Avancement Étape C:

Travail fait:

-Nous avons lu la partie GenCode du poly.
-ecrire de quelque partie de code qui sont responsables sur la génération du code en assembleur pour:
les déclarations de variable dans le programme principal.
-les instructions basiques comme Plus, Assign … .
-Nous avons eu besoin de quelques outils pour générer un code assembleur optimisé et pour gérer les registres de façon plus efficace.

Travail à faire:

-compléter toute l'étape c pour deca sans objet:
-Génération du code pour les structures de contrôle.
-Génération du code pour toutes les expressions.
-gérer le cas où on atteint le nombre de registre maximal ‘’15”. Pop and Push.

⇒Donc on aura besoin de bien gérer les labels et les registres ce qui nous ramene à bien enrechir notre classe **outlis.java** dans **fr.ensimag.deca.codegen,** contient pour l’instant une gestion simple des registres par exemple et elle désalloue les registre non utilisés.

*rem: il nous faut savoir plus sur les Label aussi vu que pour l’instant on a pas essayé de générer du code assembleur en les utilisant.

Résultat jusqu'à maintenant:

Pour un fichier_test.deca

```
{
        int a=2;
        float b=3.14;
        int c=5;
        println(“hello world”);
}
```

on a la génération du code comme suit: fichier test.ass:

```
1  ; start main program
2  ; Main program
3  ; Beginning of main decalarations:
4          LOAD #2, R2
5          STORE R2, 3(GB)
6          LOAD #0x1.91eb86p1, R2
7          STORE R2, 4(GB)
8          LOAD #5, R2
9          STORE R2, 5(GB)
10 ; Beginning of main instructions:
11         WSTR """hello world"""
12         WNL
13         HALT
14 ; end main program
```