Algorithmique et Programmation

—

Examen sur machine

—

G1: monasse*(at)*imagine.enpc.fr
G2: boulc-ha*(at)*imagine.enpc.fr
G3: yoann*(at)*le-bars.net
G4: arnaud.carayol*(at)*univ-mlv.fr
G5: liuz*(at)*imagine.enpc.fr
G6: vialette*(at)*univ-mlv.fr

—

09/12/11

—

1 Enoncé

1.1 Automate cellulaire

Un automate cellulaire est une règle de transformation d'une séquence de bits (0 ou 1), appelés cellules, en une autre séquence par une règle de transformation : chaque bit est transformé en fonction de sa valeur et de ses deux voisins immédiats. Par exemple, voici une telle règle :

voisinage	111	110	101	100	011	010	001	000
nouvelle valeur centrale	0	1	1	0	1	1	1	0

La séquence de bits de la deuxième ligne se lit en binaire comme le nombre $2^1 + 2^2 + 2^3 + 2^5 + 2^6 = 110$ et est la règle 110. Voici un exemple de transformation suivant cette règle :

automate initial	1	1	0	1
automate transformé	0	1	1	1

en considérant la séquence comme circulaire, c'est-à-dire que le voisin de gauche de la première cellule est la dernière cellule et que le voisin de droite de la dernière cellule est la première.

Pour calculer la valeur centrale, il suffit d'interprêter le voisinage comme un binaire, par exemple $101 = 2^2 + 2^0 = 5$ et de lire le bit numéro 5 en partant de la droite (et commençant à compter par 0) de la décomposition binaire de la règle : 01**1**01110.

Parmi les 256 règles possibles, certaines sont plus intéressantes que d'autres : elles exhibent une certaine régularité sans être réellement périodiques. On représente un automate en laissant en blanc ou en noir un pixel d'une ligne en fonction de la valeur de la cellule. Dans le diagramme (voir figure), chaque ligne est l'automate transformé de la ligne précédente.

Il existe en C++ des opérateurs agissant directement sur les bits. Comme nous ne les avons pas encore vus en cours, nous n'allons pas les utiliser mais représenter la valeur d'une cellule par un `bool` et gèrer nous-mêmes les opérations.

1.2 Travail demandé

Il est plus important de livrer un code clair et qui compile sans warning, même s'il ne répond pas à toutes les questions. Pour cela, vérifiez à chaque étape que votre programme compile et se lance correctement.

1. Créez un nouveau projet Imagine++ nommé `Automate` et écrivez le `main`, qui ouvre une fenêtre de taille fixe.
2. Dans un fichier `automate.h`, écrivez une structure `Automate`, comprenant un tableau de `bool` et sa taille, de type `int`.

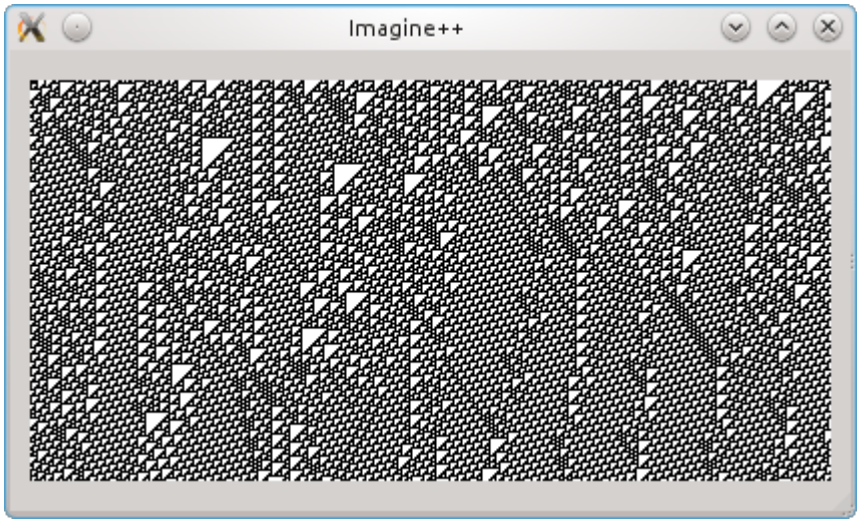

FIGURE 1 – Une réalisation de l'automate cellulaire

1.2.1 Automate

3. Dans `automate.cpp`, implémentez des fonctions :
   – `init` prenant un automate et une taille et allouant le tableau ;
   – `detruis` libérant le tableau de l'automate passé en argument ;
   – `random` mettant les cellules de l'automate aléatoirement à `true` ou `false`.
   N'oubliez pas de *déclarer* ces fonctions dans `automate.h`.
4. Implémentez une fonction `affiche` prenant un automate et un numéro de ligne j et mettant à noir ou blanc le pixel (i, j) suivant la valeur de la cellule numéro i de l'automate.
5. Dans le fichier `diagramme.h`, écrivez la structure `Diagramme` avec 3 champs : `n` (le nombre d'automates, un entier), `nmax` (taille maximum du diagramme) et un tableau d'automates.
6. Une fonction `init` crée un nombre `nmax` d'automates, le premier étant initialisé par `random`.
7. Ecrivez une fonction `affiche` dessinant les `n` cases d'un diagramme.

1.2.2 Binaires

8. Dans un fichier `binaire.h`, écrivez une structure `Binaire` contenant un tableau de 8 booléens.
9. Ecrivez dans `binaire.cpp` une fonction `decompose` décomposant un entier passé en paramètre (supposé entre 0 et 255) en sa décomposition binaire, avec la convention 1=`true`, 0=`false`. Procéder ainsi :
   – Si le nombre est impair (utiliser l'opération modulo % sur les entiers), mettre la première valeur du tableau à `true` (sinon `false`).
   – Diviser le nombre par 2 (quotient de division euclidienne).
   – Tester la parité du nombre, et affecter la deuxième valeur du tableau.
   – Diviser à nouveau par 2.
   – Réitérer pour remplir tout le tableau.
10. Ecrivez une fonction `compose` prenant un tableau de 3 booléens et retournant le nombre binaire correspondant : le premier booléen vaut 1, le deuxième 2 et le troisième 4. On obtient donc un entier entre 0 et 7.

1.2.3 Règle de transformation

11. Ecrivez une fonction `transforme` prenant deux 2 automates en paramétre et remplissant le deuxième par la règle de transformation. Pour cela, on passe aussi la décomposition binaire de la règle et il suffit d'aller chercher le bon bit dans ce tableau en fonction du voisinage.
12. Ecrivez une fonction `evolue` ajoutant une ligne au diagramme par transformation de la dernière.
13. Faites remplir progressivement toute la fenêtre avec le diagramme : la première ligne est un automate aléatoire et les suivantes sont le résultat de la transformation de l'automate de la ligne précédente.

14. Quant on arrive au bout de la fenêtre, pour ajouter une ligne on remonte tout le diagramme actuel d'une ligne pour pouvoir insérer la nouvelle. Modifiez la fonction `evolue` pour mettre a jour le diagramme quand n atteint $nmax$.
15. Faire défiler à l'infini l'automate dans la fenêtre.

Important : Quand vous avez terminé, *nettoyez la solution* et créez une archive du projet *à votre nom*. Ne vous déconnectez pas avant que le surveillant ne soit passé vous voir pour copier cette archive sur clé USB.