Nom, prénom, numéro d'étudiant : . . . . . .

Université Paris Sud, Licence MPI, Info 111 **Partiel du 2 novembre 2015 (deux heures)**

Calculatrices et autres gadgets électroniques interdits.
Seul document autorisé : une feuille au format A4 avec, au recto, la résumé de la syntaxe C++ de la fiche de TD 2 et, au verso, des notes manuscrites.
Les exercices sont indépendants les uns des autres ; il n'est pas nécessaire de les faire dans l'ordre.
Les réponses sont à donner sur le sujet.

Exercice 1 (Cours)**.**
Rappeler la syntaxe et la sémantique de la boucle **for** en C++.

Correction.
Syntaxe :

```
for ( initialisation ; condition ; incrementation ) {
    bloc d'instructions
}
```

Sémantique :

1. Exécution de l'instruction d'initialisation
2. Évaluation de la condition
3. Si sa valeur est `true` :
   (a) Exécution du bloc d'instruction
   (b) Exécution de l'instruction d'incrémentation
   (c) On recommence en 2

Exercice 2 (Cours)**.**
Quelles sont les trois étapes pour construire un tableau en C++ ?

Correction.
Un tableau se construit en trois étapes :

1. Déclaration
2. Allocation
3. Initialisation

Illustrer votre réponse avec la construction d'un tableau de 20 booléens initialisés à true :

Correction.

```
vector<bool> t;
t = vector<bool>(20);
for ( int i = 0; i < 20; i++ ) {
    t[i] = true;
}
```

Exercice 3 (Fonctions)**.**

1. Écrire une fonction qui prend en paramètres deux entiers a et b et renvoie `true` s'ils sont de même signe (c'est-à-dire tous les deux positifs ou tous les deux négatifs, en considérant que zéro est à la fois positif et négatif) et `false` sinon.
   Correction.

```
bool meme_signe(int a, int b) {
    if ( a*b >= 0 )
        return true;
    else
        return false;
}
```

2. Écrire une fonction principale `main` qui demande à l'utilisateur deux entiers et affiche s'ils sont de même signe ou non en utilisant la fonction de la question précédente. Par exemple, si l'utilisateur entre 10 et 12, le programme affiche « 10 et 12 sont de même signe », et si l'utilisateur entre 10 et -5, le programme affiche « 10 et -5 sont de signes différents ».
   Correction.

```
int main() {
    int a, b;
    bool resultat;
    cin >> a;
    cin >> b;
    resultat = meme_signe(a,b);
    if ( resultat )
        cout << a << " et " << b << " sont de même signe." << endl;
    else
        cout << a << " et " << b << " sont de signes différents." << endl;
    return 0;
}
```

Exercice 4 (Racine Carrée)**.**
On peut approximer la valeur de la racine d'un nombre a à l'aide d'une simple suite numérique, définie récursivement par

$$U_0 = 0, \qquad U_{n+1} = \frac{U_n + a}{U_n + 1}.$$

Par exemple, pour approximer la racine carrée de 10, on obtient :

$$\begin{aligned} U_0 &= 0, \\ U_1 &= \frac{10}{1} = 10, \\ U_2 &= 20/11 = 1.81818181818182, \\ U_3 &= \frac{U_2 + 10}{U_2 + 1} = 4.19354838709677. \end{aligned}$$

Au bout de 10 itérations, on a $U_{10} = 3.16227766016838$ qui est une bonne approximation de $\sqrt{10}$.

1. Compléter le code de la fonction `racine_carre_approx` dans le programme suivant :

```
/** Racine Carree
 *  @param a un nombre en double précision
 *  @param n un entier positif
 *  @return une approximation de la racine carrée de a après n itérations
 **/
double racine_carree_approx(double a, int n) {









}

int main() {
    cout << racine_carree_approx(10,50) << endl;
}
```

Correction.

```
/** Racine Carree
 *  @param a un nombre en double précision
 *  @param n un entier positif
 *  @return une approximation de la racine carrée de a après n itérations
 **/
double racine_carree_approx(double a, int n) { //Correction
    double u = 0;
    for ( int i = 0; i < n; i++ )
        u = (u+a) / (u+1.0);
    return u;                                  //FinCorrection
}
```

2. Donner une nouvelle version de la fonction `main` qui permette à l'utilisateur d'entrer le nombre a ainsi que le nombre d'itérations n et d'afficher la valeur approximée de $\sqrt{a}$ après n itérations.
Correction.

```
int main() {
    double a;
    int n;
    cin >> a;
    cin >> n;
    cout << racine_carree_approx(a, n) << endl;
    return 0;
}
```

Problème

Le Yams est un jeu de dés où l'on cherche à former des figures avec 5 dés. Les figures que nous allons considérer dans cet exercice sont les suivantes :

- le **brelan** : trois dés égaux, par exemple $\{5, 5, 2, 5, 6\}$,
- le **full** : trois dés égaux et deux dés égaux, par exemple $\{1, 6, 6, 1, 6\}$,
- le **carré** : quatre dés égaux, par exemple $\{3, 2, 2, 2, 2\}$,
- le **yams** : tous les dés égaux, par exemple $\{4, 4, 4, 4, 4\}$.

