Guide d'exercices Ada 2012

Formation complète - Du débutant à l'expert

Progression structurée pour maîtriser toutes les notions importantes d'Ada

1. Fondamentaux du langage

1.1 Premier programme et syntaxe de base

Exercice 1.1 Facile

Créez un programme Ada qui affiche "Bonjour Ada !" à l'écran.

Résultat attendu : Le programme compile et affiche exactement "Bonjour Ada !" suivi d'un retour à la ligne.

```
Bonjour Ada !
```

Exercice 1.2 Facile

Écrivez un programme qui affiche votre nom, puis sur une nouvelle ligne votre âge sous la forme "Je m'appelle [nom] et j'ai [âge] ans".

Résultat attendu : Deux lignes d'affichage avec le format demandé.

```
Alice
Je m'appelle Alice et j'ai 25 ans
```

1.2 Variables et types de base

Exercice 1.3 Facile

Déclarez des variables de types `Integer`, `Float`, `Boolean`, et `Character`. Initialisez-les avec des valeurs et affichez-les.

Résultat attendu : Affichage correct de chaque type de variable avec leurs valeurs.

```
Entier: 42
Flottant: 3.14159
Booleen: TRUE
Caractere: A
```

Exercice 1.4 Moyen

Créez un programme qui calcule l'aire d'un rectangle. Déclarez les variables largeur et hauteur, calculez l'aire, et affichez le résultat avec un message explicite.

Résultat attendu : "L'aire du rectangle de [largeur] x [hauteur] est [résultat]"

```
L'aire du rectangle de 5.0 x 3.0 est 15.0
```

1.3 Opérateurs et expressions

Exercice 1.5 Facile

Écrivez un programme qui effectue les quatre opérations arithmétiques de base (+, -, *, /) sur deux nombres entiers et affiche tous les résultats.

Résultat attendu : Quatre lignes montrant les résultats des opérations avec des messages explicites.

```
12 + 8 = 20
12 - 8 = 4
12 * 8 = 96
12 / 8 = 1
```

Exercice 1.6 Moyen

Créez un programme qui teste tous les opérateurs de comparaison (=, /=, <, >, <=, >=) sur deux nombres et affiche les résultats sous forme de booléens.

Résultat attendu : Six lignes affichant TRUE ou FALSE pour chaque comparaison.

```
15 = 10 : FALSE
15 /= 10 : TRUE
15 < 10 : FALSE
15 > 10 : TRUE
15 <= 10 : FALSE
15 >= 10 : TRUE
```

2. Structures de contrôle

2.1 Conditions (if/elsif/else)

Exercice 2.1 Facile

Écrivez un programme qui demande un nombre à l'utilisateur et indique s'il est positif, négatif ou nul.

Résultat attendu : Message approprié selon la valeur saisie ("Le nombre est positif/négatif/nul").

```
Entrez un nombre: -5
Le nombre est negatif
```

Exercice 2.2 Moyen

Créez un programme qui calcule les mentions d'un étudiant selon sa note (0-20) : Très bien (≥16), Bien (≥14), Assez bien (≥12), Passable (≥10), Insuffisant (<10).

Résultat attendu : Affichage de la mention correspondante à la note saisie.

```
Entrez votre note (0-20): 15
Mention: Bien
```

2.2 Structures case

Exercice 2.3 Moyen

Implémentez une calculatrice simple qui demande deux nombres et un opérateur (+, -, *, /) puis effectue l'opération correspondante en utilisant une structure `case`.

Résultat attendu : Calcul correct selon l'opérateur choisi, avec gestion d'erreur pour la division par zéro.

```
Entrez le premier nombre: 12
Entrez le second nombre: 4
Entrez l'operateur (+, -, *, /): *
Resultat: 12 * 4 = 48
```

Exercice 2.4 Moyen

Créez un programme qui affiche le nom du jour de la semaine selon un numéro (1=Lundi, 2=Mardi, etc.) en utilisant `case`.

Résultat attendu : Nom du jour correspondant au numéro, avec gestion des valeurs invalides.

```
Entrez un numero de jour (1-7): 3
Mercredi
```

2.3 Boucles (for, while, loop)

Exercice 2.5 Facile

Écrivez une boucle `for` qui affiche les nombres de 1 à 10.

Résultat attendu : Affichage des nombres 1, 2, 3... 10, chacun sur une ligne.

```
1
2
3
4
5
6
7
8
9
10
```

Exercice 2.6 Moyen

Implémentez le calcul de la factorielle d'un nombre avec une boucle `while`.

Résultat attendu : Calcul correct de n! (ex: 5! = 120).

```
Entrez un nombre: 5
5! = 120
```

Exercice 2.7 Moyen

Créez un programme qui trouve tous les nombres premiers entre 2 et 100 en utilisant le crible d'Ératosthène.

Résultat attendu : Liste des 25 nombres premiers : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

```
Nombres premiers entre 2 et 100:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
Total: 25 nombres premiers
```

3. Types et sous-types

3.1 Types énumérés

Exercice 3.1 Moyen

Définissez un type énuméré `Couleur` avec les valeurs Rouge, Vert, Bleu. Créez une fonction qui retourne la couleur complémentaire.

Résultat attendu : Rouge→Vert, Vert→Bleu, Bleu→Rouge (cycle des couleurs primaires).

```
Couleur: Rouge
Couleur complementaire: Vert
```

Exercice 3.2 Moyen

Créez un type énuméré `Etat_Moteur` (Arrete, Demarrage, Marche, Arret) et implémentez une machine à états simple.

Résultat attendu : Transitions d'états logiques avec affichage de l'état courant à chaque étape.

```
Etat initial: Arrete
Demarrage du moteur...
Etat: Demarrage
Moteur en marche
Etat: Marche
Arret du moteur...
Etat: Arret
Moteur arrete
Etat: Arrete
```

3.2 Types entiers et sous-types

Exercice 3.3 Moyen

Définissez un sous-type `Note` pour les entiers de 0 à 20, et un sous-type `Pourcentage` pour 0 à 100. Créez une fonction de conversion.

Résultat attendu : Conversion correcte (ex: note 15/20 = 75%).

```
Note: 15/20
Pourcentage: 75%
```

Exercice 3.4 Difficile

Implémentez un type `Annee` (1900..2100) avec validation, et créez des fonctions pour déterminer si une année est bissextile.

Résultat attendu : Détection correcte des années bissextiles (ex: 2000=oui, 1900=non, 2024=oui).

```
Annee 2000: bissextile
Annee 1900: non bissextile
Annee 2024: bissextile
```

3.3 Types à virgule flottante

Exercice 3.5 Moyen

Créez un type `Temperature` pour des flottants avec une précision spécifique, et implémentez des conversions Celsius/Fahrenheit.

