EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours d'Informatique à la section SV
Sam J.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

PROGRAMMATION I

Examen Semestre I

Instructions :

- Vous disposez de une heure quarante cinq minutes pour faire cet examen (15h15 - 17h).
- Nombre maximum de points: 75 points (+ 20 en bonus).
- Indiquez votre **NUMÉRO SCIPER** sur *chacune* des feuilles. **Une feuille sans identification ne sera pas corrigée.**
- Toute documentation est autorisée, hormis les corrigés des anciens tests ;
- Répondez sur les feuilles qui vous sont distribuées à cet effet (utilisez aussi le verso des feuilles si nécessaire). **Ne répondez pas sur l'énoncé**.
- Vous pouvez répondre aux questions en français ou en anglais.
- Veillez à **ne traiter *qu'un* exercice par feuille**.
- L'examen compte 3 exercices indépendants.

  Vous pouvez commencer par celui que vous souhaitez
  - Exercice 1: **45** points.
  - Exercice 2: **30** points.
  - Exercice 3: **20** points (bonus).

SUJET À LA PAGE SUIVANTE

Exercice 1 : Conception de programme et programmation [45 points]

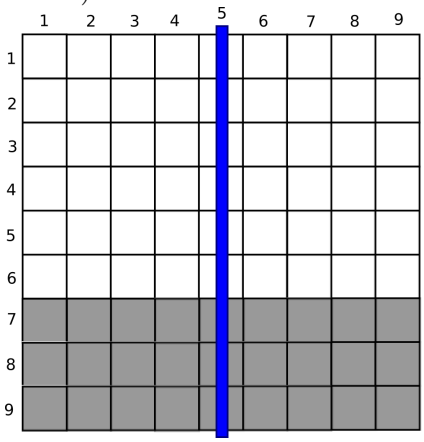
Vos talents de programmeur sont sollicités par un ami paysagiste qui souhaite que vous l'aidiez à aménager des petits jardins.

Un jardin standard est carré et peut être quadrillé en zones/cases.

Chaque jardin dispose de cases ombragées : toujours les mêmes pour un jardin donné, et occupant un certain nombre de lignes consécutives. Il n'est pas nécessaire se ce préoccuper de la raison pour laquelle les lignes consécutives de cases ombragées sont toujours les mêmes.

Chaque jardin dispose d'un tuyau d'irrigation fixe qui le traverse complètement verticalement (pas forcément au milieu du jardin).

La figure ci-dessous vous donne un exemple typique de jardin (les cases en gris sons les zones d'ombres, le tuyau d'irrigation traverse la colonne 5) :



Chaque case sera occupée par une seule plante (on supposera que toutes les plantes peuvent avoir des fleurs).

Caractéristiques des plantes

1. Chaque plante est caractérisé par un *code*, une *taille maximale* (en centimètres) et la *couleur* des fleurs. Le code et la couleur sont de simples chaînes de caratères.
2. Certaines plantes supportent l'ombre est d'autres pas.
3. Certaines plantes ont beaucoup besoin d'eau, d'autres non.

Contraintes de plantations

1. Les cases occupées par le tuyau ne sont pas plantées.
2. Les plantes qui ont beaucoup besoin d'eau doivent être plantées à proximité du tuyau d'irrigation. Les critères décidant de la proximité du tuyau ne sont pas importants ici.
3. Les plantes qui ne supportent pas l'ombre ne doivent pas être plantées en zone ombragée.
4. Un jardin est correctement aménagé si toutes ces contraintes sont respectées pour les plantes de chacune de ses cases.

SUJET À LA PAGE SUIVANTE

Question 1 : structures de données [12 points]

Quelles structures de données proposez-vous pour modéliser :

1. une plante ;
2. un jardin ;

Important : Il vous est imposé de

1. modéliser une case du jardin comme **un pointeur sur une plante** ;
2. modéliser le jardin en utilisant un tableau à deux dimensions (en utilisant le type `vector`).

   Attention : un jardin est une zone quadrillée **ainsi que** d'autres informations caractéristiques, comme la position du tuyau d'arrosage.

