

# De l'exigence au prototype

Analyser un besoin, puis lire et faire évoluer un programme C

Ressource R3.03 — BUT2 Informatique

4 TD et 4 TP de 1 h 15, en alternance — travail en groupe de 3 ou 4

## Résumé

Huit séances pour formuler des exigences, les transposer dans un programme C fourni et vérifier le résultat. Vous étudierez le besoin du laboratoire BioSeq, préciserez les scénarios, puis ferez évoluer quatre petites parties du prototype. Les structures, les en-têtes et les pointeurs de fonctions restent au programme, avec une lecture guidée et des vérifications individuelles.

Chaque TP utilise le livrable du TD qui le précède. Le parcours de répertoires, l'adaptateur POSIX, la doublure et les tests sont fournis dans la base guidée.

## Table des matières

|                                                                       |           |
|-----------------------------------------------------------------------|-----------|
| <b>Organisation et évaluation</b>                                     | <b>3</b>  |
| <b>1 TD1 : du dossier client aux exigences</b>                        | <b>5</b>  |
| 1.1 Les faits et leurs sources : 10 minutes . . . . .                 | 5         |
| 1.2 Le besoin et les questions : 15 minutes . . . . .                 | 5         |
| 1.3 Cinq exigences prioritaires : 25 minutes . . . . .                | 5         |
| 1.4 Préparer l'observation, puis lire le code : 15 minutes . . . . .  | 6         |
| 1.5 Validation et passage au TP : 10 minutes . . . . .                | 6         |
| <b>2 TP1 : lire le programme et rendre son bilan explicite</b>        | <b>7</b>  |
| 2.1 Diagnostic individuel, non noté : 10 minutes . . . . .            | 7         |
| 2.2 Compiler et observer : 10 minutes . . . . .                       | 7         |
| 2.3 Suivre une valeur entre plusieurs fichiers : 15 minutes . . . . . | 7         |
| 2.4 Modifier le bilan et ajouter un cas : 25 minutes . . . . .        | 8         |
| 2.5 Contrôle collectif : 10 minutes . . . . .                         | 8         |
| 2.6 Exercice individuel : 5 minutes . . . . .                         | 8         |
| <b>3 TD2 : du scénario à une règle locale</b>                         | <b>9</b>  |
| 3.1 Deux stories relues : 15 minutes . . . . .                        | 9         |
| 3.2 Le scénario nominal : 20 minutes . . . . .                        | 9         |
| 3.3 Une règle à préciser : 20 minutes . . . . .                       | 9         |
| 3.4 Contrat et cas de test : 15 minutes . . . . .                     | 10        |
| 3.5 Validation : 5 minutes . . . . .                                  | 10        |
| <b>4 TP2 : corriger une règle et retrouver son appel</b>              | <b>11</b> |
| 4.1 Relire le contrat : 10 minutes . . . . .                          | 11        |
| 4.2 Prédire le calcul : 15 minutes . . . . .                          | 11        |
| 4.3 Corriger et suivre l'utilisation : 25 minutes . . . . .           | 11        |
| 4.4 Observer la réutilisation réelle : 15 minutes . . . . .           | 12        |
| 4.5 Point de contrôle : 5 minutes . . . . .                           | 12        |
| 4.6 Exercice individuel : 5 minutes . . . . .                         | 12        |

|          |                                                                   |           |
|----------|-------------------------------------------------------------------|-----------|
| <b>5</b> | <b>TD3 : définir les exceptions avant de provoquer les pannes</b> | <b>13</b> |
| 5.1      | Les étapes fragiles : 10 minutes . . . . .                        | 13        |
| 5.2      | Trois incidents communs : 25 minutes . . . . .                    | 13        |
| 5.3      | Arbitrer la suite du traitement : 15 minutes . . . . .            | 14        |
| 5.4      | Préparer les tests : 20 minutes . . . . .                         | 14        |
| 5.5      | Validation : 5 minutes . . . . .                                  | 14        |
| <b>6</b> | <b>TP3 : suivre un appel indirect et corriger un repli</b>        | <b>15</b> |
| 6.1      | Vérifier le nominal : 10 minutes . . . . .                        | 15        |
| 6.2      | Décomposer l'appel : 15 minutes . . . . .                         | 15        |
| 6.3      | Provoquer la panne et modifier le repli : 25 minutes . . . . .    | 16        |
| 6.4      | Vérifier résultat et effets : 15 minutes . . . . .                | 16        |
| 6.5      | Point de contrôle : 5 minutes . . . . .                           | 16        |
| 6.6      | Exercice individuel : 5 minutes . . . . .                         | 16        |
| <b>7</b> | <b>TD4 : du modèle de domaine aux invariants</b>                  | <b>17</b> |
| 7.1      | Trier les candidats : 10 minutes . . . . .                        | 17        |
| 7.2      | Relations et multiplicités : 20 minutes . . . . .                 | 17        |
| 7.3      | Un processus met le modèle à l'épreuve : 15 minutes . . . . .     | 17        |
| 7.4      | L'automate et ses garanties : 20 minutes . . . . .                | 18        |
| 7.5      | Les tests du TP : 10 minutes . . . . .                            | 18        |
| <b>8</b> | <b>TP4 : contrôler la finalisation et relire un autre travail</b> | <b>19</b> |
| 8.1      | Lire les états et prédire : 10 minutes . . . . .                  | 19        |
| 8.2      | Corriger la garde : 20 minutes . . . . .                          | 19        |
| 8.3      | Tester l'absence d'une publication : 15 minutes . . . . .         | 19        |
| 8.4      | Revue croisée : 20 minutes . . . . .                              | 20        |
| 8.5      | Traçabilité et contrôle final : 5 minutes . . . . .               | 20        |
| 8.6      | Exercice individuel : 5 minutes . . . . .                         | 20        |

