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 stable, 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 les nouveaux fichiers rapidement (automatique ou rajout d'une ligne dans le makefile

Les fichiers se structurent de la façon suivante :

(Voir avec Barry,Driss,Florian,Valerian)

MAKEFILE

Le makefile permet en une simple ligne de commande de compiler les fichiers, de composer les étapes intermédiaires si nécessaire 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 :

- Une interface qui simplifie (et augmente) les possibilités du compilateur c++. Il s'utilise en ligne de commande et permet d'utiliser les différentes possibilités comme la redirection de la sortie et l'ajout de bibliothèque

  Il est alors généré un ensemble de fichiers C++ valides. Ces fichiers peuvent alors eux-aussi être compilés en fonction des options choisies par l'utilisateur pour donner un exécutable final.

GESTION DES INCLUSIONS

On considère le code écrit par l'utilisateur comme tournant autour d'un fichier principal dont les caractéristiques ne sont pas encore définies.
Les inclusions sont alors possibles dans trois cas différents :

- On remplace une ligne d'inclusion dans le fichier EZ qui désigne le fichier à inclure
- On détermine les fichiers à inclure dans les options de compilation. Pour cela, deux choix :
    - On détermine une option pour inclure tous les fichiers EZ contenus dans le dossier du fichier principal
    - On ajoute les fichiers par une option de compilation qui désigne à chaque fois un fichier différent.

Afin de faciliter le travail de COMPILATION, on peut alors créer un fichier temporaire qui contient tout le code EZ, permettant de travailler la compilation que sur un seul fichier.

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

Quelles sont les options de bases utilisées par le compilateur

(Voir avec Alexandre)

Organisation des bibliothèques du langage

LES BIBLIOTHEQUES

Ces fichiers seront des fichiers avec une syntaxe particulière qui permet de définir les entêtes de fonctions. Seront ensuite définis dans les fichiers bisons les traitements à effectuer pour transformer ces appels de fonctions EZ en appels de fonctions C++.

Les bibliothèques sont créées en amont en deux parties :

- Une partie HEADER avec les prototypes des fonctions et les informations permettant de vérifier la bonne utilisation de la fonction en EZ. Ces prototypes sont définis par le groupe LANGAGE et ont leur propre écriture.
- Une partie TRADUCTION contenue dans les fichiers Bison, qui traduisent les appels correspondant aux Headers en C++.

Exemple simpliste pour l'idée générale :

