Imperial College London **EEE computing**

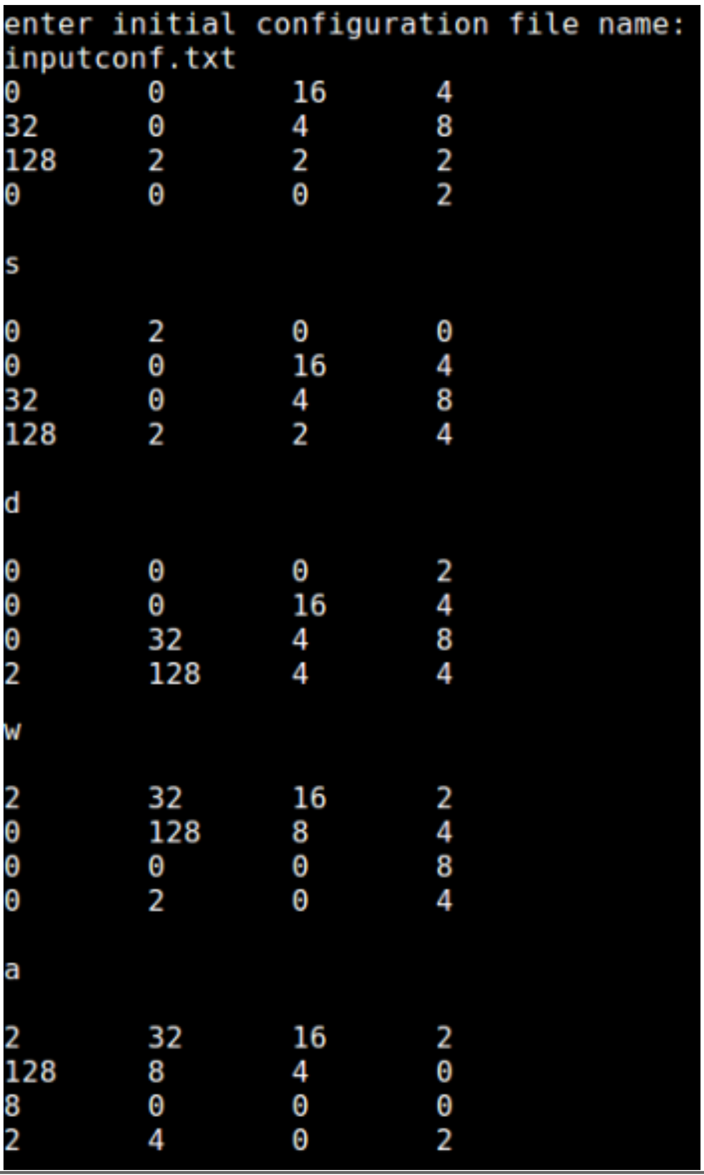
Assignment 2 -- Spring 2019

Write a C++ program for the `2048` game http://gabrielecirulli.github.io/2048/ [http://gabrielecirulli.github.io/2048/] (the submission area is linked from the "EE1-07 Software Engineering 1: Introduction to Computing (2018-2019)" page on Blackboard).

The following screenshots show the game interface and expand on more specific requirements.

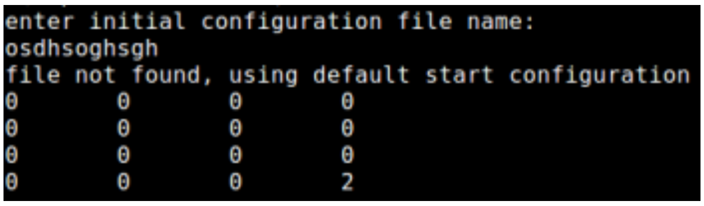
Let file `inputconf.txt` contained in the same directory of the executable (this, as usual, assuming the program is run from the command line, conditions may vary if an IDE is used) have the following content:

```
0 0 16 4
32 0 4 8
128 2 2 2
0 0 0 2
```



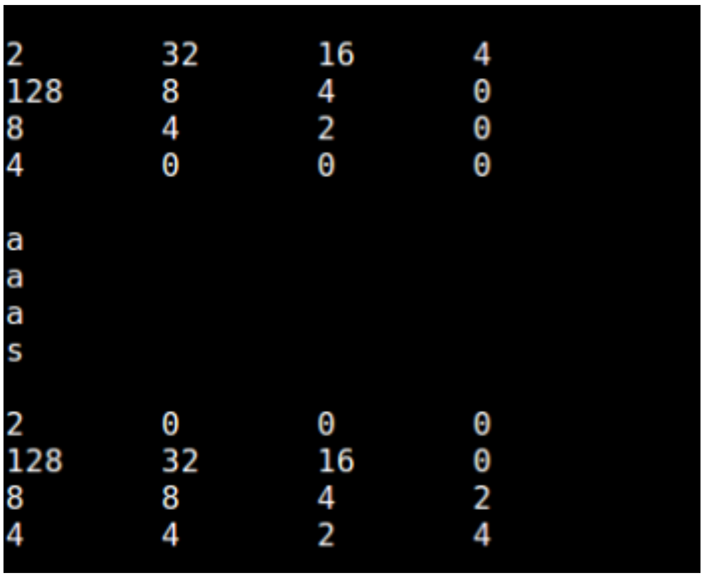
```
enter initial configuration file name:
inputconf.txt
0       0       16      4
32      0       4       8
128     2       2       2
0       0       0       2

s

0       2       0       0
0       0       16      4
32      0       4       8
128     2       2       4

d

0       0       0       2
0       0       16      4
0       32      4       8
2       128     4       4
w

2       32      16      2
0       128     8       4
0       0       0       8
0       2       0       4

a

2       32      16      2
128     8       4       0
8       0       0       0
2       4       0       2
```

