Nom, Prénom : **Numéro d'étudiant :** ␣␣␣␣␣␣␣␣␣␣

Faculté d'Orsay, Licence 1, Semestre 1, Programmation Impérative **Année 2020-2021**

Examen mi-semestre du vendredi 6 novembre 2020 (deux heures)

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5	Exercice 6	Exercice 7	Total / 100
␣␣	␣␣	␣␣	␣␣	␣␣	␣␣	␣␣	␣␣␣

copie 1

Calculatrices, téléphones mobiles et tout appareil connectable non autorisé doivent être éteints et déposés avec les affaires personnelles de l'étudiant.
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 mais font bien partie du barème sur 100.
Le barème est indicatif et sujet à ajustements.
Les réponses sont à donner, autant que possible, sur le sujet ; sinon, mettre un renvoi.
Les enseignants collecterons votre copie à votre place.

Exercice 1 (Cours (10 points))**.**
Rappeler la syntaxe et la sémantique de la boucle `do ... while` en C++.

Correction : Syntaxe :

```
do {
    bloc d'instructions;
} while ( condition )
```

Sémantique : répéter le bloc d'instructions tant que la condition est vraie :

(1) Exécution du bloc d'instruction

(2) Évaluation de la condition

(3) Si sa valeur est `true` recommencer en 1

Exercice 2 (Fonctions simples (10 points))**.**

(1) Écrire une fonction `formuleBizarre` qui prend en paramètres deux entiers et renvoie la somme du double du premier paramètre et du triple du second paramètre.

Correction :

```
int formuleBizarre(int a, int b) {
    return 2*a + 3*b;
}
```

(2) Utilisation : écrire les lignes de code qui permettent de déclarer et initialiser deux variables avec les valeurs de votre choix puis de stocker le résultat de la formule bizarre appliquée à ces nombres dans une nouvelle variable. Il est obligatoire d'utiliser un **appel** à votre fonction `formuleBizarre`.

Correction :

```
    int p = 2;
    int q = 5;
    int resultat = formuleBizarre(p, q);
```

Exercice 3 (Booléens (6 points))**.**

(1) Écrire la fonction dont la documentation et les tests sont donnés ci-dessous :

```
/** Fonction un_50_ou_deux_20
 * @param a un entier
 * @param b un entier
 * @return true si l'un des deux entiers est supérieur ou égal à 50 ou
 *   si les deux sont strictement supérieurs à 20; renvoie false sinon
 **/
```

```
    ASSERT(     un_50_ou_deux_20(52, 53) );
    ASSERT(     un_50_ou_deux_20(16, 50) );
    ASSERT(     un_50_ou_deux_20(21, 42) );
    ASSERT( not un_50_ou_deux_20(20, 49) );
    ASSERT( not un_50_ou_deux_20(47, 18) );
```

Correction :

```
bool un_50_ou_deux_20_naif(int a, int b) {
    if ( (a >= 50 or b >= 50) or (a > 20 and b > 20) ) {
        return true;
    } else {
        return false;
    }
}
```

(2) Si ce n'est pas déjà le cas, ré-écrire votre fonction pour qu'elle ne contienne pas de `if` (ni aucune autre structure de contrôle bien sûr !).

Correction :

```
bool un_50_ou_deux_20(int a, int b) {
    return ( (a >= 50 or b >= 50) or (a > 20 and b > 20) );
}
```


Exercice 4 (Boucles (15 points))**.**

(1) Écrire les lignes de code permettant d'afficher tous les diviseurs de 2310 (y compris 1 et 2310). On rappelle que a divise b si le reste de la division de b par a est nul. Le reste s'obtient avec l'opérateur % en C++.

Correction :

```
    for ( int i = 1; i <= 2310; i++ ) {
        if ( 2310 % i == 0 ) {
            cout << i << endl;
        }
    }
```

(2) Écrire une fonction nommée `somme_carres` qui prend en paramètre un entier m et qui renvoie la somme des carrés des nombres compris entre 1 et m inclus. Par exemple, le résultat pour $m = 3$ sera $1 * 1 + 2 * 2 + 3 * 3 = 14$.