## Votre travail pendant les quatre séquences

Le laboratoire BioSeq a besoin de sauvegardes consultables et d'un résultat de traitement compréhensible. Vous reprenez un prototype C fourni et vous le faites évoluer. Votre responsabilité est de **relier chaque modification à une exigence et à un test**, puis de pouvoir expliquer le chemin d'exécution concerné.

### Les règles du travail

Quatre séquences, chacune composée d'un CM, d'un TD de 75 minutes et d'un TP de 75 minutes. Groupes de **3 ou 4**. Le TP utilise directement la production du TD qui le précède. Aucun développement à la maison n'est nécessaire entre les deux.

| Séquence | Le TD produit                  | Le TP confronte au C              |
|----------|--------------------------------|-----------------------------------|
| 1        | Exigences et observations      | Bilan d'un traitement             |
| 2        | Scénario nominal et règle      | Condition de réutilisation        |
| 3        | Flux alternatifs et tests      | Transfert et repli après incident |
| 4        | Modèle, automate et invariants | Finalisation et publication       |

## Votre base C

L'archive `base-guidee.tar.gz` contient le parcours de répertoires, l'adaptateur POSIX, les entêtes, une doublure et les tests. Elle compile dès le départ. Quatre fichiers contiennent des décisions volontairement incomplètes à corriger au fil des séances. Les tests correspondants peuvent donc échouer **sans que votre installation soit défectueuse**.

| Fichier à faire évoluer         | Question posée                                    |
|---------------------------------|---------------------------------------------------|
| <code>src/rapport.c</code>      | Le bilan rend-il visible un échec de traitement ? |
| <code>src/regles.c</code>       | La décision respecte-t-elle toute la règle ?      |
| <code>src/transfert.c</code>    | Que fait-on après l'échec d'un lien ?             |
| <code>src/finalisation.c</code> | À quelles conditions publie-t-on le résultat ?    |

Vous devez aussi lire `src/sg.h`, les points d'appel indiqués dans `src/service.c` et les tests. Vous n'avez pas à réécrire `src/fs_posix.c`. Travaillez avec `-std=c11 -Wall -Wextra -Wpedantic`, sans ignorer les avertissements.

### ✗ À ne pas faire : Des données de laboratoire uniquement

Ce prototype d'apprentissage comporte des défauts annoncés. Utilisez uniquement les petits jeux jetables créés par les commandes fournies. Aucun essai sur vos documents personnels. Les tests d'intégration créent leur propre répertoire temporaire et le suppriment ensuite.

## Périmètre du prototype

Le besoin du client reste celui du dossier BioSeq. Le prototype étudie un sous-ensemble : source stable pendant l'exécution, fichiers ordinaires et répertoires, destination nouvelle et séparée de la source. Les liens symboliques et fichiers spéciaux sont ignorés. Les jeux fournis ont quelques fichiers, au plus 256 entrées par répertoire et des chemins de moins de 4096 caractères. Les instantanés précédents restent en lecture seule. Les exécutions concurrentes, la purge, les empreintes de contenu, les notifications et la résistance aux pannes électriques restent hors périmètre. Ces limites ne deviennent pas des besoins du client.

## Répartition des rôles

En TD : porte-parole du client, rédacteur, contradicteur et rapporteur. À trois, le rédacteur assure la restitution. En TP : pilote au clavier, lecteur du contrat, testeur et, à quatre, lecteur de la trace d'exécution. Les rôles tournent dans les phases de manipulation, environ toutes les 10 à 15 minutes. Chacun doit piloter au moins une fois sur deux TP successifs.

Chaque membre écrit d'abord sa prédiction de résultat, puis la confronte au groupe. L'enseignant peut demander à un autre membre que le pilote d'expliquer un appel. Un seul dépôt par groupe, des contributions identifiées et un journal très court des décisions. On ne répartit pas durablement « le code » et « le dossier » entre personnes.

## Ce qui est rendu et évalué

| Note          | Éléments sur 20                                                                                                  | Coef. |
|---------------|------------------------------------------------------------------------------------------------------------------|-------|
| G1, groupe    | Exigences justifiées (6), stories, scénario et contrats (8), exceptions (6). Consolidation après TP3, avant TD4. | 2     |
| G2, groupe    | Modifications expliquées (6), tests (6), modèle et automate (4), traçabilité (4). Rendu après TP4.               | 3     |
| G3, groupe    | Revue croisée : protocole (8), observations et conclusions (8), amélioration argumentée (4). Fin TP4.            | 1     |
| I, individuel | Quatre exercices de cinq minutes, chacun sur 5 : lecture (2), micro-correction (2), justification ou test (1).   | 2     |

$$\text{Note finale} = \frac{2G1 + 3G2 + G3 + 2I}{8}.$$

Git sert à retrouver les changements et leur auteur. Le nombre de lignes ou de commits ne donne pas directement des points. Les exercices individuels portent sur le C étudié en séance, avec une donnée ou une condition différente.

