Aide mémoire - Épreuve terminale

- Guide aide mémoire que vous pourrez apporter à l'épreuve terminale et ainsi compléter avec les notes prises en classe.
- Ce guide sera le seul document admis en examen.
- Les cellulaires devrons être fermés complètement.
- Aucune visite à la salle de bain ne sera permise alors prévoyez-y un tour avant.

Boucles

for

```
for (initialisation; condition; incrémentation) {
    // instructions à exécuter à chaque itération
}
```

- Utilisée quand on connait le nombre d'itération à faire.
- `Initialisation` : Cette section initialise une ou plusieurs variables avant que la boucle ne démarre (souvent un compteur).
- `Condition` : Tant que cette condition est vraie, la boucle continue de s'exécuter. Quand la condition devient fausse, la boucle s'arrête.
- `Incrémentation` : À chaque itération, cette étape est exécutée après le bloc de code. Elle permet de modifier la ou les variables du compteur, par exemple en les incrémentant.

while

```
while (condition) {
    // instructions à répéter
}
```

- Utilisée lorsque l'on ne connait pas le nombre total d'itération.
- `condition` : une expression booléenne qui est évaluée avant chaque itération de la boucle.
    - Si la condition est vraie (true), le bloc d'instructions est exécuté.
    - Si la condition est fausse (false), la boucle s'arrête.

do...while

```
do {
    // Instructions à exécuter
} while (condition);
```

- Utilisée lorsque l'on ne connait pas le nombre total d'itération qu'il y aura mais qu'il faut qu'au moins une itération soit faite.
- `condition` : une expression booléenne qui est évaluée avant chaque itération de la boucle.
    - Si la condition est vraie (true), le bloc d'instructions est exécuté.
    - Si la condition est fausse (false), la boucle s'arrête.

Tableaux

```
void utilisationTableau() {
    int notes[] = { 90, 89, 75, 99, 95 };
    cout << notes; // <-- ceci est l'adresse mémoire du tableau de
notes.\n";
    cout << notes[0]; // " <-- ceci est la valeur de la 1ère note (index =
0).\n";
    cout << notes[4]; // " <-- ceci est la valeur de la dernière note
(index = 4).\n";
    cout << notes[5]; // " <-- l'index 5 n'existe pas (OutOfBound).\n";
  notes[4] = 100; //Injection d'une valeur dans le tableau (à la fin)
    cout << "Il y a " << size(notes) << " notes entrées"; //size() pour
connaître le nombre d'éléments dans le tableau
}
```

Valeurs aléatoires

```
srand((unsigned int)time(0)); //Pour utiliser le temps actuel pour générer
le seed
int nombreAleatoire = (rand() % (max - min + 1)) + min
```

Lire un seul caractère du clavier

```
#include <conio.h>

void accelererDecelerer() {
    char touche;
    cout << "Utilisez les flèches W / S pour accélérer ou décélérer (esc
pour quitter)." << endl;
    do {
        touche = _getch();
        //Permettra d'effacer le texte de la console.
        system("CLS");
        switch (touche) {
            case 'w': cout << "Accélérer" << endl; break;
            case 's': cout << "Décélérer" << endl; break;
        }
    } while (touche != 27); //27 est le code ASCII pour la touche ESC
}
```

Formatter l'affichage

- `fixed`: s'assure de pouvoir fixer le nombre de décimal désirées.
- `setprecision`: si utilisée avec `fixed`, précise le nombre de décimal à afficher.
- `setw()`: fonction qui formate le texte qui suit en lui attribuant un nombre fixe de case d'affichage. Utile pour aligner des montant par exemple.
- `left`: positionnera le texte suivant le `<<`, à gauche.
- `right`: positionnera le texte suivant le `<<`, à droite.

```
#include <iomanip>

void affichageFormate() {
    double notes[] = { 100, 89.4, 75.1, 99, 95.2 };
    for (int i = 0; i < size(notes); i++) {
        cout << "Note #" << i + 1 << " : " << fixed << setprecision(1) <<
setw(5) << right << notes[i] << endl;
    }
}
```

Résultat en sortie

```
Note #1 : 100.0
Note #2 :  89.4
Note #3 :  75.1
Note #4 :  99.0
Note #5 :  95.2
```

Fonctions

- une fonction sans retour utilise le mot réservé `void` et n'a pas de `return`

Fonctions sans paramètres:

```
[type de retour] maFonction1(){
    //instructions
    return ... ; //si nécessaire
}
```

Fonctions avec paramètres

(ajouter une & entre le type de param et le nom de celui-ci pour passer par référence)

```
[type de retour] maFonction2([type de param1] nomParam1, [type de param2]
nomParam2, ...){
    //instructions
    return ...; //si nécessaire
}
```

avec tableau classique:

```
[type de retour] maFonction6([type de tableau]  tableau[], int taille,
[autre param] nomParam, ... ){
    //instructions
    return ...; //si nécessaire
}
```

Structs

Nous avons vu en classe qu'il était possible d'imbriquer des `struct` dans d'autres `struct`. Il est également possible d'y ajouter une fonction membre.

```
struct Evaluation {
    string codeEvaluation;
    string titreCourt;
    string titre;
    int ponderation;
};

struct Correction {
    string codeEvaluation;
    int noteObtenue;
};

struct Etudiant {
    string DA;
    string nom;
    string prenom;
    Correction corrections[NOMBRE_EVALUATIONS];

    float resultat() {
        /* Code de la fonction ici */
        return round(somme * 10) / 10;
    };
};
```

Fichiers

Ecriture

```
const string PLANET_DATA_FILE = "planets.csv";
const int PLANET_COUNT = 8;

struct Planet {
    string name = "";
    float weight = 0; // En masses terrestres (exemple)
    int size = 0;     // En diamètre en kilomètres
};

Planet planets[PLANET_COUNT] = {
        {"Mercure", 0.055, 4879},
        {"Vénus", 0.815, 12104},
        {"Terre", 1.0, 12756},
        {"Mars", 0.107, 6792},
        {"Jupiter", 317.8, 142984},
        {"Saturne", 95.2, 120536},
        {"Uranus", 14.5, 51118},
        {"Neptune", 17.1, 49528}
};

void ecrirePlanetesCSV() {
    ofstream fichier(PLANET_DATA_FILE);

    if (fichier.is_open()) {
        for (int i = 0; i < PLANET_COUNT; i++) {
            fichier << planets[i].name << ","
                << planets[i].weight << ","
                << planets[i].size << endl;
        }
    }
    else {
        cout << "Impossible d'ouvrir le fichier " << PLANET_DATA_FILE <<
endl;
    }
}
```

Lecture

```
  void lirePlanetesCSV() {
    Planet planets[PLANET_COUNT];
    ifstream fichier(PLANET_DATA_FILE);
    string donnee;
    for (int i = 0; i < PLANET_COUNT; i++) {
        getline(fichier, donnee, ',');
        planets[i].name = donnee;
        getline(fichier, donnee, ',');
        planets[i].weight = stof(donnee);  //stof() Transforme la string
en float
        getline(fichier, donnee);
        planets[i].size = stoi(donnee);      //stoi() Transforme la string
en int
    }
}
```

ATENTION: La fonction `file.get()` permet de retirer un caractère de la chaîne de caractère lue à partir d'un fichier. Par exemple un `enter` ou une `virgule`.

Espace de notes complémentaires