Contents

Exercices C++ - Guide Complet d'Apprentissage

De débutant à avancé - Apprentissage progressif

Table des matières

Niveau 1 : Bases fondamentales

Théorie

Variables et types de données

```cpp
int entier = 42;           // Nombre entier
double decimal = 3.14;     // Nombre décimal
char caractere = 'A';      // Caractère unique
bool booleen = true;       // Vrai ou faux
std::string texte = "Hello"; // Chaîne de caractères
```

Opérateurs arithmétiques

```cpp
int a = 10, b = 3;
int somme = a + b;      // 13
int difference = a - b; // 7
int produit = a * b;    // 30
int quotient = a / b;   // 3 (division entière)
int reste = a % b;      // 1 (modulo)
```

Structures de contrôle

```cpp
// if-else
if (condition) {
    // Code si condition vraie
} else {
    // Code sinon
}

// switch
switch (variable) {
```

```
    case valeur1:
        // Code pour valeur1
        break;
    default:
        // Code par défaut
        break;
}
```

Boucles

```
// for
for (int i = 0; i < 10; i++) {
    // Code répété 10 fois
}

// while
while (condition) {
    // Code répété tant que condition vraie
}

// do-while
do {
    // Code exécuté au moins une fois
} while (condition);
```

Exercice 1.1 : Variables et opérateurs

Objectif : Maîtriser la déclaration de variables et l'utilisation des opérateurs.

Énoncé : Écrivez un programme qui : 1. Déclare les variables suivantes : - Un entier `age` avec la valeur 25 - Un double `taille` avec la valeur 1.75 - Un char `grade` avec la valeur 'A' - Un bool `est_etudiant` avec la valeur true - Une string `nom` avec la valeur "Alice"

2. Affiche toutes ces variables avec des messages explicatifs
3. Calcule et affiche :
    - La somme de `age` et 5
    - Le produit de `taille` par 100 (pour convertir en cm)
    - Si `age` est supérieur à 18
    - Si `est_etudiant` ET `age` est inférieur à 30
4. Utilise des structures if-else pour afficher :
    - "Majeur" si age >= 18, sinon "Mineur"
    - "Étudiant" si est_etudiant est true, sinon "Non étudiant"
5. Utilise une boucle for pour afficher les nombres de 1 à 5
6. Utilise une boucle while pour afficher les nombres pairs de 2 à 10

Code de base :

```cpp
#include <iostream>
#include <string>

int main() {
    // Votre code ici
    return 0;
}
```

Exercice 1.2 : Calculateur simple

Objectif : Créer un programme interactif utilisant les structures de contrôle.

Énoncé : Créez un calculateur simple qui : 1. Demande à l'utilisateur d'entrer deux nombres 2. Demande de choisir une opération (1:+, 2:-, 3:*, 4:/, 5:%) 3. Utilise une structure switch pour effectuer l'opération 4. Affiche le résultat 5. Gère le cas de division par zéro

Code de base :

```cpp
#include <iostream>

int main() {
    double nombre1, nombre2;
    int operation;

    // Votre code ici

    return 0;
}
```

Exercice 1.3 : Algorithmes de base

Objectif : Implémenter des algorithmes classiques.

Énoncé : Créez un programme qui : 1. Trouve le plus grand de trois nombres 2. Vérifie si un nombre est premier 3. Calcule la factorielle d'un nombre 4. Affiche la table de multiplication d'un nombre

---

Niveau 2 : Fonctions et portée

Théorie

Définition de fonctions

```cpp
// Fonction simple
int addition(int a, int b) {
    return a + b;
```

```
}

// Fonction avec paramètres par défaut
void afficherMessage(std::string message = "Hello") {
    std::cout << message << std::endl;
}

// Fonction surchargée
int max(int a, int b) {
    return (a > b) ? a : b;
}

double max(double a, double b) {
    return (a > b) ? a : b;
}
```

Portée des variables

```
int variable_globale = 10;  // Variable globale

void fonction() {
    int variable_locale = 20;  // Variable locale
    std::cout << variable_globale << std::endl;  // Accès à la globale
    std::cout << variable_locale << std::endl;   // Accès à la locale
}
```

Exercice 2.1 : Fonctions mathématiques

Objectif : Créer des fonctions pour des calculs mathématiques.

