

TP04

September 26, 2021

1 Conversions binaires

On considère l'écriture binaire $n = \sum_{k=0}^{+\infty} b_k 2^k$ où les (b_k) sont presque tous nuls d'un entier naturel. En pratique, on ne considère que des nombres dont la taille est majorée donc de la forme $\sum_{k=0}^{N-1} b_k 2^k$. N est le nombre de bits sur lequel on a écrit l'entier n . Il peut être nul, c'est-à-dire qu'on impose pas que $b_{N-1} \neq 0$, par contre, on a forcément $n < 2^N$.

Les différents types de valeurs entières en C sont de la forme `t` ou `unsigned t` où :

- `unsigned` signifie qu'on ne peut stocker que des valeurs positives, si on l'omet on peut avoir des valeurs négatives sachant qu'il faut un bit pour coder l'information du signe, un même type peut prendre des valeurs deux fois plus grandes en `unsigned`
- `t` peut valoir (types principaux) **ATTENTION : c'est dépendant du compilateur, on donne ici les valeurs usuelles pour un compilateur moderne et un processeur 64bits ! :**
 - `char` : pour des valeurs sur $N = 8$ bits, donc 1 octet, donc de 0 à 255
 - `short` : pour des valeurs sur $N = 16$ bits, 2 octets, donc de 0 à $2^{16} - 1$
 - `int` : pour des valeurs sur $N = 32$ bits, donc 4 octets, donc de 0 à $2^{32} - 1$
 - `long` : pour des valeurs sur $N = 64$ bits, donc 8 octets, donc de 0 à $2^{64} - 1$

Dans le doute, on peut connaître la taille en nombre d'octets en utilisant `sizeof` :

```
[1]: sizeof(long)
```

```
[1]: 8
```

Il est aussi possible d'utiliser des types explicites comme `int32_t` ou `int64_t`.

```
[2]: void vers_binaire(unsigned int x, unsigned char b[])
{
    // On considère ici que b est un tableau de 32 unsigned char
    // cette fonction renvoie dans b la représentation
    // en binaire de x de sorte que b[i] contienne b_i de l'expression au dessus
    // (comme toujours voir tests en cas de doute)

    // BEGIN SOLUTION
    for (int i = 0; i < 32; i++)
    {
        b[i] = x % 2;
        x = x / 2;
    }
}
```

```

    }
    // END SOLUTION
}

```

```

[3]: #include <stdio.h>
unsigned char b[32];
unsigned char out[] = { 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
    ↪0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
vers_binaire(123457, b);
for(int i =0; i < 32; i++)
{
    if (b[i] != out[i])
    {
        printf("[Test 1] Erreur en %d : %d != %d\n", i, b[i], out[i]);
        throw(0);
    }
}
vers_binaire(4294967295, b);
for(int i =0; i < 32; i++)
{
    if (b[i] != 1)
    {
        printf("[Test 2] Erreur en %d : %d != 1\n", i, b[i]);
        throw(0);
    }
}
"Youpi"

```

[3]: "Youpi"

```

[4]: unsigned int depuis_binaire(unsigned char b[])
{
    // renvoie l'entier représenté par le tableau de 32 bits b
    // on fera en sorte que depuis_binaire et vers_binaire soit réciproque
    ↪l'une de l'autre

    // BEGIN SOLUTION
    unsigned int x = 0;
    unsigned int p = 1;
    for (int i = 0; i<32; i++)
    {
        if (b[i] == 1)
        {
            x = x + p;
        }
        p = 2*p;
    }
}

```

```

    return x;
    // END SOLUTION
}

```

```

[5]: #include <stdio.h>
unsigned char b[] = { 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0,
    ↪ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
if (depuis_binaire(b) != 123457) throw(0);
unsigned char c[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    ↪ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
if (depuis_binaire(c) != 4294967295) throw(0);
"Super"

```

[5]: "Super"

2 Retour sur les chaînes

On rappelle qu'une chaîne de caractère en C est juste un tableau de `char`. Les `char` sont des valeurs entières sur un octet (8 bits). En pratique, on ne s'intéressera pas à la valeur entière mais uniquement à leur représentation sous forme de caractère, donc on écrira 'a' et pas 97 bien que ce soit la même valeur.

```

[6]: 'a' == 97

```

[6]: true

On ne considère ici que les caractères ASCII dont la valeur est comprise entre 0 et 127. **ATTENTION** comme les `char` sont limités, on ne peut pas écrire `for(char i=0; i<128; i++)` car la comparaison 128 est toujours vrai.

Les chaînes finissent toutes par '\0' et donc on peut passer des chaînes en argument sans se soucier de donner leur taille grâce à ce caractère spécial. On appelle ça une **sentinelle**.

