Nom, prénom, numéro d'étudiant :

Université Paris Sud, Licence MPI, Info 111 — Partiel du 7 novembre 2016 (deux heures)

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5	Total	Note / 20

Calculatrices et autres gadgets électroniques interdits.
Seul document autorisé : une feuille au format A4 avec, au recto, le résumé de la syntaxe C++ de la fiche de TD 2 et, au verso, des notes manuscrites. Pour les étudiants inscrits en Français Langue Étrangère, un dictionnaire est autorisé.
Les exercices sont indépendants les uns des autres ; il n'est pas nécessaire de les faire dans l'ordre ; ceux marqués d'un ♣ sont plus difficiles ; le barème est indicatif.
Les réponses sont à donner, autant que possible, sur le sujet ; sinon, mettre un renvoi.

Exercice 1 (Cours : 1 point)**.**
Rappeler la syntaxe et la sémantique de la boucle `while` en C++.

Exercice 2 (Entrée - sorties / fonctions : 3 points)**.**

(1) Compléter le programme suivant pour qu'il demande à l'utilisateur d'entrer deux entiers et affiche leur minimum.

```
#include <iostream>
using namespace std;
int main() {
|
|
|
|
|
|
|
|
|
    return 0;
}
```

(2) On veut maintenant écrire un programme qui demande à l'utilisateur **quatre entiers** a, b, c, d et affiche la somme des minimums de a et b d'une part et de c et d d'autre part. Par exemple, si on a $a = 1$, $b = 3$, $c = 4$, $d = 2$, le programme doit afficher 3.

Écrire une fonction qui calcule le minimum de deux nombres et une fonction principale (fonction main) :

Exercice 3 (Boucles : 4 points)**.**

(1) Compléter le **programme** suivant pour qu'il demande à l'utilisateur un entier positif n puis **affiche** tous ses diviseurs. Par exemple, si l'utilisateur entre le nombre 10, le programme affiche `1 2 5 10`. Rappel : d divise n si et seulement si `n%d == 0`.

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

int main(){
|
|
|
|
|
|
|
    return 0;
}
```

(2) Écrire une **fonction** qui prend en **paramètre** un entier n et **renvoie** son plus grand diviseur qui n'est pas n lui-même. Par exemple, pour $n = 10$, la fonction renvoie 5, pour $n = 7$, la fonction renvoie 1. Par convention, pour $n = 1$, la fonction renvoie 1.

Exercice 4 (Tableaux simples : 9 points)**.**

(1) Observer le fragment de programme suivant :

```
1  vector<int> lireTableau(int n) {
2      vector<int> t;
3      t = vector<int>(n);
4      for ( int i=0; i < n; i++ ) {
5          cin >> t[i];
6      }
7      return t;
8  }
9
10 int main() {
11     vector<int> t = lireTableau(10);
12     for ( int i = 0; i < t.size(); i++ ) {
13         cout << t[i] << endl;
14     }
15 }
```

(a) Quel est le **type de retour** de la fonction `lireTableau` ?

(b) Indiquer le numéro de la ou les lignes de la fonction `lireTableau` qui

(i) **Déclarent** le tableau `t` :

(ii) **Allouent** le tableau `t` :

(iii) **Initialisent** le tableau `t` :

(c) En une phrase, décrire ce que fait ce programme.

(2) Compléter la fonction suivante dont on vous donne la documentation et les tests.

```
/** Renvoie le nombre de valeurs négatives dans le tableau
 * @param t un tableau d'entiers
 * @return le nombre d'entiers négatifs
 **/
int compteNegatifs(vector<int> t) {
|
|
|
|
|
|
|
}

void compteNegatifsTest() {
    ASSERT( compteNegatifs({1,-5,2,8,-3,7,-4}) == 3 );
    ASSERT( compteNegatifs({-2,-7,-8,-4})      == 4 );
}
```

(3) Écrire une **fonction** nommée `contient` qui prend en paramètre un `vector<int> t` ainsi qu'un entier v et renvoie vrai si v appartient à `t` sous forme de **booléen**. La fonction devra passer les tests suivants :