Vous indiquerez en commentaire le rôle de certaines données s'il n'est pas trivial à comprendre.

Question 2 : fonctionnalités [23 points]

Notre paysagiste aurait besoin des fonctionnalités suivantes pour gérer son jardin :

1. mettre une plante sur une case donnée (sans se préoccuper du fait que les contraintes précitées soient respectées) ;
2. Afficher l'aménagement du jardin sous un format tel que:

   ```
   <code> <code> *   -
   <code>   -    * <code>
     -    <code> * <code>
   <code> <code> * <code>
   ```

   (où `<code>` est le code d'une plante, le symbole `*` représente le tuyau et le symbole `-` représente une case sans plante).
3. tester si une case donnée du jardin est loin du tuyau d'irrigation (les critères exacts ne sont pas importants) ;
4. tester si une plante donnée supporte l'ombre ;
5. tester si une plante donnée a besoin de beaucoup d'eau ;
6. tester si une case donnée est à l'ombre ;
7. tester si une case donnée est correctement aménagée ;
8. tester si un jardin donné est correctement aménagé.

Donnez le prototype de ces fonctions. On ne vous demande pas d'écrire un programme complet ou le corps des fonctions mais uniquement les prototypes.

Question 3 : programmation [10 points]

Donnez le code de la fonction mettant en oeuvre la fonctionnalité 7.
Vous utiliserez les fonctionnalités prototypées sans avoir besoin d'en donner les corps.

suite au verso ☞

Exercice 2 : Questions de cours [30 points]

Répondez clairement et succinctement aux questions suivantes :

① **[5 points]** Soit le code suivant :

```
vector< vector<int>> tab(10, vector<int>(3,1));
```

Pour chacune des instructions suivantes, indiquez si elle est correcte et si oui, ce qu'elle affiche :

(a) `cout << tab[9] << endl;`

(b) `cout << tab[10] << endl;`

(c) `cout << tab[5].size() << endl;`

(d) `cout << tab[5][0] << endl;`

(e) `cout << tab[11][0] << endl;`

(f) `cout << tab[5][0].size() << endl;`

Justifiez brièvement votre réponse lorsque vous indiquez qu'une ligne est incorrecte.

② **[3 points]** Soit le code suivant :

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

int main()
{
  int t(20);
  Truc v(nullptr);
  v = &t;
  cout << *v << endl;

  return 0;
}
```

Quelle ligne de code proposez-vous d'ajouter avant le `main` pour que ce programme compile et affiche 20 ?

③ **[4 points]** Soit le code suivant :

```
constexpr int GRAND_NOMBRE(2014);

// première fonction :
bool grand(int i) {
  return i > GRAND_NOMBRE;
}

// deuxième fonction :
bool grand(int i1, int i2) {
  return i1 > i2;
}
```

(a) la seconde fonction `grand` est elle une surcharge de la première ? Justifiez brièvement ;

(b) proposer une surcharge de la première fonction ;

(c) proposer un prototype d'une fonction nommée `grand` qui ne serait pas une surcharge de la seconde.

④ **[3 points]** Le programme suivant est censé afficher `: 0 2 3 0 15 0` mais ne le fait pas.

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

int main()
{
  vector<int> v = {-1, 2, 3, -12, 15, 2 -4 };

  for (auto elt : v){
    if (elt < 0){elt = 0;}
  }
  for (auto elt : v){
    cout << elt << " ";
  }
  return 0;
}
```

Expliquez pourquoi et comment le corriger.

suite au verso ☞

⑤ **[6 points]** Qu'affiche le programme suivant :

```
#include <iostream>
#include <string>
using namespace std;
void f(string s1, string s2)
{
  s1 = s1.substr(4, 4);
  s1.insert(3, "s");
  s1.replace(s1.find("e"), 2, "val");
  s1  = "que les " + s1;
  s1+= s2;
}
void g(string& s1, string s2)
{
  s1 = s1.substr(5, 5);
  s1.insert(5, "s");
  s1.replace(s1.find("r"), 2, "mm");
  s1  = "que les " + s1;
  s1+= s2;
}
int main ()
{
  string s1("est-ce ainsi");
  string s2("cent hordes sauvages");

  f(s1, " meurent ?");
  g(s2, " vivent ?");

  cout << s1 << endl;
  cout << s2 << endl;
  return 0;
}
```

Justifiez brièvement.