Résultat attendu : Conversions correctes (0°C = 32°F, 100°C = 212°F).

```
0.0°C = 32.0°F
100.0°C = 212.0°F
20.0°C = 68.0°F
```

4. Tableaux et structures de données

4.1 Tableaux statiques

Exercice 4.1 Facile

Déclarez un tableau de 10 entiers, remplissez-le avec les carrés des indices (1^2, 2^2, 3^2...) et affichez le contenu.

Résultat attendu : Affichage de 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.

```
Tableau des carres:
1 4 9 16 25 36 49 64 81 100
```

Exercice 4.2 Moyen

Implémentez un algorithme de tri à bulles pour trier un tableau d'entiers.

Résultat attendu : Tableau trié par ordre croissant avec affichage avant/après tri.

```
Tableau avant tri: 64 34 25 12 22 11 90
Tableau apres tri: 11 12 22 25 34 64 90
```

Exercice 4.3 Difficile

Créez une fonction de recherche dichotomique dans un tableau trié, qui retourne l'indice de l'élément ou -1 s'il n'existe pas.

Résultat attendu : Indice correct pour les éléments présents, -1 pour les absents.

```
Tableau: 1 3 5 7 9 11 13 15
Recherche de 7: trouve a l'indice 4
Recherche de 8: non trouve (-1)
```

4.2 Tableaux multidimensionnels

Exercice 4.4 Moyen

Implémentez une matrice 3x3 et créez des fonctions pour calculer la somme des éléments de chaque ligne et colonne.

Résultat attendu : Affichage de la matrice et des sommes par ligne et par colonne.

```
Matrice 3x3:
1 2 3
4 5 6
7 8 9

Sommes par ligne: 6 15 24
Sommes par colonne: 12 15 18
```

Exercice 4.5 Difficile

Créez un jeu du morpion (tic-tac-toe) avec une grille 3x3, gestion des tours des joueurs et détection de victoire.

Résultat attendu : Jeu fonctionnel avec affichage de la grille, alternance X/O, détection de fin de partie.

```
| |
-----------
| |
-----------
| |

Joueur X, entrez ligne (1-3): 2
Joueur X, entrez colonne (1-3): 2

| |
-----------
| X |
-----------
| |

Joueur O, entrez ligne (1-3): 1
```

4.3 Chaînes de caractères

Exercice 4.6 Moyen

Implémentez une fonction qui compte le nombre d'occurrences de chaque voyelle dans une chaîne de caractères.

Résultat attendu : Comptage correct (ex: "bonjour" → a:0, e:0, i:0, o:2, u:1).

```
Texte: "bonjour"
Voyelles trouvees:
a: 0
e: 0
i: 0
o: 2
u: 1
```

Exercice 4.7 Difficile

Créez une fonction qui vérifie si une chaîne est un palindrome (se lit de la même façon dans les deux sens).

Résultat attendu : TRUE pour "radar", "kayak", FALSE pour "bonjour".

```
Test de "radar": TRUE (palindrome)
Test de "kayak": TRUE (palindrome)
Test de "bonjour": FALSE (pas un palindrome)
```

5. Enregistrements (records)

5.1 Enregistrements simples

Exercice 5.1 Moyen

Définissez un enregistrement `Personne` avec nom, prénom, âge. Créez un tableau de personnes et triez-le par âge.

Résultat attendu : Affichage des personnes triées par âge croissant.

```
Personnes triees par age:
Alice Martin, 22 ans
Bob Durant, 25 ans
Claire Dubois, 30 ans
```

Exercice 5.2 Moyen

Implémentez un enregistrement `Point_2D` avec coordonnées x, y et créez des fonctions pour calculer la distance entre deux points.

Résultat attendu : Calcul correct de la distance euclidienne entre deux points.

```
Point A: (1.0, 2.0)
Point B: (4.0, 6.0)
Distance entre A et B: 5.0
```

5.2 Enregistrements variants

Exercice 5.3 Difficile

Créez un enregistrement variant `Forme` qui peut être un cercle (rayon) ou un rectangle (largeur, hauteur), avec une fonction de calcul d'aire.

Résultat attendu : Calcul correct de l'aire selon le type de forme ($\pi \times r^2$ pour cercle, $l \times h$ pour rectangle).

```
Cercle de rayon 5.0: aire = 78.54
Rectangle 4.0 x 6.0: aire = 24.0
```

6. Procédures et fonctions

6.1 Fonctions de base

Exercice 6.1 Facile

Créez une fonction qui calcule le maximum de trois nombres entiers.

Résultat attendu : Retour du plus grand des trois nombres fournis.

```
Maximum de 15, 8, 23: 23
```

Exercice 6.2 Moyen

Implémentez une fonction récursive pour calculer le n-ième terme de la suite de Fibonacci.

Résultat attendu : Séquence correcte : F(0)=0, F(1)=1, F(2)=1, F(3)=2, F(4)=3, F(5)=5, etc.

```
Suite de Fibonacci:
F(0) = 0
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
```

6.2 Paramètres et modes

Exercice 6.3 Moyen

Créez une procédure avec paramètres `in` , `out` , et `in out` qui échange deux valeurs et calcule leur somme.

Résultat attendu : Variables échangées et somme calculée correctement.

```
Avant echange: A=10, B=20
Apres echange: A=20, B=10
Somme: 30
```

Exercice 6.4 Difficile

Implémentez une procédure générique de tri qui accepte un tableau de tout type ordonnable en paramètre.

Résultat attendu : Tri fonctionnel sur différents types (Integer, Float, Character).

```
Tri d'entiers: 5 2 8 1 9 -> 1 2 5 8 9
Tri de flottants: 3.2 1.1 4.7 -> 1.1 3.2 4.7
Tri de caracteres: z a m -> a m z
```

6.3 Surcharge de fonctions

Exercice 6.5 Moyen

Créez plusieurs versions surchargées d'une fonction "Distance" (2D, 3D, Manhattan).

Résultat attendu : Choix automatique de la bonne fonction selon les paramètres fournis.

```
Distance 2D entre (0,0) et (3,4): 5.0
Distance 3D entre (0,0,0) et (1,2,2): 3.0
Distance Manhattan entre (0,0) et (3,4): 7.0
```

7. Paquetages (packages)

7.1 Paquetages simples

Exercice 7.1 Moyen

Créez un paquetage `Mathematiques` avec des fonctions pour PGCD, PPCM, et test de primalité.

