# Documentations tests effectués :

## Note Bien :

- Les tests sont spécifiquement conçus pour analyser et modifier des fichiers ELF 32 bits (*ELF32*) au format Little Endian (*LSB*).

- Tous les tests sont limités à ce type de fichier.

- Les autres types de fichiers ELF ne sont pas pris en charge.

## Comment lancer les tests :

1. Se placer dans le répertoire elf_linker-1.0.

2. Exécuter la commande suivante :

**./test/automated_testing.sh <numéro de l'étape>**

Où <numéro de l'étape> est un chiffre entre {1…9}.

**Résultats possibles après exécution :**

- Si des fichiers .o nécessaires aux tests sont absents, un message indique :

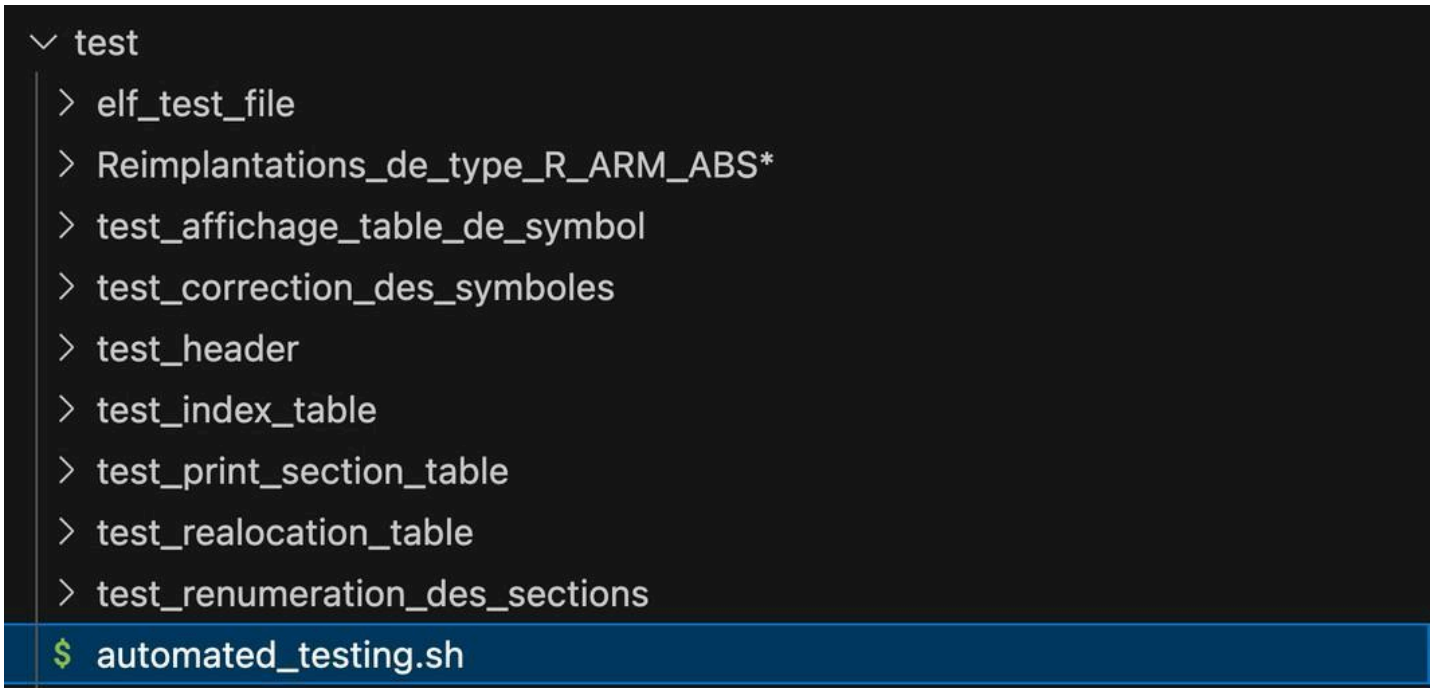
*"Des fichiers .o pour les tests n'existent pas dans le répertoire. Tapez make pour les générer."*

