Programmation pour le calcul scientifique
Année : 2018-2019
Formation : L3 Ingénieurie Mathématique

TP2 : Rappels, compilation séparée, pointeurs

1 Opération sur des matrices creuses : bis

Pour cet exercice, on structurera le programme en un programme principal dans un fichier `main.cpp`, des fonctions annexes dans un fichier `.cpp` distinct, et un fichier `.h` contenant les en-têtes des fonctions.

Pour éviter les cascades de bugs, prenez l'habitude de tester chaque fonction dans le programme principal une fois que vous l'avez écrite, et ne passez à la question suivante que lorsque vous vous êtes assuré(e) qu'elle fonctionne correctement.

On reprend l'exercice du TP précédent, mais dans ce contexte de compilation séparée, en transformant les programmes écrits précédemment en fonctions. Soit A une matrice quelconque, mais qui contient de nombreux éléments nuls, par exemple une matrice tridiagonale. Nous allons effectuer des opérations sur cette matrice, en utilisant un stockage creux.

1. Créez une structure `matcreuse` qui contient un entier n, et trois tableaux (en utilisant la classe `vector`) de taille n, respectivement appelés `ligne`, composé d'entiers, `colonne`, composé d'entiers, et `valeur`, composé de réels.
2. Créez un fichier "matrice.txt" qui contient sur sa première ligne un entier n, puis sur chacune des n lignes suivantes deux entiers positifs, et un réel quelconque.
3. Créez une fonction qui lit le fichier "matrice.txt", et qui remplit une structure `matcreuse` avec les valeurs lues sur chaque ligne du fichier. Ces trois tableaux représentent respectivement les indices des lignes, les indices des colonnes, et les valeurs des éléments non-nuls d'une matrice.
4. Ecrire une fonction `trace` permettant de calculer la trace de la matrice ainsi représentée, en utilisant uniquement les tableaux `ligne`, `colonne`, et `valeur`.
5. Ecrire une fonction `norme` permettant de calculer la norme infinie de la matrice, en utilisant uniquement les tableaux `ligne`, `colonne`, et `valeur`.
6. Ecrire une fonction `matvec` permettant de faire le produit de la matrice par un vecteur `x`, en utilisant uniquement les tableaux `ligne`, `colonne`, et `valeur`.
7. Transformer la fonction `trace` en une fonction `inline`.
8. Ecrire un programme permettant, à l'aide des fonctions ainsi écrites, d'appliquer l'algorithme de la puissance itérée.

2 Pointeurs 1

Testez le code suivant :

```cpp
#include <iostream>
using namespace std;

int main()
{
   int mon_entier(23);
   //Affichage de l'adresse de la variable en utilisant &
   //Elle sera donnée en base 16 (d'où la présence de lettres)
   cout << "L'adresse est : " << &mon_entier << endl;
   return 0;
}
```

Rajouter les lignes suivantes à votre code :

```cpp
//Ce code déclare un pointeur qui peut contenir
//l'adresse d'une variable de type int.
int *pointeurInt;
//On peut faire de même pour n'importe quel type :
double const *pointeurDoubleConst;
vector<int> *pointeurVector; //etc ...
```

Pour le moment, les pointeurs ne contiennent aucune adresse connue, on ne sait donc pas quelle case de la mémoire est manipulée quand on utilise le pointeur. Pour éviter les bug, il est préférable de donner au pointeur la valeur 0. Cela signifie qu'il ne contient l'adresse d'aucune case. Modifiez le code de la manière suivante :

```cpp
//Ce code déclare un pointeur qui peut contenir
//l'adresse d'une variable de type int.
int *pointeurInt(0);
//On peut faire de même pour n'importe quel type :
double const *pointeurDoubleConst(0);
vector<int> *pointeurVector(0); //etc ...
```

Testez le code suivant et décrivez les instructions qui y sont effectuées :

```cpp
int mon_entier(23); //Une variable de type int
int *ptr(0); //Un pointeur pouvant contenir l'adresse d'un int
ptr = &mon_entier; //L'adresse de 'mon_entier' est mise dans le pointeur 'ptr'
//On dit alors que le pointeur ptr pointe sur mon_entier
cout << "L'adresse de 'mon_entier' est : " << &mon_entier << endl;
cout << "La valeur de 'ptr' est : " << ptr << endl;
cout << "La valeur est : " << *ptr << endl;
```

3 Pointeurs 2 : allocation dynamique

Rajoutez à votre code les lignes suivantes et testez les :

```
int *pointeur(0);//Définit un pointeur pouvant contenir l'adresse d'un int
pointeur = new int;//Demande 1case et renvoie un pointeur pointant vers celle-ci
*pointeur = 2;//Accède à la case mémoire pour en modifier la valeur
delete pointeur;//Libère la case mémoire
//Attention le pointeur pointe toujours mais vers une case vide !!!
pointeur = 0;//Indique que le pointeur ne pointe plus vers rien
```

4 Pointeurs 3

Ecrivez un programme qui demande l'âge de l'utilisateur, et l'affiche à l'aide d'un pointeur. Le pointeur doit être l'unique variable utilisée.

5 Pointeurs 4 : QCM

Rajoutez à votre code les lignes suivantes et testez les :

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string repA("Euler"), repB("Runge-Kutta");
  cout << "Quel schéma en temps souhaitez-vous utiliser ? " << endl;
  cout << "A) " << repA << endl;
  cout << "B) " << repB << endl;

  char votre_reponse;
  cout << "Votre reponse (A ou B) : ";
  cin >> votre_reponse; //Récupère la réponse de l'utilisateur

  string *reponseUtilisateur(0); //Un pointeur qui pointera sur la réponse

  switch(votre_reponse)
  {
  case 'A':
      reponseUtilisateur = &repA; //Déplace le pointeur sur la réponse choisie
      break;
```

```
  case 'B':
      reponseUtilisateur = &repB;
      break;
  default:
      cout << "Ce choix n'est pas valable !" << endl;
      exit(0); //Le programme s'arrête
  }
  //Utilise le pointeur pour afficher la réponse choisie
  cout << "Vous avez choisi la reponse : " << *reponseUtilisateur << endl;
  //Le pointeur n'ayant pas été défini par un "new"
  //il n'est pas nécessaire de libérer la case mémoire
  return 0;
}
```

Que se passe-t-il si on retire la ligne 28 et que l'on ne répond ni par A ni par B ? Créez votre propre QCM.