Chapitre 4 : Les Entrées/Sorties

Objectifs pédagogiques :

- Maîtriser les flux d'entrée/sortie standard (console)
- Savoir formater l'affichage pour des résultats professionnels
- Gérer correctement la saisie utilisateur, notamment les chaînes de caractères
- Comprendre la différence entre les entrées formatées et non formatées
- Appliquer ces notions à des problèmes concrets d'automatique

1. Introduction aux flux (streams)

En C++, les entrées et sorties sont gérées par le concept de flux (streams). Un flux est une séquence d'octets qui circulent entre le programme et un périphérique (clavier, écran, fichier, etc.).

Les principaux flux standards sont :

- cout : sortie standard (généralement l'écran)
- cin : entrée standard (généralement le clavier)
- cerr : sortie d'erreur standard (également l'écran, mais non bufferisée)
- clog : version bufferisée de cerr

Pour utiliser ces flux, il faut inclure la bibliothèque <iostream> :

2. La sortie avec `cout`

2.1. Affichage simple

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

int main() {
    // Affichage de texte simple
    cout << "Bonjour le monde !";

    // Avec retour à la ligne
    cout << "Première ligne" << endl;
    cout << "Deuxième ligne\n";  // \n = retour à la ligne (plus léger que endl)

    // Chaînage multiple
    cout << "Le résultat est : " << 42 << " et c'est fini." << endl;

    return 0;
}
```

Différence entre endl et \n :

- \n : insère un retour à la ligne
- endl : insère un retour à la ligne ET vide le tampon (flush)
- En pratique, \n est plus efficace, endl est utile quand on veut forcer l'affichage immédiat

2.2. Affichage de variables

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

int main() {
    // Variables typiques en automatique
    double temperature = 23.5;
    int compteur = 42;
    bool moteur_en_marche = true;
    char code = 'A';

    // Affichage simple
    cout << "Température : " << temperature << " °C" << endl;
    cout << "Compteur : " << compteur << endl;

    // Les booléens affichent 1 (true) ou 0 (false) par défaut
    cout << "Moteur en marche : " << moteur_en_marche << endl;  // Affiche 1

    // Caractère
    cout << "Code erreur : " << code << endl;

    return 0;
}
```

2.3. Formatage de l'affichage

Pour un affichage professionnel, surtout en automatique où on manipule des mesures, le formatage est crucial.

```
#include <iostream>
#include <iomanip>  // Pour les manipulateurs de format
using namespace std;

int main() {
    double mesure1 = 12.345678;
    double mesure2 = 123.4;
    double mesure3 = 1.23456;

    // Sans formatage - affichage brut
    cout << "Sans formatage :" << endl;
```

```
    cout << mesure1 << endl;
    cout << mesure2 << endl;
    cout << mesure3 << endl;
    cout << endl;

    // Avec formatage - fixed et setprecision
    cout << fixed;  // Force l'affichage en notation décimale (pas scientifique)
    cout << setprecision(3);  // 3 chiffres après la virgule

    cout << "Avec 3 décimales :" << endl;
    cout << mesure1 << endl;  // 12.346
    cout << mesure2 << endl;  // 123.400
    cout << mesure3 << endl;  // 1.235
    cout << endl;

    // setprecision seul (sans fixed) donne le nombre de chiffres significatifs
    cout << setprecision(4);  // 4 chiffres significatifs
    cout << "4 chiffres significatifs :" << endl;
    cout << mesure1 << endl;  // 12.35
    cout << mesure2 << endl;  // 123.4
    cout << mesure3 << endl;  // 1.235
    cout << endl;

    // Largeur de champ
    cout << setprecision(3) << fixed;
    cout << "Avec largeur de champ :" << endl;
    cout << setw(10) << mesure1 << setw(10) << mesure2 << setw(10) << mesure3 << endl;
    // setw(10) : réserve 10 caractères pour l'affichage (aligné à droite par défaut)

    // Alignement à gauche
    cout << left;
    cout << setw(10) << mesure1 << setw(10) << mesure2 << setw(10) << mesure3 << endl;

    return 0;
}
```

2.4. La sortie d'erreur cerr

cerr est utilisé pour les messages d'erreur. Contrairement à cout, il n'est pas bufferisé, donc les messages apparaissent immédiatement.

