ICC MT & EL Test C++ /White mardi 19 janvier 2021

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: **16h15**

Nous recommandons de consacrer 1h20 à l'examen de C++ et 1h40 à l'examen théorique

Les deux copies d'examens sont ramassées à : **19h15**

Le contrôle 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, notes manuscrites, etc...

Vous pouvez utiliser un crayon à papier et une gomme

Ce contrôle écrit de C++ permet d'obtenir **38 points** sur un total de 100 points pour le cours complet.

1) (8 pts) structure de contrôle, opérateurs divers et récursivité

Le code suivant compile en C++11 et s'exécute correctement.

```
1   #include <iostream>
2   using namespace std;
3
4   int p ( int n, int k )
5   {
6       if(n<k)
7           return 0;
8       else if( n == k || k == 1 )
9            return 1;
10      else
11          return p( n-1, k-1) + p( n-k, k ) ;
12  }
13
14  int my_function ( int n )
15  {
16      int resultat(0), i(0);
17
18      while ( i++ < n )
19          resultat += p ( n, i );
20
21      return resultat;
22  }
23
24  int main ()
25  {
26      cout << my_function( 4 ) << endl;
27      return 0;
28  }
```

1.1.1) Donner d'abord le **nombre de passages dans la boucle** des lignes 18 et 19

1.1.2) Donner la **liste des appels de la fonction p** à la ligne 19 (sans les appels récursifs). Montrer la valeur des arguments. Ex : p(33,124), p(76,21), ….
Remarque : le détails de l'évaluation des appels de p() est pour la question suivante.

1.2) **Evaluer chaque appel de p** identifié à la question précédente. Préciser les résultats intermédiaires, en particulier les appels récursifs ; si le cas se présente, vous pouvez ré-utiliser la valeur des appels récursifs que vous avez déjà expliquée.

1.3) A partir des résultats de la question précédente, justifier ce qui est affiché par le programme en détaillant les calculs intermédiaires effectués dans **my_function**.

==========

2) (15 pts) structuration des données avec vector et struct, surcharge des fonctions

Le code fourni pour cet exercice compile en C++11 et s'exécute correctement.

On désire calculer les durées totales de communication avec différents numéros de téléphones en distinguant les appels sortants (*outgoing*) des appels entrants (*incoming*). Pour cela on a défini plusieurs structures dans le code de la page suivante :

- **Date** : mémorise une date du calendrier avec les champs **day, month, year**
- **Call** : mémorise un appel téléphonique avec les champs :
  - **number** : un numéro de téléphone représenté avec le type string
  - **date** : la date de l'appel
  - **duration** : la durée en minutes
  - **direction** : type de l'appel avec un seul caractère
    - sortant : le caractère est `'O'` comme *Outgoing*
    - entrant : le caractère est `'I'` comme *Incoming*
- **Stats** : mémorise les durées pour un numéro de téléphone avec les champs :
  - **number** : un numéro de téléphone
  - **outgoing** : durée en minutes de tous les appels sortants pour ce numéro
  - **incoming**: durée en minutes de tous les appels entrants pour ce numéro

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4
5   using namespace std;
6
7   struct Date {
8      unsigned int day;
9      unsigned int month;
10     unsigned int year;
11  };
12
13  struct Call {
14     string number;
15     Date date;
16     unsigned int duration;
17     char direction; // either 'I' for Incoming or 'O' for Outgoing
18  };
19
20  struct Stats {
21     string number;
22     unsigned int outgoing;
23     unsigned int incoming;
24  };
25
26  void display(const Date  &date);      // Question 2.1
27  void display(const Call  &call);      // Question 2.1
28  void display(const Stats &stat);      // Question 2.1
29
30  int find_number(const string& num, const vector<Stats>& vec); //Q2.2
31  void process_call(const Call& call, vector<Stats>& vec);      //Q2.3
32
33  int main()
34  {
35     vector<Call> calls ( {
36      {"0771234567", {4,   1, 2021}, 20, 'I'},
37      {"0798765432", {20, 12, 2020}, 43, 'I'},
38      {"0771234567", {3,   1, 2021}, 89, 'O'},
39      {"0771234567", {2,   4, 2020}, 37, 'O'}
40     });
41
42     for (Call c: calls)
43        display(c);
44
45     vector<Stats> stats;
46
47     for (Call c: calls)
48        process_call(c, stats);
49
50     for (Stats s: stats)
51        display(s);
52
53     return 0;
54  }
```

Après la déclaration des structures on trouve la déclaration des prototypes de fonctions aux lignes 26 à 31. Il faudra écrire le code de ces fonctions (questions 2.1 à 2.3). Mais avant cela, prenez le temps d'examiner la fonction principale qui initialise un **vector** de structures **Call** (lignes 35 à 40) puis on trouve 3 boucles et une déclaration:

- Boucle d'**affichage** de chaque structure **Call**
- Déclaration de `stats` le vector de **Stats** qui est initialement vide (ligne 45)
- Boucle de **traitement** de chaque structure **Call** pour construire `stats`
- Boucle d'**affichage** de chaque structure **Stats**

2.1) Ecrire le code des fonctions **display**

2.1.1) La première fonctions **display** (ligne **26**) doit être utilisée par la seconde (ligne **27**) pour produire un affichage comme celui illustré ici pour la première structure **Call** (ligne 36) :