Le dossier évolue au fil des séances : environ six pages utiles, hors code. Une page d'exigences, deux pages de stories/scénario/règle, une page d'exceptions, une page de modèles, une page de traçabilité. La revue croisée tient sur une page séparée.

### Travail à la maison — Avant S1 et entre les séquences

Avant S1 : 30 minutes de lecture du dossier, avec une question à apporter. Groupe et accès à la forge sont préparés en amont. Après S1, S2 et S3 : 45 minutes pour relire l'extrait C étudié, répondre aux questions indiquées et améliorer votre dossier. Après S4 : 30 minutes pour la mise au propre. On ne reporte pas à la maison un développement indispensable à la séance suivante.

## Si vous bloquez

Après avoir identifié la ligne et formulé ce que vous ne comprenez pas, demandez une aide : orientation vers un fichier, trace partielle, puis version commentée de reprise si nécessaire. Conservez votre essai et expliquez la différence. Utiliser une aide n'annule pas l'obligation de comprendre le code. Une base de reprise peut être fournie avant la séquence suivante sans effacer votre travail.

# 1 TD1 : du dossier client aux exigences

## 75 minutes : la méthode du CM1, appliquée à BioSeq

Fait et source, besoin, exigence, observation. **Production** : trois exigences fonctionnelles et deux exigences de qualité, dont une sur la visibilité du résultat. Vous lirez le code après avoir formulé ce que l'utilisateur doit pouvoir constater.

### 1.1 Les faits et leurs sources : 10 minutes

Chacun présente sa lecture du dossier BioSeq. Le groupe retient trois faits utiles. Relevez le passage ou l'interlocuteur qui les établit. Un fait chiffré n'est pas automatiquement un objectif de performance.

**Q1.1.** Pour chaque fait, indiquer qui subit le problème et une conséquence concrète. Classer séparément ce qui n'est qu'une hypothèse de votre groupe.

### 1.2 Le besoin et les questions : 15 minutes

**Q1.2.** Écrire en une phrase la demande initiale, puis le besoin à satisfaire. Donner une raison pour laquelle réaliser littéralement la demande peut laisser le besoin insatisfait.

**Q1.3.** Proposer deux questions ouvertes qui permettraient de clarifier le besoin. Choisir la plus utile et expliquer quel choix dépendrait de sa réponse. Toute réponse supplémentaire donnée par l'enseignant est ajoutée au dossier avec sa date et son statut.

### 1.3 Cinq exigences prioritaires : 25 minutes

Formuler trois EF et deux ENF. Chaque exigence exprime une seule propriété. Au moins l'une concerne l'observation d'un résultat de sauvegarde comportant une erreur.

| ID    | Exigence et catégorie justifiée | Source ou hypothèse | Observation et contexte |
|-------|---------------------------------|---------------------|-------------------------|
| EF-1  |                                 |                     |                         |
| EF-2  |                                 |                     |                         |
| EF-3  |                                 |                     |                         |
| ENF-1 |                                 |                     |                         |
| ENF-2 |                                 |                     |                         |

**Q1.4.** Une exigence dit « faire un lien dur ». Est-ce un service attendu, une contrainte imposée ou une solution envisagée ? Justifier à partir du dossier, puis reformuler le besoin auquel elle pourrait répondre.

**Q1.5.** Pour chaque seuil proposé, choisir : fait client, hypothèse à confirmer ou objectif d'essai pédagogique. Une mesure sur le petit jeu du TP ne prouve pas une performance sur les 400 Gio du client.

## 1.4 Préparer l'observation, puis lire le code : 15 minutes

Pour l'exigence sur le résultat, préparer deux situations contrastées : traitement sans erreur et traitement avec un fichier non transféré. Écrire ce que l'utilisateur doit pouvoir distinguer, indépendamment du texte exact du message.

```
1 typedef struct {  
2     int vus;  
3     int copies;  
4     int lies;  
5     int erreurs;  
6 } sg_rapport_t;  
7 const char *sg_bilan(const sg_rapport_t *r);
```

**Q1.6.** Quel champ peut contribuer à cette distinction ? À quelle exigence reliez-vous cette fonction ? Quels renseignements ce rapport ne donne-t-il pas sur la publication de l'instantané ?

## 1.5 Validation et passage au TP : 10 minutes

Un autre membre du groupe relit chaque exigence et propose un cas qui pourrait la réfuter. L'enseignant valide les observations à mettre en œuvre.

### ✓✓ Point d'arrêt — À ouvrir au début du TP1

Votre tableau de cinq exigences, vos deux situations de bilan et une première ligne de traçabilité : source, exigence, fonction concernée, observation attendue.

## 2 TP1 : lire le programme et rendre son bilan explicite

### 75 minutes : un chantier dans `rapport.c`

On vous fournit un programme qui sait déjà copier une petite arborescence. Le bilan est volontairement insuffisant. Votre modification doit permettre de distinguer les situations préparées au TD.