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

int main() {
```

```
    double tension;

    cout << "Entrez la tension (0-24V) : ";
    cin >> tension;

    if (tension < 0 || tension > 24) {
        cerr << "ERREUR : Tension hors plage !" << endl;
        return 1;  // Code retour d'erreur
    }

    cout << "Tension valide : " << tension << " V" << endl;
    return 0;
}
```

3. L'entrée avec `cin`

3.1. Saisie simple de nombres

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

int main() {
    double gain, constante_temps;
    int nb_mesures;

    cout << "=== Paramètres du système ===" << endl;

    cout << "Entrez le gain statique K : ";
    cin >> gain;

    cout << "Entrez la constante de temps tau (s) : ";
    cin >> constante_temps;

    cout << "Entrez le nombre de mesures à simuler : ";
    cin >> nb_mesures;

    cout << endl;
    cout << "Récapitulatif :" << endl;
    cout << "K = " << gain << endl;
    cout << "tau = " << constante_temps << " s" << endl;
    cout << "Nombre de mesures = " << nb_mesures << endl;

    return 0;
}
```

3.2. Saisie multiple

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

int main() {
    double kp, ki, kd;

    cout << "Entrez les gains PID (kp ki kd) : ";
    cin >> kp >> ki >> kd;  // Saisie séparée par des espaces

    cout << "PID configuré : " << endl;
    cout << "kp = " << kp << ", ki = " << ki << ", kd = " << kd << endl;

    return 0;
}
```

3.3. Problèmes avec cin

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

int main() {
    int age;
    char nom[50];

    cout << "Entrez votre age : ";
    cin >> age;  // L'utilisateur tape "25\n" -> "25" est lue, '\n' reste

    cout << "Entrez votre nom : ";
    cin.getline(nom, 50);  // Problème : lit le '\n' restant, pas le nom !

    cout << "Age: " << age << ", Nom: " << nom << endl;  // Nom vide !

    return 0;
}
```

Solution : Vider le tampon après une saisie numérique

```
#include <iostream>
#include <limits>  // Pour numeric_limits
using namespace std;

int main() {
    int age;
    char nom[50];
```

```
    cout << "Entrez votre age : ";
    cin >> age;

    // Vider le tampon (ignorer tous les caractères jusqu'au \n inclus)
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    cout << "Entrez votre nom : ";
    cin.getline(nom, 50);

    cout << "Age: " << age << ", Nom: " << nom << endl;

    return 0;
}
```

Problème 2 : Saisie erronée

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

int main() {
    double valeur;

    cout << "Entrez une valeur numérique : ";
    cin >> valeur;

    // Vérifier si la saisie a échoué
    if (cin.fail()) {
        cout << "Erreur de saisie !" << endl;

        // Réinitialiser l'état d'erreur de cin
        cin.clear();

        // Vider le tampon
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    } else {
        cout << "Valeur saisie : " << valeur << endl;
    }

    return 0;
}
```

3.4. Saisie robuste avec validation

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

double saisie_securisee(string message, double min_val, double max_val) {
    double valeur;
    bool saisie_valide = false;

    do {
        cout << message;
        cin >> valeur;

        if (cin.fail()) {
            cout << "Erreur : veuillez entrer un nombre." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            saisie_valide = false;
        } else if (valeur < min_val || valeur > max_val) {
            cout << "Erreur : la valeur doit être entre "
                 << min_val << " et " << max_val << endl;
            saisie_valide = false;
        } else {
            saisie_valide = true;
        }

    } while (!saisie_valide);

    return valeur;
}

int main() {
    double tension = saisie_securisee("Tension (0-24V) : ", 0.0, 24.0);
    double courant = saisie_securisee("Courant (0-10A) : ", 0.0, 10.0);

    double puissance = tension * courant;
    cout << "Puissance : " << puissance << " W" << endl;

    return 0;
}
```

4. Les chaînes de caractères

4.1. Chaînes C-style (ancienne méthode)

```cpp
#include <iostream>
#include <cstring>  // Pour les fonctions sur les chaînes C
using namespace std;

int main() {
    // Déclaration de chaînes C-style
    char nom[50];                // Tableau de caractères
    char message[] = "Bonjour";  // Initialisation automatique

    // Problème : pas de contrôle de dépassement
    cout << "Entrez votre nom : ";
    cin >> nom;  // Si l'utilisateur tape plus de 49 caractères -> débordement !

    // Pour lire une ligne entière (avec espaces)
    cout << "Entrez votre adresse : ";
    cin.ignore();  // Vider le \n précédent
    cin.getline(nom, 50);  // Lit au maximum 49 caractères

    // Fonctions de manipulation (dangereuses)
    strcpy(nom, "Dupont");  // Copie (risque de débordement)
    strcat(nom, " Jean");   // Concaténation (risque)
    int longueur = strlen(nom);  // Longueur sans le \0 final

    cout << "Nom: " << nom << ", longueur: " << longueur << endl;

    return 0;
}
```

Problèmes des chaînes C-style :

- Risque de débordement mémoire
- Gestion manuelle de la mémoire
- Fonctions peu sûres