Pour faire une boucle dans un tableau de taille `n`, on écrit `for(i = 0; i < n; i++)` mais ici, comme on ne connaît pas `n` il faut utiliser la sentinelle, par exemple en écrivant `for(i = 0; s[i] != '\0'; i++)`.

```

[7]: int longueur_chaine(char s[])
{
    // Renvoie la longueur de la chaîne s

    // BEGIN SOLUTION
    int i;
    for (i = 0; s[i] != '\0'; i++)
    {
    }
    return i;
    // END SOLUTION
}

```

```
}
```

```
[8]: char s1[] = "test";  
char s2[] = "";  
if(longueur_chaine(s1) != 4) throw(0);  
if(longueur_chaine(s2) != 0) throw(0);  
"Bravo"
```

```
[8]: "Bravo"
```

Cette fonction, comme d'autres sur les strings, est définie directement dans la bibliothèque standard du C. C'est `strlen`. On pourra donc l'utiliser dans la suite pour pouvoir faire des boucles avec la taille comme dans un tableau usuel.

Il est probable que pour écrire la fonction suivante vous souhaitez pouvoir sortir prématurément d'une boucle comme en Python. C'est également possible en C, on peut utiliser :

- `break`; pour sortir de la boucle
- `continue`; pour finir l'itération et passer à l'itération suivante

```
[9]: int recherche_motif(char s[], char m[])  
{  
    // Renvoie l'indice de début de la première occurrence du motif m  
    // dans s s'il est présent ou -1 sinon (allez voir les tests en dessous !)  
  
    // BEGIN SOLUTION  
    int n = strlen(s);  
    int p = strlen(m);  
    for (int i = 0; i <= n-p; i++)  
    {  
        int j;  
        for (j = 0; j < p; j++)  
        {  
            if (s[i+j] != m[j])  
            {  
                break;  
            }  
        }  
        if (j == p)  
        {  
            return i;  
        }  
    }  
  
    return -1;  
    // END SOLUTION  
}
```

```
[10]: char s[] = "tetututo";
char m1[] = "tu";
char m2[] = "to";
char m3[] = "ti";

if (recherche_motif(s, m1) != 2) throw(0);
if (recherche_motif(s, m2) != 6) throw(0);
if (recherche_motif(s, m3) != -1) throw(0);
"Génial"
```

[10]: "Génial"

```
[11]: void occurrences(char s[], int occ[])
{
    // Renvoie dans le tableau occ contenant 128 valeurs NON INITIALISEES
    // les occurrences de chaque caractère dans la chaîne s

    // BEGIN SOLUTION
    for (int i=0; i < 128; i++)
    {
        occ[i] = 0;
    }
    for (int i=0; s[i] != '\0'; i++)
    {
        occ[s[i]]++;
    }
    // END SOLUTION
}
```

```
[12]: char s[] = "test";
int occ[128];
occurrences(s, occ);
for(int i=0; i<128;i++)
{
    if (i == 't' && occ[i] != 2) throw(0);
    else if (i == 'e' && occ[i] != 1) throw(0);
    else if (i == 's' && occ[i] != 1) throw(0);
    else if (i != 't' && i != 'e' && i != 's' && occ[i] != 0) throw(0);
}
"Well done!"
```

[12]: "Well done!"

3 Automates de Wolfram

3.1 Présentation et premier automate

On va s'intéresser ici à des automates cellulaires unidimensionnels. On considère ainsi une ligne de cellules infinies. Chaque cellule a exactement deux cellules voisines, une à sa droite et une à sa gauche. Les cellules contiennent un état. Pour simplifier ici, on ne va considérer que deux états 1 et 0, noir ou blanc ou encore '#' et ' ' dans la suite.

Afin de représenter informatiquement de telles lignes infinies, on va considérer un nombre fini de cellules en supposant que toutes les autres sont dans l'état 0.

Un automate de Wolfram est la donnée d'un ensemble de règles de la forme



ce qui peut se lire sous la forme $(g, c, d) \rightarrow c'$: connaissant l'état g de la cellule gauche, c de la cellule courante et d de la cellule de droite, le nouvel état de la cellule courante est c' .

```
[13]: int regle(int g, int c, int d)
{
    // prend un triplet (g,c,d) sous la forme de valeurs 0 ou 1
    // renvoie c' selon la règle au dessus en image selon la correspondance
    // noir = 1, blanc = 0

    // BEGIN SOLUTION
    if (g == 1 && c == 1 && d == 1)
        return 0;
    if (g == 1 && c == 1 && d == 0)
        return 0;
    if (g == 1 && c == 0 && d == 1)
        return 0;
    if (g == 1 && c == 0 && d == 0)
        return 1;
    if (g == 0 && c == 1 && d == 1)
        return 1;
    if (g == 0 && c == 1 && d == 0)
        return 1;
    if (g == 0 && c == 0 && d == 1)
        return 1;
    return 0;
    // END SOLUTION
}
```

```
[14]: if (regle(0,0,0) != 0) throw(0);
      if (regle(0,0,1) != 1) throw(0);
```

```

if (regle(0,1,0) != 1) throw(0);
if (regle(0,1,1) != 1) throw(0);
if (regle(1,0,0) != 1) throw(0);
if (regle(1,0,1) != 0) throw(0);
if (regle(1,1,0) != 0) throw(0);
if (regle(1,1,1) != 0) throw(0);
"Wow"

