School of Computing and Information Systems

comp20005 Intro. to Numerical Computation in C
Semester 1, 2025
Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of loops, `if` statements, functions, and arrays; writing a program that first reads some numeric data and then processes it. You may also make use of structures if you wish (Chapter 8). But there is no rubric requirement for you to make use of `struct` types, and the required tasks can be carried out using a set of parallel one-dimensional arrays.

Spatial Data

Spatial data is 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 a building, with each room stored as one or more rectangular sections. All input will have this format:

apartment_number	an apartment number (integer, 1–999);
room_type	room type for the first room section (integer, 1–9);
room_num	ordinal room number within type for first room section (integer, 1–9);
xsize	the x dimension in meters of the first room section (positive double);
ysize	the y dimension in meters of the first room section (positive double);
...	*and so on, one group of four values for each rectangular room section*;
-1	sentinel value to indicate that there are no more rooms in this apartment
...	*and so on, one set of room sections for each apartment in the input file*;

An input file might describe one apartment, or might describe multiple apartments, each with its own block of room sections. Apartment numbers are unique non-consecutive integers.

The following *room_type* codes are used in the input files to represent the types of rooms:

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 used. 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 input lines with the same *room_type* and *room_num* values, each describing one non-overlapping rectangular section of the overall room. Any rectilinear room can be represented in this way as the sum of non-overlapping rectangles. There will never be any curved or angled walls permitted in the apartments, and all rooms will be rectilinear.

See the input files linked from the LMS Assignment 1 page for detailed examples. You should study them carefully 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 provided.

You may assume that all input files will be complete and correct according to this format, and will be error-free. You may also (if you need to) assume that no apartment will have more than 99 *separate rooms; that no room will have more than* 9 *sections; that no input file will contain more than* 999 *apartments; and that no apartment will have a total area greater than* 999.99 *square meters.*

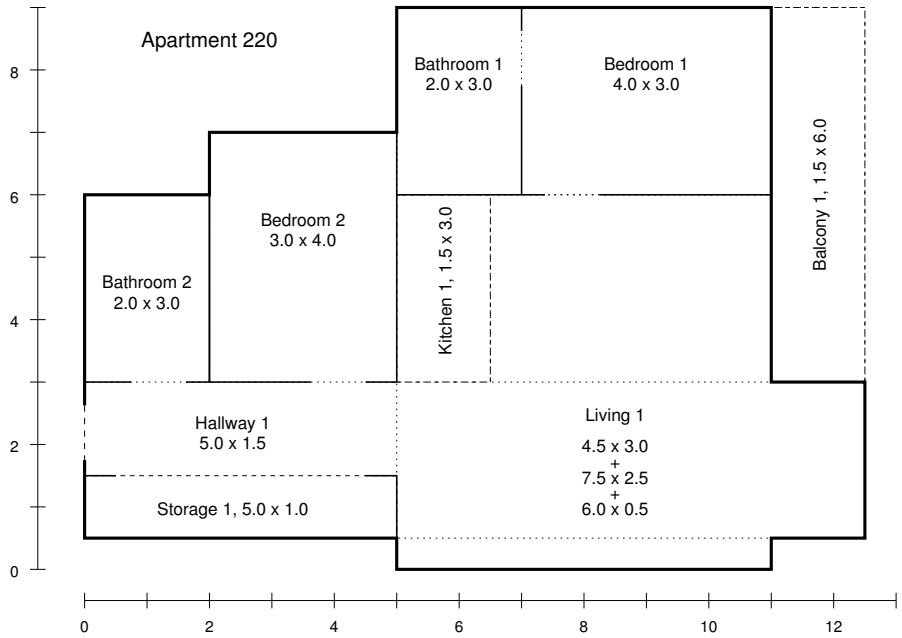

Figure 1: Example apartment configuration, corresponding to the input file `test1.txt`.

```
mac: more test1.txt
220
1 1 5.00 1.50
2 1 4.00 3.00
2 2 3.00 4.00
3 1 4.50 3.00
3 1 7.50 2.50
3 1 6.00 0.50
4 1 2.00 3.00
4 2 2.00 3.00
5 1 1.50 3.00
7 1 5.00 1.00
9 1 1.50 6.00
-1
mac:
```

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

Figure 2: File `test1.txt` (left) and required Stage 1 output (right).

Stage 1 – Reading and Printing One Apartment (marks up to 8/20)

Write a program that reads 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, with consecutive room sections amalgamated into a single total area. In this stage any input values that appear after the first sentinel value should be ignored. The required output format for `test1.txt` is shown in Figure 2.

To keep Stage 1 simple, you may assume that the room sections for the first apartment in the input file are already in sorted order according to the numeric *room_type* code, with ties broken according to the *room_num* value. That is, all of the sections for each room will appear consecutively in the input.