Énoncé : Créez les fonctions suivantes : 1. `double carre(double x)` - retourne x^2 2. `double racine_carree(double x)` - retourne $\sqrt{x}$ 3. `int puissance(int base, int exposant)` - retourne base^exposant 4. `bool est_pair(int n)` - retourne true si n est pair 5. `int pgcd(int a, int b)` - retourne le PGCD de a et b

Testez ces fonctions dans le main.

Code de base :

```
#include <iostream>
#include <cmath>

// Déclarez vos fonctions ici

int main() {
    // Testez vos fonctions ici
    return 0;
}
```

Exercice 2.2 : Fonctions avec surcharge

Objectif : Maîtriser la surcharge de fonctions.

Énoncé : Créez des fonctions surchargées pour : 1. `afficher()` - affiche “Hello” 2. `afficher(int n)` - affiche le nombre n 3. `afficher(std::string message)` - affiche le message 4. `afficher(int n, std::string message)` - affiche le nombre et le message

Créez aussi des fonctions pour : 1. `calculer_moyenne(int a, int b)` - moyenne de deux entiers 2. `calculer_moyenne(double a, double b)` - moyenne de deux doubles 3. `calculer_moyenne(int a, int b, int c)` - moyenne de trois entiers

Exercice 2.3 : Récursivité

Objectif : Comprendre et implémenter la récursivité.

Énoncé : Implémentez les fonctions récursives suivantes : 1. `int factorielle_recursive(int n)` - factorielle récursive 2. `int fibonacci_recursive(int n)` - suite de Fibonacci 3. `int pgcd_recursive(int a, int b)` - PGCD récursif (algorithme d’Euclide) 4. `void afficher_compte_a_rebours(int n)` - affiche le compte à rebours de n à 0

Niveau 3 : Tableaux et chaînes

Théorie

Tableaux statiques

```cpp
int tableau[5] = {1, 2, 3, 4, 5};
int tableau2[] = {1, 2, 3};  // Taille automatique

// Accès aux éléments
int premier = tableau[0];
int dernier = tableau[4];

// Parcours
for (int i = 0; i < 5; i++) {
    std::cout << tableau[i] << " ";
}
```

Chaînes de caractères

```cpp
std::string texte = "Hello World";
std::string texte2 = texte + " C++";

// Longueur
int longueur = texte.length();

// Sous-chaîne
```

```cpp
std::string sous_chaine = texte.substr(0, 5);  // "Hello"

// Recherche
size_t position = texte.find("World");
```

Exercice 3.1 : Manipulation de tableaux

Objectif : Maîtriser les opérations sur les tableaux.

Énoncé : Créez un programme qui : 1. Déclare un tableau de 10 entiers 2. Remplit le tableau avec des valeurs aléatoires (1-100) 3. Affiche le tableau 4. Trouve le maximum et le minimum 5. Calcule la moyenne 6. Compte les nombres pairs et impairs 7. Inverse l'ordre du tableau 8. Trie le tableau (algorithme de tri à bulles)

Code de base :

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

int main() {
    srand(time(0));  // Initialise le générateur aléatoire
    int tableau[10];

