UNIVERSITÉ DE MONTPELLIER
UNIVERSITÉ DE MONTPELLIER - FACULTÉ DES SCIENCES ET TECHNIQUES
PLACE EUGÈNE BATAILLON - 34095 MONTPELLIER CEDEX 5
Licence 2 informatique – 2018/2019

HLIN302 – Travaux Dirigés n° 5

Programmation impérative avancée
Alban MANCHERON et Pascal GIORGI

1 Gestion de données dynamiques

Dans cet exercice, nous souhaitons proposer une classe permettant de gérer des tableaux d'entiers de taille quelconque et dont celle-ci n'est pas connue à la compilation. Pour cela, votre classe va devoir gérer une zone mémoire dynamique : c'est-à-dire que votre objet devra gérer explicitement la mémoire du tableau. Pour cela, vous allez utiliser l'allocation dynamique via les appels **`new`** et **`delete`**.

Attention, qui dit objet avec mémoire dynamique dit gestion explicite des constructeurs,destructeur et copies.

1.1 Un tableau d'entiers

main-1.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include "tableau-int.h"

using namespace std;

int main(int argc, char** argv){
  if (argc !=2) {cerr<<"Usage: "<<argv[0]<<" [tab dim]"<<endl; return 1;}

  size_t n = atoi(argv[1]);
  TableauInt T(n);
  for(size_t i=0;i<n;i++){
    T.at(i)=i+1;
  }
  write(cout,T);

  return 0;
}
```

1. Proposez la signature de la classe `TableauInt` permettant de compiler le programme ci-dessus.
2. Donnez le code complet de la classe `TableauInt`

1.2 Un tableau d'entier extensible

Nous souhaitons maintenant améliorer cette classe afin de permettre l'extension du tableau par ajout d'éléments à la fin de celui-ci. En particulier, nous souhaitons définir une méthode **`void`** `push_back(`**`int`**`)` qui va agrandir l'espace mémoire du tableau et ajouter l'entier désiré à la fin de celui-ci.

Comme l'opération d'agrandissement reste coûteuse (il faut recopier l'ensemble des cases du tableau initial dans un nouveau tableau plus grand), nous allons utiliser une stratégie d'agrandissement dite paresseuse. Cette

stratégie consiste à allouer plus de mémoire que nécessaire lorsqu'on souhaite ajouter un élément au tableau. Si l'allocation est suffisamment grande, alors en moyenne il y aura peu de recopie du tableau par ajout d'élément. Une stratégie pertinente est de doubler la taille du tableau à chaque extension.

1. Quel attribut doit-on ajouter à la classe `TableauInt` pour mettre en place cette stratégie ?
2. Donner le code de la méthode `extend()` qui agrandit le tableau selon la stratégie de doublement. Attention, on ne double la taille que lorsque le tableau est réellement plein.
3. Donner le code de la méthode **`void`** `push_back(`**`int`**`)`
4. Vous modifierez le programme `main-1.cpp` pour mettre en avant l'utilisation de cette nouvelle classe.
5. Si l'on remplace les lignes 11,12 et 13 du programme `main-1.cpp` par

```
11  TableauIntExtensible T;
12  for(size_t i=0;i<n;i++)
13    { T.push_back(i+1);}
```

   combien de fois sera agrandit le tableau ?

2 Le jeu de la vie (le retour).

Dans la feuille de TD 3, nous avons proposé une classe `PopulationVivante` qui stockait uniquement les cellules vivantes d'une N-population du jeu de la vie. Afin de simplifier le code de cette classe, nous allons utiliser un tableau dynamique extensible de cellule.

2.1 Un tableau extensible de cellule

1. En vous inspirant du tableau extensible d'entiers, proposez une classe `TableauCellule` permettant l'ajout de cellule en fin de tableau.
2. Afin de permettre le lecture/écriture de configurations du jeu de la vie dans des fichiers, votre classe proposera une méthode **`void`** `read(std::istream&)` et **`void`** `write(std::ostream&)` permettant de lire/écrire un tableau de cellule dans des flux. Bien que ces méthodes ne sont pas définies pour des objets de type `ofstream` et `ifstream`, un mécanisme appelé "héritage" vous permettra d'utiliser ces deux méthodes sur ce type de flux de fichier. Donnez le code de ces deux méthodes.
3. Comme les cellules du tableau doivent être accédé par rapport à leurs coordonnées (i, j) dans la grille du jeu de la vie, il faut fournir un mécanisme permettant de retrouver un cellule précise dans le tableau. Attention, il se peut que votre cellule n'appartienne pas au tableau.
   (a) Proposez la signature d'une méthode `find` qui permet de récupérer la cellule recherchée si elle appartient au tableau. Si elle n'appartienne pas, votre méthode doit permettre de le tester rapidement.
   (b) Si vous avez utilisé les pointeurs, proposez une version sans pointeurs.
   (c) Donner le code complet d'une des deux version de la méthode `find`.
4. En accord avec votre version de la méthode `find`, proposez le code complet d'une méthode `erase` permettant d'enlever une cellule précise du tableau. En particulier, votre classe doit permettre l'appel suivant :

```
1 T.erase(T.find(Cellule(true,2,3)));
```

   qui supprime du tableau `T` la cellule vivante en position $(2, 3)$, si elle est présente sous cette forme dans le tableau (si la cellule est présente mais morte, alors elle n'est pas supprimée).

2.2 Une nouvelle classe de Population

La classe `PopulationVivante`, dont vous trouverez les premières lignes ci-dessous, permettait de stocker dans un tableau statique de taille maximum `NMAX` l'ensemble des cellules vivantes d'une N-Population. Un des

inconvénients de cette classe et que l'on est limité par le nombre de cellules vivantes, ce qui implique des restrictions sur la taille des N-Population que l'on peut couvrir.

Population Vivante

```
#ifndef __POPULATION_VIVANTE_H
#define __POPULATION_VIVANTE_H
#include "cellule.h"

#define NMAX 100

class PopulationVivante {
 private:
  Cellule T[NMAX];
  size_t alive;
  size_t N;

```

Vous trouverez les codes complets de cette classe sous l'ENT de l'UM.

1. Comment changer les attributs de cette classe pour utiliser le Tableau extensible de cellule vu à la section précédente ?
2. Quels sont les modifications à faire sur le ou les constructeurs de cette classe ?
3. Vous remarquerez que cette classe utilise des méthodes `at(i,j)` pour récupérer un pointeur sur une cellule précise de la population. Maintenant que votre classe utilise la classe `TableauCellule`, peut on supprimer ces méthodes ? Si oui, par quoi peut on les remplacer ? Modifiez les codes en conséquence.
4. Ajouter à votre classe un constructeur paramétré sur un flux de lecture fichier, qui construira une population à partir d'une configuration donnée dans un fichier. Vous ajouterez également une méthode d'écriture d'une configuration dans un flux d'écriture fichier.
5. Tester vos codes sur un programme avec des fichiers de configuration du jeu de la vie.