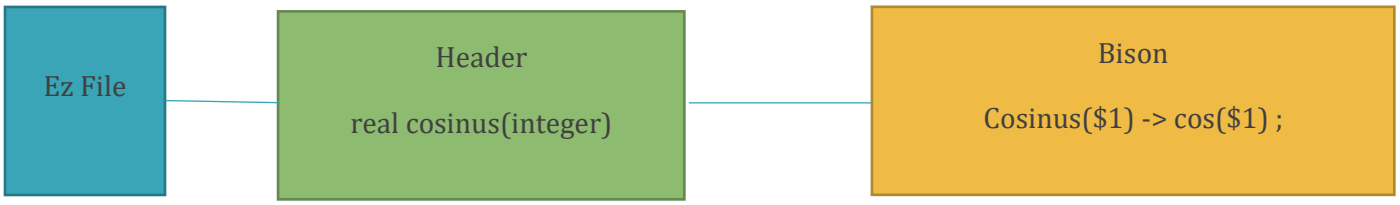
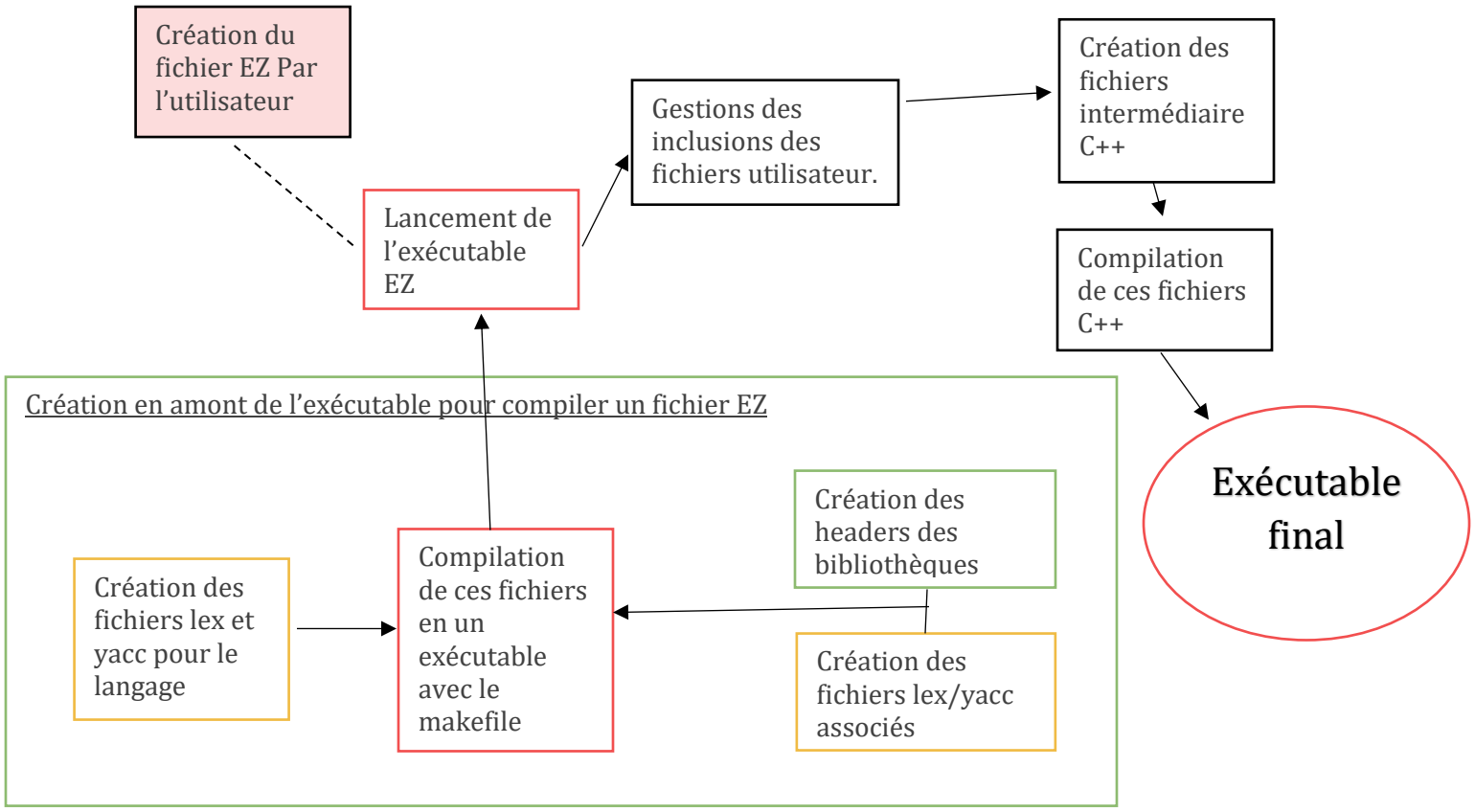


Schéma Générale de la compilation :



INCLUSION DES BILBIOTHEQUES

Les bibliothèques doivent être rangées dans un dossier particulier car elles sont différentes des fichiers EZ purs. Pour cela, elles seront rangées dans un dossier « EZ » à l'intérieur de « /usr/lib/ ». Ensuite, ces bibliothèques doivent être implémentées dans le code :

- 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

Convention de nommage :

L'extension peut-être :

- « .a » : extension pour les librairies statiques à recompiler à chaque fois en C. Elle parait correspondre à notre utilisation et serait donc cohérente avec le reste des libraires
- « .lib_ez » : extension moins conventionnelle mais plus compréhensible. Elle peut être aussi légèrement modifié, comme par exemple « .lez » ou « .ezb »

Le nom de la bibliothèque doit commencer et correspondre à :

- Commencer par « lib »
- Contenir ensuite un ensemble court décrivant brièvement la librairie (ex : « libmath.a »)
- Finir par l'extension choisie au-dessus

Gestion des versions

Les bibliothèques peuvent être rangées de deux manières :

- Par versions de paquets (ex : g++ 5.4 et g++ 6.0 ont des paquets différents). Les paquets ont donc une version, ce qui implique une plus grande stabilité, mais permet moins de mise à jour
- Par versions de librairies, qui a donc les avantages et inconvénients inverses.

Une version plus intermédiaire pourrait correspondre à nos besoins (gestions par paquets mais avec des versions temporaires, comme si on utilisait un compilateur g++ en version 5.4.0.1 qui n'est pas stable et que d'autres personnes travaillent sur la 5.4.0.2 et qu'on merge en 5.4.1)

Installation et déploiement

DEPLOIEMENT

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 compilations sont alors exécutées par l'exécutable EZ.

Toutes les étapes intermédiaires sont modulables par les options de compilation.

Pour le déploiement sur machine, il se sépare en deux parties :

- Déclaration de « EZ » comme une exécutable valide par le système dans le terminal
- Copie des librairies dans les dossiers qui ont été définis

COLORATION SYNTAXIQUE

On peut inclure une coloration syntaxique sur un éditeur de texte. Le plus courant et dont la simplicité de des colorations syntaxiques. Une coloration simple va donc être mise en place pour les deux types de fichiers.

Les codes couleurs sont encore à définir, sûrement en accord avec le groupe LANGAGE.