TD3 — Compilation séparée, Makefile

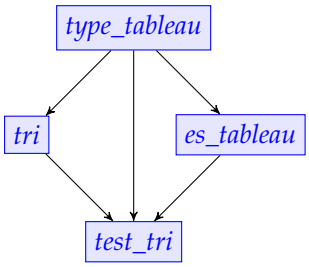
Éléments de cours et de correction

On donne ci-dessous le fichier `Makefile` du TP2 :

```
CC=clang -Wall -g

all: test_tri

test_tri.o: test_tri.c tri.h es_tableau.h type_tableau.h
	$(CC) -c test_tri.c

es_tableau.o: es_tableau.c es_tableau.h type_tableau.h
	$(CC) -c es_tableau.c

type_tableau.o: type_tableau.c type_tableau.h
	$(CC) -c type_tableau.c

tri.o: tri.c tri.h type_tableau.h
	$(CC) -c tri.c

test_tri: test_tri.o es_tableau.o type_tableau.o tri.o
	$(CC) test_tri.o es_tableau.o type_tableau.o tri.o -o test_tri

clean:
	rm -f test_tri *.o
```

Exercice 1. Rappeler la structure du programme.

La structure est donnée par les dépendances :



Exercice 2. Donner la séquence de commandes exécutées :

1. par `make tri.o`

```
clang -c tri.c
```

2. par `make`

```
clang -c test_tri.c
clang -c es_tableau.c
clang -c type_tableau.c
clang -c tri.c
clang test_tri.o es_tableau.o type_tableau.o tri.o -o test_tri
```

3. par `make` après modification du fichier `tri.c`

```
clang -c tri.c
clang test_tri.o es_tableau.o type_tableau.o tri.o -o test_tri
```

4. par `make` après modification du fichier `tri.h`

```
clang -c test_tri.c
clang -c tri.c
clang test_tri.o es_tableau.o type_tableau.o tri.o -o test_tri
```

1. Motifs, variables internes, règles génériques

On peut remarquer que dans un programme composé de plusieurs modules écrits dans le même langage de programmation, les règles (ou au moins la partie «commande» de ces règles) sont assez répétitives.

Il est possible d'utiliser le motif (ou «pattern») `%` pour écrire des règles génériques *:*

```
%.o : %.c
	<commande utilisant les variables internes $<, $@, ...>
```

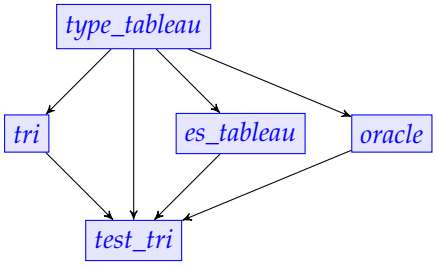
Exercice 3. Modifier le `Makefile`, en utilisant des motifs (`%`) et variables internes (`$<`, **`$@`**, `$^`), pour avoir une seule règle de compilation d'un fichier C.

```
CC=clang -Wall -g

all: test_tri

%.o : %.c
	$(CC) -c $<

# Dépendances àconserver
test_tri.o: test_tri.c tri.h es_tableau.h type_tableau.h

es_tableau.o: es_tableau.c es_tableau.h type_tableau.h

type_tableau.o: type_tableau.c type_tableau.h

tri.o: tri.c tri.h type_tableau.h

# Édition de lien
test_tri: test_tri.o es_tableau.o type_tableau.o tri.o
	$(CC) $^ -o $@

clean:
	rm -f test_tri *.o
```

Exercice 4. On souhaite ajouter un module `oracle`, contenant les fonctions permettant de tester la fonction de tri. Décrire les fichiers ajoutés (sans détailler l'implémentation des fonctions), et les modifications à apporter aux fichiers existants.

Le module `oracle` contient une fonction `oracle_tri`, et éventuellement d'autres fonctions intermédiaires (fonction de vérification de l'ordre des éléments d'un tableau par exemple). Seule la fonction `oracle_tri` est exportée *: déclarée dans le fichier `oracle.h`, c'est la seule fonction du module appelée depuis l'extérieur du module (depuis le module principal `test_tri`). Cette fonction prend en paramètre deux tableaux et renvoie un booléen : le module `oracle` dépend du module `type_tableau`.*

Structure du programme modifié :



Contenu du fichier `oracle.h` :

```
#ifndef _ORACLE_H_
#define _ORACLE_H_

#include "type_tableau.h"

int oracle_tri(tableau_entiers *t_trie, tableau_entiers *t_init);

#endif
```

Modifications à apporter :

- *au début du fichier* `test_tri.c` *: ajout de* `\#include "oracle.h"`
- *dans le* `Makefile` *:*
  - *ajout de la dépendance :*

    ```
    oracle.o: oracle.c oracle.h type_tableau.h
    ```

  - *modification de la règle pour* `test_tri.o` *:*

    ```
    test_tri.o: test_tri.c tri.h es_tableau.h type_tableau.h oracle.h
    $(CC) -c test_tri.c
    ```

  - *modification de la règle pour l'édition de liens :*

    ```
    test_tri: test_tri.o es_tableau.o type_tableau.o tri.o oracle.o
    $(CC) $^ -o test_tri
    ```