Takuzu

Assignment 2: Grid Parser

The objective of this assignment is to develop a grid parser that reads grids from files and writes grids to files. Grids are square arrays of size 4, 8, 16, 32 or 64 elements per row/column.

For this assignment you will work by groups on two again.

```
0 _ _ _ _ _         0 _ _ _     0 0 _ 1 _ _ 1 _
0 1 1 _ _ _         _ _ 1 1     1 _ 1 _ _ 0 _ _
_ _ _ _ _ _         1 _ _ _     _ _ _ 1 _ 1 _ 0
_ _ 0 _ _ _         _ _ _ _     _ _ _ _ _ _ _ _
1 _ _ _ _ _                     _ 1 _ 1 0 _ 1 0
_ 0 _ _ _ _                     0 0 1 0 _ _ _ _
                                _ _ _ _ 1 _ _ _
                                _ _ _ _ _ 1 _ 0
```

The file format is as follows (see examples above):

- Comment lines: all characters in a line that follow a '#' are ignored until '\n' or EOF
- A row has an even number of significant characters (from 4 to 64) and end with an '\n' or EOF
- The number of rows and the number of characters per row are always the same (square grids)
- Characters '0', '1' and '_' denote respectively a cell filled with a 0, a cell filled-in with a 1 and an empty cell
- Supported separators between significant characters are tabs ('\t') and spaces (' ')

If one of these rules are not satisfied the program must write a warning explaining which rule has been broken, at which line, and return EXIT_FAILURE.

Note: the functions written in this assignment will check that grids are syntactically correct only, regardless of the fact that they have one unique solution, multiple solutions or no solution at all.

1. Utility functions to manage grids

The data structure used to handle grids will be the following:

```c
typedef struct {
  int size;     // Number of elements in a row
  char *grid;   // Pointer to the grid
} t_grid;
```

Note: the data structure assumes the grid will be allocated as an array of cells (of `size*size` bytes). It is also allowed that you declare the grid as `char **grid`, with grid an array of pointers on rows, each row of `size` bytes.

1. Write the following functions
   a. Grid allocation (and initialization to empty cells):
      `void grid_allocate(t_grid *g, int size);`
   b. Grid deallocation:
      `void grid_free (const t_grid *g);`
      In case memory allocation fails, the program will stop and return EXIT_FAILURE.
      Suggested functions: `malloc(), calloc(), free()`
2. Write a function that prints the content of a grid in text format (see start of this sheet) in an opened file:
   `void grid_print(const t_grid *g, FILE *fd);`
   Suggested functions: `fputs(), fprintf()`
   In order to facilitate automated testing in the future, the output will not contain any comment line, will have exactly one line per row and will use only spaces as separators.
3. Write a function that checks if a character is in the list of significant characters: ‘0’, ‘1’, ‘_’. The function should return `true` if the character is in the list, `false` otherwise. The function signature is as follows:
   `bool check_char(const char c);`

2. Grid parser

The parser must be able to read a grid of size 4 to 64 with only one scan of the file. The scanner has to be as robust as possible when a user provides an incorrect grid file. A meaningful error message must be issued in these situations, all dynamically allocated memory has to be freed and the program has to stop and return EXIT_FAILURE.

1. Write the parser such that it starts with the first row and stores all the significant characters in a local array: `char line[MAX_GRID_SIZE];` When a newline character is read, the grid size is revealed and function `grid_allocate()` can be called. Then, the grid contents can be filled-in. The function prototype will be:
   `file_parser(t_grid *grid, char *filename);`
   Suggested functions: `fgetc();`
   Display the parsed grid using function `grid_print()` to help testing your code.
2. Add to your parser the possibility to ignore comment lines (starting by character ‘#’)
3. Modify your code to detect the following problems:
   a. Characters in the grid are not among authorized ones
   b. A row does not have the correct number of items
   c. A grid does not have the right number of columns

Error messages must be as follows:

```
sh> ./takuzu grid01.txt
takuzu: error: 'grid01.txt': wrong character 'X' at line 6!
sh> ./takuzu grid02.txt
takuzu: error: 'grid02.txt': line 42 is malformed (wrong number of columns)
sh> ./takuzu grid03.txt
takuzu: error: 'grid03.txt': grid has 4 missing lines!
```

For every error, write the error message on `stderr`, mention the line number in the input file when applicable, free all allocated memory and quit the program with the EXIT_FAILURE error code.