    // Votre code ici

    return 0;
}
```

Exercice 3.2 : Manipulation de chaînes

Objectif : Maîtriser les opérations sur les chaînes.

Énoncé : Créez un programme qui : 1. Demande à l'utilisateur d'entrer une phrase 2. Affiche la longueur de la phrase 3. Convertit la phrase en majuscules 4. Convertit la phrase en minuscules 5. Compte le nombre de voyelles 6. Compte le nombre de mots 7. Inverse la phrase 8. Vérifie si la phrase est un palindrome

Exercice 3.3 : Tableaux multidimensionnels

Objectif : Travailler avec des tableaux 2D.

Énoncé : Créez un programme qui : 1. Déclare une matrice 3x3 2. Remplit la matrice avec des valeurs 3. Affiche la matrice 4. Calcule la somme de chaque ligne 5. Calcule la somme de chaque colonne 6. Calcule la somme de la diagonale principale 7. Transpose la matrice 8. Vérifie si la matrice est symétrique

---

Niveau 4 : Programmation orientée objet

Théorie

Classes et objets

```cpp
class Rectangle {
private:
    double largeur;
    double hauteur;

public:
    // Constructeur
    Rectangle(double l, double h) : largeur(l), hauteur(h) {}

    // Méthodes
    double calculerAire() {
        return largeur * hauteur;
    }

    double calculerPerimetre() {
        return 2 * (largeur + hauteur);
    }

    // Getters et setters
    double getLargeur() { return largeur; }
    void setLargeur(double l) { largeur = l; }
};
```

Héritage

```cpp
class Forme {
protected:
    std::string couleur;

public:
    Forme(std::string c) : couleur(c) {}
    virtual double calculerAire() = 0;  // Méthode virtuelle pure
};

class Cercle : public Forme {
private:
    double rayon;

public:
    Cercle(double r, std::string c) : Forme(c), rayon(r) {}

    double calculerAire() override {
        return 3.14159 * rayon * rayon;
```

```
    }
};
```

Exercice 4.1 : Classe Compte Bancaire

Objectif : Créer une classe simple avec encapsulation.

Énoncé : Créez une classe `CompteBancaire` avec : 1. Attributs privés : `numero`, `titulaire`, `solde` 2. Constructeur avec paramètres 3. Méthodes publiques : - `deposer(double montant)` - `retirer(double montant)` - `afficherSolde()` - `getSolde()` et `getTitulaire()` 4. Gestion des erreurs (solde insuffisant, montant négatif)

Code de base :

```cpp
#include <iostream>
#include <string>

class CompteBancaire {
private:
    std::string numero;
    std::string titulaire;
    double solde;

public:
    // Constructeur
    CompteBancaire(std::string num, std::string tit, double sld);

    // Méthodes
    void deposer(double montant);
    bool retirer(double montant);
    void afficherSolde();
    double getSolde();
    std::string getTitulaire();
};

int main() {
    // Testez votre classe ici
    return 0;
}
```

Exercice 4.2 : Hiérarchie de formes géométriques

Objectif : Maîtriser l'héritage et le polymorphisme.

Énoncé : Créez une hiérarchie de classes : 1. Classe abstraite `Forme` avec : - Attribut `couleur` - Méthode virtuelle pure `calculerAire()` - Méthode virtuelle pure `calculerPerimetre()`

2. Classe `Rectangle` héritant de `Forme` :
    - Attributs `largeur` et `hauteur`

- Implémentation des méthodes

3. Classe `Cercle` héritant de `Forme` :
   - Attribut `rayon`
   - Implémentation des méthodes
4. Classe `Triangle` héritant de `Forme` :
   - Attributs `cote1`, `cote2`, `cote3`
   - Implémentation des méthodes

Testez avec un tableau de pointeurs vers `Forme`.

Exercice 4.3 : Gestion d'une bibliothèque

Objectif : Créer un système d'objets complexes.

Énoncé : Créez un système de gestion de bibliothèque avec :

1. Classe `Livre` :
   - Attributs : titre, auteur, ISBN, disponible
   - Méthodes : emprunter(), retourner(), afficher()
2. Classe `Membre` :
   - Attributs : nom, numero, livres_empruntes[]
   - Méthodes : emprunterLivre(), retournerLivre(), afficher()
3. Classe `Bibliotheque` :
   - Attributs : livres[], membres[]
   - Méthodes : ajouterLivre(), ajouterMembre(), emprunterLivre(), retourner-Livre()

Niveau 5 : Pointeurs et gestion mémoire

Théorie

Pointeurs

