UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Corrigé du Contrôle Continu #1

`http://dpnc.unige.ch/~bravar/C++2015/CC1`

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

1 Questions

1. **Quelle est la différence entre les opérateurs = et == ?**
   L'opérateur `=` est l'opérateur d'affectation. Il est utilisé pour affecter des valeurs aux variables, p.ex. `a = 5`.
   L'opératuer logique `==` est utilisé pour comparer deux variables dans des expressions logiques ou conditions, p.ex. `if (a == b) ...` . Si les valeurs de ces deux variables sont égaux, le résultat de l'opération est `true`, si elles sont différentes, le résultat est `faux`.
2. **Où sont les erreurs dans le morceau de code suivant (en a au moins 5) ?**

```
int maFonction(double x, double y),
int main() {
  double b = maFonction(a);
  return 0;
}
void maFonction(double x, int y) {
return (4*x*y); };
```

   Les erreurs sont les suivants :
   - il doit y avoir un point-virgule et non pas une virgule à la fin de la déclaration de la fonction ;
   - la variable `a` dans `main` n'a pas été déclarée ;
   - il faut passer deux variables de type `double` à la fonction `maFonction` ;
   - la fonction renvoie une valeur de type `int` et non pas `void` ;
   - la liste des paramètres dans la définition de la fonction `maFonction` ne correspond pas à sa déclaration (variables de type différent) ;
   - il n'y a pas de point virgule à la fin de la définition de la fonction ;

   Version correcte :

```
int maFonction(double x, double y);
int main() {
  double a, y;
  double b = maFonction(a,y);
  return 0;
}
double maFonction(double x, double y) {
return (4*x*y); }
```

3. **Déterminez si les deux expressions sur chaque ligne sont équivalentes ou pas !**

   1. `!(p || q)` n'est pas équivalente à `!p || !q` :

$!(p \,\|\|\, q)$				
p	v	v	f	f
q	v	f	v	f
$p \,\|\|\, q$	v	v	v	f
$!(p \,\|\|\, q)$	f	f	f	v

$!p \,\|\|!q$				
p	v	v	f	f
q	v	f	v	f
$!p$	f	f	v	v
$!q$	f	v	f	v
$!p \,\|\|\, !q$	f	v	v	v

   2. `!!!p` n'est pas équivalente à `p` : deux négations consécutives (`!!`) s'annule, trois négations consécutives (`!!!`) correspondent à une négation (`!`).

   3. `!p && !q` n'est pas équivalente à `p && !q` :

!p && !q				
p	v	v	f	f
q	v	f	v	f
!p	f	f	v	v
!q	f	v	f	v
!p && !q	f	f	f	v

p && !q				
p	v	v	f	f
q	v	f	v	f
!q	f	v	f	v
p && !q	f	v	f	f

4. **Réécrivez le code suivant avec un boucle for :**

```
int i = 0;
do {
  cout << "Hello World" << endl;
  ++i;
} while (i<10);
```

   Réponse :

```
for (int i=0; i<10; i++)
  cout << "Hello World << endl;
```

5. **Trouvez les erreurs dans le morceau de code suivant :**

```
for (int i=0, i<10, i++) {
  cout << i << endl;
}
```

   Les trois champs dans l'instruction `for` doivent être séparés par des points-virgules.
   Version correcte :

```
for (int i=0; i<10; i++) {
  cout << i << endl;
}
```

6. **Pourquoi utiliserais-t-on le passage par référence ?**
   Dans le *passage par valeur* l'on envoie à la fonction les valeurs des variables et non pas les variables. Donc on ne peut pas modifier ces variables. En plus la fonction peut renvoyer une seule valeur. Pour remédier a on utilise le *passage par référence*. Dans ce cas, on envoie à la fonction des références aux variables dans le programme qui appelle la fonction et les variables deviennent modifiables par la fonction.

7. **Qu'est-ce que le programme suivant affichera ? Et pourquoi ?**

```
int i = 0;
if (i++) cout << "c'est vrai ?";
else cout << "ou faux ?";
```

   Le programme affichera "ou faux ?".
   Dans la condition `if` d'abord on vérifie la valeur de la variable `i`. Etant donné que la valeur de `i` est `0`, le programme considère la condition comme fausse. Seulement après avoir vérifié la condition on addition 1 à `i`. Si par contre on avais écrit `++i`, d'abord on aurait additionné 1 à `i` puis vérifié la condition. Dans ce cas la condition aurait étai vrai, étant donné que la valeur de `i` est `1`.

8. **Les variables x et y ont été déclarées de type entier int. Quel est le résultat de l'expression `( x - x / y * y )` ? Quel opérateur peut-on utiliser pour accomplir cette opération plus facilement ?**
   Cette expression calcule le reste de la division de `x` par `y`. On peut obtenir le même résultat avec l'opérateur modulo `%` : `x % y`.