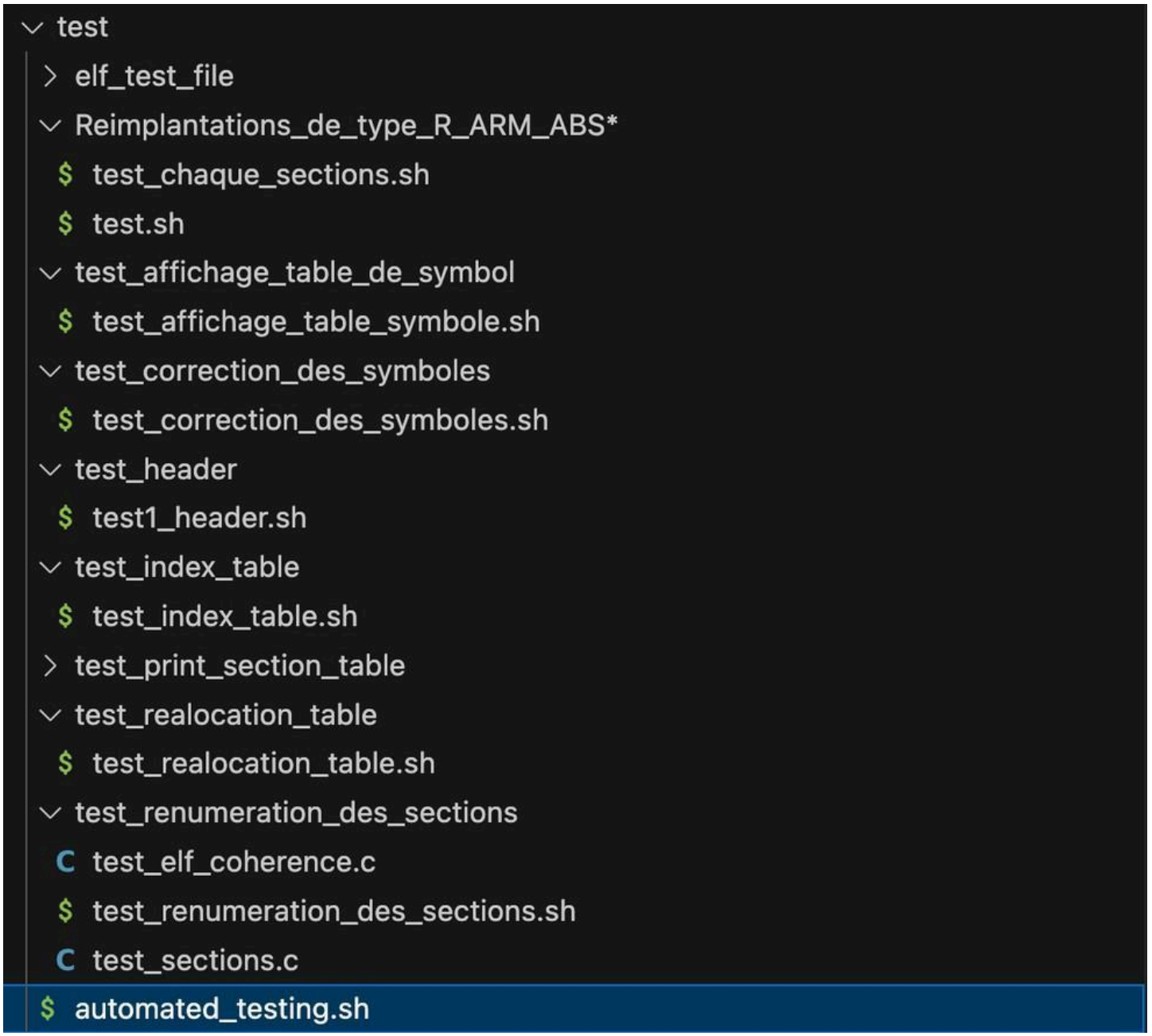
- Sinon, les tests sont lancés automatiquement.

