TP6 – Recursivité

Algorithmique 3 (Programmation en C)

Noursine Abbassi
2ème Année – ISIL C
Université des Sciences et Technologies Houari Boumediene (USTHB)

Objectif général

- Travailler la **récursivité**.
- Manipuler des **structures de données personnalisées**.
- Organiser le code en **headers (.h) + implémentations (.c)**.
- Comprendre l'intérêt de cette architecture en utilisant plusieurs `main.c` qui utilisent le même module.

Exercice 1 — Gestion d'une file d'attente simple (File FIFO)

Objectif : Créer un module `queue.h / queue.c` gérant une file d'attente de clients dans un service public.

Travail demandé

- Dans `queue.h` :

```c
typedef struct {
    int id;
    char nom[30];
} Client;

typedef struct {
    Client tab[50];
    int tete, queue, taille;
} File;

void initFile(File *f);
int estVide(File *f);
int estPleine(File *f);
void enfiler(File *f, Client c);
Client defiler(File *f);
void afficherFile(File *f);
int trouverClient(File *f, int index, char nomCherche[30]);
int trouverClientRec(File *f, int index, char nomCherche[30]);
```

- **Récursivité :** `trouverClientRec` - fonction récursive qui cherche un client par nom et retourne son ID en parcourant la file.
- **Main 1 :** `main_service.c`
  - Ajouter 5 clients.

  - Chercher récursivement un client.
  - Afficher la file.
- **Main 2 :** `main_test.c`
  - Comparaison entre la fonction de recherche itérative et la fonction de recherche récursive.

But pédagogique : Montrer qu'un même header sert à plusieurs programmes.

Exercice 2 — Gestion d'un dossier médical (liste chaînée + comparaison récursif vs itératif)

Contexte réel : Stocker les dossiers médicaux d'un hôpital avec analyse de performance avancée.

Travail demandé

- Module : `patients.h / patients.c`

```c
typedef struct Patient {
    int id;
    char nom[30];
    int age;
    char diagnostic[100];
    int gravite; // 1-5 (1=faible, 5=critique)
    struct Patient *suiv;
} Patient;

// Fonctions de base
Patient* ajouterPatient(Patient* tete, Patient p);
void afficherPatients(Patient* tete);
void libererListe(Patient* tete);

// Fonctions de comptage - comparaison
int compterPatientsIteratif(Patient* tete);
int compterPatientsRecursif(Patient* tete);

// Fonctions de recherche - comparaison
Patient* rechercherIteratif(Patient* tete, int id);
Patient* rechercherRecursif(Patient* tete, int id);

// FONCTION PLUS OPTIMALE EN RCURSIF
// Tri de la liste par gravité (récursif naturellement efficace)
Patient* trierParGraviteRecursif(Patient* tete);
Patient* trierParGraviteIteratif(Patient* tete);

// Recherche dans liste triée (meilleure en récursif)
Patient* rechercheOptimaleRec(Patient* tete, int gravite);
Patient* rechercheOptimaleIter(Patient* tete, int gravite);

// Fonctions de mesure de performance
void mesurerTempsExecution(Patient* tete);
```

```
34 void comparerAlgorithmes(Patient* tete);
```

— **Main 1 :** `main_hopital.c`
  — Ajoute des patients avec différents niveaux de gravité
  — Affiche la liste
  — Compare les temps d'exécution des différentes approches
— **Main 2 :** `main_performance.c`
  — Teste spécifiquement les performances avec différentes tailles
  — Génère un rapport comparatif détaillé

Objectif pédagogique : Montrer que pour certains types de problèmes, l'approche récursive peut être plus naturelle et parfois plus performante.

Consignes spécifiques

— Implémenter les deux versions (récursive et itérative) du tri par gravité
— Pour la fonction récursive, utiliser une approche "diviser pour régner"
— Pour la version itérative, utiliser une approche de tri classique
— Mesurer les temps d'exécution pour différentes tailles de liste
— Analyser la complexité algorithmique de chaque approche

Questions de réflexion

1. Pourquoi l'approche récursive est-elle plus naturelle pour le tri ?
2. Quel est l'impact de la taille de la liste sur les performances ?
3. Dans quels autres cas la récursivité pourrait-elle être avantageuse ?
4. Comment gérer le risque de stack overflow avec les grandes listes ?

Exercice 3 — Gestion récursive de tickets de support (pile = stack)

Contexte réel : Centre de support technique : chaque ticket ajouté va au sommet (pile de priorités).

Travail demandé

— Module : `stack.h / stack.c`

```c
typedef struct {
    int idTicket;
    char description[100];
} Ticket;

typedef struct {
    Ticket t[100];
    int sommet;
} Stack;

void initStack(Stack *s);
```

```
int isEmpty(Stack *s);
void push(Stack *s, Ticket t);
Ticket pop(Stack *s);
void afficherStack(Stack *s, int index); // récursivité
```

— **Récursivité :** `afficherStack` affiche la pile du bas vers le haut sans boucle.