```
int nombre = 42;
int* pointeur = &nombre;  // Adresse de nombre

std::cout << "Valeur: " << nombre << std::endl;
std::cout << "Adresse: " << pointeur << std::endl;
std::cout << "Valeur via pointeur: " << *pointeur << std::endl;

*pointeur = 100;  // Modifie la valeur de nombre
```

Allocation dynamique

```
// Allocation
int* tableau = new int[10];
int* nombre = new int(42);

// Utilisation
```

```
tableau[0] = 1;
*nombre = 100;

// Libération
delete[] tableau;
delete nombre;
```

Smart pointers (C++11+)

```
#include <memory>

// unique_ptr
std::unique_ptr<int> ptr1(new int(42));
auto ptr2 = std::make_unique<int>(42);

// shared_ptr
std::shared_ptr<int> ptr3 = std::make_shared<int>(42);
```

Exercice 5.1 : Manipulation de pointeurs

Objectif : Comprendre les pointeurs et les références.

Énoncé : Créez un programme qui : 1. Déclare une variable et un pointeur vers cette variable 2. Affiche la valeur, l'adresse et la valeur via le pointeur 3. Modifie la valeur via le pointeur 4. Utilise des références pour échanger deux valeurs 5. Crée un tableau dynamique de 5 entiers 6. Remplit le tableau avec des valeurs 7. Affiche le tableau 8. Libère la mémoire

Code de base :

```
#include <iostream>

void echanger(int& a, int& b) {
    // Implémentez l'échange
}

int main() {
    // Votre code ici
    return 0;
}
```

Exercice 5.2 : Allocation dynamique

Objectif : Maîtriser l'allocation et la libération de mémoire.

Énoncé : Créez un programme qui : 1. Demande à l'utilisateur la taille d'un tableau 2. Alloue dynamiquement un tableau de cette taille 3. Remplit le tableau avec des valeurs 4. Trouve le maximum et le minimum 5. Redimensionne le tableau (allocation d'un nouveau tableau plus grand) 6. Copie les anciennes valeurs 7. Ajoute de nouvelles valeurs 8. Libère correctement la mémoire

Exercice 5.3 : Smart pointers

Objectif : Utiliser les smart pointers modernes.

Énoncé : Créez un programme qui : 1. Utilise `std::unique_ptr` pour gérer un objet 2. Utilise `std::shared_ptr` pour partager un objet entre plusieurs variables 3. Utilise `std::weak_ptr` pour éviter les références circulaires 4. Crée une classe `Ressource` avec constructeur et destructeur 5. Teste la gestion automatique de la mémoire

---

Niveau 6 : STL et templates

Théorie

Conteneurs STL

```cpp
#include <vector>
#include <list>
#include <map>
#include <set>

// Vector
std::vector<int> nombres = {1, 2, 3, 4, 5};
nombres.push_back(6);
nombres.pop_back();

// Map
std::map<std::string, int> ages;
ages["Alice"] = 25;
ages["Bob"] = 30;

// Set
std::set<int> ensemble = {1, 2, 3, 3, 4};  // Pas de doublons
```

Algorithmes STL

```cpp
#include <algorithm>

std::vector<int> v = {3, 1, 4, 1, 5, 9};

// Tri
std::sort(v.begin(), v.end());

// Recherche
auto it = std::find(v.begin(), v.end(), 4);

// Comptage
int count = std::count(v.begin(), v.end(), 1);
```

```cpp
// Transformation
std::transform(v.begin(), v.end(), v.begin(),
               [](int x) { return x * 2; });
```

Templates