Résultat attendu : Fonctions accessibles depuis l'extérieur avec calculs corrects.

```
PGCD(48, 18) = 6
PPCM(48, 18) = 144
17 est premier: TRUE
18 est premier: FALSE
```

Exercice 7.2 Difficile

Implémentez un paquetage `Pile` (stack) générique avec les opérations Push, Pop, Top, Is_Empty.

Résultat attendu : Structure de pile fonctionnelle avec gestion des erreurs (pile vide/pleine).

```
Push 10
Push 20
Push 30
Top: 30
Pop: 30
Pop: 20
Pop: 10
Pile vide: TRUE
```

7.2 Paquetages enfants

Exercice 7.3 Difficile

Créez un paquetage `Geometrie` avec un enfant `Geometrie.Plans` pour les formes 2D et `Geometrie.Volumes` pour les formes 3D.

Résultat attendu : Hiérarchie fonctionnelle avec calculs d'aires et de volumes.

```
Formes 2D:
Carre 5x5: aire = 25.0
Cercle rayon 3: aire = 28.27

Formes 3D:
Cube 4x4x4: volume = 64.0
Sphere rayon 2: volume = 33.51
```

8. Types d'accès (pointeurs)

8.1 Pointeurs de base

Exercice 8.1 Difficile

Implémentez une liste chaînée simple avec insertion, suppression et parcours.

Résultat attendu : Structure de liste fonctionnelle avec gestion mémoire correcte.

```
Liste vide
Insertion de 10
Insertion de 20
Insertion de 30
Liste: 10 -> 20 -> 30
Suppression de 20
Liste: 10 -> 30
```

9. Exceptions

9.1 Gestion d'exceptions standard

Exercice 9.1 Moyen

Créez un programme de division qui gère l'exception `Constraint_Error` pour la division par zéro.

Résultat attendu : Message d'erreur approprié sans plantage du programme.

```
Entrez le dividende : 10
Entrez le diviseur : 0
Erreur : Division par zéro impossible !

Entrez le dividende : 15
Entrez le diviseur : 3
Résultat : 5.00
```

Exercice 9.2 Difficile

Implémentez un programme de lecture de fichier avec gestion complète des exceptions possibles.

Résultat attendu : Gestion de toutes les erreurs (fichier inexistant, droits insuffisants, etc.).

```
Nom du fichier à lire : fichier_inexistant.txt
Erreur : Le fichier 'fichier_inexistant.txt' n'existe pas.

Nom du fichier à lire : document.txt
Contenu du fichier :
Ligne 1 du document
Ligne 2 du document
Fin de fichier atteinte avec succès.
```

9.2 Exceptions personnalisées

Exercice 9.3 Difficile

Créez des exceptions personnalisées pour une classe `Compte_Bancaire` (solde insuffisant, compte bloqué, etc.).

Résultat attendu : Exceptions levées dans les bonnes conditions avec messages explicites.

```
=== Gestion de Compte Bancaire ===
Solde initial : 1000.00 €

Tentative de retrait : 500.00 €
Retrait effectué. Nouveau solde : 500.00 €

Tentative de retrait : 800.00 €
Erreur : Solde insuffisant pour cette opération !

Tentative d'opération sur compte bloqué :
Erreur : Compte temporairement bloqué !
```

10. Généricité

10.1 Unités génériques simples

Exercice 10.1 Difficile

Implémentez un paquetage générique de tri pour tout type ordonnable.

Résultat attendu : Tri fonctionnel sur Integer, Float, String après instanciation.

```
Tri d'entiers : [5, 2, 8, 1, 9, 3]
Résultat : [1, 2, 3, 5, 8, 9]

Tri de flottants : [3.14, 1.41, 2.71, 0.57]
Résultat : [0.57, 1.41, 2.71, 3.14]

Tri de chaînes : ["zebra", "apple", "banana", "cherry"]
Résultat : ["apple", "banana", "cherry", "zebra"]
```

Exercice 10.2 Expert

Créez une structure de données générique (tableau dynamique redimensionnable) similaire aux vectors C++.

Résultat attendu : Structure extensible avec gestion automatique de la mémoire.

```
Création d'un vecteur d'entiers
Ajout : 10, 20, 30
Taille : 3, Capacité : 4

Ajout : 40, 50, 60, 70, 80
Redimensionnement automatique !
Taille : 8, Capacité : 8

Accès à l'élément 5 : 60
Suppression du dernier élément
Taille finale : 7
```

11. Tâches et concurrence

11.1 Tâches simples

Exercice 11.1 Difficile

Créez deux tâches qui s'exécutent en parallèle : une qui compte de 1 à 10, l'autre de 10 à 1.

Résultat attendu : Affichage entrelacé des deux compteurs montrant l'exécution parallèle.

```
Tâche 1: 1
Tâche 2: 10
Tâche 1: 2
Tâche 2: 9
Tâche 1: 3
Tâche 2: 8
Tâche 1: 4
Tâche 2: 7
Tâche 1: 5
Tâche 2: 6
...
Toutes les tâches terminées
```

Exercice 11.2 Expert

Implémentez le problème producteur-consommateur avec un buffer circulaire partagé entre tâches.

Résultat attendu : Synchronisation correcte, pas de perte ni de duplication de données.

```
Producteur: Produit élément 1
Consommateur: Consomme élément 1
Producteur: Produit élément 2
Producteur: Produit élément 3
Consommateur: Consomme élément 2
Producteur: Buffer plein, attente...
Consommateur: Consomme élément 3
Producteur: Produit élément 4
...
Production/Consommation synchronisées
```

11.2 Synchronisation et communication

Exercice 11.3 Expert

Créez un système de rendez-vous entre tâches pour simuler un serveur et plusieurs clients.

Résultat attendu : Communication bidirectionnelle avec gestion des files d'attente.

```
Serveur: Démarrage, en attente de clients...
Client 1: Connexion au serveur
Serveur: Client 1 connecté, traitement de la requête
Client 2: Connexion au serveur
Serveur: Réponse envoyée au Client 1
Client 1: Réponse reçue: "Données traitées"
Serveur: Client 2 connecté, traitement de la requête
Client 3: En attente de connexion...
Serveur: Réponse envoyée au Client 2
```

Exercice 11.4 Expert

Implémentez le problème des philosophes qui dînent avec 5 philosophes et 5 fourchettes.

