# TEST final C++ ICC MT & EL 21.12.2018 EPFL / R. Boulic

## W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous

**1) (3 pts) Entrées-sorties standards**

On désire tester le fonctionnement des entrées-sorties standards en lançant l'exécution du programme ci-dessous à plusieurs reprises.

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7        double x(-1.0);
8        int    n(-1);
9
10       do
11       {
12            cin  >> n >> x ;
13            cout << n << " "  << x << endl;
14       } while (true);
15
16       return 0;
17  }
```

1.1) Chaque exécution du programme correspond à une ligne numérotée du tableau suivant à compléter. Pour chacune de ces lignes, la colonne **Entrée** indique ce qui est frappé au clavier ; chaque ligne avec des données est validée avec la frappe de la touche ***Enter***. On demande d'indiquer dans la colonne **Sortie** ce qui est affiché sur la sortie standard **cout** en précisant brièvement pourquoi on obtient cette sortie.

*Quand le ou les affichages sont effectués on suppose qu'on arrête l'exécution courante avec la combinaison de touche Ctrl-C avant de relancer une nouvelle exécution.* [Wh]

| **Exécution** | **Entrée :** chaque ligne de donnée est validée par la frappe de la touche ***Enter*** | **Sortie** + justifiez pourquoi on obtient cette sortie |
|---|---|---|
| **N°1** | `1 2.5` | `1 2.5`<br>l'input 1 est valide pour la lecture d'un entier et 2.5 est valide pour le type double |
| **N°2** | `2.5 1` | `2 0.5`<br>seulement 2 est valide pour la lecture d'un entier ; la suite de la lecture pour x commence avec « .5 » et finit avec l'espace, ce qui donne 0.5 pour la variable x. Il n'y a pas assez de donnée dans le buffer d'entrée pour compléter un second passage dans la boucle. |
| **N°3** | `4.3 2`<br>`1 5` | `4 0.3`<br>`2 1`<br>Comme ci-dessus le premier passage donne 4 et 0.3 pour n et x, puis la lecture suivante récupère 2 pour n et 1 pour x car ces valeurs sont encore dans le buffer d'entrée et sont séparées par des séparateurs (passage à la ligne, espace). |
| **N°4** | `4.4 5.3` | `4 0.4`<br>`5 0.3`<br>Comme le cas N°2 mais deux lignes sont affichées car le premier passage donne 4 et 0.4 pour n et x, puis la lecture suivante récupère 5 pour n et 0.3 pour x. |
| **N°5** | `.3 4` | Echec dès la première lecture de n à cause du point qui n'est pas accepté pour un entier. Ce caractère reste dans le buffer d'entrée ce qui cause une boucle infinie d'affichage de valeurs incorrectes pour n et x. |

**Variantes : les valeurs sont différentes mais explications sont les mêmes**

**[B]**

| | | |
|---|---|---|
| N°1 | 2 1.5 | 2 1.5 |
| N°2 | 1.5 2 | 1 0.5 |
| N°3 | 3.4 2<br>1 5 | 3 0.4<br>2 1 |
| N°4 | 4.4 3.5 | 4 0.4<br>3 0.5 |
| N°5 | .4 3 | Boucle infinie |

**[G]**

| | | |
|---|---|---|
| N°1 | 2 5.1 | 2 5.1 |
| N°2 | 5.1 2 | 5 0.1 |
| N°3 | 4.3 1<br>2 5 | 4 0.3<br>1 2 |
| N°4 | 3.3 4.5 | 3 0.3<br>4 0.5 |
| N°5 | .4 3 | Boucle infinie |

**[Y]**

| | | |
|---|---|---|
| N°1 | 5 2.1 | 5 2.1 |
| N°2 | 2.1 5 | 2 0.1 |
| N°3 | 3.4 2<br>1 5 | 3 0.4<br>2 1 |
| N°4 | 3.3 5.4 | 3 0.3<br>5 0.4 |
| N°5 | .4 3 | Boucle infinie |

1.2) On désire remplacer la condition qui pilote la boucle do-while avec une nouvelle expression permettant de quitter cette boucle en cas *d'échec de la lecture pour la variable n à cause d'une donnée incorrecte*. Donner le code C++ ci-dessous :