Les figures permettent de marquer des points. À chacune de ces figures, sont associées des bonus : 10 pour le brelan, 20 pour le full, 30 pour le carré et 60 pour le yams. A cela, on ajoute la somme des dés qui composent la figure. Par exemple, les dés $\{2, 5, 3, 5, 5\}$ permettent de marquer 10 points de bonus (brelan) et $5 + 5 + 5 = 15$ points, soit au total **25 points**. Le but des prochains exercices est de commencer l'implantation d'un jeu de yams basique.

Dans toutes la suite, on représentera les 5 dés par un tableau de type `vector<int>`.

Exercice 5 (Reconnaître les figures, compter les points)**.**

1. Écrire une fonction `afficheDes` qui prend en paramètre le tableau de dés et l'affiche.
   Correction.

```
void afficheDes(vector <int> des) {
    for ( int i = 0; i < des.size(); i++ ) {
        cout << des[i] << " ";
    }
    cout << endl;
}
```

2. La figure la plus simple à reconnaître est le Yams lui-même. La fonction `pointsFigureYams` prend en paramètre le tableau de dés et renvoie la somme des 5 dés + 60 si le tableau correspond à un yams, et 0 sinon. Ainsi, sur $\{4, 4, 4, 4, 4\}$, la fonction renverra 80 et sur $\{2, 2, 3, 2, 3\}$, elle renverra 0, car ce n'est pas un yams ; cela se traduit par les tests suivants :

```
void pointsFigureYamsTest() {
    ASSERT( pointsFigureYams(vector<int>({4,4,4,4,4})) == 80 );
    ASSERT( pointsFigureYams(vector<int>({2,2,3,2,3})) ==  0 );
}
```

   Implanter la fonction `pointsFigureYams`.
   Correction.

```
int pointsFigureYams(vector<int> des) {
    int a = des[0];
    for ( int i = 0; i < 5; i++ ) {
        if ( des[i] != a )
            return 0;
    }
    return 5*a + 60;
}
```

3. Proposer des tests pour les fonctions : `pointsFigureBrelan` et `pointsFigureFull`.
Correction.

```
void pointsFigureBerlanTest() {
    vector<int> brel = {2,2,3,2,4};
    ASSERT( pointsFigureBrelan(brel) == 23 );
    vector<int> nonbrel = {1,2,3,4,5};
    ASSERT( pointsFigureBrelan(nonbrel) == 0 );
}

void pointsFigureFullTest() {
    vector<int> brel = {2,2,3,2,3};
    ASSERT( pointsFigureBrelan(brel) == 32 );
    vector<int> nonbrel = {1,2,3,4,5};
    ASSERT( pointsFigureBrelan(nonbrel) == 0 );
}
```

4. Pour reconnaître les autres figures, il est plus facile d'avoir au préalable transformé le tableau 2D avec la fonction `mystere` suivante :

```
vector<int> mystere(vector<int> bing) {
    vector<int> yep = {0,0,0,0,0,0};
    for(int oups =0; oups< bing.size(); oups++) {
        int baba = bing[oups] -1;
        yep[baba] = yep[baba] + 1;
    }
    return yep;
}


void mystereTest() {
    ASSERT( mystere( {6,2,2,2,5} ) == vector<int>({0,3,0,0,1,1}) );
    ASSERT( mystere( {3,3,3,3,3} ) == vector<int>({0,0,5,0,0,0}) );
    ASSERT( mystere( {1,4,3,2,5} ) == vector<int>({1,1,1,1,1,0}) );
}
```

Expliquer en une phrase ce que fait cette fonction `mystere` puis **ré-écrire** la fonction en ajoutant la **documentation** et en modifiant les **noms de variables** de façon appropriée.

Correction.
Cette fonction renvoie un tableau comptant le nombre de dés de valeur 1, de valeur 2, etc (autrement dit un histogramme).

```
/** Comptage des dés par valeur
 * @param des: un tableau d'entiers
 * @return un tableau t de taille 6 tel que
 *         t[i-1] compte l nombre d'occurences du dé i
 **/
vector<int> mystere(vector<int> des) {
    vector<int> nombre = {0,0,0,0,0,0};
    for(int i =0; i < des.size(); i++) {
        int valeur = des[i] - 1;
        nombre[valeur] = nombre[valeur] + 1;
    }
    return nombre;
}
```

5. Écrire les fonctions `pointsFigureCarre`, `pointsFigureBrelan` et `pointsFigureFull` qui prennent chacune en paramètre le tableau de dés et renvoie le nombre de points de la figure si elle est présente, et 0 sinon.

   Remarque : ces fonctions sont plus simples à écrire si vous utilisez la fonction `mystere` de la question précédente. Vous pouvez cependant vous vous en passer si vous ne l'avez pas comprise.