## Documentation :

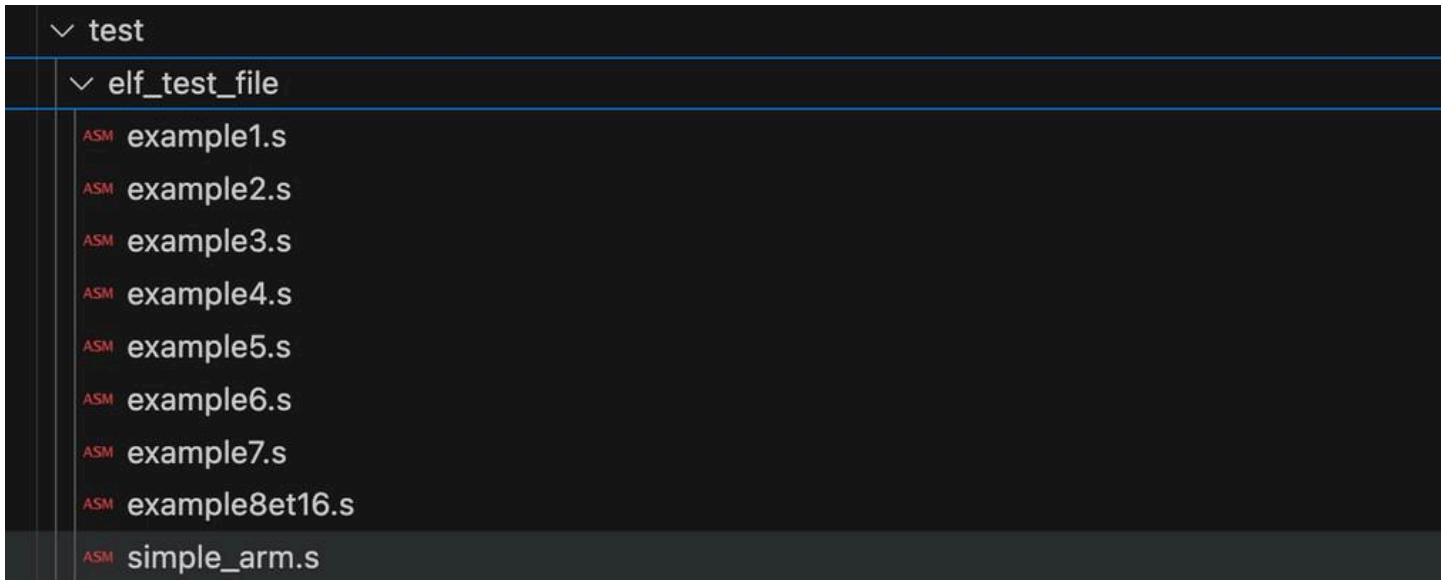
## Architecture :

Dans le répertoire ***test***, on trouve un sous-répertoire pour chaque partie, contenant les fichiers nécessaires à l'exécution des tests, ainsi qu'un fichier bash ***automated_testing.sh*** pour automatiser les tests.


test
elf_test_file
Reimplantations_de_type_R_ARM_ABS*
test_affichage_table_de_symbol
test_correction_des_symboles
test_header
test_index_table
test_print_section_table
test_realocation_table
test_renumeration_des_sections
automated_testing.sh




Un répertoire ***elf_test_file*** contenant les fichiers ARM nécessaires pour les tests.



# Liste et description des tests effectués :

# Phase 1 :

# Description :

Les tests de la phase 1 ont pour objectif de vérifier la conformité des résultats générés par notre **programme** et s'assurer que c'est cohérent avec celle de arm-none-eabi-readelf l'outils standard pour analyser des fichiers **ELF**.

**NB :** On ne vérifie pas que les sorties de notre programme et arm-none-eabi-readelf soit identique c'est pas le but dans le cadre de notre projet.

- **Test pour l'affichage de l'en-tête ELF**

  **test_header.sh : Objectif :**

  Vérifier si les en-têtes ELF d'un fichier binaire donné, générés par notre programme, correspondent aux informations fournies par la commande standard arm-none-eabi-readelf-h.

  • Le script prend un fichier ELF en entrée.

  • Exécute les deux commandes (./Program h et arm-none-eabi-readelf -h) et vérifie si les lignes de la sortie de notre programme se trouve dans la sortie de arm-none-eabi-readelf -h

  • Si une différence est détectée, les détails des divergences sont affichés, sinon le script confirme que les en-têtes sont identiques.

- **Test pour l'affichage de la table des sections et des détails relatifs à chaque section : Objectif :**

  Comparer ligne par ligne les tables de sections d'un fichier ELF, générées par notre programme, avec celles produites par la commande standard arm-none-eabi  -S.

  • Le script prend un fichier ELF en entrée.

  • Exécute les deux commandes (arm-none-eabi-readelf -S et ./Program S) et vérifie si les lignes de la sortie de notre programme se trouve dans la sortie de arm-none-eabi-readelf -S

- Identifie et affiche les différences ligne par ligne, avec les détails des divergences entre la sortie de arm-none-eabi-readelf et celle du programme.
- Si aucune différence n'est trouvée, le script indique que les tables de sections sont identiques.
- En cas de divergence, le script affiche un message signalant les différences.