The program initially asks for the name of a file containing the initial configuration. If the file is found, it is loaded and the game begins from said configuration. Commands are given in input using `a` for left, `s` for down, `d` for right and `w` for up. The letters are given in input in the usual way (i.e. followed by enter). In the original game either a `2` or a `4`, with a certain probability, is placed randomly in one of the free tiles after a move. For this assignment it should always be a `2`.



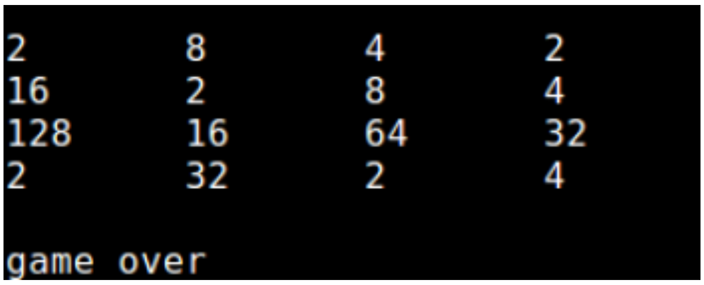
```
enter initial configuration file name:
osdhsoghsgh
file not found, using default start configuration
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       2
```

If the input file is not found then a message is printed and the game starts from a default configuration (a `4×4` grid with a `2` in the bottom right corner).



```
2       32      16      4
128     8       4       0
8       4       2       0
4       0       0       0

a
a
a
s

2       0       0       0
128     32      16      0
8       8       4       2
4       4       2       4
```

What should happen when the user enters a move in a direction that doesn't involve changes to the content of the grid. Note that this also means that, as in the original game, when this happens no additional `2` is generated.



```
2       8       4       2
16      2       8       4
128     16      64      32
2       32      2       4

game over
```

No possible moves: game over. This is the only termination condition: the game should go on even when the `2048` value is reached.

The examples above are about a `4×4` grid as in the original game, but the program should work on any `nxn` (let's say with `n > 1`) grid read from the file (the default configuration is always `4×4`).

Moreover the file may well contain numbers that are not powers of `2` (but what is generated in a random position after a move is still always a `2`) and these should be handled normally (e.g. they should be summed together if they are the same number and a move is made such that they are shifted to match together). We assume that the file always contains the right number of elements (i.e. such that it can be a square) and that it always contains at least one non-zero element.

Also, note that `inputconf.txt` is just an example, there isn't a fixed name for the file that contains the input (that's why it is read from the user and not hardcoded in the program).

Do not alter the way things are printed, and do not add anything else. Follow as closely as possible the screenshots.

Design

The **data structure** for the grid needs to be a one dimensional vector containing the data row by row as we have done for the sudoku checker program.

And, as in the sudoku checker program, one of the building blocks of your program will be a function like `twod_to_oned` in order to map the bidimensional indices we have in mind when we deal with the grid, to the corresponding one-dimensional index on the one-dimensional vector.

The following is a list of **functions** you have to define and use in your program.

These will be automatically tested, so it's important that you keep the name and parameter list as specified and follow all the instructions very carefully.

It is also important that you make good use of them in your design of the program: perfunctory inclusion in the code just to say you have followed the requirements is not enough.

You can define and use other functions in addition to these and you can define and use other functions to build the implementation of these functions.

proc_num

```
bool proc_num(std::vector<int>& v, int bi, int ei)
```

The first parameter is a `vector` of `int` (`v`) which the function uses both for input and output. Parameters `bi` and `ei` are `int` and they are used for input. The function returns a `bool`.

The examples below specify the required behaviour of the function:

Example 1:

```
v (before):

[0 0 4 0 0 0 3 0 0 3 0 3 2 0 0 5 0 6 4]

bi: 4

ei: 14

---

v (after):

[0 0 4 0 6 3 2 0 0 0 0 0 0 0 5 0 6 4]
```

```
return: true
```

Example 2:

```
v (before):

[0 0 0 5 0 2 0 0 2 0 0 0 2 0 0 2 6 0 0 5 0 6 4]

bi: 0

ei: 23 [or v.size()]

---

v (after):

[5 4 4 6 5 6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

return: true
```

Example 3:

```
v (before):

[0 0 0 5 1 2 0 2 2 0 0 0 2 0 0 2 6 0 0 5 0 6 4]

bi: 3

ei: 7

---

v (after):

[0 0 0 5 1 2 0 2 2 0 0 0 2 0 0 2 6 0 0 5 0 6 4]

return: false
```

