# Résumé

## Le compilateur

### STRUCTURE DE FICHIERS

Le groupe compilation a pour objectif de créer un ensemble de fichiers bison et lex. Ces fichiers sont alors tries dans un ensemble de structure de fichiers.

Cette structure de fichiers doit être stables, car les différents fichiers qui vont être rangés ainsi que le makefile vont se reposer sur cette structure de fichiers

Cette structure de fichiers doit donc rendre compte de plusieurs critères :

- Séparer les fichiers lex, les fichiers bisons, les exécutables et les fichiers intermédiaires
- Permettre au makefile de compiler tous nouveaux fichiers rapidement (automatique ou rajout d'une ligne dans le makefile

### MAKEFILE

Le makefile permet en une simple ligne de commande de compiler les fichiers, de composer les étapes intermédiaires si nécessaires et de créer l'exécutable à la fin.

Le makefile doit aussi être un des premiers paliers d'interface avec l'utilisateur, et doit alors être relativement exhaustif dans :

- La gestion des erreurs (remontée des erreurs défini par le groupe compilation)
- La gestion de l'aide sur les options (un --help qui soit complet)
- Les différents types de compilation -> compilation de débug, de release, d'optimisation …

### LA COMPILATION

Le lancement de la compilation se passe avec le makefile. Pour cela, deux pistes sont possibles :

- Une interface qui simplifie (et augmente) les possibilités du compilateur c++. Il se passe en ligne de commande et permet d'utiliser les différentes possibilités comme la redirection de la sortie, l'ajout de bibliothèque
- Une interface visuelle, qui permet graphiquement de se représenter les options possibles du compilateur.

Il est alors généré un ensemble de fichier C++ valides. Ces fichiers peuvent alors eux-aussi être compilés (une autre interface ? Lancer le compilateur normalement ?) pour donner un exécutable.

# Options en ligne de commande

## COMMENT GERER LES OPTIONS

L'objectif des options de compilation est de pouvoir aiguiller le compilateur sur les différentes actions qu'il exécute. Ces options se gèrent normalement directement par le makefile, ou directement ou en passant par un script de traitement externe.

Les seules options qui pourront avoir une communication avec l'intérieur du code sont :

- L'action de certaines options de code (peut-on ajouter du code C++ dans le code)
- La gestion des arguments en ligne de commande
- Les options de verboses, qui activent certaines parties qui affichent alors plus d'informations
- Les bibliothèques incluent dans le code qui doivent être reliées à une option de compilation

## QUELLES SONT LES OPTIONS DE BASES UTILISEES PAR LE COMPILATEUR

(Voir avec Alexandre)