Résultat attendu : Pas d'interblocage, tous les philosophes peuvent manger à tour de rôle.

```
Philosophe 1: Réfléchit...
Philosophe 2: Prend la fourchette gauche
Philosophe 3: Réfléchit...
Philosophe 2: Prend la fourchette droite, commence à manger
Philosophe 1: Prend la fourchette gauche
Philosophe 2: Termine de manger, repose les fourchettes
Philosophe 4: Prend la fourchette gauche
Philosophe 1: Prend la fourchette droite, commence à manger
...
Aucun interblocage détecté
```

12. Entrées/Sorties

12.1 Fichiers texte

Exercice 12.1 Moyen

Créez un programme qui lit un fichier texte ligne par ligne et compte le nombre de mots et de lignes.

Résultat attendu : Statistiques correctes (nombre de lignes, mots, caractères).

```
Analyse du fichier 'document.txt'

Contenu du fichier :
Ligne 1: Bonjour le monde
Ligne 2: Ceci est un test
Ligne 3: Fin du document

=== Statistiques ===
Nombre de lignes : 3
Nombre de mots : 9
Nombre de caractères : 47
```

Exercice 12.2 Difficile

Implémentez un programme qui copie un fichier en remplaçant toutes les occurrences d'un mot par un autre.

Résultat attendu : Fichier de sortie identique à l'entrée avec remplacements effectués.

```
Fichier source : texte.txt
Fichier destination : texte_modifie.txt
Mot à remplacer : "ancien"
Nouveau mot : "nouveau"

Traitement en cours...
5 occurrences de "ancien" remplacées par "nouveau"
Fichier 'texte_modifie.txt' créé avec succès !
```

12.2 Fichiers binaires

Exercice 12.3 Difficile

Créez un système de sauvegarde/chargement d'enregistrements `Personne` en fichier binaire.

Résultat attendu : Données correctement sérialisées et désérialisées.

```
=== Sauvegarde des données ===
Ajout : Alice, 25 ans
Ajout : Bob, 30 ans
Ajout : Claire, 28 ans
Sauvegarde dans 'personnes.dat' terminée

=== Chargement des données ===
Chargement depuis 'personnes.dat'...
Personne 1: Alice, 25 ans
Personne 2: Bob, 30 ans
Personne 3: Claire, 28 ans
3 enregistrements chargés avec succès
```

13. Attributs et pragmas

13.1 Attributs de types

Exercice 13.1 Moyen

Créez un programme qui affiche tous les attributs importants d'un type (First, Last, Range, Size, etc.).

Résultat attendu : Affichage des limites et propriétés des types Integer, Character, Boolean.

```
=== Attributs du type Integer ===
First: -2147483648
Last: 2147483647
Size: 32 bits
Range: -2147483648 .. 2147483647

=== Attributs du type Character ===
First: ASCII.NUL
Last: ASCII.DEL
Size: 8 bits

=== Attributs du type Boolean ===
First: FALSE
Last: TRUE
Size: 1 bit
```

Exercice 13.2 Difficile

Implémentez une fonction générique qui utilise les attributs pour parcourir automatiquement tout type énuméré.

Résultat attendu : Parcours complet de n'importe quelle énumération fournie.

```
Énumération Jour_Semaine:
Lundi
Mardi
Mercredi
Jeudi
Vendredi
Samedi
Dimanche

Énumération Couleur:
Rouge
Vert
Bleu
Jaune
Noir
Blanc
```

13.2 Pragmas utiles

Exercice 13.3 Difficile

Utilisez `pragma Assert` pour valider les préconditions d'une fonction de division euclidienne.

Résultat attendu : Assertions déclenchées sur les cas invalides (diviseur nul, etc.).

```
Division euclidienne de 17 par 5
Vérification : diviseur non nul ✓
Vérification : dividende positif ✓
Résultat : quotient = 3, reste = 2

Division euclidienne de 10 par 0
Assertion échouée : Le diviseur ne peut pas être nul !
Programme arrêté pour violation de précondition
```

14. Représentation des données

14.1 Clauses de représentation

Exercice 14.1 Expert

Définissez un enregistrement avec clause de représentation pour interfacer avec du matériel (registre 32 bits).

Résultat attendu : Contrôle précis de la disposition mémoire, compatible avec spécifications matérielles.

```
=== Configuration Registre Matériel ===
Définition du registre 32 bits:
- Bit 0-7 : Code état (8 bits)
- Bit 8-15 : Données (8 bits)
- Bit 16-23 : Contrôle (8 bits)
- Bit 24-31 : Flags (8 bits)

Test d'écriture:
Registre = 16#AB12CD34#
État: 52, Données: 205, Contrôle: 18, Flags: 171
Vérification disposition mémoire: OK
```

14.2 Interfaçage avec C

Exercice 14.2 Expert

Créez une interface Ada pour appeler des fonctions mathématiques C standard (sin, cos, sqrt).

Résultat attendu : Appels réussis aux fonctions C avec résultats corrects.

```
=== Interface Ada-C : Fonctions Mathématiques ===
Calcul de sin(π/4) = 0.7071067812
Calcul de cos(π/3) = 0.5000000000
Calcul de sqrt(16) = 4.0000000000
Calcul de log(2.718) = 0.9999896316

Test de l'interface C : tous les appels réussis ✓
Précision des calculs : conforme aux attentes
```

15. Programmation orientée objet (Ada 2012)

15.1 Types étiquetés et héritage

Exercice 15.1 Difficile

Créez une hiérarchie `Vehicule` → `Voiture`, `Moto` avec méthodes virtuelles.

Résultat attendu : Polymorphisme fonctionnel, appel des bonnes méthodes selon le type réel.

```
=== Test Polymorphisme Véhicules ===
Création d'une Voiture (4 portes):
Démarrage: Le moteur de la voiture démarre
Klaxon: Bip bip ! (klaxon de voiture)

Création d'une Moto (2 roues):
Démarrage: Le moteur de la moto vrombît
Klaxon: Tut tut ! (klaxon de moto)
Action spéciale: Wheelie spectaculaire !

Polymorphisme testé avec succès ✓
```

Exercice 15.2 Expert

Implémentez un système de formes géométriques avec interfaces et méthodes abstraites.

Résultat attendu : Hiérarchie extensible avec contrats d'interfaces respectés.

```
=== Système de Formes Géométriques ===
Cercle (rayon=5.0):
- Aire: 78.54 unités²
- Périmètre: 31.42 unités
- Dessiner: ○ (cercle)

Rectangle (L=8.0, l=3.0):
- Aire: 24.00 unités²
- Périmètre: 22.00 unités
- Dessiner: ▭ (rectangle)

Triangle (base=6.0, hauteur=4.0):
- Aire: 12.00 unités²
- Périmètre: 16.25 unités
- Dessiner: △ (triangle)
```

15.2 Interfaces et méthodes abstraites

Exercice 15.3 Expert

Créez une interface `Serializable` et implémentez-la pour différents types de données.