Correction :

```
int somme_carres(int m) {
    int somme = 0;
    for ( int i = 1; i <= m; i = i+1) {
        somme = somme + i*i;
    }
    return somme;
}
```

(3) Écrire une fonction nommée `affichePuissances` qui prend en paramètres un entier k et un entier p, et qui affiche toutes les k-ièmes puissances des nombres naturels (à partir de 1^k) qui sont inférieures ou égales à p si $k > 0$ et rien sinon. Par exemple `affichePuissances(2,15)` doit afficher 1, 4 et 9, `affichePuissances(2,16)` doit afficher 1, 4, 9 et 16, et `affichePuissances(0,16)` ne doit rien afficher.

Vous devez utiliser des appels à la fonction `puissance` dont la documentation est donnée ci-dessous :

```
/** La fonction puissance
 *  @param x un nombre entier
 *  @param n un nombre entier positif
 *  @return la n-ième puissance x^n de x
 **/
```

La fonction puissance est considérée déjà définie, vous n'avez pas à écrire son code.

Correction :

```
void affiche_puissances(int k, int p) {
    if (k > 0){
        int nombre = 1;
        while ( puissance(nombre, k) <= p ) {
            cout << puissance(nombre,k) << endl;
            nombre++;
        }
    }
}
```

Exercice 5 (Tableaux (12 points))**.**

(1) Écrire une fonction nommée `contientNegatif` qui prend en paramètre un tableau d'entiers `t` et renvoie `true` si `t` contient au moins un nombre strictement négatif et `false` sinon. La fonction devra passer les tests suivants :

```
    ASSERT(     contientNegatif( {-2, 8, 6} ) );
    ASSERT(     contientNegatif( {2, 8, -6} ) );
    ASSERT( not contientNegatif( {2, 0, 6} ) );
```

```
bool contientNegatif(vector<int> t) {
    for ( int i = 0; i < t.size(); i++ ) {
        if ( t[i] < 0 ) {
            return true;
        }
    }
    return false;
}
```

(2) ♣ Si ce n'est déjà fait, réécrivez la fonction précédente en utilisant une boucle *for each* : `for (int v:t){...}`.

```
bool contientNegatif_foreach(vector<int> t) {
    for ( int w: t )
        if ( w < 0 )
            return true;
    return false;
}
```

(3) Écrire une fonction nommée `positionMultiple` qui prend en paramètres un tableau d'entiers `t` et un entier `m`, et renvoie l'indice du premier nombre multiple de `m` dans `t`, ou -1 si `t` ne contient pas de multiple de m. On rappelle que le reste s'obtient avec % en C++. La fonction devra passer les tests suivants :

```
    ASSERT( positionMultiple( {4, 6, 3}, 2) ==  0 );
    ASSERT( positionMultiple( {1, 15, 3}, 5) ==  1 );
    ASSERT( positionMultiple( {3, 5, 2}, 2) ==  2 );
    ASSERT( positionMultiple( {1, 4, 3}, 5) == -1 );
```

Correction :

```
int positionMultiple(vector<int> t, int k) {
    for ( int i = 0; i < t.size(); i++ ) {
        if ( t[i] % k == 0 ) {
            return i;
        }
    }
    return -1;
}
```

Exercice 6 (Pile et Tas (12 points))**.**
On considère le fragment de programme suivant :

