UNIVERSITÉ DE MONTPELLIER
UNIVERSITÉ DE MONTPELLIER - FACULTÉ DES SCIENCES ET TECHNIQUES
PLACE EUGÈNE BATAILLON - 34095 MONTPELLIER CEDEX 5

Licence 2 informatique – 2019/2020

HLIN302 – Travaux Dirigés n° 4

Programmation impérative avancée

Alban MANCHERON et Pascal GIORGI

La gestion des entrées sorties est un incontournable de tous les langages de programmation. En effet, cela permet de récupérer de l'information (lecture) et de stocker des résultats (écriture).

Mais avant de commencer, il est grand temps de démystifier les arguments passés à la fonction `main` de vos programmes.

Des arguments de poids

Lorsque vous ouvrez un *shell*, la plupart des commandes que vous pouvez exécuter acceptent des paramètres optionnels, voire certaines nécessitent de fournir un ou plusieurs arguments obligatoires.

```
prompt$ ls -l
.  ..  TD4.h
prompt$ cp -i TD4.h TD4.cpp
prompt$ emacs TD4.cpp
prompt$ g++ -Wall -ansi -pedantic -g -O0 -c TD4.cpp
```

Lorsque vous écrivez vos propres programmes `C++`, vous pouvez récupérer les arguments passés à votre exécutable et les analyser.

En effet, la fonction `main` admet deux paramètres, le premier de type entier (traditionnellement appelé `argc` pour *argument count*), le second de type tableau de tableaux de caractères (traditionnellement appelé `argv` pour *argument vector*). Lorsque votre binaire est exécuté, ces deux variables sont mises à jour de sorte que le tableau `argv` contient exactement `argc` chaînes de caractères, toutes terminées par le caractère `'\0'`. Ainsi vous pouvez *a minima* récupérer le nom du binaire qui a été exécuté car c'est la première chaîne stockée dans le tableau.