- **Test pour l'affichage du contenu d'une section :**

  **Objectif :**

  Vérifier si les tables de sections d'un fichier ELF, générées par notre programme, correspondent aux informations fournies par la commande standard arm-none-eabi-readelf -x.

  - Le script prend un fichier ELF en entrée.
  - Vérifie l'existence du fichier et extrait le nombre total de sections via arm-none-eabi-readelf -h.
  - Compare les sections de deux manières :
  - Par index : chaque index de section est comparé entre les sorties de ./Program x et arm-none-eabi-readelf -x.
  - Par nom : les noms des sections sont extraits via arm-none-eabi-readelf -WS, et leurs contenus respectifs sont comparés.
  - Les différences, s'il y en a, sont affichées pour chaque section avec les détails correspondants (contenu généré par le programme et par arm-none-eabi-readelf).
  - Si aucune différence n'est détectée, le script confirme que les tables de sections sont identiques.

- **Test pour l'affichage de la table des symboles et des détails relatifs à chaque symbole :**

  **Objectif :**

  Comparer la table des symboles d'un fichier ELF, générée par notre programme, avec celle produite par la commande standard arm-none-eabi-readelf -s.

  - Le script prend un fichier ELF en entrée.

- Exécute les deux commandes (./Program s et arm-none-eabi-readelf -s) pour récupérer les tables des symboles respectives.

- Compare les deux sorties

- En cas de différence, le script affiche la ligne concernée pour chaque outil (programme personnalisé et arm-none-eabi-readelf).

- Si aucune différence n'est détectée, le script indique que les résultats sont identiques avec un message en vert.

- Si des divergences existent, un message en rouge signale les différences.

- **Test pour l'affichage des tables de réimplantation et des détails relatifs à chaque entrée** :

  **Objectif :**

  Comparer la table des relocations d'un fichier ELF, générée par notre programme, avec celle produite par la commande standard arm-none-eabi-readelf -r.

  - Le script prend un fichier ELF en entrée.

  - Exécute les deux commandes (./Program r et arm-none-eabi-readelf -r) pour récupérer les tables des relocations respectives.

  - Compare les deux sorties :

  - Si une différence est détectée, les détails de l'erreur sont affichés, incluant les valeurs produites par le programme et arm-none-eabi-readelf.

  - Si les tables des relocations sont identiques, un message en vert confirme la réussite du test.

  - En cas de divergence, le test échoue avec un message d'erreur et les valeurs en conflit.

# Phase 2 :

- **Test pour la renumérotation des sections :**

  **Objectif :**

  Vérifier la cohérence des en-têtes et des sections d'un fichier ELF après avoir effectué une opération sur ce fichier, en comparant les résultats avant et après l'opération.

- Le script prend un fichier ELF en entrée et un fichier de sortie pour stocker les résultats après transformation.
- Vérifie l'existence du fichier ELF et récupère le nombre d'en-têtes de sections avant l'opération (avant).
- Exécute un programme secondaire (./Program2) pour modifier le fichier ELF et le sauvegarder dans un nouveau fichier de sortie.
- Compare le nombre d'en-têtes de sections du fichier ELF avant et après l'opération.
- Vérifie si le nombre de sections supplémentaires créées (avec Program S) est cohérent avec la modification effectuée.
- Utilise un script de cohérence (test_elf_coherence) pour s'assurer qu'aucune assertion n'a échoué pendant le processus.
- Si toutes les vérifications sont réussies, le test est passé avec succès. Sinon, le test échoue.

- **Test pour la correction des symboles :**

  **Objectif :**

  Vérifier si les symboles d'un fichier ELF ont été correctement modifiés en comparant les valeurs avant et après modification.

  - Le script prend un fichier ELF en entrée et effectue une modification à l'aide d'un programme secondaire (Program2), qui génère un fichier ELF modifié.
  - Il utilise la commande arm-none-eabi-readelf pour récupérer la table des symboles avant et après modification.
  - Les symboles sont comparés ligne par ligne en extrayant les indices de section et les valeurs associées.
  - Pour chaque ligne, le script vérifie que les noms de section sont identiques entre les deux versions du fichier ELF.
  - Il convertit les valeurs hexadécimales en décimales et vérifie si la différence entre les valeurs correspond à l'offset de la section.
  - Si une erreur est détectée dans les valeurs, le script affiche un message d'erreur avec les détails. Si tout est correct, un message de succès est affiché.