Rappels :

— `s.substr(int position, int length)` retourne la sous-chaîne de `s` commençant à `position` (comprise) et de longueur `length` ;
— `s.replace(int position, int length, string other)` remplace dans `s` la séquence commençant à `position` (compris), de longueur `length` par la chaîne `other` ;
— `s.insert(int position, string other)` insère dans `s` la chaîne `other` à partir de la position `position` ;
— `s.find(string seq)` retourne la position dans `s` de la première occurence de `seq`.

⑥ **[4 points]** Le programme suivant devrait afficher : `Mon nom est personne`

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

void f(string nom) {
  cout << nom << endl;
}

int main() {
  cout << "Mon nom est  " << f("personne");
  cout << endl;
  return 0;
}
```

Il ne compile cependant pas.

Expliquez pourquoi et comment le corriger.

⑦ **[5 points]** Soit le programme suivant :

```
1.  #include <vector>
2.  #include <string>
3.  #include <iostream>
4.  using namespace std;

5.  int f(vector<int> v)
6.  {
7.    if (!v.empty()) {
8.      int sum(0);
9.      for (auto n : v) {
10.       sum += n;
11.     }
12.     return sum;
13.   }
14.    return 0;
15. }

16. int main()
17. {
18.   vector<int> v1({ -15, 3, -9, -2, 5 });
19.   vector<int> v2({ 2, 5 });
20.   cout << "calcul 1 : " << f(v1) << endl;
21.   cout << "calcul 2 : " << f(v2) << endl;
22.   return 0;
23. }
```

Quelles lignes de code ajouter et à quels endroits pour faire en sorte que le calcul d'une somme négative dans `f` provoque le lancement d'une exception rattrapée et traitée dans le `main`.

Le traitement pourra simplement consister à afficher un message indiquant le calcul d'une somme négative

suite au verso ☞

Exercice 3 : Déroulement de programme [20 points] (bonus)

Le programme suivant compile et s'exécute sans erreurs (C++11).

```
1.  #include <iostream>
2.  #include <string>
3.  #include <vector>

4.  using namespace std;

5.  typedef unsigned int J;
6.  typedef unsigned int M;
7.  typedef unsigned int A;

8.  struct D {
9.    J jj;
10.   M mm;
11.   A aa;
12. };

13. typedef vector<D> DS;

14. void e(DS& d, int a, int b){
15.   J temp;
16.   temp=d[a].jj;
17.   d[a].jj = d[b].jj;
18.   d[b].jj=temp;
19. }

20. void op(DS& d) {
21.   for (size_t i(0); i<d.size(); ++i)
22.     for (size_t j(i); j<d.size(); ++j)
23.       if (d[i].jj > d[j].jj) e(d,i,j);
24. }

25. void a(const DS& dt){
26.   for (auto v : dt){
27.      cout << v. jj << "/" << v.mm << "/";
28.      cout << "20"  << v.aa << endl;
29.    }
30.   cout << endl;
31. }
```

```
31. int main ()
32. {
33.   DS dts;

34.   cout << "Init" << endl;
35.   D dt = {25,1,17};
36.   dts.push_back(dt);
37.   dt.jj = 4;
38.   dts.push_back(dt);
39.   dt.jj = 30;
40.   dts.push_back(dt);
41.   dt.jj = 1;
42.   dts.push_back(dt);

43.   cout << "Process" << endl;
44.   op(dts);
45.   a(dts);
46.   return 0;
}
```

1. Que font les fonctions `e`, `op` et `a` ? Il ne s'agit pas ici de paraphraser le code, mais bien d'*expliquer* les étapes et le déroulement de ces fonctions.
2. Qu'affiche le programme ?