# Introduction

Your task is to write an ANSI-C program (called `match`) which allows the user to play a game (described later). This will require I/O from both the user and from files. Your assignment submission must comply with the C style guide (v1.7) available on the course website.

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

In this course we will use the subversion (svn) system to deal with assignment submissions. Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

# The Game

`match` will display a grid of cells, each containing a symbol. The user will nominate a cell (by entering its coordinates). If the symbol in the cell matches the symbol in any of the cells directly above, below, left or right, then the selected cell and any of its matching neighbours will be removed. This effect is also applied to neighbours of matching neighbours (and so on). As an example, suppose the grid contents (surrounded by a border) were:

```
+-----+
|ZAAAA|
|ZBBCC|
|ZWaaX|
|XXXXX|
|Zcdef|
+-----+
```

and the cell 31 was chosen (that is, Row 3 and Column 1, with the top left being 0 0). In this case, all of Row 3 would be removed as well as the last value on Row 2.

```
+-----+
|ZAAAA|
|ZBBCC|
|ZWaa.|
|.....|
|Zcdef|
+-----+
```

If a move leaves a gap in the middle of a column, the symbols above it will be moved down to fill the the gap.

```
+-----+
|.....|
|ZAAA.|
|ZBBCA|
|ZWaaC|
|Zcdef|
+-----+
```

If 4 1 was chosen, nothing would happen because the 'c' in that cell is not touching any other 'c's. If a removal leaves one or more whole columns empty, any columns to the right of the empty ones, will be moved over to fill the gap. For example, if 1 0 was chosen in the above, the result would be:

```
+-----+
|.....|
|AAA..|
|BBCA.|
|WaaC.|
|cdef.|
+-----+
```

If the grid looked like:

```
+-----+
|AAAAA|
|ZABAC|
|ZAaAX|
|XAXAX|
|ZAdAf|
+-----+
```

and 0 0 was chosen, the result would be:

```
+-----+
|.....|
|ZBC..|
|ZaX..|
|XXX..|
|Zdf..|
+-----+
```

The game ends when either all the symbols are removed, or there are no more legal moves.

## Invocation

When run with no arguments, `match` should print usage instructions to stderr:

```
Usage: match height width filename
```

and exit (see the error table).

`height` and `width` must be positive integers greater than 1 and less than 1,000. `filename` must be a path to a readable file containing a valid grid (see later) with dimensions matching `height` and `width`. See the error table for what to do if these constraints are not met.

When run with valid arguments, `match` should read in the map file. It should then display the grid. If there are no legal moves, it should exit with the appropriate message; otherwise it should display the prompt:

```
>
```

(note, a space follows the prompt). The user should then enter either:

- the row and column they wish to remove (space separated). eg 3 2
- `w` followed immediately by a filename(or path). eg `whello` This will result in the current grid (including the border) being written to a file called "hello".

If there is no more input from the user, a message should be displayed (see error table) and the program should exit normally. If the user enters anything else, the prompt should be displayed again.

After the user has entered a legal (non-w) move, the grid should be updated and displayed again. If the game is over, display the appropriate message and exit. If not, display the prompt again and wait for another move.

## Input file format

A valid grid file will consist of rows of symbols (and blanks) terminated by newlines (`\n`). You may use whichever characters you want for grid symbols except the following:

- `\0` — the null character. This should trigger an error in your program.
- `\n` — newlines will always be interpretted as the end of a line.
- `.` — a dot indicates a blank. You can include these in your input but the gaps will not be filled until a valid move has been completed.

## Errors and messages

When one of the conditions in the following table happens, the program should print the error message and exit with the specified status. All error messages in this program should be sent to standard error. Error conditions should be tested in the order given in the table. All messages are followed by a newline.

| **Condition** | **Exit Status** | **Message** |
|---|---|---|
| Program started with incorrect number of arguments | 1 | Usage: match height width filename |
| Invalid grid dimensions | 2 | Invalid grid dimensions |
| Cannot open grid file | 3 | Invalid grid file |
| Error reading grid. Eg: bad chars in input, not enough lines, short lines | 4 | Error reading grid contents |

For a normal exit, the status will be 0 and no special message is printed.