```
while( not(cin.fail()));
```

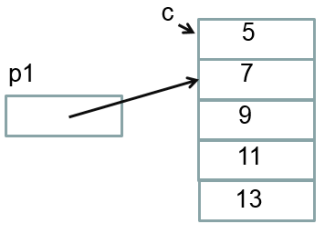
**2) (3 pts) Pointeurs et tableau à-la-C**

Le code suivant compile en C++11 mais présente un bug à l'exécution :

```
1   #include <iostream>
2
3   using namespace std;
4
5   void avance(int* ptr);
6
7   int main()
8   {
9        int c[5] = {5,7,9,11,13};
10
11       int *p1(nullptr);
12
13       p1 = c;
14       ++p1;
15
16       cout << *p1 <<  endl;
17
18       avance(p1);
19
20       cout << *p1 << endl;
21
22       return 0;
23  }
24
25  void avance(int* ptr)
26  {
27      ptr = ptr + 2;
28  }
```

2.1) Dessiner avec le formalisme **boîte + flèche** l'état du pointeur **p1** et du **tableau c** lorsque les instructions jusqu'à la ligne 14 (comprise) ont été exécutées (et pas plus).

Dessin de l'état du pointeur p1 et du tableau c =>




2.2) Quel est l'affichage produit par la ligne **16**, et pourqu

La ligne 16 envoie sur cout la valeur de l'élément de c pointé par p1, c'est-à-dire c[1], c'est-à-dire la valeur **7**. L'affichage est immédiat car il y a **endl** qui force cet affichage au lieu de conserver la valeur dans le buffer de sortie.

2.3) Quel est l'affichage produit par la ligne **20** ? Justifiez votre réponse en expliquant ce qui est fait par l'appel de la fonction avance() à la ligne 18.

L'affichage est EXACTEMENT LE MEME que pour la question précédente car la valeur du

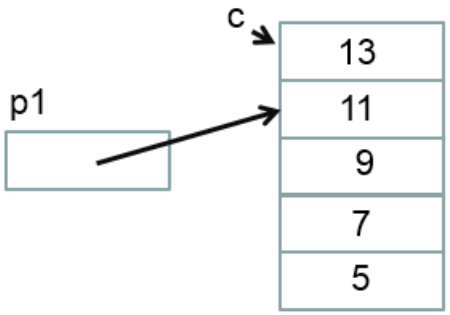
pointeur p1 n'a pas été modifiée par l'appel de la fonction **avance**() : il y a affichage immédiat de la valeur **7**.

En effet il s'agit d'un passage par valeur qui modifie la valeur du paramètre formel **ptr** sans aucune répercussion sur la valeur de p1. Il aurait fallu faire une version de la fonction avance() avec un passage par référence de p1.

**Variantes : les valeurs du tableau c sont différentes à la ligne 9, ce qui produit un affichage différent pour les questions 2.1 à 2.3 mais le comportement est le même**

**[B]**

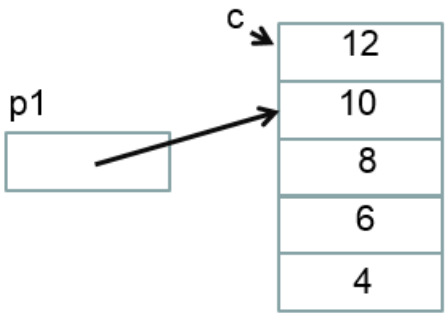
```
int c[5] = {13,11,9,7,5};
```




**Affiche 11 au lieu de 7**

**[G]**

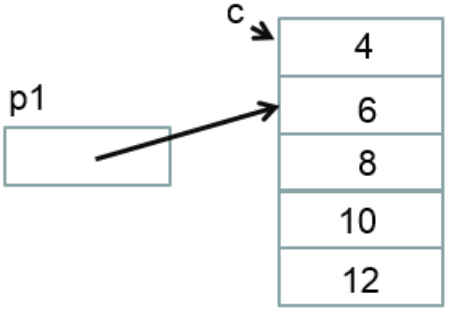
```
int c[5] = {12,10,8,6,4};
```




**Affiche 10 au lieu de 7**

**[Y]**

