highly simplified) data that represents rooms and apartments in an apartment building, with each room stored as one or more rectangular regions. Each section of input for your program will have the following structure, describing a single apartment that has n "rooms":

apartment_number	an apartment number (positive integer);
$room\_type_0$	an indicator of the type of room (positive integer);
$room\_num_0$	the ordinal room number within this type (positive integer);
$xsize_0$	the x dimension in meters of this room (double);
$ysize_0$	the y dimension in meters of this room (double);
...	*and so on, groups of four values, one group for each room or part room*;
$room\_type_{n-1}$	an indicator of the type of the last room (positive integer);
$room\_num_{n-1}$	the ordinal room number within the last type (positive integer);
$xsize_{n-1}$	the x dimension in meters of the last room (double);
$ysize_{n-1}$	the y dimension in meters of the last room (double);
-1	sentinel value to indicate that there are no more rooms in this apartment.

There may be multiple such sections in the input file, each one describing a different apartment. Some test files structured according to this format are provided on the LMS, and you should study them while reading this handout. Different data will be used during the post-submission testing process, but will have exactly the same structure as the examples shown. *You may assume through all four stages that all test input files will be complete and correct, and error-free.*

The following codes are used in the input files to represent room types:

Dry areas		Wet areas		Utility areas	
1	Hallway	4	Bathroom	7	Storage
2	Bedroom	5	Kitchen	8	Garage
3	Living	6	Laundry	9	Balcony

No other room types will be considered in this project. For example, the input file `test1.txt` (available on the LMS) describes the room types and sizes of the apartment shown in Figure 1. Notice how the irregular-shaped living area is represented by three consecutive input lines, each with the same *room_type* and *room_num* values. Any rectilinear room can be represented in this way as the sum of non-overlapping rectangles. You may assume that no apartment will have more than 100 rooms; that any particular input file will have fewer than 100 apartments described in it; and that, for simplicity, there will never be any curved or angled walls permitted (these are budget apartments, not luxury penthouses). In the example files provided `test0.txt` has three rooms in a simple hotel-style apartment; `test1.txt` has several rooms, but still just a single apartment; and `test2.txt` lists multiple rooms across multiple apartments.

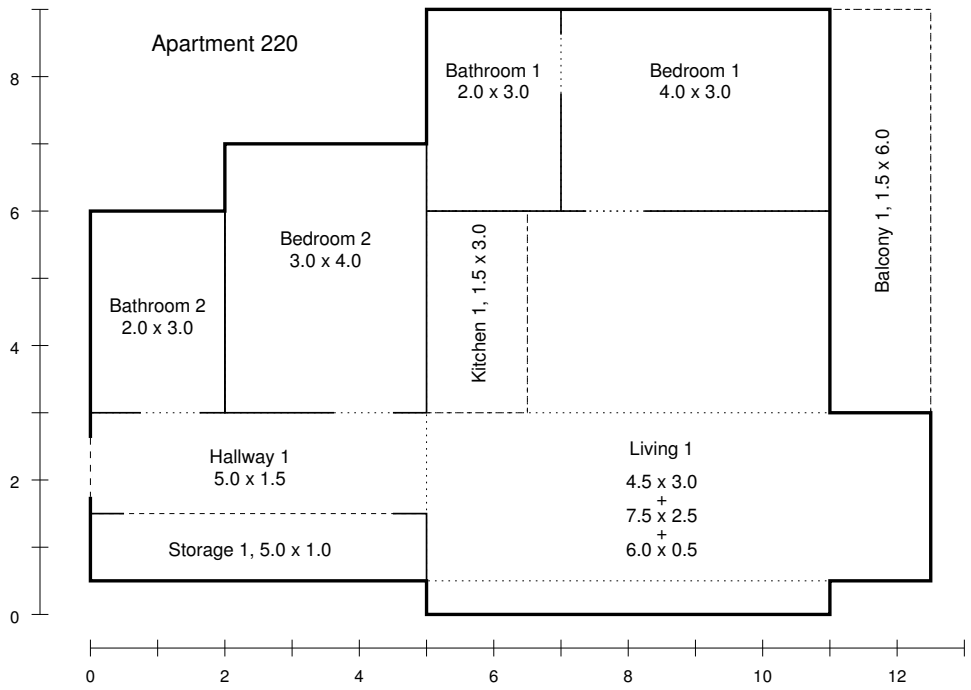

Figure 1: Example apartment configuration, leading to the input file `test1.txt`.

Stage 1 – Reading and Printing (marks up to 4/10)

It is possible to complete this stage without using arrays. However, you may also make use of arrays in this stage if you wish to, and if you are confident that you will be proceeding right through to Stage 4, you will need to employ arrays in your program.