— **Main :** `main_support.c`

  — Ajouter plusieurs tickets

  — Afficher récursivement

  — Retirer un ticket

But pédagogique :

— Manipulation d'une structure différente (pile)

— Recursion pour parcourir la pile

Exercice 4 — DEVOIR MAISON : Système bancaire complet avec persistance

Contexte réel : Créer un système bancaire complet avec persistance des données et multiples interfaces.

Modules nécessaires

— `queue.h / queue.c` (file d'attente des clients)

— `compte.h / compte.c` (gestion des comptes bancaires)

— `transaction.h / transaction.c` (gestion des transactions)

— `fichier.h / fichier.c` (persistance des données)

Structures de données

```
// compte.h
typedef struct Compte {
    int numero;
    char titulaire[50];
    double solde;
    char date_creation[20];
} Compte;

typedef struct ListeComptes {
    Compte *comptes;
    int nombre;
    int capacite;
} ListeComptes;

// transaction.h
typedef struct Transaction {
    int id;
    int numero_compte;
```

```
    char type[20]; // "depot", "retrait", "virement"
    double montant;
    char date[20];
    struct Transaction *suiv;
} Transaction;
```

Mains multiples

- **Main 1 :** `main_creation.c`
  - Création de nouveaux comptes
  - Sauvegarde automatique dans `comptes.dat`
- **Main 2 :** `main_depot.c`
  - Effectuer des dépôts
  - Mise à jour du solde et historique
- **Main 3 :** `main_retrait.c`
  - Effectuer des retraits avec vérification de solde
  - Gestion des découverts
- **Main 4 :** `main_consultation.c`
  - Consultation des soldes
  - Affichage de l'historique des transactions
  - Calculs récursifs des totaux
- **Main 5 :** `main_performance.c`
  - Comparaison récursif vs itératif sur l'historique

Fonctions de persistance

```
// fichier.h
int sauvegarderComptes(ListeComptes *liste);
int chargerComptes(ListeComptes *liste);
int sauvegarderTransaction(Transaction *tete);
Transaction* chargerTransactions();
```

Fonctions récursives vs itératives

```
// transaction.c - Comparaison de performance
double totalDepotsIteratif(Transaction *tete) {
    double total = 0;
    Transaction *courant = tete;
    while (courant != NULL) {
        if (strcmp(courant->type, "depot") == 0) {
            total += courant->montant;
        }
        courant = courant->suiv;
    }
    return total;
```

```
}

double totalDepotsRecursif(Transaction *tete) {
    if (tete == NULL) return 0;
    double total = totalDepotsRecursif(tete->suiv);
    if (strcmp(tete->type, "depot") == 0) {
        total += tete->montant;
    }
    return total;
}
```

Exemple d'exécution

```
=== Système Bancaire Complet ===
[main_creation] Compte 1001 créé pour Pierre Martin
[main_depot] Dépôt de 500.00 euro sur le compte 1001
[main_retrait] Retrait de 200.00 euro du compte 1001
[main_consultation] Solde actuel : 300.00 euro

=== Analyse de Performance ===
Calcul des dépôts :
- Itératif : 0.034 ms, Total : 1500.00 euro
- Récursif : 0.089 ms, Total : 1500.00 euro

Données sauvegardées dans comptes.dat et transactions.dat
```

Structure du projet modifié

```
banque_project/
    include/
        queue.h
        compte.h
        transaction.h
        fichier.h
    src/
        queue.c
        compte.c
        transaction.c
        fichier.c
        main_creation.c
        main_depot.c
        main_retrait.c
        main_consultation.c
        main_performance.c
    data/
        comptes.dat
        transactions.dat
    bin/
    Makefile
```

Makefile adapté

```
# Cibles pour chaque main
creation: $(OBJDIR)/main_creation.o $(OBJDIR)/compte.o $(OBJDIR)/fichier.o
	$(CC) $(CFLAGS) -o $(BINDIR)/main_creation $^

depot: $(OBJDIR)/main_depot.o $(OBJDIR)/compte.o $(OBJDIR)/transaction.o $(OBJDIR)/fichier.o
	$(CC) $(CFLAGS) -o $(BINDIR)/main_depot $^

retrait: $(OBJDIR)/main_retrait.o $(OBJDIR)/compte.o $(OBJDIR)/transaction.o $(OBJDIR)/fichier.o
	$(CC) $(CFLAGS) -o $(BINDIR)/main_retrait $^

consultation: $(OBJDIR)/main_consultation.o $(OBJDIR)/compte.o $(OBJDIR)/transaction.o $(OBJDIR)/fichier.o
	$(CC) $(CFLAGS) -o $(BINDIR)/main_consultation $^

performance: $(OBJDIR)/main_performance.o $(OBJDIR)/compte.o $(OBJDIR)/transaction.o $(OBJDIR)/fichier.o
	$(CC) $(CFLAGS) -o $(BINDIR)/main_performance $^

all: creation depot retrait consultation performance
```