```
int c[5] = {4,6,8,10,12};
```




Affiche 6 au lieu de 7

**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

**3) (4 pts) pointeur : exécution pas à pas**

Soit les déclarations suivantes

```
1
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7        int a(0) ;
8        int b(2);
9        int c(4);
10       int* p1(nullptr);
11       int* p2(nullptr);
12
13       // les instructions sont dans
14       // le tableau à compléter
15
16       return 0;
17  }
18
```

Compléter le tableau suivant en indiquant la valeur des variable **APRES** l'exécution de l'instruction indiquée dans la colonne de gauche. *Les cases grises doivent rester vides*.

Attention, les instructions sont exécutées l'une après l'autre comme dans un programme ; il faut prendre en compte les éventuelles modifications des variables par les instructions qui précèdent.

| instruction | a | b | c | *p1 | *p2 |
|---|---|---|---|---|---|
| Ligne 11 après les initialisations | 0 | 2 | 4 | | |
| p1 = &a ; | 0 | 2 | 4 | 0 | |
| p2 = &c ; | 0 | 2 | 4 | 0 | 4 |
| (*p1)=++(*p2); | 5 | 2 | 5 | 5 | 5 |
| p1 = p2 ; | 5 | 2 | 5 | 5 | 5 |
| p2 = &b ; | 5 | 2 | 5 | 5 | 2 |
| *p1-=*p2 ; | 5 | 2 | 3 | 3 | 2 |
| ++*p2 ; | 5 | 3 | 3 | 3 | 3 |
| *p1*=*p2 ; | 5 | 3 | 9 | 9 | 3 |
| a=++*p2**p1 ; | 36 | 4 | 9 | 9 | 4 |

W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous

**Variantes : les valeurs initiales des variables a, b et c sont différentes aux lignes 7-9, mais les instructions sont les mêmes**

**[B]**

```
int a(2) ;
int b(0);
int c(4);
```

| instruction | a | b | c | *p1 | *p2 |
|---|---|---|---|---|---|
| Ligne 11 après les initialisations | 2 | 0 | 4 | | |
| p1 = &a ; | 2 | 0 | 4 | 2 | |
| p2 = &c ; | 2 | 0 | 4 | 2 | 4 |
| (*p1)=++(*p2); | 5 | 0 | 5 | 5 | 5 |
| p1 = p2 ; | 5 | 0 | 5 | 5 | 5 |
| p2 = &b ; | 5 | 0 | 5 | 5 | 0 |
| *p1-=*p2 ; | 5 | 0 | 5 | 5 | 0 |
| ++*p2 ; | 5 | 1 | 5 | 5 | 1 |
| *p1*=*p2 ; | 5 | 1 | 5 | 5 | 1 |
| a=++*p2**p1 ; | 10 | 2 | 5 | 5 | 2 |

**[G]**

```
int a(2) ;
int b(4);
int c(0);
```

| instruction | a | b | c | *p1 | *p2 |
|---|---|---|---|---|---|
| Ligne 11 après les initialisations | 2 | 4 | 0 | | |
| p1 = &a ; | 2 | 4 | 0 | 2 | |
| p2 = &c ; | 2 | 4 | 0 | 2 | 0 |
| (*p1)=++(*p2); | 1 | 4 | 1 | 1 | 1 |
| p1 = p2 ; | 1 | 4 | 1 | 1 | 1 |
| p2 = &b ; | 1 | 4 | 1 | 1 | 4 |
| *p1-=*p2 ; | 1 | 4 | -3 | -3 | 4 |
| ++*p2 ; | 1 | 5 | -3 | -3 | 5 |
| *p1*=*p2 ; | 1 | 5 | -15 | -15 | 5 |
| a=++*p2**p1 ; | -90 | 6 | -15 | -15 | 6 |

**[Y]**