Example 4:

```
v (before):

[0 0 0 5 1 2 0 2 2 0 0 0 2 0 0 2 6 0 0 5 0 6 4]

bi: 1

ei: 4

---

v (after):

[0 5 0 0 1 2 0 2 2 0 0 0 2 0 0 2 6 0 0 5 0 6 4]

return: true
```

Note that `ei` is not the index of the last element involved in the transformation, rather it's 1 + the index of the last element involved in the transformation. (It is quite common, when ranges are expressed in C++, that the left extreme is included in the interval while the right one isn't.)

rotate_anti_clock

```
void rotate_anti_clock(std::vector<int>& v)
```

This function performs an “anti-clockwise rotation” of the grid contained in vector `v` (again this is both an input and output parameter).

For example if `v` initially contains `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16`, the function should change the content to `4 8 12 16 3 7 11 15 2 6 10 14 1 5 9 13`.

Note that this is a typical example in which `twod_to_oned`, as mentioned above, must be used for the sake of good design and readability.

game_over

```
bool game_over(const std::vector<int>& v)
```

This function should return true if it's true that the content of `v` represents a game over situation, and false otherwise.

(An input file may well contain a grid already in a game over situation, in which case the program should just print game over as in the screenshot above.)

print_grid

```
void print_grid(const std::vector<int>& v)
```

This function should print the content of `v` as an `nxn` grid (as shown in the screenshots).

Functions other than the `main` and `print_grid` should not contain user or file input or output (`cin`, `cout`, `ifstream` etc) in their implementation.

Guidelines

All the variables should be declared in the scope of a function (either the main or some other one). In other words, global variables are not allowed.

All the loops should be controlled/terminated either by the loop condition or by return. Statements such as break, continue, goto are not allowed anywhere in the program.

Do not use the switch statement.

Functions should not `return` vectors provided as output. In other words, vectors must be provided in output using output parameters (i.e. passing by reference).

Only headers from the standard library are allowed (if you are not sure whether a header is part of the standard library or not, try compiling on the Linux install on the lab computers, if it works it's part of the standard library), however the use of the header `<algorithm>` is not allowed.

Do not use the `using namespace` directive.

Our reference standard in terms of compiler and runtime environment is the Linux install on the lab computers. In other words, if your program doesn't compile or work correctly on Linux on the lab computers, for our purposes it means it doesn't compile or work correctly and it will be assessed as such (no matter the behaviour shown on your own computer or other computers). It is your responsibility to ensure compliance with this standard and you are strongly advised to compile and test your program on Linux on the lab computers before the submission.

It's ok to use C++11 features; if you do, you will need to compile your program on the lab computers including the `-std=c++11` flag.

Q&A:

How are the submissions going to be assessed? Is it enough if the program works?

Correctness is of course important. Automated tests will be run for each of the functions to check correctness (including the tests provided as example in the comments of the source file). Each function will need to pass all the tests in order to be considered correct.

The submissions will also be checked to make sure that all the instructions and guidelines are followed.

For instance if a program uses global variables, or if there are changes in the provided function interfaces, or if the functions don't actually do what is requested in the descriptions provided, etc. this is going to result in a (potentially much) lower mark (even if the program “works”).

It is important in general that the code is clear and readable. Indentation needs to be followed very carefully. Variables should have useful names.

To some extent I believe that you have already done the experience of writing a piece of code, getting to a version which works and then doing some refinement or tidying up. As you can imagine you are expected to perform this process also for your submission. If you are not sure about how to do this, ask me to go through the process with you for one of your exercises (not the assignment).

What about efficiency?

The algorithms that should be implemented in this program are quite straightforward and don't really offer much for real efficiency differences. Please note that declaring one less variable and other things like that are not more efficient for our purposes and fall in the category of the “clever tricks” (see question below).

Of course it's a good touch to avoid wastefulness as in the examples in the notes. Think of the prime number example, in which the loop terminates as soon as the first factor is found (instead of going on to try other potential factors) or as soon as the square root of the number is reached (instead of trying all the numbers up to the input).

(For EIE1 students:) In Introduction to Computer Architecture we often need to minimize the number of lines of code. Does the same apply here?

No and in fact aggressively shorter code (or code with “clever tricks”) might be penalised as not very readable and error prone.

What about comments?

We are not really going to consider comments in this submission. Feel free to add some if you think they would be useful for yourself to understand your own code or to explain some aspects of your program. But it is more important that your code is clear and readable (minimising the need for comments).

Do not add too many comments: that will only make us waste time by having to remove them before we check your submission.

When will the grades be available?

By Monday 4th March grades and feedback for the submissions not presenting particular issues will be available on Blackboard (I will send a notification email).

Can you or a TA help me with the assignment?

No because it's assessed. We can however help you with other exercises on the topics covered by the assignment.

I have another question.

Sure, you can send me an email but:

1) First please reread this page and double check that the answer isn't already here or elsewhere on this website or Blackboard.

2) Make sure you send your email **by Thursday 7th February** to allow time for a reply.

---

introcomp/a2sp19.txt · Last modified: 2019 by mc