```
void contientTest() {
    ASSERT(     contient({1,4,3}, 1) );
    ASSERT(     contient({1,4,3}, 4) );
    ASSERT(     contient({1,4,3}, 3) );
    ASSERT( not contient({1,4,3}, 2) );
}
```

(4) Compléter la fonction suivante dont on vous donne la documentation et les tests :

```
/** Renvoie le tableau des valeurs absolues
 * @param t1 un tableau d'entier
 * @return un nouveau tableau contenant les valeurs absolues des éléments de t1
 **/
vector<int> absTableau(vector<int> t1) {
|
|
|
|
|
|
}

void absTableauTest() {
    vector<int> t = {1,4,3};
    ASSERT( absTableau({1,4,3})    == t );
    ASSERT( absTableau({-1,4,3})   == t );
    ASSERT( absTableau({-1,-4,-3}) == t );
}
```

(5) ♣ Écrire une fonction qui prend en paramètre un tableau d'entiers et renvoie un nouveau tableau d'entiers contenant les mêmes valeurs, mais où les valeurs négatives sont toutes à gauche des valeurs positives. Par exemple, si le tableau donné en paramètre est `{1, -6, 7, 2, -7, -3}`, alors la fonction renvoie `{-6, -7, -3, 1, 7, 2}`.

Exercice 5 (Tableaux doubles : 5 points)**.**
Une salle de réunion peut être utilisée par différents employés d'une entreprise. La réservation se fait par plage d'une heure, de 8H00 du matin à 19H00. Chaque plage d'une heure commence à l'heure pile (par exemple, il y a une plage 9H00-10H00 mais il n'y a pas de plage 9H15-10H15). Un tableau de booléens à deux dimensions est utilisé pour représenter si la salle est occupée (valeur `true`) ou disponible (valeur `false`) pendant une semaine. Une dimension est utilisée pour coder les jours ouvrables de 0 (lundi) à 4 (vendredi). L'autre dimension est utilisée pour les plages horaires de 0 (8H00-9H00) à 10 (18H00-19H00). Chaque case correspond à la réservation de la salle pour une plage d'un jour donné.

Par exemple, si `s` est la variable contenant le tableau, la valeur `s[0][2]` est égale à `true` si la salle est occupée le lundi (indice 0) de 10h à 11h (indice 2).

(1) (a) Si `s[1][4] == false`, quelle information a-t-on sur la disponibilité de la salle ?

(b) Si la salle est occupée le jeudi de 10h à 11h, quelle case du tableau est égale à `true` ?

(2) Compléter les quatre trous du programme suivant :

(a) La déclaration et le début de l'allocation du tableau `salle` dans la fonction `main`.

(b) L'initialisation des valeurs en fonction des indices donnés par l'utilisateur dans la boucle `while` de la fonction `main`.

(c) La fonction `nbPlagesOccupees` qui compte le nombre de plages horaires occupées dans le tableau de la salle.

(d) La fonction `occupation` qui calcule le *taux d'occupation*, c'est-à-dire le pourcentage de plages occupées parmi toutes les plages.

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

/** Compte le nombre de plages horaires occupées
 * @param s un double tableau de booléens
 * @return le nombres de plages horaires où la salle est occupée
 **/
int nbPlagesOccupees(vector<vector<bool>> s) { // (c)
```

```
}
```

Suite au dos !

```
/** Renvoie le taux d'occupation de la salle
 * @param s un double tableau de booléens
 * @return le taux d'occupation, en pourcentage
 **/
double occupation (vector<vector<bool>> s) {    // (d)



}

int main() {
    // (a) Déclaration et allocation du tableau



    for ( int i = 0; i < salle.size(); i++ ) {
        salle[i] = vector<bool> (11);
        for ( int j = 0; j < 11; j++ ) {
            salle[i][j] = false;
        }
    }

    int i = 0;
    int j = 0;
    while( i  != -1 ) {
        cout << "Entrez les indices du jour et de l'heure d'utilisation "
             << "de la salle" << endl;
        cin >> i >> j;
        if (i >= 0 and i < 5 and j >= 0 and j < 11) {
            // (b) Initialisation des valeurs  du tableau



        }
    }

    cout << "taux d'occupation: " << occupation(salle) << "%" << endl;

    return 0;
}
```