```

[14]: "Wow"

On voit dans l'exemple précédent qu'il y a 8 cas de figure, on peut les numéroté de 0 à 7, en fait, on peut considérer l'entier $4g + 2c + d$ pour associer à un tel triplet ce numéro. Par exemple, la configuration (1,1,0) correspond à $4 + 2 = 6$.

```

[15]: int regle(int config)
{
    // prend le triplet sous la forme d'un numéro comme décrit au dessus et
    ↪ renvoie la nouvelle valeur
    // On ne fera PAS appel à la fonction regle précédente
    // BEGIN SOLUTION
    if (config == 0 || config == 5 || config == 6 || config == 7)
        return 0;
    return 1;
    // END SOLUTION
}

```

```

[16]: if (regle(0) != 0) throw(0);
if (regle(1) != 1) throw(0);
if (regle(2) != 1) throw(0);
if (regle(3) != 1) throw(0);
if (regle(4) != 1) throw(0);
if (regle(5) != 0) throw(0);
if (regle(6) != 0) throw(0);
if (regle(7) != 0) throw(0);
"Top"

```

[16]: "Top"

```

[17]: void suivant(char in[], char out[])
{
    // prend une configuration in de cellules SOUS LA FORME D'UNE CHAINE
    // (donc terminée par '\0' et dont on peut obtenir la taille avec strlen)
    // applique la règle précédente sur chacune
    // et stocke la nouvelle configuration dans la chaîne out DE MEME LONGUEUR
    // ATTENTION : on utilise à la place de 0 le caractère ' '
    // et à la place de 1 le caractère '#'
    // ATTENTION : en dehors du tableau on considère que les cellules sont à ' '
}

```

```

// BEGIN SOLUTION
int n = strlen(in);
for(int i = 0; i < n; i++)
{
    int config = 0;
    if (i > 0 && in[i-1] == '#')
    {
        config += 4;
    }
    if (in[i] == '#')
    {
        config += 2;
    }
    if (i < n-1 && in[i+1] == '#')
    {
        config += 1;
    }
    if (regle(config) == 1)
    {
        out[i] = '#';
    }
    else
    {
        out[i] = ' ';
    }
}
// END SOLUTION
}

```

```

[18]: char in[] = "    ##  ## ##    #  #  #  #";
      char out[] = "                                "; // de meme longueur
      suivant(in, out);
      if(strcmp(out, "    ## # ##  # #  ##### # #") != 0) throw(0);
      "Trop bien !"

```

[18]: "Trop bien !"

```

[19]: void copie_chaine(char tgt[], char src[])
      {
          // prend en entrée deux chaines src et tgt de même longueur
          // copie la chaine src dans la chaine tgt
          // BEGIN SOLUTION
          int n = strlen(src);
          for (int i = 0; i < n; i++)
          {
              tgt[i] = src[i];
          }
      }

```

```

    }
    // END SOLUTION
}

```

```

[20]: char src[] = "toto";
char tgt[] = "abcd";
copie_chaine(tgt, src);
if (strcmp(src, tgt) != 0) throw(0);
"OK."

```

[20]: "OK."

```

[21]: void itere_regle(char depart[], char temp[], char sortie[], int iterations)
{
    // applique iterations fois la fonction suivant en partant de la
    ↪ configuration depart
    // et place la configuration finale dans sortie
    // comme on ne sait pas encore allouer une telle chaine dynamiquement, on
    // vous donne aussi une chaine temp comme espace de travail
    // ATTENTION : les chaines depart, temp et sortie sont de même longueur

    // BEGIN SOLUTION
    copie_chaine(sortie, depart);
    copie_chaine(temp, depart);
    for (int i = 0; i < iterations; i++)
    {
        suivant(temp, sortie);
        copie_chaine(temp, sortie);
    }
    // END SOLUTION
}

```

```

[22]: char init[81];
for(int i = 0; i < 80; i++)
{
    init[i] = ' ';
}
init[80] = '\0';
char temp[81], sortie[81];
temp[80] = '\0';
sortie[80] = '\0';
strcpy(temp, init);
strcpy(sortie, init);
init[40] = '#';
itere_regle(init, temp, sortie, 30);
if(strcmp(sortie, "          ## #  # ###  # ## #  ##### # ##### #  #_
↪ ##  #  #  #          ") != 0)