Résultat attendu : Sérialisation/désérialisation uniforme pour tous les types implémentant l'interface.

```
=== Test Interface Serializable ===
Personne: Alice, 30 ans
Sérialisation: "Personne|Alice|30"
Désérialisation: Alice, 30 ans ✓

Produit: Ordinateur, 1200.50€
Sérialisation: "Produit|Ordinateur|1200.50"
Désérialisation: Ordinateur, 1200.50€ ✓

Date: 15/06/2024
Sérialisation: "Date|15|06|2024"
Désérialisation: 15/06/2024 ✓

Interface implémentée avec succès pour tous les types
```

16. Contrats et assertions (Ada 2012)

16.1 Préconditions et postconditions

Exercice 16.1 Difficile

Implémentez une fonction de racine carrée avec pré/postconditions utilisant `Pre` et `Post`.

Résultat attendu : Validation automatique des contraintes à l'entrée et à la sortie.

```
Test de la fonction Racine_Carree:

Calcul de sqrt(16.0):
Précondition: X >= 0.0 ✓
Résultat: 4.000000
Postcondition: Result * Result = X ✓

Calcul de sqrt(-4.0):
Précondition échouée: X >= 0.0 ✗
Exception: Constraint_Error (précondition violée)

Calcul de sqrt(25.0):
Précondition: X >= 0.0 ✓
Résultat: 5.000000
Postcondition: Result * Result = X ✓
```

Exercice 16.2 Expert

Créez un ADT `Pile` complet avec tous les contrats (invariants de type inclus).

Résultat attendu : Structure robuste avec vérification automatique de la cohérence.

```
=== Test ADT Pile avec Contrats ===
Création d'une pile (capacité max: 5)
Invariant: Taille = 0, Pile_Vide = True ✓

Empiler(10):
Précondition: not Est_Pleine ✓
Postcondition: Sommet = 10, Taille = 1 ✓

Empiler(20, 30, 40, 50):
Pile: [10, 20, 30, 40, 50]
Invariant: Taille = 5, Pile_Pleine = True ✓

Dépiler():
Précondition: not Est_Vide ✓
Résultat: 50
Postcondition: Taille = 4 ✓
Invariant vérifié ✓
```

17. Expressions et quantificateurs (Ada 2012)

17.1 Expressions conditionnelles

Exercice 17.1 Moyen

Réécrivez des conditions complexes en utilisant les expressions `if` et `case` d'Ada 2012.

Résultat attendu : Code plus concis et lisible avec même logique métier.

```
=== Test Expressions Conditionnelles ===
Note de l'étudiant: 16/20

Mention (expression if):
"Bien" (car note >= 14 et note < 16)

Catégorie (expression case sur âge=22):
"Jeune adulte" (car âge dans 18..25)

Tarif cinema (expression if imbriquée):
Âge: 65 ans, Étudiant: Non
Tarif: 6.50€ (tarif senior)

Expressions Ada 2012: code plus concis ✓
```

17.2 Quantificateurs for all/for some

Exercice 17.2 Expert

Utilisez les quantificateurs pour valider qu'un tableau est trié, qu'il contient des doublons, etc.

Résultat attendu : Validation logique exprimée de façon déclarative et claire.

```
=== Test Quantificateurs Ada 2012 ===
Tableau 1: [1, 3, 5, 7, 9]
- Est trié (for all): True ✓
- Contient doublons (for some): False ✓
- Tous positifs (for all): True ✓

Tableau 2: [2, 4, 3, 8, 10]
- Est trié (for all): False ✓
- Contient doublons (for some): False ✓
- Tous pairs (for all): False ✓

Tableau 3: [1, 2, 2, 4, 5]
- Est trié (for all): True ✓
- Contient doublons (for some): True ✓
- Valeur > 10 existe (for some): False ✓

Quantificateurs: validation logique claire ✓
```

18. Projets intégrateurs

Note de progression : *Ces exercices finaux combinent plusieurs concepts pour des projets complets.*

18.1 Système de gestion

Exercice 18.1 Expert

Créez un système de gestion de bibliothèque avec livres, auteurs, emprunts, utilisateurs, persistance fichier.

Résultat attendu : Application complète avec menu, CRUD, recherche, rapports, sauvegarde/chargement.

```
=== SYSTÈME DE GESTION BIBLIOTHEQUE ===
1. Gestion des livres 2. Gestion des emprunts
3. Recherche 4. Rapports
5. Sauvegarde 0. Quitter

Choix: 1
--- Ajout d'un livre ---
Titre: "Le Petit Prince"
Auteur: Saint-Exupéry
ISBN: 978-2-07-040857-4
Livre ajouté avec succès !

Choix: 2
--- Nouvel emprunt ---
Utilisateur: Alice Dupont
Livre: "Le Petit Prince"
Date retour prévue: 25/06/2024
Emprunt enregistré !

=== RAPPORT MENSUEL ===
Livres en circulation: 12
Retards en cours: 3
Nouveaux utilisateurs: 8
Taux d'occupation: 85%
```

18.2 Simulateur temps réel

Exercice 18.2 Expert

Implémentez un simulateur de trafic routier avec feux, véhicules, statistiques, interface utilisateur.

Résultat attendu : Simulation temps réel avec multitâches, affichage dynamique, collecte de métriques.

```
=== SIMULATEUR DE TRAFIC ROUTIER ===
Initialisation: 4 carrefours, 8 voies
Génération aléatoire de véhicules: ON

[Temps: 00:03:45]
Carrefour A:  Rouge (30s restant)
File: [] (3 véhicules)

Carrefour B:  Vert (15s restant)
Véhicules passent: → →

Carrefour C:  Orange (3s restant)
File: [] (5 véhicules)

=== STATISTIQUES TEMPS RÉEL ===
Véhicules générés: 127
Temps d'attente moyen: 45.2s
Embouteillages détectés: 2
Débit global: 2.3 véh/s

Simulation en cours... [CTRL+C pour arrêter]
```

18.3 Algorithmes avancés

Exercice 18.3 Expert

Créez une bibliothèque d'algorithmes de graphes (Dijkstra, Floyd-Warshall, A*) avec visualisation.