```
int a(0) ;
int b(4);
int c(2);
```

| instruction | a | b | c | *p1 | *p2 |
|---|---|---|---|---|---|
| Ligne 11 après les initialisations | 0 | 4 | 2 | | |
| p1 = &a ; | 0 | 4 | 2 | 0 | |
| p2 = &c ; | 0 | 4 | 2 | 0 | 2 |
| (*p1)=++(*p2); | 3 | 4 | 3 | 3 | 3 |
| p1 = p2 ; | 3 | 4 | 3 | 3 | 3 |
| p2 = &b ; | 3 | 4 | 3 | 3 | 4 |
| *p1-=*p2 ; | 3 | 4 | -1 | -1 | 4 |
| ++*p2 ; | 3 | 5 | -1 | -1 | 5 |
| *p1*=*p2 ; | 3 | 5 | -5 | -5 | 5 |
| a=++*p2**p1 ; | -30 | 6 | -5 | -5 | 6 |

**4) (5 pts) Généricité des pointeurs :**

Le code fourni dans les pages suivantes compile avec C++11. Dans cet exercice il s'agit de compléter 5 fonctions dont le prototype est fourni. Les commentaires avant chaque fonction indiquent leur but et donnent une indication de la taille de la fonction.

Les noms des types, des champs des structures et des noms des fonctions devraient suffire pour comprendre le but du programme. Voici néanmoins un résumé :

- Le type **Student** mémorise un identificateur numérique **sciper** pour un(e) étudiant(e) et utilise un **vector** de pointeurs vers des structures **Course** pour mémoriser la liste des cours suivis par cette personne. Quatre variables de ce type sont déclarées et initialisées dans les lignes 30-33 de la page suivante.

- Le type **Course** mémorise le titre du cours dans le champ **title** et utilise un **vector** de pointeurs vers des structures **Student** pour mémoriser la liste des étudiants qui suivent ce cours. Quatre variables de ce type sont déclarées et initialisées dans les lignes 35-38 de la page suivante.

- La fonction **update_course_and_student()** doit permettre de mettre à jour les deux structures **Student** et **Course** en ajoutant le cours à la liste des cours suivis par l'étudiant et en ajoutant l'étudiant à la liste des étudiants qui suivent ce cours. Cette fonction va tirer parti de quatre autres fonctions qu'il faut écrire (questions 4.1 à 4.4) avant d'écrire cette fonction (4.5). la fonction main() donne des exemples d'appels corrects de cette fonction.

```
1   #include <iostream>
2   #include <vector>
3   #include <string>
4   using namespace std;
5
6   struct Course ;
7   struct Student ;
8
9   struct Course{
10      string title;
11      vector<const Student*> audience;
12  } ;
13  struct Student{
14      unsigned int sciper;
15      vector<const Course*> cursus;
16  } ;
17
18  void update_course_and_student(Course& c, Student& s);
19  void add_course(Student& s, const Course* pc) ;
20  void add_student(Course& c, const Student* ps) ;
21  bool belong_to_cursus(const Student& s,  const Course* pc);
22  bool belong_to_audience(const Course& c, const Student* ps);
23  void display(Course  c);
24  void display(Student s);
25
26  int main()
27  {
28      Student julie= {123456, {}};
29      Student malik= {234567, {}};
30      Student konrad={345678, {}};
31      Student sophie={456789, {}};
32
33      Course ch_505 ={"cuisine moleculaire",{}};
34      Course mgt_100={"startup revisited",{}};
35      Course ph_777 ={"quantum vibes",{}};
36      Course sc_123 ={"crypto trip",{}};
37
38      update_course_and_student(ph_777,sophie);
39      update_course_and_student(sc_123,sophie);
40      update_course_and_student(ch_505,sophie);
41      update_course_and_student(ph_777,malik);
42      update_course_and_student(mgt_100,malik);
43      update_course_and_student(ph_777,konrad);
44      update_course_and_student(mgt_100,julie);
45
46      display(sophie) ;
47      cout << endl;
48      display(ph_777) ;
49      return 0;
50  }
51  void display(Student s)
52  {
53      cout << "Cursus of student: " << s.sciper << endl;
54      for (size_t i(0); i < s.cursus.size() ; i++)
55          cout << s.cursus[i]->title << endl;
56  }
57  void display(Course  c)
58  {
59      cout << "Audience of course: " << c.title << endl;
60      for (size_t i(0); i < c.audience.size() ; i++)
61          cout << c.audience[i]->sciper << endl;
62  }
63  // suite du code source à la page suivante
```

Compléter le code source ci-dessous selon les indications données en commentaires