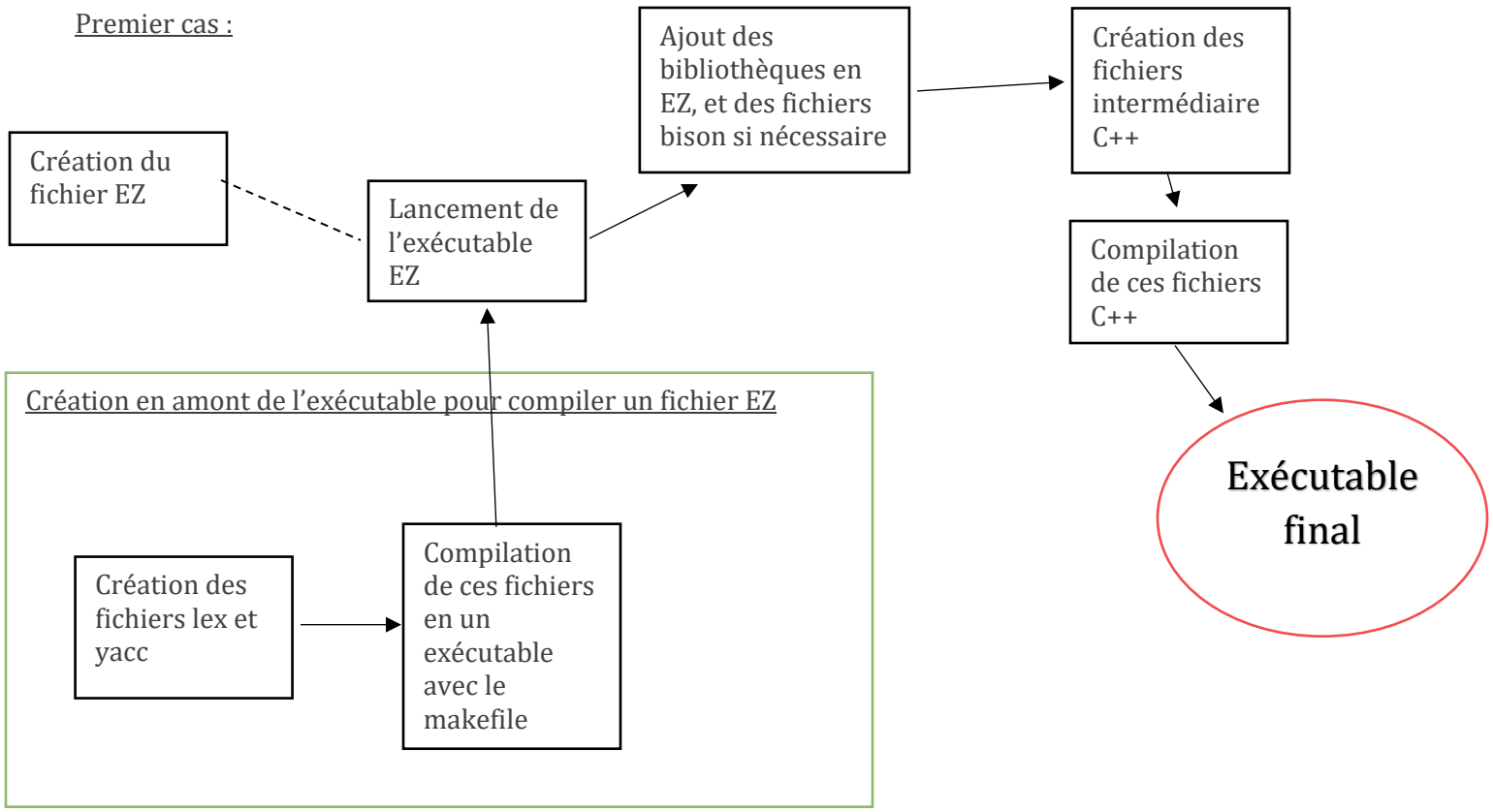
## Organisation des bibliothèques du langage

### LES BIBLIOTHEQUES

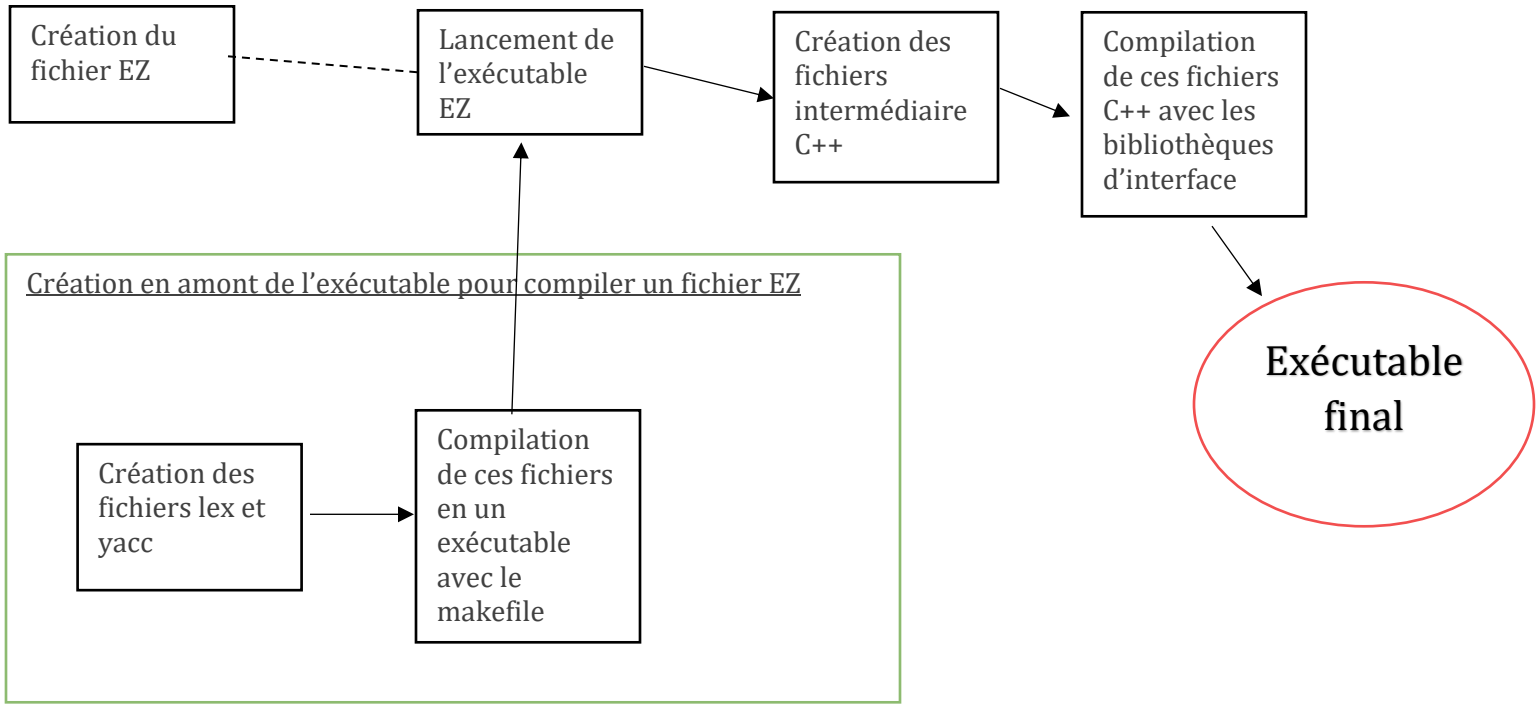
Les différentes possibilités du langage (tel que les fonctions de tri sur les vecteurs) peuvent se retrouver lors de la compilation de plusieurs manières :