### 2.1 Diagnostic individuel, non noté : 10 minutes

Sans exécuter le code, répondre aux six questions. Le diagnostic sert à choisir une aide, pas à attribuer une note.

```
1 typedef struct { int n; } boite_t;
2 int lire(const boite_t *p);
3 boite_t b = { .n = 3 };
4 boite_t copie = b;
5 copie.n = 8;
6 const boite_t *p = &b;
7 int (*operation)(const boite_t *) = lire;
```

1. Que valent `b.n` et `copie.n` ?
2. Que représente `&b` ?
3. Écrire la lecture de `n` à travers `p`.
4. Quelle ligne est un prototype ? Dans quel type de fichier le partager ?
5. Quel type de résultat `lire` renvoie-t-elle ?
6. Dans `operation(p)`, comment la fonction appelée a-t-elle été choisie ?

### 2.2 Compiler et observer : 10 minutes

Extraire la base fournie dans le dépôt du groupe, puis se placer dans son dossier.

```
make depart
make observer
```

`make depart` doit réussir : il compile et vérifie une copie nominale. `make observer` présente deux rapports A et B. Noter les champs et les deux étiquettes. Comparer aux observations du TD avant d'exécuter :

```
make test-s1
```

Le test R2 doit échouer dans la version initiale. Lire la condition testée dans `tests/test_seances.c`, fonction `sequence1`.

### 2.3 Suivre une valeur entre plusieurs fichiers : 15 minutes

**Q2.1.** Retrouver le type du rapport et le prototype dans `src/sg.h`. Chercher ensuite la définition dans `src/rapport.c` et l'appel dans `src/main.c`.

**Q2.2.** Le programme transmet-il le rapport entier ou son adresse à `sg_bilan` ? Expliquer `&rapport`, `r->erreurs` et la différence avec `rapport.erreurs`.

**Q2.3.** Dessiner une trace courte : le service remplit le rapport, le programme principal appelle le bilan, puis affiche son résultat. Indiquer qui écrit le champ et qui le lit. Vous n'avez pas à détailler la récursion.

## 2.4 Modifier le bilan et ajouter un cas : 25 minutes

**Contrat technique du TP, correspondant aux observations validées en TD :** `sg_bilan` renvoie la chaîne "OK" si le traitement ne comporte aucune erreur, et "ERREURS" sinon. Cette étiquette décrit le traitement des fichiers. La publication a son propre résultat.

1. Prédire les valeurs renvoyées par la version actuelle pour A et B.
2. Modifier seulement `src/rapport.c` pour respecter le contrat. Conserver la signature et expliquer le rôle de `const`.
3. Rejouer `make observer` et `make test-s1`.
4. Dans `tests/mes_tests.c`, ajouter un rapport avec plusieurs erreurs et vérifier le bilan avec `strcmp`. Ajouter `#include <string.h>`.
5. Exécuter `make test-perso`. Indiquer l'exigence du TD que votre assertion vérifie.

## 2.5 Contrôle collectif : 10 minutes

Faire expliquer le changement par un membre qui ne l'a pas tapé. Conserver le diagnostic en échec avant modification et le résultat après correction, sous forme de quelques lignes dans le journal. Créer un commit identifié par l'exigence.

### ✓✓ Point d'arrêt — Avant de quitter la séance

`make depart`, `make test-s1` et votre test personnel passent. Chacun sait retrouver déclaration, définition et appel, puis expliquer le champ utilisé. Les échecs des tests S2 à S4 seront étudiés plus tard.

## 2.6 Exercice individuel : 5 minutes

L'enseignant distribue un extrait voisin : prédiction, micro-correction et justification. Travail sans échange avec le groupe.

### 🏠 Travail à la maison — Après S1 : 45 minutes

Améliorer les cinq exigences à partir du retour. Relire `rapport.c` et le type du rapport. Écrire individuellement trois phrases : ce que transmet `&rapport`, ce que lit `r->erreurs`, et ce que prouve le test R2. Préparer une story pour un acteur du dossier BioSeq.

### 3 TD2 : du scénario à une règle locale

#### 75 minutes : la méthode du CM2

Objectif utilisateur, scénario, décision locale, table de décision, contrat et tests. **Production :** deux stories, un scénario nominal, une table et un contrat.

#### 3.1 Deux stories relues : 15 minutes

À partir des propositions individuelles, retenir deux stories avec des acteurs ou objectifs différents. Utiliser la forme « En tant que... , je veux... , afin de... ». Au moins une porte sur la création d'une sauvegarde ou la disponibilité de son résultat.

**Q3.1.** Relire les stories avec INVEST. Identifier une formulation trop technique ou trop large et expliquer votre reformulation. Les deux stories restent dans le dossier, même si une seule est développée en scénario.

#### 3.2 Le scénario nominal : 20 minutes

Décrire « Créer un instantané consultable » en cinq à sept étapes, au niveau des interactions et des résultats. Indiquer acteur, déclencheur, préconditions et postcondition de succès. Les cas d'erreur seront développés au TD3.

**Q3.2.** Quelle étape garantit la disponibilité du résultat ? Pourquoi une liste d'appels open, read et write ne constitue-t-elle pas le scénario attendu ?

