Université Gustave Eiffel

# Travaux Dirigés de Compilation n°6

## Licence d'informatique

———

---

## Assembleur. Arbres abstraits

Les objectifs de ce TD sont de pratiquer l'assembleur et de construire des arbres abstraits en Bison.

---

### ▸ Exercice 1. *Lire et écrire en assembleur*

*Écrivez un programme en assembleur qui lit un caractère au clavier, le sauvegarde en mémoire, lui ajoute 1 et affiche le caractère obtenu suivi d'un caractère de fin de ligne. Par exemple, si l'utilisateur tape* `V`*, le programme doit répondre* `W`*. Pour lire et écrire, utilisez l'instruction* `syscall`*, avec :*

- *dans* `rax` *le code 0 pour lire ou 1 pour écrire,*
- *dans* `rdi` *le code 0 pour* `stdin` *ou 1 pour* `stdout`*,*
- *dans* `rsi` *l'adresse de la chaine de caractères en mémoire,*
- *et dans* `rdx` *la taille de la chaine de caractères en octets.*

### ▸ Exercice 2. *Utiliser de l'assembleur dans du code C*

1. *Écrivez en assembleur une fonction* ***mult2*** *qui renvoie le double d'un nombre.*
2. *Écrivez un programme en C qui utilise la fonction* ***mult2()*** *sans la définir. Pour éviter les avertissements, déclarez le prototype de la fonction.*
3. *Faites un makefile qui compile séparément les deux sources pour obtenir deux fichiers objet* ***.o****, puis qui lie les deux. Testez.*

### ▸ Exercice 3. *Construire un arbre abstrait*

*Ajoutez à votre projet d'analyse syntaxique des actions* `bison` *et éventuellement* `flex` *qui construisent un arbre abstrait pour chaque non-terminal* `F` *qui ne contient aucun* `Exp` *ni aucun* `Arguments`*. Vous pouvez partir de l'ébauche de module* `abstract-tree.c` *pour manipuler les arbres abstraits. Pour vérifier le résultat, ajoutez des actions qui affichent les arbres abstraits.*

### ▸ Exercice 4. (Crible d'Ératosthène)

*Le crible d'Ératosthène est une méthode rapide pour déterminer quels sont les nombres premiers plus petits qu'un entier positif $n$. Il consiste à initialiser un tableau tab de $n$ cases*

*en mettant toutes ses valeurs à 1, sauf les cases 0 et 1 qui sont mises à 0. Ensuite, pour chaque case $i \geq 2$, si $tab[i]$ vaut 0, on ne fait rien, et s'il vaut 1, on met à 0 les cases $k \times i$ avec $k > 1$ et $k \times i < n$.*

1. *Écrivez un programme en C qui implémente le crible d'Ératosthène avec les caractéristiques suivantes :*
   - *Le programme demande à l'utilisateur le $n$ souhaité et affiche à la fin du calcul le nombre et la liste des nombres premiers trouvés.*
   - *Le code comportera trois fonctions :* `main()` *qui effectue les entrées-sorties et réserve la mémoire, une fonction* `init_crible_c()` *qui initialise avec des 1 un tableau dont l'adresse et la taille sont transmis en argument, et enfin une fonction* `crible_c()` *qui déroule l'algorithme du crible comme décrit ci-dessus.*
   - *Le tableau servant au crible utilisera un octet par case.*
   - *Mesurez la performance du code obtenu en déterminant la valeur maximale de $n$ qui peut être atteinte en 1 s, 5 s, 10 s et 1 mn. Pour mesurer les performances, utilisez les fonctions* `clock` *et* `CLOCKS_PER_SEC` *de la bibliothèque* `time.c`*. La fonction* `clock` *vous donne le nombre de tics écoulés depuis une valeur de base (qui ne varie pas), et* `CLOCKS_PER_SEC` *est le nombre de tics en une seconde.*
2. *Codez en assembleur une fonction* `init_crible_a` *équivalente à* `init_crible_c()`*, avec les mêmes arguments. De même pour* `crible_a`*. Mesurez la performance du code obtenu comme pour la version 1.*
3. *On peut arrêter le crible dès que $i^2 > n$, parce que le tableau ne change plus.*
4. *Modifiez les fonctions* `main()`*,* `init_crible_a` *et* `crible_a` *afin que le tableau consacre non plus 1 octet mais 1 bit à chaque nombre. On arrondira le $n$ transmis par l'utilisateur au multiple de 8 supérieur ou égal. Mesurez la performance du code obtenu.*