```

```

    throw(0);
    "Cool"

```

[22]: "Cool"

```

[23]: // Vous pouvez maintenant profiter et évaluer cette cellule et voir l'affichage
      ↪ des 30 premières itérations !
char init[81];
for(int i = 0; i < 80; i++)
{
    init[i] = ' ';
}
init[80] = '\0';

for(int i = 0; i < 30; i++)
{
    char temp[81], sortie[81];
    temp[80] = '\0';
    sortie[80] = '\0';
    strcpy(temp, init);
    strcpy(sortie, init);
    init[40] = '#';
    itere_regle(init, temp, sortie, i);
    printf("%s\n", sortie);
}

```

```

          #
        ###
       ## #
      ## ####
     ## #  #
    ## ##### ###
   ## #    # #
  ## ##### #####
 ## #   ###   #
## ##### ## #  ###
## #   # ##### ## #
## ##### ## #   # #####
## #   ### ## ## #  #
## ##### ##   ### ## ##
## #   #   ### #   ### # #
## ##### ## #   ##### #####
## #   ### ##### #   ###   #
## ##### ##   ###   ## #   ###
## #   #   ### #   ## ### ##### ## #
## ##### ## #   ##### #   #   ### #####
## #   ### #####   ##### ### ## #  #

```

```

      ## ##### ##   ###   #   ##   #   #   #   ## ## ##
      ## #   #   ###   #   ### ## #   ### ## #   #   #
      ## ##### ## #   ###   #   #   #####   #   #   ## #####
      ## #   ###   #####   ## #####   #   ##### #   #   #
      ## ##### ##   ###   #   ##   #   #   ## #   #####   ###
      ## #   #   ###   #   ## #   #####   ## #   ##   ##   #   ## #
      ## ##### ## #   ###   #   #   #   ###   ##### #   ## #   ## #   #####
      ## #   ###   #####   ##### ## ##   ###   #   #   #####   #   #   #
      ## ##### ##   ###   #   ##   #   #   ###   #   ##   #####   ##### ##   ###

```

Il se trouve que la règle qu'on vient d'implémenter s'appelle la règle 30, en effet si on construit l'entier $\sum_{i=0}^7 regle(i)2^i$ où $regle(i)$ a été définie au dessus, on obtient l'entier :

$$0.128 + 0.64 + 0.32 + 1.16 + 1.8 + 1.4 + 1.2 + 0.1 = 30$$

On va maintenant définir une fonction permettant d'obtenir la règle associée à un nombre entre 0 et 255. Par exemple, comme 111 s'écrit 01111011 en binaire, cela signifie qu'on associe la valeur 0 aux configurations de numéro 7 et 2 (i.e. (g, c, d) valant $(1, 1, 1)$ et $(0, 1, 0)$) et la valeur 1 à toutes les autres.

```

[24]: int regle_gen(int gen, int config)
{
    // renvoie la valeur obtenue en appliquant la règle gen
    // sur la configuration de numéro config
    // On pourra utiliser vers_binaire, même si sur 32 bits, beaucoup de
    ↪valeurs
    // seront à 0

    // BEGIN SOLUTION
    unsigned char r[32];
    vers_binaire(gen, r);
    return r[config];
    // END SOLUTION
}

```

```

[25]: if (regle_gen(30, 2) != 1) throw(0);
      if (regle_gen(188, 6) != 0) throw(0);
      if (regle_gen(188, 7) != 1) throw(0);
      if (regle_gen(188, 2) != 1) throw(0);
      if (regle_gen(188, 3) != 1) throw(0);
      "Nice !"

```

```

[25]: "Nice !"

```

```

[26]: void suivant_gen(int gen, char in[], char out[])
{
    // prend une configuration in de cellules SOUS LA FORME D'UNE CHAÎNE
    // (donc terminée par '\0' et dont on peut obtenir la taille avec strlen)

```

```

// applique la règle gen sur chacune
// et stocke la nouvelle configuration dans la chaîne out DE MEME LONGUEUR
// ATTENTION : on utilise à la place de 0 le caractère ' '
// et à la place de 1 le caractère '#'
// ATTENTION : en dehors du tableau on considère que les cellules sont à ' '

// REMARQUE : AVEC UN COPIER-COLLER de ce que vous avez fait pour suivant
// ça prend une seconde

// BEGIN SOLUTION
int n = strlen(in);
for(int i = 0; i < n; i++)
{
    int config = 0;
    if (i > 0 && in[i-1] == '#')
    {
        config += 4;
    }
    if (in[i] == '#')
    {
        config += 2;
    }
    if (i < n-1 && in[i+1] == '#')
    {
        config += 1;
    }
    if (regle_gen(gen, config) == 1)
    {
        out[i] = '#';
    }
    else
    {
        out[i] = ' ';
    }
}
// END SOLUTION
}

```

```

[27]: char in[] = "    ##  ## ##    # # # #";
char out[] = "                                "; // de meme longueur
suivant_gen(30, in, out);
if(strcmp(out, "    ## # ## # #  ##### # #") != 0) throw(0);
suivant_gen(188, in, out);
if(strcmp(out, "    # # # ## #  ## #####") != 0) throw(0);
"Génial !"