#### 3.3 Une règle à préciser : 20 minutes

##### Complément au dossier : décision pédagogique du client fictif

Pour ce prototype, un fichier peut être réutilisé depuis l'instantané précédent si un fichier ordinaire au même chemin relatif existe et si sa taille et sa date de modification sont identiques. Sinon, on copie le fichier courant. Le client fictif accepte ce compromis pour l'exercice, en sachant qu'il ne garantit pas l'identité du contenu.

**Conventions techniques fournies :** l'appelant a déjà rapproché les chemins. Il passe l'adresse des métadonnées précédentes si elles existent, sinon NULL. Les objets sont initialisés. Le type distingue ici fichiers ordinaires et répertoires. On ne modifie aucun objet pour décider.

| Situation                                | Réutiliser ? | Justification |
|------------------------------------------|--------------|---------------|
| Référence absente                        |              |               |
| Deux fichiers, taille et date identiques |              |               |
| Taille différente, date identique        |              |               |
| Taille identique, date différente        |              |               |
| Taille et date différentes               |              |               |
| Référence qui est un répertoire          |              |               |

**Q3.3.** Compléter la table avant de lire l'implémentation. Proposer un changement de contenu qui conserverait taille et date. Quel risque le compromis laisse-t-il subsister ?

### 3.4 Contrat et cas de test : 15 minutes

La future lecture du code portera sur :

```
1 int sg_reutilisable(const sg_entree_t *courant,  
2                     const sg_entree_t *ancien);
```

**Q3.4.** Écrire le contrat : entrées, résultat 0/1, interprétation de NULL, répertoires et effets interdits. Par convention défensive, un courant absent donne également 0.

**Q3.5.** Proposer au moins cinq tests issus de la table, avec des tailles et dates concrètes. Choisir deux cas qui distinguent une comparaison de taille seule de la règle complète.

### 3.5 Validation : 5 minutes

#### ✓✓ Point d'arrêt — À emporter au TP2

Les deux stories, le nominal, la table et le contrat. Une ligne de traçabilité rattache la règle au besoin d'économie d'espace et à ses limites. Le mécanisme de lien dur est une solution fournie, pas une nouvelle exigence du client.

## 4 TP2 : corriger une règle et retrouver son appel

75 minutes : le chantier `regles.c`

Lire une condition avec deux pointeurs de structure, confronter ses résultats à la table du TD, corriger puis vérifier dans le programme complet. Le point d'appel et les primitives de copie et de lien sont déjà présents.

### 4.1 Relire le contrat : 10 minutes

Ouvrir la table du TD, puis `src/sg.h`. Retrouver les champs de `sg_entree_t`. Comparer le contrat écrit en TD avec celui de `sg_reutilisable`. Signaler une divergence avant de changer le code.

**Q4.1.** Pourquoi la décision prend-elle deux adresses ? Quels champs lit-elle ? Pourquoi ne faut-il pas interpréter « réutilisable » comme « contenu certainement identique » ?

### 4.2 Prédire le calcul : 15 minutes

Lire `src/regles.c`. Chaque membre remplit d'abord seul les colonnes suivantes pour trois cas de sa table : résultat attendu par le contrat, résultat calculé par le code, justification.

**Q4.2.** Expliquer le rôle de la garde sur `NULL`. Que provoquerait une lecture de `ancien->mtime` avant cette garde lorsque la référence est absente ?

```
make test-s2
```

Dans la base initiale, D4 échoue. Relier son assertion à une ligne de votre table.

### 4.3 Corriger et suivre l'utilisation : 25 minutes

1. Modifier la condition métier dans `src/regles.c`. Conserver les gardes et la signature.
2. Relancer `make test-s2`. Expliquer les résultats D1 à D5.
3. Dans `src/service.c`, retrouver l'appel à `sg_reutilisable`. Annoter la trace : `p = NULL`, renseignement de `info`, `p = &info`, décision, appel à `sg_transferer`.
4. Expliquer dans quel cas `reference` devient `NULL` et ce que cela signifie pour le transfert.
5. Ajouter dans `tests/mes_tests.c` un cas absent de D1–D5. Commenter sa justification, puis lancer `make test-perso`.

**Q4.3.** Dans le point d'appel, distinguer l'objet `info`, son adresse `&info` et le pointeur `p`. La fonction pourrait-elle conserver `p` pour l'utiliser après la fin de la fonction appelante ? Justifier à partir de la durée de vie de l'objet.

#### 4.4 Observer la réutilisation réelle : 15 minutes

Un lien dur ajoute un nom pour les mêmes données sur un même système de fichiers. Ici, le lien vise **l'instantané précédent**, jamais la source en cours de travail. Modifier un fichier d'un instantané lié modifierait aussi les données visibles par ses autres liens : les instantanés sont donc traités en lecture seule.

```
make integration-s2
```

Le script fourni crée une source et un premier instantané. Il vérifie les dates conservées. Une deuxième exécution doit réutiliser les fichiers inchangés. Il modifie ensuite un fichier en conservant sa taille mais en changeant sa date : la troisième exécution doit copier ce fichier et préserver l'ancien instantané.

**Q4.4.** Pourquoi le test emploie-t-il une date ancienne fixée plutôt que l'heure courante ? Quelle erreur l'observation du contenu permet-elle de détecter en plus du comptage ? Pourquoi un test local ne suffit-il pas à vérifier l'ensemble de cette coopération ?

