TD 6 : Jeu de Yam's

Exercice 1 (Aparté : boucles imbriquées)**.**
Devinez l'affichage du fragment de programme mystère suivant :

```
for ( int i = 1; i <= 4; i++ ) {
    for ( int j = 1; j <= i; j++ ) {
        cout << "(" << i << "," << j << ") ";
    }
    cout << endl;
}
```

Correction : `(1,1)`
`(2,1) (2,2)`
`(3,1) (3,2) (3,3)`
`(4,1) (4,2) (4,3) (4,4)`

Problème : le jeu de Yam's

Le jeu de Yam's (ou Yahtzee) est un jeu de dés dont le but est d'enchaîner les combinaisons à l'aide de cinq dés pour remporter un maximum de points.

Nous ne nous intéressons ici qu'à une version simplifiée du Yam's et chercherons à reconnaître les figures suivantes :

— **brelan** : 3 dés identiques parmi les 5 dés
— **yam's** : les 5 dés identiques.

Les figures permettent de marquer des points. À chacune de ces figures sont associés des bonus : **10 pour le brelan**, et **60 pour le yam's**. À 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 Yam's basique.

Exercice 2 (Échauffement : trois fonctions utilitaires)**.**

(1) Spécifiez (documentation) et implantez (code) une fonction `afficheDes` qui affiche le contenu d'un tableau d'entiers, en affichant chaque entier du tableau suivi d'un point-virgule. Ainsi l'appel `afficheDes({1,2,3,5,4})` devra entraîner l'affichage `1 2 3 5 4 .`

Correction :

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

(2) Spécifiez et implantez une fonction `chercheDansTableau` qui cherche l'emplacement d'un entier donné dans un tableau d'entiers. Si l'entier est présent dans le tableau, `chercheDansTableau` renvoie **l'indice d'une case du tableau le contenant**. Si l'entier n'est pas présent dans le tableau on renverra **-1**. Ainsi
— l'appel `chercheDansTableau(3, {1,2,3})` devra renvoyer `2`,
— l'appel `chercheDansTableau(4, {1,2,3})` devra renvoyer `-1`, etc.

Correction :

```
/** Cherche si un entier est présent dans un tableau
 * @param n un entier
 * @param des un tableau d'entiers
 * @return -1 si l'entier n n'est pas dans le tableau,
 *         l'indice de son emplacement dans le tableau sinon
 **/
int chercheDansTableau(int n, vector<int> des) {
    for (int i = 0; i < des.size(); i++) {
        if (des[i] == n) {
            return i;
        }
    }
    // Si on n'a pas trouvé l'entier on renvoie -1.
    return -1;
}
```

(3) Spécifiez et implantez une fonction `nombreOccurrences` qui prend en paramètre un tableau d'entiers `t` et un entier `v`, et qui renvoie le nombre d'occurrences de `v` dans `t` (combien de fois il apparaît).

Correction :

```
/** Compte le nombre d'occurrences d'un entier dans un tableau
 * @param un tableau d'entiers t
 * @param un entier v
 * @return un entier
 **/
int nombreOccurences(vector<int> des, int v) {
    int res = 0;
    for (int i = 0; i < des.size(); i++) {
        if ( des[i] == v ) {
            res ++;
        }
    }
    return res;
}
```

Exercice 3.

(1) Observez les tests suivants et déduisez-en la spécification (rôle, entrées et sortie) de la fonction `compteDes` :

```
    CHECK( compteDes({1, 1, 1, 1, 1}) == vector<int>({5, 0, 0, 0, 0, 0}) );
    CHECK( compteDes({2, 2, 2, 2, 2}) == vector<int>({0, 5, 0, 0, 0, 0}) );
    CHECK( compteDes({3, 3, 3, 3, 3}) == vector<int>({0, 0, 5, 0, 0, 0}) );
    CHECK( compteDes({4, 4, 4, 4, 4}) == vector<int>({0, 0, 0, 5, 0, 0}) );
    CHECK( compteDes({5, 5, 5, 5, 5}) == vector<int>({0, 0, 0, 0, 5, 0}) );
    CHECK( compteDes({6, 6, 6, 6, 6}) == vector<int>({0, 0, 0, 0, 0, 5}) );
    CHECK( compteDes({1, 2, 3, 4, 5}) == vector<int>({1, 1, 1, 1, 1, 0}) );
    CHECK( compteDes({2, 2, 6, 2, 2}) == vector<int>({0, 4, 0, 0, 0, 1}) );
    CHECK( compteDes({4, 1, 4, 1, 1}) == vector<int>({3, 0, 0, 2, 0, 0}) );
```

(2) Proposez une implantation de cette fonction.

