ICC MT & EL Test final C++ /White vendredi 21 décembre 2018

Ne PAS retourner ces feuilles avant d'en être autorisé!

Merci de poser votre carte CAMIPRO en évidence sur la table.

Vous pouvez déjà compléter et lire les informations ci-dessous:

NOM _____

Prénom _____

Numéro SCIPER _____

Signature _____

BROUILLON : Ecrivez aussi votre NOM-Prénom sur la feuille de brouillon fournie. Toutes vos réponses doivent être sur cette copie d'examen. Les feuilles de brouillon sont ramassées pour être immédiatement détruites.

Le test écrit commence à:	**14h15**
Nous recommandons de passer à l'autre examen à :	**15h30**
Les deux copies d'examens sont ramassées à :	**16h40**

Le contrôle final de ICC
reste SANS appareil électronique

Vous avez le droit d'avoir tous vos documents **personnels** sous forme papier: dictionnaire, livres, cours, exercices, code, projet, etc...

Vous pouvez utiliser un crayon à papier et une gomme

Ce contrôle écrit de C++ permet d'obtenir **19 points** sur un total de 100 points pour le cours complet.

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.

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`	
N°2	`2.5 1`	
N°3	`4.3 2` `1 5`	
N°4	`4.4 5.3`	
N°5	`.3 4`	

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(……);
```

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).

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

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.

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					
p1 = &a ;					
p2 = &c ;					
(*p1)=++(*p2);					
p1 = p2 ;					
p2 = &b ;					
*p1-=*p2 ;					
++*p2 ;					
p1=*p2 ;					
a=++*p2**p1 ;					

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
2   #include <iostream>
3   #include <vector>
4   #include <string>
5   using namespace std;
6
7   struct Course ;
8   struct Student ;
9
10  struct Course{
11      string title;
12      vector<const Student*> audience;
13  } ;
14
15  struct Student{
16      unsigned int sciper;
17      vector<const Course*> cursus;
18  } ;
19
20  void update_course_and_student(Course& c, Student& s);
21  void add_course(Student& s, const Course* pc) ;
22  void add_student(Course& c, const Student* ps) ;
23  bool belong_to_cursus(const Student& s,  const Course* pc);
24  bool belong_to_audience(const Course& c, const Student* ps);
25  void display(Course  c);
26  void display(Student s);
27
28  int main()
29  {
30      Student julie= {123456, {}};
31      Student malik= {234567, {}};
32      Student konrad={345678, {}};
33      Student sophie={456789, {}};
34
35      Course ch_505 ={"cuisine moleculaire",{}};
36      Course mgt_100={"startup revisited",{}};
37      Course ph_777 ={"quantum vibes",{}};
38      Course sc_123 ={"crypto trip",{}};
39
40      update_course_and_student(ph_777,sophie);
41      update_course_and_student(sc_123,sophie);
42      update_course_and_student(ch_505,sophie);
43      update_course_and_student(ph_777,malik);
44      update_course_and_student(mgt_100,malik);
45      update_course_and_student(ph_777,konrad);
46      update_course_and_student(mgt_100,julie);
47
48      display(sophie) ;
49      cout << endl;
50      display(ph_777) ;
51      return 0;
52  }
53  void display(Student s)
54  {
55      cout << "Cursus of student: " << s.sciper << endl;
56      for (size_t i(0); i < s.cursus.size() ; i++)
57          cout << s.cursus[i]->title << endl;
58  }
59  void display(Course  c)
60  {
61      cout << "Audience of course: " << c.title << endl;
62      for (size_t i(0); i < c.audience.size() ; i++)
63          cout << c.audience[i]->sciper << endl;
64  }
65  // 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
7
8
9
10
11
12
13  }
14
15  // 4.2 renvoie true si ps fait déjà partie de l'audience
16  //     du cours c, et renvoie false sinon.
17  // → 3 lignes de code
18  bool belong_to_audience(const Course& c, const Student* ps){
19
20
21
22
23
24
25
26
27  }
28
29  // 4.3 ajoute le pointeur pc au cursus de l'étudiant s SEULEMENT
30  //     si ce pointeur n'est pas déjà présent dans son cursus.
31  // → utiliser une des fonctions précédentes pour le test
32  // → 2 lignes de code
33  void add_course(Student& s, const Course* pc){
34
35
36
37
38
39  }
40
41  // 4.4 ajoute le pointeur ps à l'audience du cours c SEULEMENT
42  //     si ce pointeur n'est pas déjà présent dans son audience.
43  // → utiliser une des fonctions précédentes pour le test
44  // → 2 lignes de code
45  void add_student(Course& c, const Student* ps){
46
47
48
49
50
51  }
52
53  // 4.5 utilise les fonctions précédentes pour ajouter le cours c
54  //     à l'étudiant s et l'étudiant s au cours c
55  // → 2 lignes de code
56  void update_course_and_student(Course& c, Student& s){
57
58
59
60
61
62  }
63
```

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

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

5.1) Qu'affiche-t-il ?

5.2) quelle est le but de la fonction f() ?

```
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  }
```