- Création de toutes les fonctions en EZ langage, qui ensuite seront traduits en C++ avec le reste du code. Cela permet de garder une homogénéité avec le code de l'utilisateur. Cette façon de faire amène alors deux cas :
    - Premier cas : tout ce qui est utilisé dans ce fichier externe EZ a été prévu par le compilateur. Il est alors compilé de la même façon qu'un fichier utilisateur
    - Deuxième cas : des parties de codes ne peuvent être interprétées par le compilateur, on ajoute alors fichier Bison spécifique qui permet au compilateur de vérifier ce code inhabituel.
- Création de toutes les fonctions en C++. Les différentes fonctions utilisées seront alors vérifiées uniquement puis transformées en C++. Le code se basera ensuite sur le code C++ pour faire fonctionner ces fonctions.

Ces deux possibilités sont résumées dans le schéma qui suit :



Deuxième cas :




**INCLUSION DES BILBIOTHEQUES**

Les bibliothèques doivent être rangées dans un dossier particulier (« /include/bibliotheque/ »). Ensuite, ces bibliothèques doivent être implémentées dans le code, pour ceci trois possibilités :

- Intégration complète des bibliothèques :
  Lors de la compilation, c'est lourd mais cela permet d'éviter une gestion complète des bibliothèques. Cela peut-être une solution temporaire avant de passer à une autre méthode
- Intégration par les inclusions :
  Durant le code, une inclusion d'une bibliothèque pourra être faite par l'utilisateur et ainsi les bibliothèques seront listées dans le Bison afin de déterminer quelle bibliothèque intégrée quand.
- Intégration par un système de FLAG :
  Lors de la découverte de certains mots clés (« vector », « tree », …), le Bison les reconnait, puis active un FLAG. Ce FLAG active ensuite l'inclusion des différentes bibliothèques. Cette méthode demande à être réfléchi sur la forme (premier passage pour déterminer ce qu'il faut inclure ou une inclusion au fur et à mesure de l'interprétation du code, comment ajouter simplement une autre librairie et donc avoir un code qui soit le moins possible entrelacé entre le Bison et les bibliothèques)

## Installation et déploiement

Sur les différents schémas, on rend compte de deux compilations différentes :

- Celle d'EZ vers C++ avec notre exécutable
- Celle de C++ vers Exécutable avec G++

Ces deux compilations sont séparées mais s'active en série avec le lancement de l'exécutable EZ. Les deux peuvent donc faire partie du même makefile.

Dans le déploiement, trois étapes sont à définir :

- Définir le lieu qui va contenir les libraires (en EZ ou C++ en fonction du choix des bibliothèques)
- Mettre une variable externe remontant à ce dossier
- Le programme peut alors utiliser les différents fichiers présents sur la machine, sans avoir à être dans un lieu particulier.