```

[27]: "Génial !"

```
[28]: void itere_regle_gen(int gen, char depart[], char temp[], char sortie[], int
    ↪ iterations)
{
    // applique iterations fois la fonction suivant_gen en partant de la
    ↪ configuration depart
    // et place la configuration finale dans sortie
    // comme on ne sait pas encore allouer une telle chaine dynamiquement, on
    // vous donne aussi une chaine temp comme espace de travail
    // ATTENTION : les chaines depart, temp et sortie sont de même longueur

    // REMARQUE : AVEC UN COPIER-COLLER de ce que vous avez fait pour
    ↪ suivant_gen
    // ça prend une seconde

    // BEGIN SOLUTION
    copie_chaine(sortie, depart);
    copie_chaine(temp, depart);
    for (int i = 0; i < iterations; i++)
    {
        suivant_gen(gen, temp, sortie);
        copie_chaine(temp, sortie);
    }
    // END SOLUTION
}
```

```
[37]: char init[81];
for(int i = 0; i < 80; i++)
{
    init[i] = ' ';
}
init[80] = '\0';
char temp[81], sortie[81];
temp[80] = '\0';
sortie[80] = '\0';
strcpy(temp, init);
strcpy(sortie, init);
init[40] = '#';
itere_regle_gen(30, init, temp, sortie, 30);
if(strcmp(sortie, "          ## # # ### # ## # # ##### # ##### # #
    ↪ ## # # # " ) != 0)
    throw(0);
itere_regle_gen(188, init, temp, sortie, 30);
if(strcmp(sortie, "          # ### ### ### ###
    ↪ ### ### ### # " ) != 0)
    throw(0);
"Dernière question : maintenant savourez en évaluant la cellule suivante"
```

[37]: "Dernière question : maintenant savourez en évaluant la cellule suivante"

[36]: *// On va afficher les 256 règles sur 30 itérations !*

```
char init[81];
for(int i = 0; i < 80; i++)
{
    init[i] = ' ';
}
init[80] = '\0';
char temp[81], sortie[81];
temp[80] = '\0';
sortie[80] = '\0';
strcpy(temp, init);
strcpy(sortie, init);
init[40] = '#';

for (int i = 0; i < 255; i++)
{
    printf("Regle %d\n", i);
    for(int j = 0; j < 30; j++)
    {
        itere_regle_gen(i, init, temp, sortie, j);
        printf("%s\n", sortie);
    }
    printf("\n");
}
```

Regle 0

#

[illegible]

#

#


```

#
#####
#
#####
#
#####
#
#####
#
#####
#
#####

```

Regle 4

[illegible]

Regle 5

```
#####  
##### # #####
```

[illegible]

Regle 6

[illegible]


```
#
#####
#
#####
# #
### #
##### # #####
### #
# ##### # #####
### #
# # ##### #
### #
### # ##### #
# ##### #
### #
### # ##### #
# ### #
### # ##### #
# # ### #
### # ##### #
### #
# ### #
```

Regle 10

Regle 11

21

Regle 12

22

Regle 13

Regle 14

23

Regle 16

Regle 17

25

Regle 18

26

Regle 21

28

Regle 22

Regle 23

29

Regle 24

30

Regle 27

32

Regle 28

Regle 29

33

Regle 30

34

Regle 31

Regle 32

Regle 33

[illegible]

```
#####  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

Regle 34

Regle 35

```
#####
#####
```



```

#
#####
#
#####
#
#####
#
#####
#
#####
#
#####

```

Regle 40

```

#

```

Regle 41

```

#
#####

```

Regle 42

42

Regle 43

Regle 44

43

Regle 46

Regle 47

45

Regle 48

46

47

48

```

#
#####

#
#####

#
#####

#
#####

#
#####

#
#####

#
#####

```

Regle 52

[illegible]

Regle 53

```
#####  
#####
```

Regle 54

50

Regle 55

Regle 56

Regle 57

52

Regle 58

Regle 59

53

```

#                                     ##
#####
##                                     ##
# #####
##                                     ##
## #####
# ##                                     ##
## #####
## ##                                     ##
# ## #####
## ##                                     ##
## ## #####
## ## ##                                     ##
# ## ## #####
## ## ##                                     ##
## ## ## #####
## ## ## ##                                     ##
# ## ## ## #####
## ## ## ##                                     ##
## ## ## ## #####
## ## ## ## ##                                     ##
# ## ## ## ## #####
## ## ## ## ##                                     ##
# ## ## ## ## ## #####

```

Regle 60