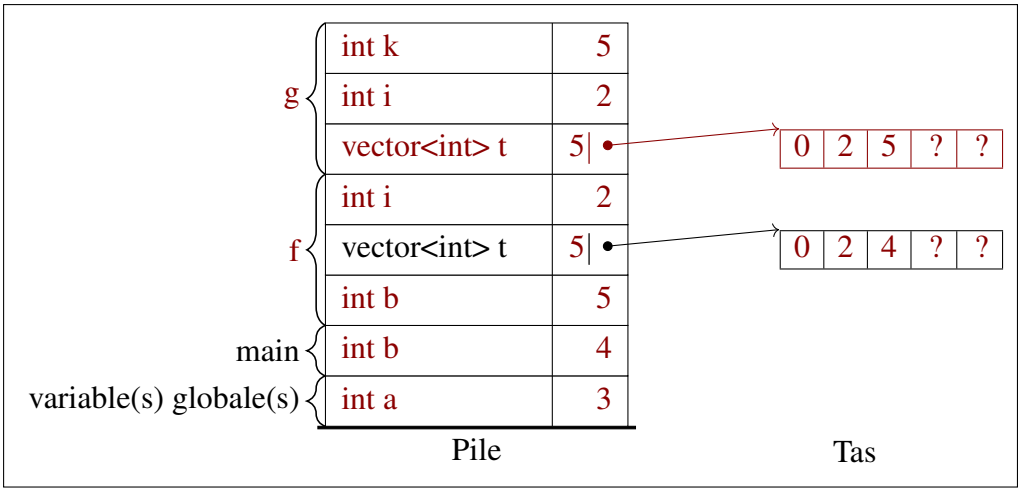
```
int a;

void g(vector<int> t, int i) {
    int k = a + i;
    t[i] = k;
    cout << t[i] << " ";
}

int f(int b) {
    a = a - 1;
    b = b + 1;
    vector<int> t;
    t = vector<int>(b);
    for ( int i = 0; i < t.size(); i++ ) {
        t[i] = 2*i;
        g(t, i);
    }
    return t[2];
}

int main() {
    a = 4;
    int b = a;
    cout << f(b) << endl;
    return 1;
}
```

(1) Soulignez la ou les déclarations de paramètres formels.

(2) Encadrez d'un rectangle la ou les déclarations de variables locales.

(3) Entourez d'un rond la ou les déclarations de variables globales.

(4) Sur votre brouillon, exécutez pas-à-pas le programme. En suivant les conventions du cours, complétez ci-dessous le dessin de la pile et du tas au moment où l'exécution atteint la dernière ligne de la fonction `g` et où `i` vaut 2.



copie 1
page 6

(5) Qu'affiche le programme au cours de l'ensemble de son exécution ?

```
3 4 5 6 7 4
```

Exercice 7 (Master Mind (35 points))**.**
Le MASTERMIND est un jeu de société où un joueur doit deviner le code secret choisi par un arbitre. L'arbitre choisit un code composé de quatre jetons de couleurs parmi six couleurs possibles et placés dans l'ordre voulu. Il peut choisir plusieurs fois la même couleur dans son code et cache son code à l'adversaire. Le joueur doit deviner le code en faisant des propositions. À chaque proposition, l'arbitre doit donner les informations suivantes :

— Le nombre de pions qui est de la bonne couleur et à la bonne position (indiqués par autant de fiches rouges sur le plateau de jeu).
— Le nombre de pions de la bonne couleur mais à la mauvaise position (indiqués par autant de fiches blanches).

Si aucun pion n'est de la bonne couleur, l'abitre ne donne aucun indice au joueur.

