

# 1102 – Introduction à l'algorithmique et à la programmation

```

**/
int required_nb_monsters(game game, content monster);

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

bool is_game_over (cgame g);

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

void restart_game(cgame 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 mirror
 **/

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 to consider (N, S, E, or W)
 * @param pos the coordinate on that side (from 0 to N or from N to 0)
 * @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
 * @return 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[i][0] == current_nb_seen(g,E,i));
        res = res && (g->labels[i][W-1] == current_nb_seen(g,W,i));
    }
    return res && (g->nb_monsters == g->nb_monsters[EMPTY]);
}

bool is_mirror(cgame game, int col, int line){
    return (game->labels[col][line] == 'M' || game->labels[line][col] == 'M');
}

res = res && (g->nb_spirits == g->cur_nb_item[SPIRIT]);

return res;

void restart_game(cgame g){
    restart_board(g->width, g->height, g->board);
    g->cur_nb_item[EMPTY] = g->cur_nb_item[EMPTY];
    g->cur_nb_item[EMPTY] = 0;
    g->cur_nb_item[EMPTY] = g->cur_nb_item[WAMPFIRE];
    g->cur_nb_item[WAMPFIRE] = 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 == WAMPFIRE && monster == SPIRIT) || monster == EMPTY)
        )
    )
        if(!g->board[line][col] == MIRROR && g->board[line][col] != MIRROR){

```

Cours : Paul Dorbec

TD-TP : + Pierrick Meignen, Pascal Morin, Albrecht

Zimmermann.

## Objectifs du Module

Dans le programme national :

*Savoir décomposer un problème en sous-problèmes plus simples et définir des types simples pour structurer les données d'un problème en étant attentif aux critères de qualité de programmation*

1. **Algorithmique** : décrire la méthode employée avec des opérations élémentaires.
2. **Programmation** : transcrire l'algorithme dans un langage contraint, interprétable par une machine (ici le **C**)
3. **Qualité** : garantir un programme sans bug, et facile à maintenir et modifier.

## Un algorithme

- ▶ description détaillée de la méthode pour atteindre un objectif.  
Ex : recette de cuisine, notice de montage d'un meuble, ...
- ▶ naturel mais pas toujours facile à expliciter  
Ex : conduire une voiture, trier sa main au tarot, reconnaître un chat, ...
- ▶ niveau de précision à définir, dépend un peu du contexte (de ce qu'on sait déjà faire)  
Ex : pour faire des lasagnes, faites une béchamelle...
- ▶ prend d'autant plus de sens quand la méthode n'est pas unique !

## Exemple d'algorithme

L'entier  $n$  est-il premier ?

- 1 Si  $n$  est premier :
- 2     renvoyer vrai
- 3 sinon :
- 4     renvoyer faux

- 1 pour chaque  $i$  entre 2 et  $n$  :
- 2     si  $i$  **divise**  $n$ , renvoyer faux
- 3     sinon, continuer
- 4 si aucun ne divise  $n$ , renvoyer vrai

## Exemple d'algorithme

L'entier  $n$  est-il premier ?

- 1 Si  $n$  est premier :
- 2     renvoyer vrai
- 3 sinon :
- 4     renvoyer faux

- 1 pour chaque  $i$  entre 2 et  $n$  :
- 2     pour chaque  $j$  entre 2 et  $n$  :
- 3         si  $i * j = n$ , renvoyer faux
- 4 si aucun ne divise  $n$ , renvoyer vrai

- 1 pour chaque  $i$  entre 2 et  $n$  :
- 2     si  $i$  **divise**  $n$ , renvoyer faux
- 3     sinon, continuer
- 4 si aucun ne divise  $n$ , renvoyer vrai

- 1 pour chaque  $i$  entre 2 et  $n$  :
- 2     mettre  $k$  à 0,
- 3     tant que  $k < n$  :
- 4          $k$  devient  $k + i$
- 5         si  $k = n$ , renvoyer faux
- 6 à l'issue de tout, renvoyer vrai

## Exemple d'algorithme

L'entier  $n$  est-il premier ?

```

1 Si  $n$  est premier :
2   renvoyer vrai
3 sinon :
4   renvoyer faux

