1102 – Compilation et bibliothèques

```
 **/

int required_nb_monsters(cgame game, content monster);



/**
 * @brief Test if the game is over (that is the grid is filled according to the require
ments).
 * @return true if all the constraints are satisfied
 **/

bool is_game_over (cgame g);



/**
 * @brief Restart a game by cleaning monsters from the board.
 **/

void restart_game(game g);



/**
 * @brief adds a monster on the game board.
 * Can also be used to remove any monster by adding EMPTY.
 * This function does not have effect on mirrors so it can be used safely in the course
 of the game.
 * @param game the game board where to add the monster
 * @param monster the type of monster to add
 * @param col the column where to insert the monster
 * @param line and the line where to insert the monster
 * @return false if the monster was not placed since the square was occupied by a mirro
r.
 **/

bool add_monster(game game, content monster, int col, int line);

/**
 * @brief says how many monsters can be seen on the current game board
 * @param game the game board to look at
 * @param side the side of the board we consider (N, S, E, or W)
 * @param pos the coordinate on that side (from S to N or from W to E)
 * @return the number of monsters that can be seen through all the mirrors from a given
 side at position x
 **/

int current_nb_seen(cgame game, direction side, int pos);
```

```
/**
 * @brief check whether all squares are occupied
 * @param game the game board
 **/

bool is_filled(cgame game)
{
  return (game->cur_nb_item[EMPTY] == 0);
}


bool is_game_over(cgame g){
  bool res = is_filled(g);
  for(int i=0; i < g->width; i++){
     res = res && (g->labels[N][i] == current_nb_seen(g,N,i));
     res = res && (g->labels[S][i] == current_nb_seen(g,S,i));
  }
  for(int i=0; i<g->height; i++){
     res = res && (g->labels[E][i] == current_nb_seen(g,E,i));
     res = res && (g->labels[W][i] == current_nb_seen(g,W,i));
  }
  res = res && (g->nb_ghosts == g->cur_nb_item[GHOST]);
  res = res && (g->nb_vampires == g->cur_nb_item[VAMPIRE]);
  res = res && (g->nb_zombies == g->cur_nb_item[ZOMBIE]);
  res = res && (g->nb_spirits ==g->cur_nb_item[SPIRIT]);

  return res;
}

void restart_game(game g){
  restart_board(g->width, g->height, g->board);
  g->cur_nb_item[EMPTY] += g->cur_nb_item[GHOST];
  g->cur_nb_item[GHOST] = 0;
  g->cur_nb_item[EMPTY] += g->cur_nb_item[VAMPIRE];
  g->cur_nb_item[VAMPIRE] = 0;
  g->cur_nb_item[EMPTY] += g->cur_nb_item[ZOMBIE];
  g->cur_nb_item[ZOMBIE] = 0;
  g->cur_nb_item[EMPTY] += g->cur_nb_item[SPIRIT];
  g->cur_nb_item[SPIRIT] = 0;
  update_current_nb_seen(g);
}

bool add_monster(game g, content monster, int col, int line){
  if(
     (line>=0 && line<g->height && col>=0 && col<g->width)
     &&
     ((monster>=VAMPIRE && monster<= SPIRIT) || monster == EMPTY)
     )
  {
    if(g->board[line][col]>=MIRROR && g->board[line][col]<=HMIRROR){
```

Le compilateur

gcc

- Utilisation la plus simple : `gcc nom_du_programme.c`
  - Compile et crée le lien d'exécution.
  - Nécessite une fonction `main` dans le code
  - Génére un programme *a.out*
- L'option `-o` (pour *output*)
  - Permet de définir le fichier de sortie
  - par exemple, `gcc programme.c -o programme`
- L'option `-Wall` (pour *Warning all*)
  - Affiche un tas de messages pour indiquer les éléments suspects du code.
  - Souvent très pertinent.

$\longrightarrow$ `gcc -Wall programme.c -o programme`

Les bibliothèques

Les bibliothèques standard

Il est possible d'utiliser des fonctions écrites ailleurs dans un programme avec
`#include <bibliotheque.h>`
dans le code source.

<stdio.h>

La bibliothèque d'entrée/sortie (*input/output* = *i/o*) standard.

- Permet d'utiliser la fonction `printf` pour afficher.
- Permet d'utiliser la fonction `scanf` pour lire des entrées au clavier.

⟶ `#include <stdio.h>`

⟶ Rien à préciser pour la compilation.

<stdio.h> printf

`printf`, qui signifie *print formatted* :

- permet d'afficher une chaîne de caractère formatée.
- dans l'argument,forcément une chaîne de caractère :

  ⟶ `printf("Bonjour à tous");`
- le retour à la ligne s'écrit avec le caractère `'\n'`

  ⟶ `printf("Bonjour\nComment allez vous?\n");`
- certains codes commençant par % sont interprétés :
  - `%d` int, `%f` double, `%c` char, `%s` chaîne de caractère...
  - ils sont substitués par les arguments suivants de la fonction

  ⟶ `printf("%d + %f vaut %f \n",11,4.64,15.64);`

<stdio.h> scanf

`scanf`, qui signifie *scan formatted* :

- permet de lire dans l'entrée standard une chaîne formatée.
  ⟶ la chaîne en argument est la chaîne devant être lue !
- renvoie le nombre de champs correctement lus (0 si échec)
- enregistre les conversions effectuées aux *adresses* indiquées ensuite.

D'un usage pas si facile. . .

<stdio.h> scanf

Lire des nombres

Lire des nombres est relativement facile.

```
int variable, retour;
retour = scanf("%d", &variable);
```

- Si l'utilisateur rentre un nombre,
  - `variable` contiendra le nombre.
  - `retour` contiendra 1, pas très utile.
- Si l'utilisateur rentre autre chose,
  - `variable` ne sera pas initialisée (contient n'importe quoi)
  - `retour` contiendra 0, indiquant l'échec de la lecture.

<stdio.h> scanf

Lire des caractères

Lire des caractères est beaucoup plus difficile.

```
int retour;
char caractere;
retour = scanf("%c", &caractere);
```

- Quand l'utilisateur rentre un caractère, puis valide la ligne en tapant `entrée`
  - `caractere` contiendra le caractère.
  - `retour` contiendra 1, pas très utile.
- Mais si vous avez déjà lu autre chose avant...
  - Il est fort probable qu'il reste des choses dans le tampon de lecture (par exemple un `\n`)
  - Vous ne serez alors pas consulté, et la valeur sera immédiatement attribuée.

$\longrightarrow$ plein de pièges avec `scanf`

<math.h>

La bibliothèque `<math.h>` contient des outils mathématiques

- elle permet par exemple de calculer une racine carrée (comme `sqrt`, `exp`, `log`, ...)
- elle nécessite d'être signalée à la compilation (avec l'option `-lm` (en fin de ligne par exemple))

<stdlib.h> et <time.h>

Générer des nombres aléatoires

Il est possible de générer des nombres aléatoires en `C` avec les deux bibliothèques `<stdlib.h>` et `<time.h>`.

- `<stdlib.h>` donne accès à la fonction `rand()` qui renvoie des entiers aléatoires entre `0` et `RAND_MAX`
- le point de départ de cette fonction s'initialize avec `srand(int graine)`.
- pour que le point de départ change à chaque fois, utiliser l'horloge (dans <time.h>), une fois dans le `main` : `srand(time(NULL));`
- puis utiliser `rand()` autant que nécessaire.