```
1   // 4.1 renvoie true si pc fait déjà partie du cursus
2   //     de l'étudiant s, et renvoie false sinon.
3   // → 3 lignes de code
4   bool belong_to_cursus(const Student& s, const Course* pc){
5
6       for (size_t i(0); i < s.cursus.size() ; i++)
7           if(s.cursus[i] == pc) return true;
8
9       return false;
10
11  }
12
13  // 4.2 renvoie true si ps fait déjà partie de l'audience
14  //     du cours c, et renvoie false sinon.
15  // → 3 lignes de code
16  bool belong_to_audience(const Course& c, const Student* ps){
17
18      for (size_t i(0); i < c.audience.size() ; i++)
19          if(c.audience[i] == ps) return true;
20
21      return false;
22
23
24  }
25
26  // 4.3 ajoute le pointeur pc au cursus de l'étudiant s SEULEMENT
27  //     si ce pointeur n'est pas déjà présent dans son cursus.
28  // → utiliser une des fonctions précédentes pour le test
29  // → 2 lignes de code
30  void add_course(Student& s, const Course* pc){
31
32        if(not belong_to_cursus(s,pc))
33              s.cursus.push_back(pc);
34
35  }
36
37  // 4.4 ajoute le pointeur ps à l'audience du cours c SEULEMENT
38  //     si ce pointeur n'est pas déjà présent dans son audience.
39  // → utiliser une des fonctions précédentes pour le test
40  // → 2 lignes de code
41  void add_student(Course& c, const Student* ps){
42
43        if(not belong_to_audience(c,ps))
44              c.audience.push_back(ps);
45
46  }
47
48  // 4.5 utilise les fonctions précédentes pour ajouter le cours c
49  //     à l'étudiant s et l'étudiant s au cours c
50  // → 2 lignes de code
51  void update_course_and_student(Course& c, Student& s){
52
53        add_course(s,  &c);
54        add_student(c, &s);
55
56  }
```

**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

### 5) (4 pts) programme mystère (sans pointeur)

Le programme suivant compile avec C++11 et s'exécute correctement.

5.1) Qu'affiche-t-il ? [Wh]

**1 2**
**2 4**
**3 3**
**4 1**

5.2) quelle est le but de la fonction f() ?
Si la valeur **v** se trouve déjà dans le vector **val** alors l'élément de même indice du vector **nb** est incrémenté et la fonction renvoie FAUX. Sinon elle renvoie VRAI et (optionnel) main() est chargée d'ajouter **v** à **tabv** et d'initialiser son compteur à **1** dans **tabc**.

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   bool f(int v, vector<int>& nb, const vector<int>& val);
7   void g(const vector<int>& cnb, const vector<int>& val);
8
9   int main()
10  {
11        vector<int> list({1,2,2,3,3,3,4,2,2,1});
12        vector<int> tabc;
13        vector<int> tabv;
14
15        for (size_t i(0); i< list.size() ;++i)
16        {
17              if (f(list[i],tabc,tabv))
18              {
19                    tabv.push_back((list[i]));
20                    tabc.push_back(1);
21              }
22        }
23        g(tabc,tabv);
24
25        return 0;
26  }
27
28  bool f(int v, vector<int>& nb, const vector<int>& val)
29  {
30        bool s(1);
31
32        for (size_t i(0);i<val.size(); i++)
33        {
34              if (val[i]== v)
35              {
36                    ++nb[i];
37                    s=0;
38              }
39        }
40        return s;
41  }
42
43
44  void g(const vector<int>& nb, const vector<int>& val)
45  {
46        for (size_t i(0); i< nb.size() ; i++)
47        {
48              cout << val[i] << "     " << nb[i] << endl;
49        }
50  }
```

**Variantes de la ligne 11 du code et réponse 5.1:**

**[B]**

```
vector<int> list({1,2,3,3,3,3,4,2,2,1});
```

5.1) Qu'affiche-t-il ?

**1 2**
**2 3**
**3 4**
**4 1**

**[G]**

```
vector<int> list({1,2,3,3,3,2,4,2,2,1});
```

5.1) Qu'affiche-t-il ?

**1 2**
**2 4**
**3 3**
**4 1**

**[Y]**

```
vector<int> list({2,2,3,3,3,2,4,4,2,1});
```

5.1) Qu'affiche-t-il ?

**2 4**
**3 3**
**4 2**
**1 1**