Pour mieux comprendre, le plus simple est de compiler le petit programme suivant et de l'exécuter avec et sans paramètres.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char** argv) {
6
7    cout << "Le programme a été exécuté avec argc = " << argc << " et :" << endl;
8    for (int i = 0; i < argc; i++) {
9       cout << "- argv[" << i << "] = '" << argv[i] << "'" << endl;
10    }
11    cout << "Au revoir." << endl;
12   return 0;
13
14 }
```

1 Un chat dans une gouttière

1.1 Cat's in the cradle

La commande `cat` de votre OS préféré permet d'afficher sur un terminal le contenu des fichiers qui lui sont passés en paramètres. Son nom est provient de l'anglais *[con]cat[enate]*, qui se traduit par « concaténer » [1].

1. Écrire un programme qui prend en entrée un nom de fichier et qui affiche son contenu ligne par ligne à l'écran, en vérifiant bien évidemment l'existence du fichier et sa lisibilité.
2. Modifier ce programme permettre d'afficher le contenu de plus d'un fichier passé en paramètre.

1.2 l'heure du 'T'

La commande `tee` de votre OS préféré permet quant à elle d'afficher sur un terminal le contenu de ce qui est fourni entrée tout en le recopiant dans les fichiers qui lui sont passés en paramètres. Son nom lui vient de la graphie de la lettre `'T'` qui symbolise le principe d'un tuyau qui duplique son entrée (la partie gauche de la barre horizontale) vers les deux sorties (la partie droite de la barre horizontale et la barre verticale).

1. Écrire un programme qui recopie le contenu de l'entrée standard (ce qui est saisi au clavier) sur la sortie standard ainsi que dans le fichier dont le nom est fourni en argument du programme, en vérifiant bien évidemment l'existence du fichier et sa lisibilité.
2. Modifier ce programme pour permettre de copier le contenu de l'entrée dans plus d'un fichier passé en paramètre.

2 Retour au jeu de la vie

Par défaut, le jeu de la vie peut-être initialisé au hasard avec une probabilité fixée *a priori* qu'une cellule soit vivante. Il est intéressant de démarrer avec une configuration précise.

Une manière de procéder est d'écrire la configuration dans un fichier qui, s'il est fourni en ligne de commande définira l'état de départ du jeu de la vie.

2.1 Arguments de la ligne de commande

De manière générale, il est souhaitable de pouvoir gérer plusieurs options en ligne de commande, telles que `--help` (qui affiche l'aide), `--version` (qui affiche la version), `--dimension <entier>` (qui initialise une matrice carrée de la dimension spécifiée), `--probability <réel>` (qui définit la probabilité d'être en vie d'une cellule au démarrage), `--config <fichier>` (qui charge la configuration initiale du jeu), ainsi que leurs versions courtes respectives (`-h`, `-v`, `-N`, `-p` et `-f`). Il est bien évidemment possible d'inventer d'autres options.

1. Écrire la déclaration d'une classe qui permet de représenter une option (un identifiant numérique, son intitulé, son raccourci, le type éventuel de son argument et sa description).
2. Écrire la déclaration d'une classe qui gère un tableau d'options (vide au départ), qui permette :
   — d'ajouter une option ;
   — d'afficher les options enregistrées ;
   — de renvoyer l'identifiant correspondant à une chaîne de caractères donnée
   — de savoir, étant donné un nom (court ou long) d'option, si un argument est requis et son type le cas échéant.
3. Écrire un programme qui déclare les options du jeu de la vie et qui affiche un message spécifique pour chaque option passée en ligne de commande (la classe gérant les options devra s'assurer que l'identifiant fourni pour chaque option est unique).

1. Du latin *cum* (« avec ») et *catena* (« enchaînement, lien, … »).

2.2 Configuration initiale

Un fichier de configuration doit suivre un formalisme précis. En ce qui concerne le jeu de la vie, nous poserons de le définir ainsi :

- tout ce qui suit le caractère `'#'` est considéré comme un commentaire (et donc est ignoré) ;
- chaque ligne (nettoyée des éventuels commentaires) est de la forme `clé : valeur` (la `valeur` peut-être vide auquel cas une valeur par défaut est appliquée) ;
- ne pas indiquer un mot clé revient à lui assigner sa valeur par défaut ;
- la casse des caractères est prise en compte et les espaces superflus sont ignorés ;
- les mots-clés reconnus sont :
  - `Dimension` qui reçoit un entier compris entre 3 et N_{max} ;
  - `Probability` qui reçoit un réel compris entre 0 et 1 ;
  - `Cell` qui reçoit un couple d'entiers séparés par une (ou plusieurs) espace(s) `'␣'`, une virgule (`','`) ou la lettre x (`'x'`).
- chaque cellule déclarée est crée vivante, les autres sont crées vivantes ou mortes en fonction de la probabilité fournie.
- si le mot clé `Dimension` ou `Probability` est fourni plusieurs fois, c'est sa dernière occurence qui sera prise en compte. Dès qu'un de ses mots-clés est rencontré, l'ensemble de la population doit être regénérée aléatoirement, en fonction de la dimension et de la probabilité.
- par défaut, `Dimension` est fixé à 8 et `Probability` est fixé à 25%.
- si un le mot clé `Cell` apparait avant `Dimension` ou `Probability` cette cellule ne pourra pas être garantie dans la constituion aléatoire de la nouvelle population.

Questions :

1. Écrire un exemple de fichier de configuration.
2. Modifier la classe **`PopulationVivante`** pour ajouter la possibilité de stocker une probabilité et une dimension, et de pouvoir les modifier.
3. Ajouter une méthode de regénération de la population.

2.2.1 Parcours de fichiers de configuration

Afin de parcourir un fichier de configuration et d'en extraire les informations pertinentes, nous avons besoin de définir des fonctions dite de "parsing". Vue les spécificités de notre fichier de configuration, il va falloir parcourir chaque ligne du fichier et en extraire les couples (clé/valeur) si l'information est présente, et les répercuter sur la population du jeu de la vie. Pour cela, nous allons nous appuyer sur la classe `JeuDeLaVie` suivante :