#### 4.5 Point de contrôle : 5 minutes

##### ✓✓ Point d'arrêt — Votre preuve

La table, le défaut initial, le changement de condition, un test personnel justifié et une trace du point d'appel. `make test-s1 test-s2 test-perso integration-s2` doit réussir.

#### 4.6 Exercice individuel : 5 minutes

Lecture d'une condition portant sur deux structures, micro-correction et choix d'un cas discriminant. L'enseignant distribue l'extrait.

##### 📖 Travail à la maison — Après S2 : 45 minutes

Relire le contrat et la fonction corrigée. Écrire le résultat pour trois cas sans exécuter. Compléter le scénario nominal et relever deux étapes où un incident modifierait son déroulement. Préparer une question sur les pointeurs de fonctions à partir de la capsule du CM3.

## 5 TD3 : définir les exceptions avant de provoquer les pannes

### 75 minutes : la méthode du CM3

Repérer une étape fragile, décrire un flux alternatif, décider des effets autorisés et choisir un test.

**Production** : trois flux alternatifs et leur protocole de test.

### 5.1 Les étapes fragiles : 10 minutes

Relire votre scénario nominal. Pour chaque étape, chercher une entrée absente, une opération impossible ou un résultat dont on ignore le succès.

**Q5.1.** Choisir l'étape concernée par chacune des trois cartes suivantes. Les incidents ne se décrivent pas seulement par un message d'erreur : ils doivent aboutir à un résultat défini.

### 5.2 Trois incidents communs : 25 minutes

#### Carte A : source absente

Le chemin de la source ne désigne plus un répertoire disponible au début de la sauvegarde. Que doit-on annoncer ? Quelles écritures doivent être absentes ?

#### Carte B : lien impossible

Un fichier semble réutilisable, mais la création du lien échoue. Cela peut notamment arriver si les deux instantanés ne sont pas sur le même système de fichiers. Quel autre moyen peut satisfaire le besoin ?

#### Carte C : copie impossible

La copie d'un fichier échoue. Elle peut suivre ou non un échec de lien. Comment rendre l'échec visible sans prétendre que l'instantané est complet ?

Pour chaque carte : numéro de l'étape, condition de déclenchement, actions, point de reprise ou arrêt, postcondition et effet interdit. Le contradicteur propose un cas qui ferait apparaître une ambiguïté dans votre texte.

### 5.3 Arbitrer la suite du traitement : 15 minutes

**Q5.2.** Comparer les conséquences de « arrêter au premier fichier non transféré » et « poursuivre pour établir un rapport ». Distinguer cette décision de la question « peut-on publier un instantané incomplet comme complet ? ».

#### Décision pédagogique commune pour le TP

Le client fictif accepte qu'on poursuive le parcours pour obtenir un rapport. Si un lien échoue, on essaie une copie. Un repli réussi n'ajoute pas d'erreur au rapport. Si le transfert du fichier échoue malgré cela, on ajoute une erreur et on n'annonce pas une sauvegarde complète. La séquence 4 précisera le contrôle de publication. Conserver votre comparaison des politiques dans le dossier.

### 5.4 Préparer les tests : 20 minutes

| Incident                        | Données et panne imposée | Résultat attendu | Effet interdit |
|---------------------------------|--------------------------|------------------|----------------|
| Source absente                  |                          |                  |                |
| Lien impossible, copie possible |                          |                  |                |
| Lien et copie impossibles       |                          |                  |                |

**Q5.3.** Pour chaque cas, dire ce qu'une doublure permet de contrôler. Quel test avec de vrais fichiers faudra-t-il conserver en complément ?

**Q5.4.** Pour un fichier, quels compteurs doivent changer après un lien réussi, un repli réussi, ou un transfert échoué ? Écrire une assertion sur le résultat et une sur un effet interdit. Ne pas compter un lien échoué puis remplacé par une copie réussie comme un fichier perdu.

### 5.5 Validation : 5 minutes

#### ✓✓ Point d'arrêt — À emporter au TP3

Trois flux alternatifs rattachés au nominal, la politique commune identifiée comme complément au dossier, les configurations de panne et les observations attendues. Consolidation du rendu G1 après le TP3.

## 6 TP3 : suivre un appel indirect et corriger un repli

75 minutes : le chantier `transfert.c`

La doublure est écrite. Vous configurez ses réponses, remplacez une fonction dans son port et corrigez la décision qui suit l'échec d'un lien. Vous n'avez pas à construire un système de fichiers en mémoire.

### 6.1 Vérifier le nominal : 10 minutes

```
make test-s1 test-s2 integration-s2
make test-s3
```

Les premiers tests doivent réussir, éventuellement à partir d'une base de reprise. Dans S3, identifier les cas en échec et les rattacher à vos flux alternatifs. Les tests peuvent s'exécuter sans droits particuliers ni second disque.

### 6.2 Décomposer l'appel : 15 minutes

Dans `src/transfert.c`, lire :

```
1 fs->lier(fs->ctx, ancien, dst)
```

**Q6.1.** Dire ce que désignent `fs`, `fs->lier`, `fs->ctx`, `ancien` et `dst`. Retrouver la signature du champ dans `src/sg.h`.