Résultat attendu : Implémentation correcte des algorithmes avec comparaison de performances et affichage des chemins.

```
=== BIBLIOTHÈQUE ALGORITHMES DE GRAPHES ===
Chargement du graphe: 8 nœuds, 15 arêtes

1. Algorithme de Dijkstra (A → H):
Chemin optimal: A → C → F → H
Distance totale: 23
Nœuds explorés: 6/8
Temps d'exécution: 0.45ms

2. Algorithme A* (A → H):
Chemin optimal: A → C → F → H
Distance totale: 23
Nœuds explorés: 4/8 (heuristique efficace)
Temps d'exécution: 0.31ms

3. Floyd-Warshall (toutes paires):
Matrice des plus courts chemins calculée
Plus long chemin: B → G (distance: 31)
Temps d'exécution: 1.23ms

=== VISUALISATION ASCII ===
A---5---C---8---F
|   |   |
3   4   7
|   |   |
B---9---D---6---H
|   |
2   4
|   |
E---3---G

Performance: A* > Dijkstra > Floyd-Warshall ✓
```

19. Outils de développement et écosystème Ada

Note importante : *Cette section couvre les outils modernes essentiels pour le développement Ada professionnel.*

19.1 Alire - Gestionnaire de paquets Ada

Exercice 18.1 Facile

Installez Alire et créez votre premier projet Ada avec `alr init mon_projet`. Explorez la structure générée et compilez le projet.

Résultat attendu : Projet Ada fonctionnel avec fichier alire.toml, compilation réussie avec `alr build`.

Exercice 18.2 Moyen

Ajoutez des dépendances à votre projet Alire : `aunit` pour les tests et `ada_util` pour les utilitaires. Créez un test unitaire simple.

Résultat attendu : Dépendances ajoutées automatiquement, test unitaire exécutable avec `alr test`.

Exercice 18.3 Moyen

Créez un projet bibliothèque avec Alire, définissez ses métadonnées dans alire.toml et publiez-le localement avec `alr publish`.

Résultat attendu : Bibliothèque correctement packagee, métadonnées complètes, validation réussie.

19.2 GNAT et compilation

Exercice 18.4 Facile

Compilez un programme Ada avec différentes options GNAT : `-gnat2012`, `-Wall`, `-O2`, `-g`. Comparez les résultats.

Résultat attendu : Compréhension des options de compilation, différences de performance et de débogage observées.

Exercice 18.5 Moyen

Utilisez `gnatmake` avec des options avancées pour un projet multi-fichiers. Explorez `-j` pour la compilation parallèle.

Résultat attendu : Compilation efficace d'un projet complexe, gestion automatique des dépendances.

Exercice 18.6 Difficile

Configurez GNAT pour la compilation croisée vers une architecture différente (ARM, par exemple). Créez un projet simple et compilez-le.

Résultat attendu : Binaire généré pour l'architecture cible, configuration cross-compilation fonctionnelle.

19.3 GPRbuild et fichiers projet

Exercice 18.7 Moyen

Créez un fichier projet GPR (.gpr) pour un projet avec plusieurs exécutables et bibliothèques. Définissez les chemins sources et objets.

Résultat attendu : Fichier .gpr syntaxiquement correct, compilation réussie avec `gprbuild`.

Exercice 18.8 Difficile

Configurez un projet GPR avec plusieurs configurations (Debug/Release), variables d'environnement et options de compilation conditionnelles.

Résultat attendu : Basculement facile entre configurations, optimisations appropriées appliquées.

Exercice 18.9 Difficile

Créez un projet GPR hiérarchique avec un projet parent et plusieurs sous-projets. Gérez les dépendances entre projets.

Résultat attendu : Architecture modulaire fonctionnelle, compilation incrémentale efficace.

19.4 Outils d'analyse et de qualité

Exercice 18.10 Moyen

Utilisez `gnatcheck` pour analyser la qualité de code selon des règles de codage. Créez un fichier de règles personnalisé.

Résultat attendu : Rapport de qualité détaillé, code conforme aux standards définis.

Exercice 18.11 Moyen

Analysez la complexité cyclomatique de votre code avec `gnatmetric`. Identifiez les fonctions à refactoriser.

Résultat attendu : Métriques de complexité, recommandations d'amélioration du code.

Exercice 18.12 Difficile

Utilisez `gnatprove` pour la vérification formelle d'un algorithme critique (tri, recherche). Ajoutez les contrats nécessaires.

Résultat attendu : Preuve formelle de la correction de l'algorithme, contrats vérifiés automatiquement.

19.5 Débogage et profilage

Exercice 18.13 Moyen

Déboguez un programme Ada avec GDB, utilisez les extensions Ada pour l'inspection des types complexes.

Résultat attendu : Débogage efficace, inspection des enregistrements et tableaux Ada.

Exercice 18.14 Difficile

Profilez un programme Ada avec `gprof` pour identifier les goulots d'étranglement. Optimisez le code critique.

Résultat attendu : Profil d'exécution détaillé, amélioration mesurable des performances.

19.6 Documentation et maintenance

Exercice 18.15 Moyen

Générez la documentation de votre code Ada avec `gnatdoc`. Ajoutez des commentaires de documentation appropriés.

Résultat attendu : Documentation HTML complète, API clairement documentée.

Exercice 18.16 Difficile

Configurez un pipeline CI/CD pour un projet Ada avec tests automatisés, analyse de qualité et déploiement.

Résultat attendu : Pipeline fonctionnel avec GitHub Actions ou GitLab CI, déploiement automatique.

19. Projets intégrateurs

Note de progression : *Ces exercices finaux combinent plusieurs concepts pour des projets complets.*

Exercice 19.1 : Système de gestion de bibliothèque

Objectif : Créer un système complet de gestion de bibliothèque utilisant les types abstraits, les exceptions, et la programmation générique.

Spécifications :

- Gestion des livres (ajout, suppression, recherche)
- Gestion des emprunts avec dates
- Système d'authentification des utilisateurs
- Rapports et statistiques
- Sauvegarde/chargement des données

Structure proposée :

```
-- Spécification du package principal
```

```
package Library_System is
   type Book_Type is private;
   type User_Type is private;
   type Loan_Type is private;

   -- Exceptions personnalisées
   Book_Not_Found : exception;
   User_Not_Found : exception;
   Book_Already_Loaned : exception;
   Loan_Overdue : exception;

   -- Opérations sur les livres
   procedure Add_Book(ISBN : String; Title : String; Author : String);
   procedure Remove_Book(ISBN : String);
   function Search_Book(ISBN : String) return Book_Type;

   -- Opérations sur les utilisateurs
   procedure Register_User(ID : String; Name : String);
   function Authenticate_User(ID : String; Password : String) return Boolean;

   -- Opérations d'emprunt
   procedure Loan_Book(User_ID : String; ISBN : String);
   procedure Return_Book(User_ID : String; ISBN : String);
   procedure Check_Overdue_Loans;

   -- Rapports
   procedure Generate_Report;

private
   type Book_Type is record
      ISBN : String(1..13);
      Title : String(1..100);
      Author : String(1..50);
      Is_Available : Boolean := True;
   end record;

   type User_Type is record
      ID : String(1..10);
      Name : String(1..50);
      Active_Loans : Natural := 0;
   end record;

   type Loan_Type is record
      User_ID : String(1..10);
      ISBN : String(1..13);
      Loan_Date : Ada.Calendar.Time;
      Due_Date : Ada.Calendar.Time;
   end record;
end Library_System;
```