Correction :

```
vector<int> compteDes(vector<int> des) {
    vector<int> res = {0, 0, 0, 0, 0, 0};
    for (int i = 0; i < des.size(); i++) {
        // les indices des cases vont de 0 à 5, les valeurs de dé de 1 à
6
        int indice = des[i] - 1;
        res[indice] += 1;
    }
    return res;
}
```

Exercice 4 (Yam's !)**.**

(1) Le yam's (cinq chiffres identiques) est la figure la plus facile à reconnaître. Spécifiez et implantez une fonction `pointsFigureYams` qui, lorsqu'on lui donne en entrée un tableau contenant 5 entiers, renvoie les points obtenus (**somme des 5 dés + 60**) s'il s'agit d'un yam's, **0** sinon.

Correction :

```
int pointsFigureYams(vector<int> des) {
    // Variable locale :
    int val = des[0];  // Pour vérifier que tous les dés sont égaux au premier

    for (int i = 1; i < des.size(); i++) {
        if (val  != des[i]) {
            // Si un dé est différent du premier alors pas de yams
            return 0;
        }
    }
    return val * 5 + 60 ;
}
```

(2) Complétez la liste de tests suivante avec au moins deux autres cas que vous jugez intéressants :

```
CHECK( pointsFigureYams({4,4,4,4,4}) == 80 );
CHECK( pointsFigureYams({1,1,1,1,1}) == 65 );
```

Correction :

```
    CHECK( pointsFigureYams({4, 4, 4, 4, 5}) == 0 );
    CHECK( pointsFigureYams({2, 1, 1, 1, 1}) == 0 );
```

(3) Pour simplifier la fonction `pointsFigureYams`, on peut utiliser les fonctions `compteDes` et `chercheDansTableau`. Donnez une nouvelle implantation de `pointsFigureYams`.

Correction :

```
int pointsFigureYams2(vector<int> des) {
    int indiceYams = chercheDansTableau(5, compteDes(des));

    if (indiceYams >= 0) {
        return (indiceYams + 1) * 5 + 60 ;
        // +1 car l'indice d'une case du tableau correspond à la valeur - 1
    } else {
        return 0;
    }
}
```

Exercice 5 (Brelan)**.**

(1) À l'image de la question 2 de l'exercice précédent, proposez des tests pour une fonction `pointsFigureBrelan` qui, lorsqu'on lui donne en entrée un tableau contenant 5 entiers, renvoie les points obtenus (**somme des 3 dés qui forment un brelan + 10**) s'il s'agit d'un brelan, **0** sinon.

Correction :

```
    CHECK( pointsFigureBrelan({4, 4, 4, 1, 1}) == 22 );
    CHECK( pointsFigureBrelan({1, 1, 4, 4, 5}) == 0 );
    CHECK( pointsFigureBrelan({1, 1, 1, 1, 5}) == 13 );
```

(2) Spécifiez et implantez cette fonction en vous aidant de la fonction `compteDes`.

Correction :

```
int pointsFigureBrelan(vector<int> des) {
    // On commence par compter les dés.
    vector<int> compte = compteDes(des);

    // On regarde si on a un brelan (3 valeurs identiques au moins).
    for (int i = 0; i < compte.size(); i ++) {
        if (compte[i] >= 3) {
            // Si oui, on renvoie la valeur du brelan correspondant.
            // Rappelons que l'entier stocké en position i du tableau "compte"
            // correspond au nombre de dés qui affichent (i + 1)
            return (i + 1) * 3 + 10;
        }
    }
    // En l'absence de brelan on renvoie un score nul.
    return 0;
}
```

Exercice 6 (Le jeu)**.**

(1) Implantez une fonction `pointsFigure` qui, étant donné un tableau de cinq dés et le nom d'une figure parmi « brelan » et « yams », renvoie le score associé en appelant respectivement la fonction `pointsFigureBrelan` ou la fonction `pointsFigureYams`. Cette fonction doit renvoyer **0** si le nom de figure entré n'est pas valide.

Correction :

```
int pointsFigure(vector<int> des, string figure) {
    if (figure == "brelan") {
        return pointsFigureBrelan(des);
    } else if (figure == "yams") {
        return pointsFigureYams(des);
    } else if (figure == "full") {  // En question supplémentaire du TP
        return pointsFigureFull(des);
    } else if (figure == "carre") { // En question supplémentaire du TP
        return pointsFigureCarre(des);
    }  else {
        return 0;
    }
}
```

(2) Donner la spécification et l'implantation d'une fonction `lanceDes` qui renvoie un tableau contenant cinq entiers choisis aléatoirement entre 1 et 6. Pour cela vous pouvez utiliser une fonction `int aleaInt(int a, int b)` qui étant donné deux entiers a et b renvoie un entier aléatoire n tel que $a \leq n \leq b$.