   Correction.

```
int pointsFigureBrelan(vector<int> des) {
    vector<int> nombre;
    nombre = mystere(des);
    for ( int i = 0; i < nombre.size(); i++ ) {
        if (nombre[i] == 3)
            return (i+1)*3 + 10;
    }
    return 0;
}

int pointsFigureCarre(vector<int> des) {
    vector<int> nombre;
    nombre = mystere(des);
    for ( int i = 0; i < nombre.size(); i++ ) {
        if( nombre[i] == 4 )
            return (i+1)*4 + 30;
    }
    return 0;
}

int pointsFigureFull(vector<int> des) {
    vector<int> nombre;
    int paire = 0;
    int brelan = 0;
    nombre = mystere(des);
    for ( int i = 0; i < nombre.size(); i++ ) {
        if ( nombre[i] == 3 )
            brelan = i+1;
        if ( nombre[i] == 2 )
            paire = i+1;
    }
    if ( paire == 0 or brelan == 0 )
        return 0;
    else
        return 2*paire + 3*brelan + 20;
}
```

Exercice 6 (le Jeu)**.**
On souhaite à présent écrire le jeu de yams lui-même en utilisant les fonctions de l'exercice précédent. Le principe est de lancer les dés puis de choisir une figure pour marquer des points (si les dés ne correspondent pas à la figure choisie, on marque 0 points). Voici une fonction `pointsFigure` et une fonction `main` qui permettent de faire un tour de jeu avec un unique lancé.

```
int main() {
    srand(time(0));
    string figure;
    vector<int> des;

    des = lanceDes();
    cout << "Voici vos dés" << endl;
    afficheDes(des);
    do {
        cout << "Quelle figure choisissez-vous ?";
        cin >> figure;
    } while ( figure != "brelan" and figure != "carre" and
              figure != "full"   and figure != "yams" );
    cout << "Vous avez marqué " << pointsFigure(des, figure) << " points";
    return 0;
}
```

1. Écrire une fonction `lanceDes` qui renvoie un tableau d'entiers de taille 5 et dont les valeurs correspondent à un lancer de dés. On supposera qu'on a accès à une fonction `int aleaint(int a, int b)` qui renvoie un entier n tiré aléatoirement tel que $a \leq n \leq b$.
   Correction.

```
vector<int> lanceDes() {
    vector<int> resultat(5);
    for ( int i = 0; i < 5; i++ )
        resultat[i] = aleaint(1, 6);
    return resultat;
}
```

2. Modifier la fonction `main` pour qu'elle laisse le joueur faire plusieurs tours de jeu en accumulant les points à chaque tour. Il a le droit de choisir plusieurs fois la même figure. La partie s'arrête lorsque le joueur le décide.

   Lorsque l'on réutilisera plusieurs lignes consécutives de la fonction `main` d'origine sans les changer, on pourra se contenter d'écrire la première et la dernière avec « ... » entre les deux.

   Correction.

```
int main2() {
    srand(time(0));
    string figure;
    vector<int> des;
    int points = 0;
    string jeu;
    do {
        des = lanceDes();
        cout << "Voici vos dés" << endl;
        afficheDes(des);
        do {
            cout << "Quelle figure choisissez-vous ?";
            cin >> figure;
        } while (figure != "brelan" and figure != "carre" and
                 figure != "full"   and figure !="yams");
        points += pointsFigure(des, figure);
        cout << "Vous avez marqué " << points << " points" << endl;
        cout << "Voulez-vous continuer à jouer ? (oui/non)" << endl;
        cin >> jeu;
    } while ( jeu != "non" );
    return 0;
}
```

3. ♣ Modifier la fonction main pour que le joueur ne puisse choisir qu'une seule fois chaque figure par partie. La partie s'arrêtera donc obligatoirement au bout de quatre tours.
Correction.

```
int main3() {
    srand(time(0));
    string figure;
    vector<int> des;
    int points = 0;
    string jeu;
    vector<bool> figures_dispo = {true, true, true, true};
    vector<bool> figures_vide  = {false, false, false, false};
    bool continuer;
    do {
        des = lanceDes();
        cout << "Voici vos dés" << endl;
        afficheDes(des);
        do {
            cout << "Quelle figure choisissez-vous ?";
            cin >> figure;
            continuer =
                (figure == "brelan" and figures_dispo[0]) or
                (figure == "carre"  and figures_dispo[1]) or
                (figure == "full"   and figures_dispo[2]) or
                (figure == "yams"   and figures_dispo[3]);
        } while ( continuer );
        points += pointsFigure(des, figure);
        if (figure == "brelan")
            figures_dispo[0] = false;
        else if ( figure == "carre" )
            figures_dispo[1] = false;
        else if ( figure == "full" )
            figures_dispo[2] = false;
        else
            figures_dispo[3] = false;
        cout << "Vous avez marqué " << points << " points" << endl;
        if ( figures_dispo == figures_vide ) {
            cout << "C'est fini pour vous!";
            return 0;
        }
        cout << "Voulez-vous continuer à jouer ? (oui/non)" << endl;
        cin >> jeu;
    } while ( jeu != "non" );
    return 0;
}
```

4. ♣ Modifier la fonction main pour qu'à trois reprises dans le tour et avant de choisir sa figure, le joueur puisse choisir certains de ses dés et les relancer. Vous pourrez par exemple faire que le programme demande au joueur les indices des dés à conserver.