```

#
##
# #
####
# #
## ##
# # # #
#####
# #
## ##
# # # #
#### ##
# # # #
## ## ## ##
# # # # # # #
#####
# #
## ##

```

Regle 61

Regle 62

55

56

#####

Regle 64

#

Regle 65

#

Regle 66

58

Regle 69

60

Regle 70

Regle 71

61

[illegible]

#

#

Regle 76

Regle 77

65

Regle 78

66

Regle 79

Regle 80

Regle 82

Regle 83

69

Regle 84

70

Regle 85

Regle 86

71


```
#####  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

Regle 88

```

#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#

```

Regle 89

```
#####  
#####
```

Regle 90

74

Regle 91

Regle 92

75

Regle 93

76

```

#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #
#           # # # # # # # # # #           # # # # # # # # #
##### # # # # # # # # # # ##### # # # # # # # # #

```

Regle 94

```

#
###
## ##
### ##
## # # ##
### # # ###
## # # # # ##
### # # # # ###
## # # # # # # ##
### # # # # # # ###
## # # # # # # # # ##
### # # # # # # # # ###
## # # # # # # # # # # ##
### # # # # # # # # # # ###
## # # # # # # # # # # # # ##
### # # # # # # # # # # # # ###
## # # # # # # # # # # # # # # ##
### # # # # # # # # # # # # # # ###
## # # # # # # # # # # # # # # # # ##
### # # # # # # # # # # # # # # # # ###
## # # # # # # # # # # # # # # # # # # ##
### # # # # # # # # # # # # # # # # # # ###
## # # # # # # # # # # # # # # # # # # # # ##
### # # # # # # # # # # # # # # # # # # # # ###
## # # # # # # # # # # # # # # # # # # # # # # ##
### # # # # # # # # # # # # # # # # # # # # # # ###

```

Regle 95

```

#
#####

```



```
#  
#####  
# #  
##### #  
# #  
##### #  
# #  
##### ###  
# # #  
##### # #  
# # #  
##### #####  
# # # #  
##### # ### #  
# # #  
##### ##### ###  
# # # #  
##### # ### # #  
# # #  
##### #####  
# # # #  
##### # ### # ### #  
# # #  
##### # ### # ### #  
# # #  
##### # ### # ### #  
# # #  
##### #####  
# # # #
```

#

#

Regle 100

[illegible]

Regle 101

```
#####  
#####
```

Regle 102

82

```

      # #                # #
    #####              #####
      #   #             #   #
     ##  ##           ##  ##
    # # # #          # # # #
   #####            #####
      #       #        #       #
     ##      ##       ##      ##
    # #      # #      # #      # #
   #####    #####    #####    #####
  #   #    #   #    #   #    #   #
##  ##    ##  ##    ##  ##    ##  ##
#   #      #   #      #   #      #   #
#####    #####    #####    #####
#   #    #   #    #   #    #   #
##  ##    ##  ##    ##  ##    ##  ##

```

Regle 103

[illegible]

Regle 104

#

```

#####
# #####
# # # #
##### # # #####
# # # # # # # #
# ##### # # #####
# # # # # # #
### ##### # # # # #####
# # # # # # # # # # # #
# # # ##### ##### # # #
# # # # # # # # # # # #
# # # ##### # # # # # # #
# # # # # # # # # # # #
##### # ##### # ##### # #####
# # # # # # # # # # # #
### ### ##### # # # # #
# # # # # # # # # # # #
# # # # # # # # # # # #

```

Regle 106

Regle 107

85

Regle 108

86

#

Regle 109

#

Regle 110

##

```

#####
## #
#####
## #
### ##
## # ###
##### #
## ###
### ## #
## # #####
##### ## #
## # ### ##
### ##### # ###
## # ## ##### #
##### ## ##
## ##### ## #
### ## # #####
## # ### ##### #
##### ## ### # ##
## # ##### # ## ##
### ## ## ##### #
## # ##### ## ###
##### # ### ## #
## # ##### ## ##### #

```

Regle 111

```

#
#####
# #####
# #####
### #####
# # #####
#####
# # #####
# # #####
### # #####
# # #####
##### # #####
# # #####
# # #####
# #####
### # #####
# # #####
##### # #####
# # #####
# # #####
##### # #####
# # #####

```

Regle 112

Regle 113

89

Regle 115

Regle 116

91

Regle 117

92

Regle 120

94


```

#                                     ###                                #
#####
##                                     ###                                ##
#####
#                                     ###                                #
#####
##                                     ###                                ##
#####
#                                     ###                                #
#####
##                                     ###                                ##
#####