**Q6.2.** Retrouver l'affectation du champ `lier` dans `src/fs_posix.c`, puis dans `tests/doublure.c`. Quel fichier le programme appelle-t-il dans chacun des deux montages ?

**Q6.3.** Dans la doublure, `sg_simulation_t *s = ctx` retrouve l'état du test. Qui a fourni l'adresse ? Pourquoi cet objet doit-il encore exister pendant l'appel ?

### 6.3 Provoquer la panne et modifier le repli : 25 minutes

Lire le test E2 de `tests/test_seances.c`. Prédire la valeur de retour, les appels effectués et les compteurs avec le code initial.

1. Changer temporairement `echec_lien` dans le montage d'un test personnel et observer le chemin exécuté. Revenir à la configuration du test étudié.
2. Modifier `src/transfert.c` pour appliquer la décision du TD : essayer la copie après échec du lien.
3. Contrôler les trois issues : lien réussi, copie réussie, transfert échoué. Chaque transfert incrémente exactement un des compteurs `lies`, `copies`, `erreurs`.
4. Dans un test personnel, construire un port avec `sg_doublure`, puis affecter `fs.copier = sg_copie_impossible`. Appeler le transfert sans précédent. Expliquer pourquoi cette affectation est compatible avec le type du champ.

### 6.4 Vérifier résultat et effets : 15 minutes

```
make test-s3 test-perso
```

**Q6.4.** E2 peut-il réussir si le bilan compte à la fois une copie et une erreur ? Pourquoi ?

**Q6.5.** Pour E3, expliquer pourquoi l'on compte une erreur de transfert, bien que deux opérations aient échoué. Pour E5, expliquer l'intérêt de `creations == 0`.

Dans `mes_tests.c`, ajouter le transfert nominal sans précédent avec une copie réussie. Vérifier le résultat, les compteurs du rapport et l'absence d'appel au lien. Chaque montage de test part d'un état neuf.

### 6.5 Point de contrôle : 5 minutes

#### ✓✓ Point d'arrêt — La preuve du repli

`make test-s1 test-s2 test-s3 test-perso` réussit. Chacun peut nommer la fonction appelée par le pointeur et expliquer le rôle du contexte. Le dossier contient la configuration d'une panne, la trace avant correction et les assertions après correction.

### 6.6 Exercice individuel : 5 minutes

L'enseignant distribue un montage avec deux fonctions compatibles : suivre l'appel, corriger une affectation et choisir une assertion.

#### 🏠 Travail à la maison — Après S3 : 45 minutes

Rendre G1 après intégration du retour sur les trois exceptions. Relire un appel indirect et dessiner ses adresses de données et de fonctions. Dans le scénario complet, relever les mots décrivant des concepts métier et ceux décrivant une technique. Préparer une question sur l'état d'un résultat après interruption.

## 7 TD4 : du modèle de domaine aux invariants

### 75 minutes : la méthode du CM4

Trier les concepts, interroger les relations, relire un processus, puis décrire la vie d'un objet.

**Production** : modèle de quatre à six concepts, automate et deux invariants reliés à des tests.

### 7.1 Trier les candidats : 10 minutes

Relever les substantifs du scénario complet. Pour chacun : concept, attribut, fusion ou rejet. Justifier au moins un rejet technique.

**Q7.1.** Quelle différence faites-vous entre la tentative de sauvegarde, son rapport et l'instantané que l'utilisateur peut consulter ? Décider quels concepts votre périmètre doit distinguer. Un lien dur ou une structure C n'est pas automatiquement un concept du métier.

### 7.2 Relations et multiplicités : 20 minutes

Construire un modèle de quatre à six concepts. Nommer les relations et leurs multiplicités. Pour chacune, écrire une phrase qui se lit dans les deux sens. Marquer les hypothèses à confirmer.

**Q7.2.** Une source peut-elle donner lieu à plusieurs tentatives dans le temps ? Une tentative échouée produit-elle nécessairement un instantané disponible ? Vérifier que les multiplicités n'imposent pas un résultat absent après échec.

Aucun héritage ni classe-association n'est obligatoire. En proposer seulement si cela sert une règle du périmètre, puis justifier son utilité.

### 7.3 Un processus met le modèle à l'épreuve : 15 minutes

1. Un opérateur choisit une source et une destination nouvelle.
2. Le système vérifie la source et ouvre une tentative.
3. Il traite les entrées et enregistre un rapport.
4. Il examine si le résultat peut devenir disponible.
5. Il tente la publication et annonce le résultat réel.

**Q7.3.** Pour chaque étape, retrouver les informations dans votre modèle. Signaler une donnée ou une relation manquante et expliquer son ajout. La reprise, la purge et les notifications automatiques ne sont pas exigées ici.

## 7.4 L'automate et ses garanties : 20 minutes

Choisir l'objet dont vous modélisez le cycle de vie. Définir les états, les événements, les gardes et les actions. Distinguer fin du parcours et réussite de la publication.

### Complément client et périmètre de validation

Un résultat incomplet ne doit pas être présenté comme un instantané complet. Un échec de publication doit rester visible. Une tentative terminale n'est pas réécrite pour devenir une nouvelle tentative. Après interruption avant publication, le travail temporaire n'est pas une référence valide pour la sauvegarde suivante.