Write a program that reads (only) the first apartment in the input file, and prints out a summary of the room types, sizes, and areas in that apartment in the order that the lines appear in the input file. Any other input sections can be ignored. The required output format is shown in this example:

```
mac: ./my-ass1 < test1.txt
Apartment 220
-------------
    Hallway  1  5.00  1.50   7.50
    Bedroom  1  4.00  3.00  12.00
    Bedroom  2  3.00  4.00  12.00
    Living   1  4.50  3.00  13.50
    Living   1  7.50  2.50  18.75
    Living   1  6.00  0.50   3.00
    Bathroom 1  2.00  3.00   6.00
    Bathroom 2  2.00  3.00   6.00
    Kitchen  1  1.50  3.00   4.50
    Storage  1  5.00  1.00   5.00
    Balcony  1  1.50  6.00   9.00
    Total area             97.25 metres^2
```

The best way to get data into your program is to edit it in to a text file (with a `.txt` extension, `jEdit` can create/edit these too), and then execute your program from the command-line, feeding the data in via input redirection (using `<`). You should also construct other test inputs too, of course.

Because all input data will come from a file (including during the submission testing), you should not print any prompts at all. All input values should be read using either "`%d`" or "`%lf`" format descriptors,

as appropriate to the value being read. That way you don't need to worry about the exact placement of blank and newline characters in the input stream.

Stage 2 – Combining Zones (marks up to 6/10)

It is also possible to complete this stage without using arrays if you do not plan to proceed right through to Stage 4.

Modify your program so that the components of the same room are still listed with their individual sizes, but with a single total area for each room as a whole. For the same test data, your program should now generate:

```
mac: ./my-ass1 < test1.txt
Apartment 220
-------------
    Hallway  1  5.00  1.50   7.50
    Bedroom  1  4.00  3.00  12.00
    Bedroom  2  3.00  4.00  12.00
    Living   1  4.50  3.00   ---
    Living   1  7.50  2.50   ---
    Living   1  6.00  0.50  35.25
    Bathroom 1  2.00  3.00   6.00
    Bathroom 2  2.00  3.00   6.00
    Kitchen  1  1.50  3.00   4.50
    Storage  1  5.00  1.00   5.00
    Balcony  1  1.50  6.00   9.00
    Total area             97.25 metres^2
```

The input lines for each apartment may be assumed to always arrive in strictly sorted order according to the numeric *room_type* code, and then according to the *room_num* value as a secondary key.

Note that your Stage 2 program does not need to also generate the Stage 1 output. The Stage 2 modifications to your program are to ***replace*** *the Stage 1 output.*

Stage 3 – Multiple Loops (marks up to 8/10)

Now extend your program so that it reads all of the apartment descriptions in the input stream, each of them ended by a -1 sentinel, and prints out the Stage 2 summary for each apartment. Examples of input files and the corresponding required output are provided on the LMS. This stage can also be done without the use of arrays. But by now, you should be getting tempted – they are not necessary, but they sure make the job easier to do (and the program easier to understand).

Wherever appropriate, code should be shared between the stages through the use of functions. In particular, there shouldn't be long (or even short) stretches of repeated or similar code appearing in different places in your program.

Stage 4 – Overall Reporting (marks up to 10/10)

To complete this stage you will need to make use of arrays. If you have completed Stages 1, 2, and 3 without arrays, you should be sure to save a copy of that version of your program in to a separate file, as an insurance policy that can be submitted again later if you are unable to get your Stage 4 solution operational.

Extend your program again, so that once the end of input is reached, an overall summary table is generated, with one row per apartment. The spaces making up that apartment should be added up according to three categories: *dry* areas are hallways, bedrooms, and living areas; *wet* areas are bathrooms, kitchens,

and laundries; and *utility* areas are storage rooms, garages, and balconies. The size in that apartment of the three categories should be listed, together with the percentage of the apartment's total. For example,

```
mac: ./my-ass1 < test2.txt

<<<<< the previous stage2/stage3 output >>>>>

+-------+---------------+---------------+---------------+
| Apart |    Dry areas  |    Wet areas  |  Utility areas |
+-------+---------------+---------------+---------------+
|  220  |  66.75  68.6% |  16.50  17.0% |  14.00  14.4% |
|  221  |  38.00  73.1% |   8.00  15.4% |   6.00  11.5% |
+-------+---------------+---------------+---------------+
```

Further examples showing the full output that is required are provided on the LMS.

The boring stuff...

This project is worth 10% of your final mark. A rubric explaining the marking expectations will be provided on the LMS.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. Submission will *not* be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software system known as `submit`. You can (and should) use `submit` **both early and often** – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. *Failure to follow this simple advice is highly likely to result in tears.* Only the last submission that you make before the deadline will be marked.

You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be individual work, not copied from anyone else. So, do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your "Uni backup" memory stick to others for any reason at all; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" when they ask for a copy of, or to see, your program, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs are so identified will be referred to the Student Center for possible disciplinary action without further warning. This message **is** the warning.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **10:00am on Tuesday 26 April** will lose penalty marks at the rate of two marks per day or part day late. Students seeking extensions for medical or other "outside my control" reasons should email `ammoffat@unimelb.edu.au` as soon as possible after those circumstances arise. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops in to something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Marks and a sample solution will be available on the LMS by Wednesday 11 May.

And remember, programming is fun!