TP n°2 : Compilation et édition des liens

Exercice 1 : Phases de compilation

```
/* Programme phases de compilation */
#include <stdio.h>
#include <stdlib.h>
#define MAX 12
int ab = 3;
const int cn = 5;
extern int vex;
int main()
{
int xyz; /* declaration de xyz */
xyz = MAX;
printf("Bonjour : %d\n", ab + cn + vex + xyz);
exit(EXIT_SUCCESS); }
```

1. Affichez le résultat de la phase de préprocesseur avec : *gcc -E exo1.c*
2. Redirigez le résultat dans un fichier avec : *cpp exo1.c -o exo1.i*
3. Visualisez le contenu du fichier avec *: cat exo1.i*, et expliquez sont résultat

Remarque : -E effectue la phase de préprocesseur (cpp) sur le fichier source .c spécifié

4. Générez le code assembleur avec *: gcc -S exo1.c* ou *gcc -S exo1.i -o exo1.s*
5. Dans le code assembleur, identifiez les sections *.data, .rodata* et *.text*
6. Quels symboles sont associés à la section *.data* ? À quoi correspond cette section ?
7. Quels symboles et valeurs sont associés à la section *.rodata* ? À quoi correspond cette section ?
8. Quels sont les identificateurs à exporter (internes ou connus) et à importer (externes ou inconnus) ?
9. Générez le fichier objet *exo1.o* à partir du fichier du code assembleur *exo1.s* avec : *gcc -c exo1.s -o exo1.o*
10. Générez le fichier objet *exo1.o* à partir du fichier source avec : *gcc exo1.c -o exo1.o* (cela donnera une erreur, expliquez pourquoi ?)
11. Générez le fichier exécutable avec : *gcc exo1.c -o exo1*, que remarquez-vous ? Ajoutez un fichier nommé *myheader.h* qui contient la déclaration de la variable *vex* et regénérez l'exécutable
12. Visualisez les symboles avec : *nm exo1* expliquez ses résultats
13. Où ces symboles sont-ils définis ? Quel programme a la charge de rechercher ces symboles ?
14. Que renvoi la commande *strings* sur *exo1.o* ?
15. Par quelles lettres commence *exo1.o*?
16. Affichez les différentes sections du fichier objet avec : *objdump -s exo1.o*
17. Affichez la table de symbole du fichier objet avec : *objdump -t exo1.o*
18. Affichez la table de symbole du fichier exécutable avec : *objdump -t exo1*
19. Recompilez de nouveau le code source avec : *gcc -c exo1.c -o exo1.o*, et réaffichez la table de symbole avec *objdump*, que remarquez-vous ?

Remarque : avec cette commande *gcc -c exo1.c -o exo1.o*, le fichier source sera compilé pour générer un fichier objet sans effectuer la phase de liaison

Exercice 2 : Edition des liens

Il est possible de créer un fichier exécutable à partir de plusieurs fichiers objets. On parle alors de compilation séparée. Ceci permet de découper un programme, et d'éviter d'avoir un seul (immense) fichier source : il est possible d'avoir plusieurs fichiers source en .c, chacun sera compilé pour donner un fichier objet en .o, et enfin l'édition de liens de tous ces fichiers .o produira l'exécutable.

Nous allons illustrer ce concept à l'aide du programme exo2.c :

```
#include <stdio.h>
int main(void){
int op1, op2 ;
printf("Entrez le premier entier ") ;
scanf("%d",&op1) ;
printf("Entrez le second entier ") ;
scanf("%d",&op2) ;
printf("Le plus grand est %d\n", plusgrand(op1,op2)) ;
return 0 ;
}
```

Qui utilise la fonction plusgrand, définie dans le fichier plusgrand.c :

```
int plusgrand(int arg1, int arg2){

return arg1<arg2 ? arg2 : arg1 ; }
```

1. Produisez les fichiers objets plusgrand.o et exo2.o, correspondant respectivement aux fichiers source plusgrand.c et exo2.c
2. Réalisez l'édition de lien avec : *gcc -o exec exo2.o plusgrand.o -o exo2* à partir des deux fichiers objets, pour créer l'exécutable exo2, et testez-le (Qu'est-ce que vous remarquez ?),
3. En utilisant l'utilitaire objdump, inspectez la table des symboles de plusgrand.o et exo2.o avec : *objdump -t plusgrand.o* et *objdump -t exo2.o* Intéressez-vous particulièrement au symbole plusgrand dans ces deux fichiers objets : que signifie la mention *UND* à gauche de plusgrand ?

Exercice 3 : Création et utilisation des bibliothèques statiques et dynamiques

1. Création d'un fichier lib.c contenant deux procédures : addition (add) et soustraction (sous)
2. Compilation du code lib.c : *gcc -c lib.c -o lib.o*
3. Création de la bibliothèque statique avec : *ar rcs lib.a lib.o*
4. Créez un code source calcul.c qui fait appel au deux procédure add et sous qui contient un fichier en tete nommé calcul.h (contenant les prototypes des procédures créer)
5. Compilez le code calcul.c : *gcc -c calcul.c -o calcul.o*
6. Liaison avec la bibliothèque statique : *gcc -o calcule calcul.o -L. lib.a*
7. Exécutez le code de calcul : *./calcule*

1. Création d'un fichier lib.c contenant deux procédures : addition (add) et soustraction (sous)
2. Compilation du code lib.c avec : *gcc -c lib.c -fpic*
3. Création de la bibliothèque dynamique : *gcc lib.o -shared -o libmath.so*
4. Compilation du code source calcule.c : *gcc -c calcul.c -o calcul.o*
5. Liaison de la bibliothèque avec le code calcule : *gcc -o calcul calcul.o -L. libmath.so*
6. Spécification du chemin de la bibliothèque : *sudo cp libmath.so /usr/lib*
7. Exécutez le code *: ./calcul*