4.2. La classe string (recommandée)

Le C++ moderne utilise la classe string (bibliothèque <string>), beaucoup plus sûre et pratique.

```cpp
#include <iostream>
#include <string>  // Pour la classe string
using namespace std;

int main() {
    // Déclaration et initialisation
```

```
    string nom;
    string prenom = "Jean";
    string message("Bonjour le monde");

    // Saisie simple (s'arrête au premier espace)
    cout << "Entrez votre nom : ";
    cin >> nom;
    cout << "Bonjour " << nom << endl;

    // Saisie d'une ligne entière (avec espaces)
    string adresse;
    cout << "Entrez votre adresse complète : ";
    cin.ignore();  // Vider le tampon
    getline(cin, adresse);  // Fonction spéciale pour string
    cout << "Adresse : " << adresse << endl;

    return 0;
}
```

4.3. Opérations sur les string

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

int main() {
    string nom = "Martin";
    string prenom = "Sophie";

    // Concaténation
    string nom_complet = prenom + " " + nom;
    cout << "Nom complet : " << nom_complet << endl;

    // Longueur
    cout << "Longueur du nom : " << nom.length() << endl;
    cout << "Longueur du prénom : " << prenom.size() << endl;  // équivalent

    // Accès aux caractères
    cout << "Première lettre : " << nom_complet[0] << endl;
    cout << "Dernière lettre : " << nom_complet[nom_complet.length() - 1] << endl;

    // Modification
    nom_complet[0] = 'm';  // Attention : 'm' pas "m"
    cout << "Après modification : " << nom_complet << endl;
```

```
    // Comparaison
    string entree1 = "abc";
    string entree2 = "abd";

    if (entree1 == entree2) {
        cout << "Identiques" << endl;
    } else if (entree1 < entree2) {
        cout << entree1 << " < " << entree2 << endl;
    } else {
        cout << entree1 << " > " << entree2 << endl;
    }

    // Recherche
    string phrase = "Le système est en marche";
    size_t pos = phrase.find("système");

    if (pos != string::npos) {  // npos = "not found"
        cout << "Trouvé à la position " << pos << endl;
    } else {
        cout << "Non trouvé" << endl;
    }

    // Extraction de sous-chaîne
    string extrait = phrase.substr(3, 7);  // 7 caractères à partir de l'index 3
    cout << "Extrait : '" << extrait << "'" << endl;

    return 0;
}
```

4.4. Conversion string ↔ nombres

```
#include <iostream>
#include <string>
#include <sstream>  // Pour stringstream
using namespace std;

int main() {
    // Conversion string -> nombre
    string str_valeur = "123.45";

    // Méthode 1 : stringstream
    double valeur1;
    stringstream(str_valeur) >> valeur1;
```

```
    cout << "Valeur1 = " << valeur1 << endl;

    // Méthode 2 : stod (C++11)
    double valeur2 = stod(str_valeur);
    cout << "Valeur2 = " << valeur2 << endl;

    // Conversion nombre -> string
    double mesure = 45.67;

    // Méthode 1 : stringstream
    stringstream ss;
    ss << "Mesure : " << mesure << " °C";
    string message = ss.str();
    cout << message << endl;

    // Méthode 2 : to_string (C++11)
    string str_mesure = to_string(mesure);
    cout << "Chaîne obtenue : " << str_mesure << endl;

    // Gestion d'erreurs de conversion
    string texte = "abc123";
    try {
        double valeur = stod(texte);
        cout << "Valeur = " << valeur << endl;
    } catch (const invalid_argument& e) {
        cerr << "Erreur : conversion impossible" << endl;
    }

    return 0;
}
```

4.5. Exemple pratique : Saisie de commandes

```
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
using namespace std;

int main() {
    string commande;
    double consigne = 0.0;
    bool continuer = true;
```

```
    cout << "=== Interface de commande ===" << endl;
    cout << "Commandes disponibles :" << endl;
    cout << "  set <valeur> - fixe la consigne" << endl;
    cout << "  status       - affiche l'état" << endl;
    cout << "  quit         - quitter" << endl;
    cout << endl;

    while (continuer) {
        cout << "> ";
        getline(cin, commande);

        if (commande == "quit") {
            continuer = false;
            cout << "Arrêt du programme." << endl;
        }
        else if (commande == "status") {
            cout << fixed << setprecision(2);
            cout << "Consigne actuelle : " << consigne << endl;
        }
        else if (commande.substr(0, 3) == "set") {
            // Extraire la valeur après "set "
            string str_valeur = commande.substr(4);

            try {
                consigne = stod(str_valeur);
                cout << "Consigne fixée à " << consigne << endl;
            } catch (...) {
                cout << "Erreur : valeur invalide" << endl;
            }
        }
        else if (!commande.empty()) {
            cout << "Commande inconnue : " << commande << endl;
        }
    }