```

Regle 124

```

#
##
###
# ##
#####
#  ##
##  ###
### # ##
# #####
###    ##
# ##   ###
#####  # ##
#   ##  #####
##   ### #   ##
### #  #####  ###
#  #####  ## # ##
###   ##  #####
# ##   #####  ##
##### #  ##   ###
#   #####  ###  # ##
##   #   ###  ##  #####
### ## #  #####  #   ##
#  #####   ## ##  ###
###   ##  #####  #  ##
#####  ###  #####  #   ##

```

Regle 125

```

#
#####

```

Regle 126

98

Regle 127

Regle 128

Regle 129

```

#
#####
##### # #####
#####
# ##### ##### #
##### ### #####
### ##### ## # ##### ###
# ##### ##### #
## ##### ##### ##
## ##### ##### ##
#### ##### ## #####
## ## ##### ##### ##### ## ##
## ##### ##### ##### #####
##### ##### ### #####
#### ##### ## ## # ## ## ##### #####
## ## ##### ##### ## ##
# ## ## ##### ##### ##### ## ## #
## ## ##### ##### ##### ## ##
```


Regle 132

102

Regle 133

Regle 134

103

104

Regle 136

Regle 137

#####

Regle 138

106

Regle 141

108

Regle 142

Regle 143

109

Regle 144

110

Regle 150

114

```

      # # #      # # #      # # #
    ## # ##    ## # ##    ## # ##
      # # #      # # #      # # #
    ### ### ###  ### ### ###  ### ### ###
      # # #      # # #      # # #
    ## ## ### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
      #      #      #      #      #
    ###      ###      ###      ###      ###
      # # #      # # #      # # #      # # #      # # #
    ## # ## ## # ##  ## # ##  ## # ## ## # ##
      # #      # #      # #      # #      # #
    ### ###      ### ###  ### ###  ### ###  ### ###

```

Regle 151

```

#
#####
#####
# ##### #
# ##### #
### ##### ###
# ##### #
##### #####
### ##### ###
# # ### ##### # #
# # # ##### # #
# ##### ##### #
# ##### ##### #
### ### ### ##### ### ### ###
# # # ##### # # #
##### ##### #####
##### ##### #####
# ##### # # ##### # # ##### #
# ##### # ##### # ##### #
### ##### ##### ##### ##### #
# ### ##### ##### ##### #
##### # ### ### ### ##### # # ### #
### # # # ##### # # # #
# # ##### ##### ##### ##### # #
# # ##### ##### ##### ##### # #
# ### ##### # # ##### # # ##### #
# # ##### # ##### # ##### #
##### ##### ##### ##### #####
##### ##### #####
# ### # # # # # ##### # # # #
# # ##### ##### ##### ##### # #
# # ##### ##### ##### ##### # #
# ### ##### # # ##### # # ##### #
# # ##### # ##### # ##### #
##### ##### ##### ##### #####
##### ##### #####
# ### # # # # # # # # # # # # # # #

```

Regle 152

```

#
#

```

Regle 153

116

Regle 154

Regle 155

117

Regle 156

118

Regle 157

Regle 158

119

120

```
#####  ##  ##  ##  ##
#####  ###  ###  ###  ##
#####  ##  ##  ##  ##
#####  ###  ###  ###  ###
#####  ##  ##  ##  ##
#####  ###  ###  ###  ##
#####  ##  ##  ##  ##
#####  ###  ###  ###  ###
#####  ##  ##  ##  ##
#####  ###  ###  ###  ###
#####  ##  ##  ##  ##
#####  ###  ###  ###  ##
#####  ##  ##  ##  ##
#####  ###  ###  ###  ###
```

Regle 160

#

Regle 161

#

```
#####  #####
```

Regle 162

122

Regle 163

Regle 164

123

124

Regle 166

Regle 167

125

[illegible]

#

#

128

Regle 172

[illegible]

Regle 173

129

Regle 174

130

Regle 175

Regle 176

131

Regle 177

132

Regle 178

Regle 179

133

Regle 180

134


```

# # #
#####
# ##### #
### ### ###
# # # # # #
#####
# ##### #
### ##### ###
# # # ##### # # #
##### #####
# ##### # ##### #
### ### ### ### ### ###
# # # # # # # # # # # #
#####
# ##### #
### ### ### ### ### ###
# # # # # # # # # # # #
#####
# ##### # ##### # ##### #
### ### ### ##### ### ### ###
# # # # # # # ##### # # # # # #
#####
# ##### # ##### # ##### #
### ##### ### ##### ### #####
# # # ##### # # # ##### # # # ##### # # #
##### ##### ##### ##### #####
# ##### # ##### # ##### # ##### # ##### #
### ### ### ### ### ### ### ### ### ###