Correction :

```
vector<int> lanceDes() {
    vector<int> des;
    des = vector<int>(5);

    for (int i = 0; i < des.size(); i++) {
        des[i] = aleaInt(1, 6);
    }

    return des;
}
```

(3) Complétez le squelette de code présent dans la **Figure 1** (3 endroits à modifier) pour :
— lancer les dés ;
— afficher le tirage au joueur et lui demander d'entrer une figure **tant que** sa réponse est différente de « brelan », « yams » et « exit » ;
— si le joueur choisit « brelan » ou « yams », afficher les points qu'il marque.

Correction :

```
    string reponseJoueur = "";
    vector<int> des;

    des = lanceDes();

    while (reponseJoueur  != "brelan"
           and reponseJoueur  != "yams"
```

```
            and reponseJoueur != "exit") {

        afficheDes(des);
        cout << "Quelle figure choisissez-vous ? (brelan ou yams, exit pour quitter)
             << endl;

        // L'instruction suivante attend que le joueur tape
        // une phrase au clavier, puis stocke cette phrase
        // dans une chaine de caractères "reponseJoueur"
        cin >> reponseJoueur;
    }

    if (reponseJoueur != "exit") {
        cout << "Vous marquez " << pointsFigure(des,
                                                reponseJoueur) << " points !" << end
    }
    return 0;
```

(4) Une partie de Yam's consiste en de nombreux lancers successifs des dés. Introduisez une boucle supplémentaire pour refléter ce comportement tant que le joueur ne tape pas « exit ». Ajoutez un calcul du score total de la partie, qui est la somme des scores de chaque lancer.

Correction :

```
    string reponseJoueur = "";
    int score = 0;
    vector<int> des;

    do {
        reponseJoueur = "";
        des = lanceDes();

        while (reponseJoueur != "brelan"
               and reponseJoueur != "yams"
               and reponseJoueur != "exit") {

            afficheDes(des);
            cout << "Quelle figure choisissez-vous ? (brelan ou yams, exit pour quit
                 << endl;

            // L'instruction suivante permet d'attendre que le joueur
            // entre une phrase dans le terminal et stocke sa réponse
            // dans une chaine de caractères "reponseJoueur"
            cin >> reponseJoueur;
        }

        if (reponseJoueur != "exit") {
            int points = pointsFigure(des, reponseJoueur);
            score += points;
            cout << "Vous marquez " << points << " points. Score total: " << score <
                 endl;
        }
    } while (reponseJoueur != "exit");
```

```
    string reponseJoueur = "";
    vector<int> des;
    // INSERER VOTRE CODE ICI

    while (     reponseJoueur  != "brelan"
                and reponseJoueur  != "yams"
                and reponseJoueur  != "exit") {

        // INSERER VOTRE CODE ICI

        // L'instruction suivante permet d'attendre que le joueur
        // entre une phrase dans le terminal et stocke sa réponse
        // dans la chaine de caractères "reponseJoueur"
        cin >> reponseJoueur;
    }
    // INSERER VOTRE CODE ICI
```

Figure 1

Exercice ♣ 7 (Relance)**.**
Dans le vrai jeu de Yam's, le joueur peut relancer jusqu'à trois fois un ou plusieurs dés avant de choisir une figure.

(1) Ajoutez une fonction `vector<int> relance(int numDe, vector<int> des)` qui « relance » uniquement le dé numéro `numDe` choisi en premier argument et le remplace donc par un nouvel entier aléatoire entre 1 et 6.

Correction :

```
vector<int> relance(int numDe, vector<int> des) {
    des[numDe - 1] = aleaInt(1, 6);
    return des;
}
```

(2) Dans la boucle de jeu, ajoutez les instructions nécessaires pour que le joueur puisse choisir jusqu'à trois dés à relancer et les relancer.

Exercice ♣ 8 (Scores)**.**
La partie de Yam's se termine lorsqu'un joueur a marqué des points pour toutes les figures possibles.

(1) Ajoutez dans la fonction `main` un tableau de scores contenant une case pour chaque figure.

(2) Lorsque le joueur choisit une figure, les points qu'il gagne doivent être stockés dans la partie correspondante du tableau. Une fois une case du tableau remplie, elle ne peut plus être modifiée.

(3) La partie se termine lorsque toutes les cases du tableau sont remplies. Le score du joueur correspond à la somme des cases du tableau.