Travail à réaliser :

1. Implémenter le corps du package
2. Créer un programme principal avec menu interactif
3. Ajouter la gestion des fichiers pour la persistance
4. Implémenter les vérifications de dates pour les retards
5. Créer des tests unitaires pour chaque fonctionnalité

Exercice 19.2 : Simulateur de réseau de transport

Objectif : Développer un simulateur de transport public utilisant les tâches, la programmation temps réel, et les structures de données avancées.

Fonctionnalités requises :

- Simulation de véhicules en temps réel
- Calcul d'itinéraires optimaux
- Gestion des correspondances
- Interface utilisateur pour consultation
- Logs et monitoring du système

Architecture avec tâches :

```
 -- Tâche de simulation des véhicules
task type Vehicle_Simulator is
   entry Start_Route(Route_ID : Integer);
   entry Update_Position(X, Y : Float);
   entry Stop_Simulation;
end Vehicle_Simulator;

-- Tâche de gestion du trafic
task Traffic_Manager is
   entry Register_Vehicle(Vehicle_ID : Integer);
   entry Update_Traffic_Status;
   entry Get_Optimal_Route(Start, Destination : String; Route : out String);
end Traffic_Manager;

-- Tâche d'interface utilisateur
task User_Interface is
   entry Display_Status;
   entry Process_User_Request(Request : String);
end User_Interface;
```

Exercice 19.3 : Système de trading algorithmique

Objectif : Créer un système de trading avec analyse en temps réel, utilisant la programmation concurrente et les calculs numériques.

Composants du système :

- Collecteur de données de marché en temps réel
- Analyseur technique avec indicateurs
- Moteur de décision de trading
- Gestionnaire de risques
- Interface de monitoring

Algorithmes à implémenter :

- Moyennes mobiles (simple et exponentielle)
- RSI (Relative Strength Index)
- MACD (Moving Average Convergence Divergence)
- Bandes de Bollinger
- Calcul de Value at Risk (VaR)

20. Exercices de compilation et déploiement

Note de progression : *Ces exercices couvrent la compilation, les makefiles, et le déploiement d'applications ADA.*

Exercice 20.1 : Compilation basique avec GNAT

Objectif : Maîtriser les commandes de compilation GNAT et comprendre le processus de build.

Fichier source simple (hello.adb) :

```
with Ada.Text_IO;

procedure Hello is
begin
   Ada.Text_IO.Put_Line("Bonjour le monde !");
end Hello;
```

Commandes de compilation :

```
 -- Compilation simple
gnatmake hello.adb

-- Compilation avec options
gnatmake hello.adb -gnat2012 -gnateE -gnatwa

-- Compilation avec optimisation
gnatmake hello.adb -O2 -gnatpn

-- Vérification syntaxique uniquement
gcc -c -gnatc hello.adb

-- Compilation avec débogage
gnatmake hello.adb -g -gnateE
```

Questions :

1. Expliquez la différence entre gcc et gnatmake
2. À quoi servent les options -gnat2012, -gnateE, -gnatwa ?
3. Comment activer les warnings supplémentaires ?

Exercice 20.2 : Projet multi-fichiers

Objectif : Gérer la compilation d'un projet avec plusieurs packages et dépendances.

Structure du projet :

```
projet_math/
├── src/
│   ├── math_utils.ads
│   ├── math_utils.adb
│   ├── statistics.ads
│   ├── statistics.adb
│   └── main.adb
├── obj/
```

```
├── bin/
└── Makefile
```

Fichier math_utils.ads :

```
package Math_Utils is
   function GCD(A, B : Integer) return Integer;
   function LCM(A, B : Integer) return Integer;
   function Factorial(N : Natural) return Long_Long_Integer;
   function Power(Base : Float; Exponent : Integer) return Float;
end Math_Utils;
```

Fichier statistics.ads :

```
with Math_Utils;
package Statistics is
   type Float_Array is array (Positive range <>) of Float;

   function Mean(Data : Float_Array) return Float;
   function Median(Data : Float_Array) return Float;
   function Std_Deviation(Data : Float_Array) return Float;
   function Correlation(X, Y : Float_Array) return Float;
end Statistics;
```

Makefile proposé :

```
GNATMAKE = gnatmake
SRC_DIR = src
OBJ_DIR = obj
BIN_DIR = bin
MAIN = main
EXECUTABLE = $(BIN_DIR)/$(MAIN)

ADAFLAGS = -gnat2012 -gnateE -gnatwa -O2
INCLUDES = -I$(SRC_DIR) -D $(OBJ_DIR)

all: $(EXECUTABLE)

$(EXECUTABLE): $(SRC_DIR)/$(MAIN).adb
        mkdir -p $(OBJ_DIR) $(BIN_DIR)
        $(GNATMAKE) $(SRC_DIR)/$(MAIN).adb $(ADAFLAGS) $(INCLUDES) -o $@

clean:
        rm -rf $(OBJ_DIR)/* $(BIN_DIR)/*

rebuild: clean all

.PHONY: all clean rebuild
```

Travail à réaliser :

1. Implémenter tous les corps de packages
2. Créer un programme principal qui utilise toutes les fonctions
3. Tester la compilation avec le Makefile
4. Modifier le Makefile pour ajouter des règles de test

Exercice 20.3 : Fichier de projet GNAT (.gpr)

Objectif : Utiliser les fichiers de projet GNAT pour gérer des projets complexes.