**Q7.4.** Écrire deux invariants en phrases, puis identifier une transition qui les violerait. Pour chacun, proposer un état initial et un événement qui doivent conduire à un refus.

**Q7.5.** Le prototype ne conserve pas son état C après l'arrêt du processus. Que faudra-t-il observer sur les répertoires pour vérifier une garantie après interruption ? Quelles informations un statut persistant apporterait-il en plus ?

## 7.5 Les tests du TP : 10 minutes

Préparer une table avec : état initial, erreurs du rapport, résultat de publication simulé, état final, retour et nombre d'appels à la publication. Couvrir succès, traitement en erreur, publication impossible et appel sur un état terminal.

### ✓✓ Point d'arrêt — À emporter au TP4

Un diagramme de domaine lisible, un automate, deux invariants et une table de tests. Vous pourrez accepter plusieurs modèles si leurs concepts, leurs règles et leurs observations sont cohérents.

## 8 TP4 : contrôler la finalisation et relire un autre travail

### 75 minutes : le chantier finalisation.c

Le prototype construit dans un répertoire suffixé `.en-cours`. La primitive fournie de publication renomme ce répertoire vers la destination définitive. Vous corrigez la décision qui autorise cet appel.

### 8.1 Lire les états et prédire : 10 minutes

Lire le type des états dans `src/sg.h`. Retrouver l'appel dans `service.c`, puis la définition de `sg_finaliser`. Associer les constantes C aux états de votre automate. La fonction est appelée après la fin du parcours.

**Q8.1.** Qu'est-ce qui change dans l'objet pointé par `etat` ? Pourquoi ce paramètre n'est-il pas `const` ? Comparer avec le rapport, qui est seulement lu.

### 8.2 Corriger la garde : 20 minutes

```
make test-s4
```

1. Relier A1 à A5 à votre table du TD. Dans la version initiale, A2 met une garantie en défaut.
2. Écrire une trace de ce cas avant de modifier le code : garde, publication appelée ou non, affectation de l'état, retour.
3. Corriger uniquement la garde manquante dans `src/finalisation.c`. Conserver le traitement de l'échec de publication et des états terminaux.
4. Rejouer les tests. Expliquer pourquoi `SG_COMPLET` ne doit être affecté qu'après une publication réussie.

### 8.3 Tester l'absence d'une publication : 15 minutes

```
make test
make integration-s4
```

Les tests unitaires imposent les échecs. Le test d'intégration utilise de vrais fichiers et un arrêt contrôlé exactement avant la publication. Le processus sort avec le code 75. Le répertoire temporaire existe, la nouvelle destination publiée n'existe pas et l'ancien instantané reste intact. Ce test ne simule pas toutes les pannes possibles.

**Q8.2.** Pourquoi la primitive fournie place-t-elle le répertoire temporaire à côté de la destination ? Quel avantage le renommage apporte-t-il par rapport au déplacement fichier par fichier ? Il faut rester sur le même système de fichiers. Cette propriété de publication ne prouve pas la conservation après panne électrique.

**Q8.3.** Quelle différence faites-vous entre le résultat du traitement des fichiers, l'état final de la tentative et les noms de répertoires observés après interruption ? Ne déduisez pas « instantané publié » de la seule absence d'erreurs de copie.

Ajouter un test personnel qui rejoue la finalisation après un premier résultat terminal et vérifie qu'aucune nouvelle publication n'a lieu. Rattacher l'assertion à un invariant.

#### 8.4 Revue croisée : 20 minutes

Deux groupes échangent leurs invariants, un jeu de tests et leurs quatre fichiers modifiés. Répartition conseillée : lecture du contrat (4 minutes), exécution et observation (8), rapport (8).

1. Choisir un invariant de l'autre groupe et prédire le résultat d'un test.
2. Exécuter une commande reproductible et noter version, données, résultat attendu et résultat obtenu.
3. Relire la fonction concernée et sa justification.
4. Rédiger une conclusion limitée à ce que vous avez vérifié, puis une amélioration argumentée : test manquant, contrat ambigu ou correction.

L'absence de défaut sur les cas essayés est un résultat recevable. N'inventez pas d'anomalie pour compléter le rapport. Évitez de conclure « tout est correct » à partir d'un seul test.

### 8.5 Traçabilité et contrôle final : 5 minutes

| Source/ID | Règle ou invariant | Fonction modifiée | Test et observation |
|-----------|--------------------|-------------------|---------------------|
|           |                    |                   |                     |

## ✓✓ Point d'arrêt — Rendu final

Les quatre fonctions sont justifiées, `make test integration-s4` réussit, le dossier relie les invariants aux tests et la revue croisée est remise. Conserver aussi les limites identifiées et les aides utilisées.

### 8.6 Exercice individuel : 5 minutes

Un extrait de transition vous est donné. Retrouver un invariant violé, déplacer ou compléter une garde et proposer un test qui distingue les versions.

👉 **Travail à la maison — Après S4 : 30 minutes**

Mettre au propre G2 sans ajouter une nouvelle fonctionnalité. Identifier la version rendue du dépôt. Vérifier que les commandes annoncées se lancent et que chaque membre sait expliquer un changement dont il n'était pas le pilote.