```cpp
template<typename T>
T maximum(T a, T b) {
    return (a > b) ? a : b;
}

template<typename T>
class Pile {
private:
    std::vector<T> elements;

public:
    void empiler(T element) {
        elements.push_back(element);
    }

    T depiler() {
        T element = elements.back();
        elements.pop_back();
        return element;
    }
};
```

Exercice 6.1 : Utilisation des conteneurs STL

Objectif : Maîtriser les conteneurs de la STL.

Énoncé : Créez un programme qui : 1. Utilise `std::vector` pour stocker des nombres 2. Utilise `std::list` pour stocker des chaînes 3. Utilise `std::map` pour créer un dictionnaire (mot → définition) 4. Utilise `std::set` pour stocker des nombres uniques 5. Utilise `std::queue` pour simuler une file d'attente 6. Utilise `std::stack` pour simuler une pile

Pour chaque conteneur : - Ajoutez des éléments - Parcourez les éléments - Recherchez des éléments - Supprimez des éléments

Exercice 6.2 : Algorithmes STL

Objectif : Utiliser les algorithmes de la STL.

Énoncé : Créez un programme qui : 1. Utilise `std::sort` pour trier un vecteur 2. Utilise `std::find` pour rechercher un élément 3. Utilise `std::count` pour compter les occurrences 4. Utilise `std::copy` pour copier des éléments 5. Utilise `std::transform` pour modifier des éléments 6. Utilise `std::accumulate` pour calculer une somme 7.

Utilise `std::max_element` et `std::min_element` 8. Utilise `std::unique` pour supprimer les doublons

Exercice 6.3 : Templates

Objectif : Créer des templates de fonctions et de classes.

Énoncé : Créez : 1. Template de fonction `echanger<T>` pour échanger deux valeurs 2. Template de fonction `maximum<T>` pour trouver le maximum 3. Template de classe `Pile<T>` pour une pile générique 4. Template de classe `File<T>` pour une file générique 5. Template de classe `Tableau<T>` pour un tableau dynamique 6. Template de fonction `trier<T>` qui utilise `std::sort`

Testez avec différents types (int, double, string).

Solutions

Solution Exercice 1.1

```cpp
#include <iostream>
#include <string>

int main() {
    // Déclaration des variables
    int age = 25;
    double taille = 1.75;
    char grade = 'A';
    bool est_etudiant = true;
    std::string nom = "Alice";

    // Affichage des variables
    std::cout << "Nom: " << nom << std::endl;
    std::cout << "Age: " << age << " ans" << std::endl;
    std::cout << "Taille: " << taille << " m" << std::endl;
    std::cout << "Grade: " << grade << std::endl;
    std::cout << "Est étudiant: " << (est_etudiant ? "Oui" : "Non") << std::endl;

    // Calculs et opérateurs
    std::cout << "Age + 5 = " << age + 5 << std::endl;
    std::cout << "Taille en cm = " << taille * 100 << " cm" << std::endl;
    std::cout << "Age > 18: " << (age > 18 ? "Vrai" : "Faux") << std::endl;
    std::cout << "Est étudiant ET age < 30: " << (est_etudiant && age < 30 ? "Vrai" : "

    // Structures conditionnelles
    if (age >= 18) {
        std::cout << "Statut: Majeur" << std::endl;
    } else {
```

```
        std::cout << "Statut: Mineur" << std::endl;
    }

    if (est_etudiant) {
        std::cout << "Statut académique: Étudiant" << std::endl;
    } else {
        std::cout << "Statut académique: Non étudiant" << std::endl;
    }

    // Boucle for
    std::cout << "Nombres de 1 à 5: ";
    for (int i = 1; i <= 5; i++) {
        std::cout << i;
        if (i < 5) std::cout << ", ";
    }
    std::cout << std::endl;