```
Number: 0771234567, date: 4/1/2021, duration: 20, direction: I
```

```
55 // répartir la ou les instructions d'affichage sur
56 // plusieurs lignes pour une bonne lisibilité
57 void display(const Date &date) // environ 2 lignes
58 {
59
60
61
62
63 }
64 // Cette fonction doit utiliser la précédente.
65 // Terminer l'affichage par un passage à la ligne
66 void display(const Call &call) // environ 4 lignes
67 {
68
69
70
71
72
73
74
75
76
77
78 }
```

2.1.2) la dernière fonction **display** (ligne **28**) affiche une structure **Stats** sur une seule ligne dans le terminal et termine par un passage à la ligne comme dans l'exemple illustré ci-dessous :

```
Number: 0771234567, outgoing minutes: 126, incoming minutes: 20
```

```
79 void display(const Stats &stat) // environ 3 lignes
80 {
81
82
83
84
85
86
87
88 }
```

2.2) Ecrire le code de la fonction **find_number** qui renvoie ***l'indice*** de l'élément du vector **vec** qui contient un numéro de téléphone égal au paramètre **num**. Si le numéro de téléphone recherché **num** n'est pas dans un des éléments de **vec** alors cette fonction doit renvoyer la valeur entière **-1**. Quatre à huit lignes suffisent pour cette fonction.

```
89  int find_number(const string& num, const vector<Stats>& vec)
90  {
91
92
93
94
95
96
97
98
99
100
101
102 }
```

2.3) Ecrire le code de la fonction **process_call** qui met à jour le vector **vec** en analysant l'appel **call**. Il faut utiliser la fonction **find_number** écrite à la question précédente pour déterminer si le numéro de **call** est déjà dans un élément du vector **vec**. S'il n'est pas présent il faut ajouter un nouvel élément à **vec**. Dans tous les cas il faut mettre à jour les champs **outgoing** / **incoming**. Une quinzaine de lignes suffisent pour écrire cette fonction.

Rappel : une structure **Stats** mémorise les durées totales de communication avec un numéro de téléphone en distinguant les appels sortants (*outgoing*) des appels entrants (*incoming*). Par exemple l'affichage obtenu pour le numéro 0771234567 du vector calls (lignes 35-40) est visible en 2.1.2).

```
103 void process_call(const Call& call, vector<Stats>& vec)
104 {
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127 }
```

2.4) Supposons que la fonction **process_call** soit déclarée et définie avec cette en-tête :

```
void process_call(const Call& call, vector<Stats> vec)
```

Le code que vous avez écrit à la question 2.3) compile-t-il avec cette en-tête différente ? (oui/non)

Si oui, l'exécution donne-t-elle le résultat attendu ? Pourquoi ?
Sinon existe-t-il un moyen simple de modifier votre code (sans changer l'en-tête proposée ici) pour obtenir un résultat correct ? (donner un exemple d'instruction modifiée).

2.5) Supposons que la fonction **process_call** soit déclarée et définie avec cette en-tête:

```
void process_call(const Call* call, vector<Stats>& vec)
```

Le code que vous avez écrit à la question 2.3) compile-t-il avec cette en-tête différente ? (oui/non)

Si oui, l'exécution donne-t-elle le résultat attendu ? Pourquoi ?
Sinon existe-t-il un moyen simple de modifier votre code (sans changer l'en-tête proposée ici) pour obtenir un résultat correct ? (donner un exemple d'instruction modifiée).

2.6) Supposons que la fonction **process_call** soit déclarée et définie avec cette en-tête:

```
void process_call(const Call& call, vector<Stats>* vec)
```

Le code que vous avez écrit à la question 2.3) compile-t-il avec cette en-tête différente ? (oui/non)

Si oui, l'exécution donne-t-elle le résultat attendu ? Pourquoi ?
Sinon existe-t-il un moyen simple de modifier votre code (sans changer l'en-tête proposée ici) pour obtenir un résultat correct ? (donner un exemple d'instruction modifiée).

3) (7 pts) évaluation d'expression et boucle for

Le code suivant compile en C++11 et affiche une valeur entière à l'exécution.

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int a(0);
7      for(int x(0); !x&&a<=1 ; ++a)
8         a++;
9
10     cout << a << endl;
11     return 0;
12 }
```

3.1) Préciser **comment** les priorités entre opérateurs s'appliquent pour l'évaluation de la condition de la boucle (ligne 7)

3.2) combien de passages sont effectués dans cette boucle ? :……..
Préciser votre réponse en **évaluant la condition de boucle** à partir de la valeur de **a** et **x** à chaque passage dans la boucle ;

3.3) En vous appuyant sur votre réponse à la question précédente, quelle valeur est affichée à la ligne 10 ?

4) (8 pts) pointeur et appel de fonction

Le code suivant compile en C++11 et affiche 2 valeurs entières à l'exécution.

```
1  #include <iostream>
2  using namespace std;
3
4  void f2(int* p)
5  {
6      *p = -*p ;
7  }
8
9  int f1(int x)
10 {
11     f2(&x);
12     return  x+1;
13 }
14
15 int main()
16 {
17     int x(3), y(2);
18     y = f1(x);
19     cout << x << " , " << y << endl;
20     return 0;
21 }
```

4.1) Quel est le résultat de l'exécution de la ligne 6 ?

Justifier votre réponse de manière détaillée

4.2) Donner et justifier l'affichage produit à la ligne 19