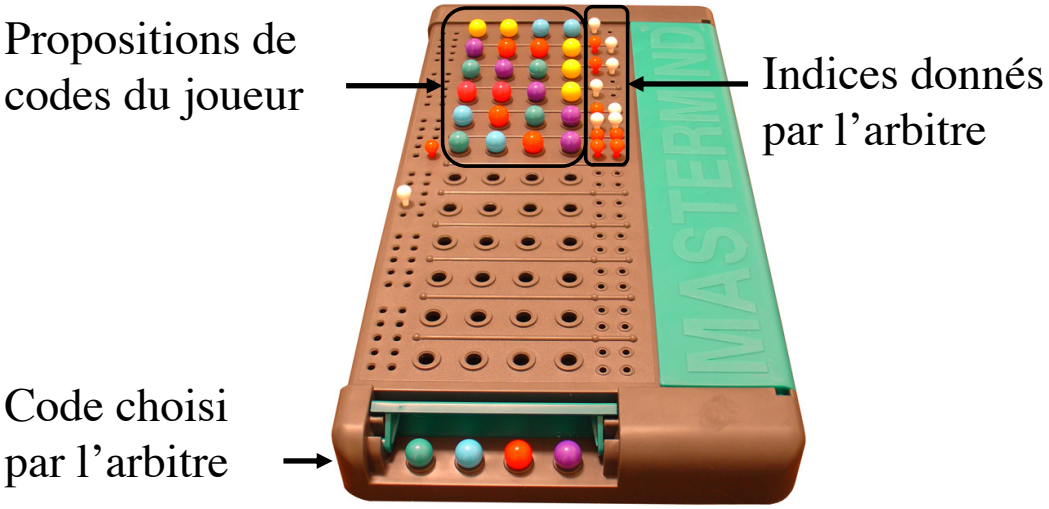


FIGURE 1. Plateau de jeu de Master Mind

La manche se termine lorsque le joueur découvre le code ou au bout d'un certain nombre de propositions (12 sur le plateau de la figure 1). L'arbitre et le joueur inversent leurs rôles à chaque nouvelle manche.

On se propose d'implanter MASTERMIND en automatisant l'arbitre, qui choisit le code et donne des indices. On encodera les six couleurs par des entiers allant de 0 à 5.

(1) Implanter une fonction `genererCodeSecret` décrite par la documentation ci-dessous. On considère que l'on a déjà une fonction `int aleaint(int a, int b)` qui renvoie un entier aléatoire compris entre a et b inclus.

```
/** Génère un code secret aléatoire
 * @return tableau contenant 4 entiers tirés aléatoirement entre 0 et 5
 **/
vector<int> genererCodeSecret() {
    vector<int> code;
    code = vector<int>(4);
    for(int i = 0; i < code.size(); i++) {
        code[i] = aleaint(0, 5);
    }
    return code;
}
```

(2) On s'intéresse dans un premier temps aux éléments du code qui sont à la bonne position. On souhaite écrire une fonction `compterPositionsCorrectes` qui prend en paramètre deux tableaux d'entiers de même taille et renvoie le nombre de cases identiques, c'est à dire avec la même valeur à la même position.

(a) Écrire trois tests pour la fonction `compterPositionsCorrectes` :

Correction :

```
ASSERT(compterPositionsCorrectes({0,1,2,3}, {0,1,2,3}) == 4);
ASSERT(compterPositionsCorrectes({0,1,2,3}, {0,1,4,5}) == 2);
ASSERT(compterPositionsCorrectes({5,4,3,2}, {2,3,4,5}) == 0);
```

(b) Implanter la fonction `compterPositionsCorrectes` :

```
/** Compte le nombre de pions qui ont la même valeur et la même
 *  position dans les deux codes
 * @param code1 tableau d'entiers
 * @param code2 tableau d'entiers de même taille que code1
 * @return un entier
**/

int compterPositionsCorrectes(vector<int> code1, vector<int> code2){
    int res = 0;
    for (int i = 0; i < code1.size(); i++){
        if (code1[i] == code2[i]){
            res++;
        }
    }
    return res;
}
```

(3) On s'intéresse à présent aux éléments de la proposition du joueur qui sont présents dans le code secret de l'arbitre mais pas forcément à la bonne position.

(a) Observer les tests suivants et déduisez-en la documentation (rôle, entrées et sortie) de la fonction `compterOccurences`. Écrire la documentation en dessous des tests :

```
ASSERT( compterOccurences( {0,1,2,3} ) == vector<int>({1,1,1,1,0,0}) );
ASSERT( compterOccurences( {4,5,5,2} ) == vector<int>({0,0,1,0,1,2}) );
ASSERT( compterOccurences( {1,1,1,1} ) == vector<int>({0,4,0,0,0,0}) );
ASSERT( compterOccurences( {5,5,5,5} ) == vector<int>({0,0,0,0,0,4}) );

    /* compte le nombre d'occurence de chaque chiffre
    * de 0 à 5 dans un code
    * @param un code à quatre éléments
    * @return un tableau à six éléments contenant le nombre d'occurence de chaque chiffre
    **/
```

(b) Implanter la fonction `compterOccurences` :

```
vector<int> compterOccurences(vector<int> code) {
        vector<int> res = {0, 0, 0, 0, 0, 0};
        for ( int i = 0; i < code.size(); i++ ) {
            int indice = code[i];
            res[indice] += 1;
        }
        return res;
    }
```