```
1
2 class JeuDeLaVie {
3  private:
4   PopulationVivante POP;
5
```

La méthode de "parsing" étant complexe, nous allons la définir en plusieur étapes au travers de plusieurs méthodes (bien évidemment, celles-ci resteront privé car elles n'auront qu'un usage interne à la classe `JeuDeLaVie`).

- Chaque ligne du fichier de configuration doit être extraite dans une chaîne de caractère
- Chaque chaîne de caractère devra être nettoyer, c-a-d, on doit enlever les espaces inutiles ainsi que les commentaires.

Pour cela, vous pourrez utiliser les méthodes `find_first_of(`**`std::string`**`)` et `substr(`**`size_t`** `debut,` **`size_t`** `n)` de la classe **`std::string`**.

La méthode `find_first_of(`**`std::string`**`)` permet de trouver la première occurence d'un caractère dans une chaine et de renvoyer sa position dans la chaine sous forme d'un entier. Le paramère de cette méthode spécifie un ou plusieurs caractère à recherché, la position de la première occurence d'un des caractères sera alors renvoyé. Ex :

```
1     string s("bonjour il # fait beau");
2     size_t n1= s.find_first_of("#");
3     size_t n2= s.find_first_of("i#");
```

Dans cet exemple, $n1 = 11$ car le caractère # est à la position 11 dans la chaine s et $n2 = 8$ car la première occurence d'un espace ou d'un # apparait à la position 8 dans s. Si aucun des caractères n'est présent dans la chaine, la méthode renvoie l'entier **string**::npos.

La méthode substr(**size_t** debut, **size_t** n) permet de récupérer la sous-chaine formé des n caractères à partir de la position debut. (Rq : si on ne donne pas le nombre de caractère, tous les caractères restants seront récupérés). Ex :

```
1     string s("bonjour il # fait beau");
2     size_t n1= s.find_first_of("#");
3     size_t n2= s.find_first_of("i#");
4     string s1= s.substr(0,n1); // contient "bonjour il "
5     string s2= s.substr(n2); // contient "il # fait beau"
```

Questions :

1. Ajouter la méthode nettoie(**std::string** s) à la classe JeuDeLaVie pour permettre de nettoyer la chaine de caractère correspondant à une ligne d'un fichier de configuration.
2. Ajouter une méthode findCleVal qui permet d'extraire les deux chaines de caractères correspondant à la clé et à la valeur. Si la ligne ne contient pas de clé/valeur (chaine vide) alors cette méthode renverra false (si il y a un couple clé/valeur elle renvoie true). Attention, il faudra penser à nettoyer les chaines de caractères ! ! !

2.2.2 Traitement des informations du fichier de configuration

Maintenant que nous pouvons extraire des couples "clé/valeur", il faut répercuter les informations sur la population du jeu de la vie : si la clé est Dimension ou Probability alors on modifie ces valeurs dans la population et on la regénère (Attention, vous devrez avoir modifié la classe **PopulationVivante** pour intègrer ces possibilités de modification et de regénération). Si la clé est Cell, alors on doit extraire la positions (x,y) de la cellule dans la chaine valeur et ajouter la cellule correspondante à la population.

Questions :

1. Ajouter la méthode traiteOption qui permet à partir d'une clé et d'une valeur de modifier la population du jeu de la vie.
2. Ajouter le constructeur vide à la classe JeuDeLaVie permettant de définir les valeurs par défaut du jeu de la vie.
3. Ajouter la méthode permettant d'initialiser le plateau en fonction d'un fichier de configuration. Cette méthode s'appuierau sur les méthodes de parsing et de traitement écrites précédemment.
4. Ajouter la méthode permettant de lancer le jeu de la vie pour un nombre d'étapes spécifié.
5. Faite un programme pour tester votre classe JeuDeLaVie.

2.3 Un jeu de la vie avec arguments en ligne de commande

Dans les deux sections précédentes, nous avons tous d'abord défini un programme permettant de spécifier les options du jeu de la vie, puis nous avons défini une classe permettant de gérer l'initialisation du jeu de la vie avec un fichier de configuration. Il est temps maintenant de fusionner ces deux approches pour offrir un vrai programme de jeu de la vie.

1. Créez une nouvelle classe JeuDeLaVie en dupliquant celle du dernier exercice.
2. Ajouter un tableau d'option comme attribut de cette classe. Modifiez le constructuer par défaut en conséquence (il devra construire le tableau de paramètres du jeu de la vie).

3. Ajouter la méthode `parseOptions` qui récupérera les paramètres en ligne de commande et qui appliquera les arguments sur le jeu de la vie.
4. Proposez un programme permettant d'utiliser le jeu de la vie avec des arguments en ligne de commande.