There are a number of conditions which should cause messages to be displayed but which should not immediately terminate the program. These messages should also go to standard error.

| **Condition** | **Action** | **Message** |
|---|---|---|
| Error opening file for saving grid | Prompt again | Can not open file for write |
| Save of grid successful | Prompt again | Save complete |
| End of input while waiting for user input | exit program normally | End of user input |

If after a move, all the symbols have been removed, the program should display "Complete" to stdout and exit normally. If the game has reached a configuration where completion is impossible and there are no more legal moves, the program will display "No moves left" to stdout and exit normally.

### Compilation

Your code must compile with command:

```
gcc -Wall -ansi -pedantic ass1.c -o match
```

You must not use flags or pragmas to try to disable or hide warnings.

If any errors result from the compile command (ie the executable cannot be created), then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted (and if compilation fails, you will receive 0 marks for functionality). If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs or use non-standard headers/libraries. It must consist of a single, properly commented and indented C file called `ass1.c`.

## Late Penalties

Late penalties will apply as outlined in the course profile.

## Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. Clarifications may be issued via the the course newsgroup. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

## Test Data

Test data and scripts for this assignment will be made available. The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments. Testing that your assignment complies with this specification is still your responsibility.

## Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment the markers will check out `https://source.eait.uq.edu.au/svn/csse2310-s???????/trunk/ass1`. Code

checked in to any other part of your repository will not be marked. Note that no submissions can be made more than 96 hours past the deadline under any circumstances.

Test scripts will be provided to test the code on the trunk. Students are strongly advised to make use of this facility after committing.

# Marks

Marks will be awarded for both functionality and style.

## Functionality (42 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely. Your programs should not take a long time to complete.

Please note that some features referred to in the following scheme may be tested in other parts of the scheme. For example, if you can not display the inital grid, you will fail a lot of tests. Not just the one which refers to "initial grid". Students are advised to pay close attention to their handing of end of input situations.

- Command args — correct response to
  - incorrect number of args (2 mark)
  - invalid dimensions (2 mark)
  - invalid grid filename (2 mark)
- Correctly display initial grid and prompt (4 marks)
- Reject illegal moves on the initial grid (2 marks)
- Correctly process a single move:
  - Single removal from top line (2 marks)
  - Single removal of whole rightmost column (2 marks)
  - Single removal covering multiple rows/columns (4 marks)
- Detect no more legal moves (4 marks)
- Correctly save grid (4 marks)
- Play complete games with multiple removals (14 marks)

## Style (8 marks)

If $g$ is the number of style guide violations and $w$ is the number of compilation warnings, your style mark will be the minimum of your functionality mark and:

$$8 \times 0.9^{g+w}$$

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of version 1.7 of the C Programming Style Guide. A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final — it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark — this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

## Notes and tips

1. A well written program should gracefully handle any input it is given. We should not be able to cause your program to `crash`.
2. Remember that the functionality of your program will be marked by comparing its output for certain inputs against the expected output. Your program's output must match exactly.
3. You assume that any characters on a line after the $79^{\text{th}}$ character are not relevant and ignore them.
4. Be sure to handle unexpected end of file properly. A number of marking tests will rely on this working.
5. Debug using small grids if possible.
6. Do not hardcode grid dimensions.
7. This assignment only deals with single byte characters (ie. characters which are stored in the `char` data type).
8. The grid files loaded by the program do not have a border. Whenever the grid is displayed or written to a file, the border should be included.

## Updates

1.3

1. Note: Legal (non-w) moves must change the board.

1.2

1. Clarified what happens when the user loads a map with no moves.

1.1.1

1. If the user input is `w` (on its own) then treat it as an empty filename and display the relevant error message.
2. Corrected mark totals.
3. Clarified language describing input file format.
4. Widened table.