2 Exercices

Exercice 1 : Boucles
Ecrivez un programme, qui utilise la boucle `for` pour additionner les premiers 10 nombres (de 1 à 10) et la boucle `while` pour calculer le produit des nombres de 1 à 10. Puis, affichez les résultats à l'écran.
Voici le programme ! La variable `somme` doit être initialisée à zéro et la variable `produit` à un.
Boucles.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main () {
6    int somme = 0;
7    for (int i=1; i<=10; i++)
8      somme += i;       //somme = somme + i;
9    cout << "La somme des 10 premiers entiers est : " << somme << endl;
10
11   int produit = 1;
12   int j = 1;
13   while (j<=10) {
14     produit *= j;       //produit = produit * j;
15     j++;
16   }
17   cout << "Le produit des 10 premiers entiers est : " << produit << endl;
18
19   return 0;
20 }
```

Exercice 2 : Ecrivez un programme, qui demande à l'utilisateur d'entrer deux nombres entiers par le clavier. Ajoutez au programme une fonction, qui prend ces deux nombres comme arguments et qui renvoie le nombre le plus petit. Ensuite, modifiez le programme : le programme demande à l'utilisateur d'entrer 10 nombres entiers et à la fin affiche à l'écran le nombre le plus petit.
Dans notre programme on utilise une boucle `for` pour saisir les 10 nombres. A chaque itération on compare le nombre saisi avec le nombre plus petit déterminé dans l'itération précédente. Si ce nombre est plus petit, on remplace le nombre plus petit déterminé précédemment avec ce-ci. Pour trier les deux nombres on utilise la fonction `petit`.
Petit.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  int petit (int, int);
6
7  int main () {
8    cout << "Entrez le premier entier    : ";
9    int min;
10   cin >> min;
11   for (int i=2; i<=10; i++) {
12     cout << "Entrez le prochain entier  : ";
13     int k;
14     cin >> k;
15     min = petit(k,min);
```

```
16    }
17    cout << "L'entier le plus petit est : " << min;
18
19    return 0;
20  }
21
22  int petit (int a, int b) {
23    if (a<b) return a;
24    else return b;
25  }
```

Exercice 3 : Nombres de Mersenne
Ecrivez une fonction correspondant au prototype `int premier (int n)` qui vérifie si le nombre entier n est premier. Les nombres de Mersenne sont définis par $M_n = 2^n - 1$ où n est un entier. Ecrivez un programme, qui affiche à l'écran les nombres de Mersenne parmi les premiers 25 ($1 \leq n \leq 25$) qui sont premiers. Utilisez la fonction `premier`.
Afin de vérifier si n est un nombre premier, il faut voir s'il y a des diviseurs d entre 2 et n. Il suffit en fait de rechercher les diviseurs parmi les nombres impairs (si le nombre est pair, il est divisible par 2 et donc il ne s'agit pas d'un nombre premier) inférieurs à $\sqrt{n}$ (si un diviseur entier d de n existe, $n = d \cdot q$ et $d \leq \sqrt{n}$, $q \geq \sqrt{n}$, tandis que si aucun $d \leq \sqrt{n}$ n'a pas été trouvé, il est impossible de trouver deux valeurs $d > \sqrt{n}$ et $q > \sqrt{n}$ telles que $d \cdot q = n$).
Dans notre programme on utilise la fonction `int premier (int)` pour vérifier si un nombre donné est premier ou pas. La fonction renvoie `1` si le nombre est premier, dans le cas contraire la fonction renvoie `0`. On aurait pu utiliser une variable booléenne pour indiquer, si le nombre est premier ou pas. Dans ce cas le prototype de la fonction serait `bool premier (int n)`. Dans la fonction `premier` on considère d'abord trois cas particulier :
- le nombre 1 n'est pas premier,
- le nombre 2 est premier,
- tous les nombres paires > 2 ne sont pas premiers.

Puis on recherche les diviseurs du nombre donné entre 3 et $\sqrt{n}$ avec une boucle `for`. Si un diviseur a été trouvé, on sort de la boucle et au même temps on revient de la fonction avec l'instruction `return`.
Dans le programme principale (`main`) on utilise une autre boucle `for` pour parcourir tous les nombres de Mersenne $M_n = 2^n - 1$ pour $1 \leq n \leq 25$ et vérifier, si ils sont premiers avec la fonction `premier`.
Mersenne.cpp

```
1   #include <iostream>
2   #include <cmath>
3
4   using namespace std;
5
6   //cette fonction verifie si n est premier
7   int premier (int);
8
9   int main() {
10    int mersenne;
11    //boucle sur les nombres de Mersenne de 2 a 25
12    for (int n=1; n<=25; n++) {
13      mersenne = pow(2,n) - 1;
14      int reponse = premier (mersenne);
```

```
15      if (reponse > 0) cout << "Le " << n << "eme nombre de Mersenne " <<
            mersenne << " est premier" << endl;
16      else cout << "Le " << n << "eme nombre de Mersenne " << mersenne << " n
            'est pas premier" << endl;
17    }
18
19    return 0;
20  }
21
22  int premier (int n) {
23    if (n == 1) return 0;    //1 n'est pas premier
24    if (n == 2) return 1;    //2 est premier
25    if (n%2 == 0) return 0;    //nombre pair > 2
26    //il suffit de verifier tous les impairs < sqrt(n)
27    int nMax = sqrt(double(n));
28    for (int i=3; i<=nMax; i+=2)
29      if (n%i == 0) return 0;    //j'ai trouve un diviseur
30    return 1;
31  }
```