```

Regle 183

```

#
#####
#####
# ##### #
## ##### ##
# ##### #
#### #####
## # ##### # ##
# ### ##### ### #
### # # ##### # # ##
# ##### #####
### ### # ##### # ## ###
# # # ### ##### ### # # #
##### # # ##### # # #####
##### #####
# ### # ### # ##### # ### # ### #
## # ### # ### ##### ### # ### # ##
### # ### # # ##### # # ### # ##

```

Regle 184

Regle 185

137

Regle 186

138

Regle 187

Regle 188

139

Regle 189

140

Regle 190

Regle 191

141

[illegible]

#

#

Regle 196

Regle 197

145

Regle 198

146

Regle 199

Regle 200

Regle 201

[illegible]

Regle 202

Regle 203

149

Regle 204

150

Regle 205

Regle 206

151

Regle 207

152

Regle 208

Regle 209

153

Regle 210

154

Regle 211

Regle 212

155

Regle 213

156

```

## ## ## ## #####
## ## ## ## #####
# ## ## ## ## #####
## ## ## ## ## #####
## ## ## ## ## #####
# ## ## ## ## ## #####
## ## ## ## ## ## #####
## ## ## ## ## ## #####
## ## ## ## ## ## #####
# ## ## ## ## ## ## #####
## ## ## ## ## ## ## #####

```

Regle 214

```

#
###
# ###
## ###
# ### ###
## ## ###
# ### ### ###
## ## ## ###
# ### ### ### ###
## ## ## ## ###
# ### ### ### ### ###
## ## ## ## ## ###
# ### ### ### ### ### ###
## ## ## ## ## ## ###
# ### ### ### ### ### ###
## ## ## ## ## ## ## ###
# ### ### ### ### ### ### ###
## ## ## ## ## ## ## ## ###
# ### ### ### ### ### ### ###
## ## ## ## ## ## ## ## ###
# ### ### ### ### ### ### ###
## ## ## ## ## ## ## ## ## ###
# ### ### ### ### ### ### ###
## ## ## ## ## ## ## ## ## ###

```

Regle 215

```

#
#####

```

Regle 216

158

Regle 219

160

Regle 220

Regle 221

161

Regle 222

162

Regle 223

[illegible]

Regle 224

#

Regle 225

```

#
#####
##### # #####
#####
# ##### ## #####
##### # #####
### ##### # #####
## #####
# # ##### ####
# # ##### ## #####
## ##### # # #####
#### # ##### ## #####
### # ##### # ## #####
## ## ##### # # #####
# ## # ## ##### # # #####
## # ## #####
### ## # # #####
## ## # # #####
```

Regle 226

Regle 227

165

Regle 228

166

Regle 229

Regle 230

167

```

# #
####
# ###
### ##
# ### #
### #####
# ### ###
### ### ##
# ### ### #
### ### #####
# ### ### ###
### ### ### ##
# ### ### ### #
### ### ### #####
# ### ### ### ###
### ### ### ### ##
# ### ### ### ### #
### ### ### ### #####
# ### ### ### ### ###
### ### ### ### ### ##
# ### ### ### ### ### #
### ### ### ### ### #####
# ### ### ### ### ### ###
### ### ### ### ### ### ##

```

Regle 231

```

#
#####
#####
# #####
## #####
## #####
# ## #####
## ## #####
## ## ## #####
# ## ## ## #####
## ## ## ## #####
## ## ## ## ## #####
# ## ## ## ## ## #####
## ## ## ## ## ## #####
## ## ## ## ## ## ## #####
## ## ## ## ## ## ## ## #####

```


#####

Regle 232

#

Regle 233

#

#####

Regle 234

170

Regle 237

172

Regle 238

Regle 239

173

Regle 240

174

#

Regle 241

#####

Regle 242

#

176

Regle 244

Regle 245

177

Regle 246

178

```

# ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ### ### ### ###
# ### ### ### ### ### ### ### ### ### ### ### ### ###
### ### ### ### ### ### ### ### ### ### ### ### ###

```

Regle 247

```

#
#####
#####
# #####
## #####
## #####
# ## #####
## ## #####
## ## #####
# ## ## #####
## ## ## #####
## ## ## ## #####
# ## ## ## ## #####
## ## ## ## ## #####
## ## ## ## ## #####
# ## ## ## ## ## #####
## ## ## ## ## ## #####
## ## ## ## ## ## #####
# ## ## ## ## ## ## #####
## ## ## ## ## ## ## #####
## ## ## ## ## ## ## #####
# ## ## ## ## ## ## ## #####
## ## ## ## ## ## ## ## #####
## ## ## ## ## ## ## ## #####
# ## ## ## ## ## ## ## ## #####
## ## ## ## ## ## ## ## ## #####
## ## ## ## ## ## ## ## ## #####

```

Regle 248

```

#
#

```

Regle 249

180

Regle 250

Regle 251

181

Regle 252

182

Regle 253

[illegible]

###