Fichier projet.gpr :

```
project Projet is
   for Source_Dirs use ("src", "src/utils", "src/gui");
   for Object_Dir use "obj";
   for Exec_Dir use "bin";
   for Main use ("main.adb");

   type Mode_Type is ("distrib", "debug", "optimize");
   Mode : Mode_Type := external ("MODE", "debug");

   package Compiler is
      Common_Options := ("-gnat2012", "-gnateE", "-gnatwa", "-gnatU");

      case Mode is
         when "distrib" =>
            for Default_Switches ("Ada") use Common_Options & ("-O2", "-gnatn");
         when "debug" =>
            for Default_Switches ("Ada") use Common_Options & ("-g", "-gnateE");
         when "optimize" =>
            for Default_Switches ("Ada") use Common_Options & ("-O3", "-gnatp");
      end case;
   end Compiler;

   package Binder is
      for Default_Switches ("Ada") use ("-Es");
   end Binder;

   package Linker is
      case Mode is
         when "distrib" =>
            for Default_Switches ("Ada") use ("-s");
         when others =>
            null;
      end case;
   end Linker;
end Projet;
```

Compilation avec gprbuild :

```
 -- Mode debug (par défaut)
gprbuild -P projet.gpr

-- Mode optimisé
gprbuild -P projet.gpr -XMODE=optimize

-- Mode distribution
gprbuild -P projet.gpr -XMODE=distrib

-- Nettoyage
gprclean -P projet.gpr
```

Exercice 20.4 : Tests unitaires et intégration continue

Objectif : Mettre en place des tests automatisés et un système de build continu.

Structure des tests :

```
tests/
├── unit_tests/
│   ├── test_math_utils.adb
│   ├── test_statistics.adb
│   └── test_runner.adb
├── integration_tests/
│   └── test_complete_workflow.adb
└── performance_tests/
    └── benchmark.adb
```

Exemple de test unitaire :

```
with Ada.Text_IO;
with Math_Utils;

procedure Test_Math_Utils is
   procedure Test_GCD is
   begin
      if Math_Utils.GCD(12, 8) /= 4 then
         raise Program_Error with "Test GCD failed: expected 4";
      end if;
      Ada.Text_IO.Put_Line("Test GCD: PASSED");
   end Test_GCD;

   procedure Test_Factorial is
   begin
      if Math_Utils.Factorial(5) /= 120 then
         raise Program_Error with "Test Factorial failed: expected 120";
      end if;
      Ada.Text_IO.Put_Line("Test Factorial: PASSED");
   end Test_Factorial;

begin
   Ada.Text_IO.Put_Line("=== Tests Math_Utils ===");
   Test_GCD;
   Test_Factorial;
   Ada.Text_IO.Put_Line("Tous les tests Math_Utils ont réussi !");
exception
   when E : Program_Error =>
      Ada.Text_IO.Put_Line("ÉCHEC: " & Ada.Exceptions.Exception_Message(E));
end Test_Math_Utils;
```

Script de build automatisé (build.sh) :

```
#!/bin/bash
set -e

echo "=== Build automatisé ADA ==="

# Nettoyage
echo "Nettoyage..."
gprclean -P projet.gpr

# Compilation en mode debug
echo "Compilation debug..."
gprbuild -P projet.gpr -XMODE=debug

# Exécution des tests unitaires
echo "Tests unitaires..."
./bin/test_runner
```

```
# Compilation optimisée
echo "Compilation optimisée..."
gprbuild -P projet.gpr -XMODE=optimize

# Tests de performance
echo "Tests de performance..."
./bin/benchmark

# Compilation finale
echo "Build de distribution..."
gprbuild -P projet.gpr -XMODE=distrib

echo "Build terminé avec succès !"
```

Exercice 20.5 : Déploiement et packaging

Objectif : Créer des packages pour la distribution de l'application.

Structure de déploiement :

```
deploy/
├── bin/
│   └── mon_application
├── lib/
│   └── libmath_utils.so
├── config/
│   └── app.conf
├── docs/
│   ├── README.md
│   └── INSTALL.md
├── scripts/
│   ├── install.sh
│   └── uninstall.sh
└── package/
    ├── debian/
    └── rpm/
```

Script d'installation (install.sh) :

```
#!/bin/bash

APP_NAME="mon_application"
INSTALL_DIR="/opt/$APP_NAME"
BIN_DIR="/usr/local/bin"

echo "Installation de $APP_NAME..."

# Création des répertoires
sudo mkdir -p $INSTALL_DIR
sudo mkdir -p $INSTALL_DIR/bin
sudo mkdir -p $INSTALL_DIR/lib
sudo mkdir -p $INSTALL_DIR/config

# Copie des fichiers
sudo cp bin/* $INSTALL_DIR/bin/
sudo cp lib/* $INSTALL_DIR/lib/
sudo cp config/* $INSTALL_DIR/config/

# Création du lien symbolique
sudo ln -sf $INSTALL_DIR/bin/$APP_NAME $BIN_DIR/$APP_NAME
```

```
# Configuration des permissions
sudo chmod +x $INSTALL_DIR/bin/*
sudo chown -R root:root $INSTALL_DIR

echo "Installation terminée."
echo "Utilisez '$APP_NAME' pour lancer l'application."
```

Création d'un package Debian :

```
 # Structure debian/
debian/
├── control
├── changelog
├── rules
├── compat
└── install

# Fichier control
Package: mon-application
Version: 1.0.0
Section: utils
Priority: optional
Architecture: amd64
Depends: libc6
Maintainer: Votre Nom <email@example.com>
Description: Application ADA exemple
 Description détaillée de l'application
 développée en ADA.
```

Commandes de packaging :

```
 # Création du package Debian
dpkg-buildpackage -us -uc

# Création d'un tarball
tar -czf mon-application-1.0.0.tar.gz deploy/

# Vérification du package
lintian ../mon-application_1.0.0_amd64.deb
```

21. Conclusion et ressources

Félicitations !

Vous avez terminé ce parcours complet d'exercices ADA. Vous devriez maintenant maîtriser :

- Les concepts fondamentaux du langage ADA
- La programmation orientée objet et générique
- La programmation concurrente et temps réel
- Les aspects avancés comme les contrats et la vérification formelle
- La compilation, les tests, et le déploiement

Ressources pour aller plus loin :

- **Documentation officielle :** Ada Reference Manual (ARM)
- **Compilateur :** GNAT Community Edition
- **IDE :** GPS (GNAT Programming Studio), AdaCore GNAT Studio
- **Livres recommandés :**

    - "Programming in Ada 2012" par John Barnes
    - "Ada as a Second Language" par Norman Cohen
    - "Concurrent and Real-Time Programming in Ada" par Alan Burns
- **Communautés :**
    - Ada Information Clearinghouse
    - comp.lang.ada newsgroup
    - AdaCore forums

Projets suggérés pour continuer :

1. Contribuer à des projets open source en ADA
2. Développer des applications temps réel
3. Explorer SPARK pour la vérification formelle
4. Participer aux concours de programmation ADA

Document d'exercices ADA - Version complète

Bonne continuation dans votre apprentissage d'ADA !