    return 0;
}
```

5. Applications pratiques en automatique

5.1. Acquisition et affichage de mesures

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
using namespace std;

int main() {
    const int NB_MESURES = 5;
    double mesures[NB_MESURES];
    double somme = 0.0;

    cout << "=== Acquisition de mesures ===" << endl;
    cout << fixed << setprecision(3);

    // Saisie des mesures
    for (int i = 0; i < NB_MESURES; i++) {
        bool saisie_valide = false;

        do {
            cout << "Mesure " << (i+1) << " : ";
            cin >> mesures[i];

            if (cin.fail()) {
                cout << "Erreur : veuillez entrer un nombre" << endl;
                cin.clear();
                cin.ignore(10000, '\n');
            } else {
                saisie_valide = true;
                somme += mesures[i];
            }
        } while (!saisie_valide);
    }

    // Calculs statistiques
    double moyenne = somme / NB_MESURES;
    double variance = 0.0;

    for (int i = 0; i < NB_MESURES; i++) {
        variance += pow(mesures[i] - moyenne, 2);
    }
```

```
    variance /= NB_MESURES;
    double ecart_type = sqrt(variance);

    // Affichage des résultats
    cout << "\n=== Résultats ===" << endl;
    cout << setw(15) << left << "Mesure"
         << setw(15) << right << "Valeur" << endl;
    cout << string(30, '-') << endl;

    for (int i = 0; i < NB_MESURES; i++) {
        cout << setw(15) << left << ("Mesure " + to_string(i+1))
             << setw(15) << right << mesures[i] << endl;
    }

    cout << string(30, '-') << endl;
    cout << setw(15) << left << "Moyenne"
         << setw(15) << right << moyenne << endl;
    cout << setw(15) << left << "Écart-type"
         << setw(15) << right << ecart_type << endl;

    return 0;
}
```

5.2. Interface de réglage de PID

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;

class RegulateurPID {
public:
    double kp, ki, kd;
    double consigne;
    double sortie;

    RegulateurPID() : kp(1.0), ki(0.0), kd(0.0), consigne(0.0), sortie(0.0) {}

    void afficher_parametres() {
        cout << fixed << setprecision(3);
        cout << "\n=== Paramètres PID actuels ===" << endl;
        cout << setw(10) << left << "Kp:" << setw(10) << right << kp << endl;
        cout << setw(10) << left << "Ki:" << setw(10) << right << ki << endl;
```

```
        cout << setw(10) << left << "Kd:" << setw(10) << right << kd << endl;
        cout << setw(10) << left << "Consigne:" << setw(10) << right << consigne << endl;
    }

    void regler(string parametre, double valeur) {
        if (parametre == "kp") kp = valeur;
        else if (parametre == "ki") ki = valeur;
        else if (parametre == "kd") kd = valeur;
        else if (parametre == "consigne") consigne = valeur;
        else cout << "Paramètre inconnu" << endl;
    }
};

int main() {
    RegulateurPID pid;
    string commande;

    cout << "=== Interface de réglage PID ===" << endl;
    cout << "Commandes :" << endl;
    cout << "  show                  - afficher paramètres" << endl;
    cout << "  set <param> <valeur> - régler (kp, ki, kd, consigne)" << endl;
    cout << "  quit                  - quitter" << endl;
    cout << endl;

    while (true) {
        cout << "\nPID> ";
        getline(cin, commande);

        if (commande == "quit") {
            break;
        }
        else if (commande == "show") {
            pid.afficher_parametres();
        }
        else if (commande.substr(0, 3) == "set") {
            stringstream ss(commande);
            string cmd, param;
            double valeur;

            ss >> cmd >> param >> valeur;

            if (ss.fail()) {
                cout << "Format : set <param> <valeur>" << endl;
```

```
            } else {
                pid.regler(param, valeur);
                cout << param << " = " << valeur << endl;
            }
        }
        else if (!commande.empty()) {
            cout << "Commande inconnue" << endl;
        }
    }

    return 0;
}
```

6. Synthèse

Élément	Rôle	Points clés
`cout`	Sortie écran	`<<`, `endl`, `\n`
`cerr`	Sortie erreur	Non bufferisé
`cin`	Entrée clavier	`>>`, attention au tampon
`cin.ignore()`	Vider tampon	Après saisie numérique
`cin.fail()`	Test erreur saisie	À vérifier systématiquement
`<iomanip>`	Formatage	`setprecision`, `setw`, `fixed`
`string`	Chaînes modernes	Sûr, pratique, `getline()`
`stod()`, `to_string()`	Conversions	C++11, gestion d'erreurs

Règles d'or :

- Toujours vérifier cin.fail() après une saisie critique
- Utiliser cin.ignore() après cin >> si vous utilisez getline() ensuite
- Préférer string aux char[] pour les chaînes
- Formater les sorties pour les rendre lisibles
- En automatique, toujours afficher les unités !