(c) Écrire une fonction `compterValeursCorrectes` qui prend en paramètre deux tableaux d'entiers et renvoie le nombre d'éléments présents dans les deux tableaux, pas forcément au même endroit (attention, cette fonction prendra également en compte les valeurs correctement positionnées). La fonction `compterValeursCorrectes` doit vérifier les tests suivants :

```
ASSERT(compterValeursCorrectes({0,0,1,1}, {2,2,3,3}) == 0);
ASSERT(compterValeursCorrectes({0,0,1,1}, {2,1,2,2}) == 1);
ASSERT(compterValeursCorrectes({0,0,1,1}, {0,1,2,2}) == 2);
ASSERT(compterValeursCorrectes({0,0,1,1}, {0,0,0,1}) == 3);
ASSERT(compterValeursCorrectes({0,0,1,1}, {1,1,0,2}) == 3);
ASSERT(compterValeursCorrectes({0,0,1,1}, {0,1,0,1}) == 4);
ASSERT(compterValeursCorrectes({0,0,1,1}, {0,0,1,1}) == 4);
ASSERT(compterValeursCorrectes({0,0,1,1}, {1,1,0,0}) == 4);
```

Indice : On pourra utiliser par exemple la fonction `compterOccurences` et une fonction `min(int a, int b)` que l'on considère déjà définie et qui renvoie le minimum entre deux entiers.

```
/** Compte le nombre de pions identiques entre les deux codes,
 *  quelles que soient leurs positions
 * @param code1 tableau d'entiers
 * @param code2 tableau d'entiers de même taille que code1
 * @return un entier
**/

int compterValeursCorrectes(vector<int> code1, vector<int> code2){
    vector<int> cptCode1 = compterOccurences(code1);
    vector<int> cptCode2 = compterOccurences(code2);
    int res = 0;
    for (int i = 0; i < cptCode1.size(); i++){
        if (cptCode1[i] > 0 and cptCode2[i]>0){
            res+=min(cptCode1[i], cptCode2[i]);
        }
    }
    return res;
}
```

(4) Compléter le code de la fonction principale du jeu présentée ci-dessous :

```
int main(){
    srand(time(0)); // Initialisation de la fonction rand
    vector<int> codeSecret = genererCodeSecret();
    int N = 12; // Nombre maximal de propositions
    vector<int> codeJoueur;
    int bienPlaces;
    int malPlaces;

    for (                                                    ){
        codeJoueur = entrerCode();
        bienPlaces = compterPositionsCorrectes(codeSecret, codeJoueur);
        malPlaces =

        cout << bienPlaces << " pions bien placés" << endl;
        cout << malPlaces << " pions corrects mais mal placés" << endl;

        if (                                                 ){
            cout << "Vous avez         !" << endl;
            return 0;
        }
    }
    cout << "Vous avez         !" << endl;
    return 0;
}
```

(5) ♣ Expliquer une manière d'automatiser les propositions du joueur pour trouver le code avec le moins de propositions possibles. On ne demande pas d'écrire de code C++ dans cette question. **Indice :** Considérez l'ensemble des propositions possibles et comment réduire cet ensemble au fur et à mesure des indices de l'arbitre.

Correction :

— Générer l'ensemble des 6^4 solutions possibles.
— Proposer une code au hasard, par exemple $[0, 0, 1, 1]$.
— Si la proposition est juste, le jeu est terminé. Sinon supprimer de l'ensemble tous les codes qui donneraient un indice différent avec la même proposition. Par exemple, avec la proposition $[0, 0, 1, 1]$, si l'indice est *"Un élément bien placé et 2 éléments mal placés"*, alors $[0, 0, 2, 2]$ est supprimé de l'ensemble tandis que $[2, 1, 0, 1]$ est conservé dans l'ensemble (car il donnerait le même indice de la part de l'arbitre).
— Pour chaque code restant dans l'ensemble des solutions, calculer combien de possibilités seraient éliminées pour chaque indice possible (combinaisons des bien placés et mal placés). Proposer le code qui élimine le plus de solutions et reprendre à l'étape 3.