```

```

1 pour chaque  $i$  entre 2 et  $n$  :
2   pour chaque  $j$  entre 2 et  $n$  :
3     si  $i * j = n$ , renvoyer faux
4 si aucun ne divise  $n$ , renvoyer vrai

```

```

1 pour chaque  $i$  entre 2 et  $n$  :
2   si  $i$  divise  $n$ , renvoyer faux
3   sinon, continuer
4 si aucun ne divise  $n$ , renvoyer vrai

```

```

1 pour chaque  $i$  entre 2 et  $n$  :
2   mettre  $k$  à 0,
3   tant que  $k < n$  :
4      $k$  devient  $k + i$ 
5     si  $k = n$ , renvoyer faux
6 à l'issue de tout, renvoyer vrai

```

pour choisir, nécessaire de savoir ce qu'on sait déjà faire !

## Exemple de programme C

L'entier  $n$  est-il premier ?

```
int est_premier(int n){
    for(int i=2; i<n; i++){
        if(n%i == 0){
            return 0;
        }
    }
    return 1;
}
```

- ▶ langage très contraint (mots clés limités)
- ▶ très précis
- ▶ lié par des conventions  
( 0 = faux, 1 = vrai )
- ▶ possibilité d'utiliser d'autres bouts de programmes.

# La compilation

Que devient ce bout de programme C ?

<pre> int est_premier(int n){   for(int i=2; i&lt;n; i++){     if(n%i == 0){       return 0;     }   }   return 1; } </pre>	<pre> .file "toto.c" .text .globl est_premier .type est_premier, @function est_premier: .LFB0: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 movl %edi, -20(%rbp) movl \$2, -4(%rbp) jmp .L2 .L5: movl -20(%rbp), %eax cld idivl -4(%rbp) movl %edx, %eax testl %eax, %eax jne .L3 movl \$0, %eax jmp .L4 </pre>	<pre> @UH&lt;89&gt;â&lt;89&gt;}iÇEü^B^@^@^@ë^X&lt;8b&gt;Ei&lt;99&gt;+}ü &lt;89&gt;ð&lt;85&gt;Äu^G ^@^@^@ë^0&lt;83&gt;Eü^A&lt;8b&gt;Eü;Ei  à,^A^@^@^@ Ä^GCC: (Ubuntu 7.5.0-3ubuntu1 -18.04) 7.5.0^@^@^@^@^@^@^@T^@^@^@^@^@ ^@^@AzR^@^@Ax^P^A[^L^G^H&lt;90&gt;^A^@^@^@^@^@ ^@^@^@^@^@^@7^@^@^@^@A^N^P&lt;86&gt;^BC^M^Fr ^L^G^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@A^@^@^@D^ñy^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@C^@A^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@C^@B ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^C^@C^@C^@E^@E^@E^@E^@E^@E^@E^@E^@E^@ ^@^@^@C^@E^@E^@E^@E^@E^@E^@E^@E^@E^@ ^@^@^@^@^@^@C^@F^@F^@F^@F^@F^@F^@F^@F^@ ^@^@^@^@^@^@^@^@^@C^@D^@D^@D^@D^@D^@D^@ ^@^@^@^@^@^@^@^@^@H^@E^@R^@A^@E^@E^@E^@ ^@^@^@7^@^@^@^@^@^@^@toto.c^est premier ^@^@^@^@^@^@ ^@^@^@^@^@^@B^@E^@E^@B^@E^@ ^@^@^@^@^@^@^@.symtab^@.strtab^@.shstr ab^@.text^@.data^@.bss^@.comment^@.note.GN U-stack^@.rela.eh_frame^@^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ </pre>
toto.c (+) 9,0-1 Tout	toto.s 1,2-9 Haut	toto.o 2,610-1006 Bas

code source

assembleur

code binaire

[petite démo]



# Compilation, conséquences

## Conséquences

Le code obtenu n'est pas une traduction directe du code écrit.

Tout ce qui est :

- ▶ nom de variable
- ▶ indentation
- ▶ saut de ligne
- ▶ et autre mise en forme

c'est pour vous, pour permettre de travailler sur votre code.

# Fin de l'intro

## Au programme

- ▶ la syntaxe du **C**
- ▶ le processus de compilation (beaucoup en TP) et les bibliothèques
- ▶ les attentes de qualité
- ▶ et tout un tas de points de vigilance : portée des variables, passage par valeur, tableaux et structures...