    // Boucle while
    std::cout << "Nombres pairs de 2 à 10: ";
    int nombre = 2;
    while (nombre <= 10) {
        std::cout << nombre;
        if (nombre < 10) std::cout << ", ";
        nombre += 2;
    }
    std::cout << std::endl;

    return 0;
}
```

Solution Exercice 2.1

```
#include <iostream>
#include <cmath>

double carre(double x) {
    return x * x;
}

double racine_carree(double x) {
    return sqrt(x);
}

int puissance(int base, int exposant) {
    int resultat = 1;
    for (int i = 0; i < exposant; i++) {
        resultat *= base;
    }
```

```
    return resultat;
}

bool est_pair(int n) {
    return n % 2 == 0;
}

int pgcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    std::cout << "Carré de 5: " << carre(5) << std::endl;
    std::cout << "Racine carrée de 16: " << racine_carree(16) << std::endl;
    std::cout << "2^3: " << puissance(2, 3) << std::endl;
    std::cout << "8 est pair: " << (est_pair(8) ? "Oui" : "Non") << std::endl;
    std::cout << "PGCD de 12 et 18: " << pgcd(12, 18) << std::endl;

    return 0;
}
```

Solution Exercice 3.1

```
#include <iostream>
#include <cstdlib>
#include <ctime>

int main() {
    srand(time(0));
    int tableau[10];

    // Remplir le tableau
    for (int i = 0; i < 10; i++) {
        tableau[i] = rand() % 100 + 1;
    }

    // Afficher le tableau
    std::cout << "Tableau: ";
    for (int i = 0; i < 10; i++) {
        std::cout << tableau[i] << " ";
    }
    std::cout << std::endl;
```

```
    // Trouver max et min
    int max = tableau[0], min = tableau[0];
    for (int i = 1; i < 10; i++) {
        if (tableau[i] > max) max = tableau[i];
        if (tableau[i] < min) min = tableau[i];
    }
    std::cout << "Maximum: " << max << std::endl;
    std::cout << "Minimum: " << min << std::endl;

    // Calculer la moyenne
    int somme = 0;
    for (int i = 0; i < 10; i++) {
        somme += tableau[i];
    }
    double moyenne = (double)somme / 10;
    std::cout << "Moyenne: " << moyenne << std::endl;

    // Compter pairs et impairs
    int pairs = 0, impairs = 0;
    for (int i = 0; i < 10; i++) {
        if (tableau[i] % 2 == 0) {
            pairs++;
        } else {
            impairs++;
        }
    }
    std::cout << "Pairs: " << pairs << ", Impairs: " << impairs << std::endl;

    return 0;
}
```

Solution Exercice 4.1

```
#include <iostream>
#include <string>

class CompteBancaire {
private:
    std::string numero;
    std::string titulaire;
    double solde;

public:
    CompteBancaire(std::string num, std::string tit, double sld)
        : numero(num), titulaire(tit), solde(sld) {}

    void deposer(double montant) {
        if (montant > 0) {
```

```
            solde += montant;
            std::cout << "Dépôt de " << montant << "€ effectué." << std::endl;
        } else {
            std::cout << "Erreur: Montant négatif!" << std::endl;
        }
    }

    bool retirer(double montant) {
        if (montant > 0 && montant <= solde) {
            solde -= montant;
            std::cout << "Retrait de " << montant << "€ effectué." << std::endl;
            return true;
        } else {
            std::cout << "Erreur: Solde insuffisant ou montant négatif!" << std::endl;
            return false;
        }
    }

    void afficherSolde() {
        std::cout << "Solde: " << solde << "€" << std::endl;
    }

    double getSolde() { return solde; }
    std::string getTitulaire() { return titulaire; }
};

int main() {
    CompteBancaire compte("12345", "Alice", 1000.0);

    compte.afficherSolde();
    compte.deposer(500.0);
    compte.retirer(200.0);
    compte.afficherSolde();

    return 0;
}
```

Solution Exercice 5.1

```
#include <iostream>

void echanger(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
```

```
    int nombre = 42;
    int* pointeur = &nombre;

    std::cout << "Valeur: " << nombre << std::endl;
    std::cout << "Adresse: " << pointeur << std::endl;
    std::cout << "Valeur via pointeur: " << *pointeur << std::endl;