- **Test pour réimplantations de type R ARM ABS* :**

  **Objectif :**

  Vérifier que les sections et les addends d'un fichier ELF sont correctement traités et que les valeurs extraites correspondent aux attentes.

  • Le script analyse un fichier ELF en lisant ses sections, puis effectue une série de vérifications.

  • Il extrait chaque section et la sauvegarde dans des fichiers temporaires pour un traitement ultérieur.

  • Chaque section est analysée pour rechercher des valeurs spécifiques, comme les symboles, les offsets et les addends.

  • Pour chaque valeur trouvée, le script la compare avec les valeurs attendues et s'assure que les addends sont corrects.

  • Il utilise des fonctions pour convertir les valeurs hexadécimales en décimales, calculer des sommes et effectuer des recherches dans les sections extraites.

  • Si les valeurs ne correspondent pas, le script affiche un message d'erreur ; sinon, il signale un succès.

- **Test pour la réimplantation de type R ARM JUMP24 et R ARM CALL :**

  **Objectif** :

  Vérifier que les valeurs extraites d'un fichier, en particulier les offsets et addends dans un dump hexadécimal, sont correctement traitées et correspondent aux attentes.

  • **Préparation des parties du fichier** :

  Le script commence par extraire des sections spécifiques du fichier (partie1, partie2, partie3) afin de se concentrer sur des segments de données précises.

  • **Extraction des valeurs et des calculs** :

  Il extrait l'offset, la valeur du symbole, et l'addend de la deuxième partie du fichier. Ces valeurs sont ensuite converties en formats appropriés (hexadécimal et décimal).

  • **Recherche de la valeur** :

  Le script recherche une valeur cible (calculée à partir des différences entre le symbole et l'offset) dans une autre section du fichier. Il effectue cette recherche à une position précise

et extrait la valeur hexadécimale.

- **Comparaison des valeurs** :

Après avoir extrait la valeur à la position donnée, elle est comparée à la valeur cible. Si elles correspondent, un message de succès est affiché, sinon une erreur est générée avec des détails sur les valeurs attendues et extraites.

- **Test de toutes les sections** :

Le script continue en analysant chaque section du fichier, en exécutant les tests et en signalant si le test réussit ou échoue pour chaque section. Si une erreur est détectée, le script arrête l'exécution.

# Automatisation :

# automated_testing.sh :

. **Gestion des séquences de tests** : Il exécute une série de tests, spécifiés soit par un argument donné au script, soit par défaut pour toutes les séquences de tests.

. **Assemblage des fichiers** : Pour chaque fichier .s dans le répertoire elf_test_file, le script les assemble en fichiers .o avec arm-none-eabi-gcc.

. **Exécution des tests** : Le script parcourt les fichiers .o et exécute différents tests pour chaque fichier selon les séquences spécifiées.

- Test d'interfaçage avec le simulateur ARM et analyse des cycles

Les deux programmes en ARM sont conçus pour tester différentes opérations sur des symboles de tailles variées (32, 16 et 8 bits) et pour vérifier le comportement du code en fonction des instructions ARM spécifiques, telles que R_ARM_ABS32, R_ARM_ABS16, et R_ARM_ABS8. Le premier programme effectue des opérations de base, telles que l'addition et la comparaison, avant d'appeler une fonction qui double la valeur d'un registre. Il utilise également des branchements conditionnels et stocke des valeurs dans des variables de 8 bits pour tester les manipulations de mémoire. Le deuxième programme génère des instructions spécifiques pour tester les types de relocalisation et leur impact sur les symboles définis dans la section .data. Les tests sont exécutés en utilisant un simulateur ARM, avec un script comme programme_run.sh pour analyser les résultats et observer les cycles d'exécution du programme. Cela permet de vérifier que les manipulations des symboles et les calculs respectent les attentes en termes de performance et de gestion des adresses mémoire.

- La Vérification de la création d'un fichier ELF non relogeable avec gestion des segments mémoire

Les deux scripts analysent les segments LOAD des fichiers ELF à l’aide de la commande arm-none-eabi-readelf et vérifient qu’il n’y a pas de chevauchement entre les segments mémoire. Enfin, il affiche le résultat des tests, indiquant si les fichiers ELF respectent les contraintes ou s’il y a des erreurs liées aux chevauchements.