Make sure that you know how to execute your program from the command-line, reading from a text file with input redirection (using `<` at the shell command level). Three test files are available from the LMS Assignment 1 page: `test0.txt` is a simple student dormitory-style (Scape/Journal/Unilodge etc) apartment; `test1.txt` as shown above, a two-bed two-bath apartment (Figure 1); and `test2.txt` describing multiple rooms across three different apartments. You should also create other test files of

```
mac: ./ass1-soln < test2.txt
<<<<< first comes three sets of Stage1/Stage2 output >>>>>

+-------+----------------+----------------+----------------+
| Apart |    Dry areas   |    Wet areas   |  Utility areas |
+-------+----------------+----------------+----------------+
|  108  |  12.25  65.3%  |   5.00  26.7%  |   1.50   8.0%  |
|  220  |  66.75  68.6%  |  16.50  17.0%  |  14.00  14.4%  |
|  221  |  38.00  63.3%  |   8.00  13.3%  |  14.00  23.3%  |
|  222  |  79.50  51.1%  |  33.50  21.5%  |  42.50  27.3%  |
+-------+----------------+----------------+----------------+

tadaa!
```

Figure 3: Required output for Stage 3 on input file `test2.txt`.

your own – maybe measure up your own apartment and share via the Discussion?

Because all input data will come from a file you should not print any prompts. Note also that all input values can be read using either "`%d`" or "`%lf`" descriptors. That means that you don't need to worry about the exact placement of blank and newline characters in the input stream.

Stage 2 – Nested Loops (marks up to 16/20)

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 1 summary for each apartment that is in the input. File `test2.txt` provides an example input file.

In this stage you may not assume that the room sections for each apartment are already sorted by *room_type* and *room_num*. Instead, you must implement a sorting function that puts the room sections of each apartment into the required order before the output summary is printed. If both *room_type* and *room_num* are tied, the ordering from the input file should be preserved. You should use insertionsort to do the sorting.

One output block similar to Figure 2 is required for each of the apartments in the input stream. Note that you are expected to share code between stages through the use of functions. Long (or even short) stretches of repeated or similar code appearing in different places in your program will incur mark deductions.

The LMS Assignment 1 page includes a detailed marking rubric; be sure to read it carefully!

Stage 3 – The Final Tabulation (marks up to 20/20)

Extend your program a second time, so that once the end of input is reached and the details of the last apartment have been read, an overall summary table is generated, with one row per apartment. The rooms in each 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 totals (square meters) of the three categories should be listed, together with the percentage of the apartment's total. Figure 3 provides an example of what is required. The summary lines should be in the same order as the apartments appeared in the input file. The last output line is also required :-)

Examples showing the full output that is required are linked from the "Assignment 1" page in the LMS.

Refinements to the Specification

There may be areas where this specification needs clarification or even correction, and you should check the "Assignment 1" Ed Discussion page regularly for updates to these instructions. There is also a range of information linked from the "Assignment 1" LMS page that you need to be aware of.

The Boring Stuff...

This project is worth **20% of your final mark**, and is due at **6:00pm on Friday 2 May**.

Submissions made after the deadline will incur penalty marks at the rate of two marks per (working) day or part day late. Students seeking extensions for medical or other "outside my control" reasons should follow the process that is described in the LMS Assignment page as soon as possible after those circumstances arise.

Submissions will be completely closed one week after the original deadline, and no extensions of more than five working days will be granted. If you are disadvantaged by more than five working days other forms of mark adjustment will then be considered.

Submission: Your `.c` file must be uploaded to GradeScope via the LMS "Assignment" page. *Don't forget to include, sign, and date the Authorship Declaration that is required at the top of your program.*

Multiple submissions may be made; only the last submission that you make before the deadline will be marked. If you make any late submission at all, your on-time submissions will be ignored, and if you have not been granted an extension, the late penalty will be applied.

Marking Rubric: The marking expectations are linked from the assignment LMS page. Feedback and a sample solution will be made available approximately two weeks after submissions close.

Academic Honesty: You may *discuss* your work during your workshop and with other students, but what gets typed into your program must be individual work, not copied from anyone else, and not developed jointly with anyone else. So, do **not** give hard copy or soft copy of your work to anyone else, no matter what they tell you (the usual line is "I just want to take a look and get some ideas, I won't copy, honest"); and do **not** ask others to give you their programs. If a friend does ask to see your program, the friendliest thing you can do is to respond with a firm "**no**". And if they keep on asking, they are not actually a friend. Nor may you make any use at all of AI tools such as ChatGPT. *The work you submit must be completely your own*. See `https://academicintegrity.unimelb.edu.au` for more information.

Solicitation of solutions via posts to "tutoring" sites or online forums is also Academic Misconduct, whether or not there is payment, and whether or not you actually employ any solutions that may result. In the past students have had their enrollment terminated for such behavior. Nor should you allow your code to to be visible at any public location (`github`, `codeshare.io`, etc) prior to the marks release.

The LMS page links to a program skeleton that includes an Authorship Declaration that you must "sign" and date and include at the top of your submitted program. Marks will be deducted (see the rubric linked from the LMS page) if you do not include the declaration, or do not sign it, or do not comply with its expectations. A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions. Students whose programs are identified as containing significant overlaps will have substantial mark deductions applied for failure to comply with instructions, or risk being referred to the Student Center for possible disciplinary action, without further warning.

And remember, programming is fun!