    *pointeur = 100;
    std::cout << "Nouvelle valeur: " << nombre << std::endl;

    // Échange avec références
    int a = 10, b = 20;
    std::cout << "Avant échange: a=" << a << ", b=" << b << std::endl;
    echanger(a, b);
    std::cout << "Après échange: a=" << a << ", b=" << b << std::endl;

    // Tableau dynamique
    int* tableau = new int[5];
    for (int i = 0; i < 5; i++) {
        tableau[i] = i + 1;
    }

    std::cout << "Tableau: ";
    for (int i = 0; i < 5; i++) {
        std::cout << tableau[i] << " ";
    }
    std::cout << std::endl;

    delete[] tableau;

    return 0;
}
```

Solution Exercice 6.1

```
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <set>
#include <queue>
#include <stack>

int main() {
    // Vector
    std::vector<int> nombres = {1, 2, 3, 4, 5};
    nombres.push_back(6);
    std::cout << "Vector: ";
```

```cpp
for (int n : nombres) {
    std::cout << n << " ";
}
std::cout << std::endl;

// List
std::list<std::string> mots = {"Hello", "World", "C++"};
mots.push_back("Programming");
std::cout << "List: ";
for (const std::string& mot : mots) {
    std::cout << mot << " ";
}
std::cout << std::endl;

// Map
std::map<std::string, int> ages;
ages["Alice"] = 25;
ages["Bob"] = 30;
ages["Charlie"] = 35;

std::cout << "Map:" << std::endl;
for (const auto& paire : ages) {
    std::cout << paire.first << ": " << paire.second << " ans" << std::endl;
}

// Set
std::set<int> ensemble = {1, 2, 3, 3, 4, 4, 5};
std::cout << "Set: ";
for (int n : ensemble) {
    std::cout << n << " ";
}
std::cout << std::endl;

// Queue
std::queue<int> file;
for (int i = 1; i <= 5; i++) {
    file.push(i);
}

std::cout << "Queue: ";
while (!file.empty()) {
    std::cout << file.front() << " ";
    file.pop();
}
std::cout << std::endl;

// Stack
std::stack<int> pile;
```

```
    for (int i = 1; i <= 5; i++) {
        pile.push(i);
    }

    std::cout << "Stack: ";
    while (!pile.empty()) {
        std::cout << pile.top() << " ";
        pile.pop();
    }
    std::cout << std::endl;

    return 0;
}
```

Conseils pour l'apprentissage

1. Pratique régulière

- Codez au moins 30 minutes par jour
- Commencez par les exercices simples
- Augmentez progressivement la difficulté

2. Bonnes pratiques

- Commentez votre code
- Utilisez des noms de variables descriptifs
- Testez vos programmes avec différentes entrées
- Gérez les cas d'erreur

3. Outils de développement

- Utilisez un IDE moderne (VSCode, CLion, etc.)
- Apprenez à utiliser le debugger
- Utilisez Valgrind pour détecter les fuites mémoire
- Formatez votre code automatiquement

4. Ressources supplémentaires

- cppreference.com - Documentation officielle
- learncpp.com - Tutoriel interactif
- cplusplus.com - Référence et tutoriels
- Stack Overflow - Questions et réponses

5. Projets pratiques

Une fois les exercices maîtrisés, essayez de créer : - Un jeu simple (Tic-tac-toe, Devinettes) - Un gestionnaire de contacts - Un calculateur d'expressions - Un anal-

yseur de fichiers texte - Un jeu de cartes

Bonne chance dans votre apprentissage de C++ !