

Architecture (ASR1 et ASR2)

TP N° 10

1 Objectifs :

- Désassemblage de programmes
- Table des symboles
- Edition de liens

2 Organisation du TP :

Le travail effectué durant cette séance sera à rendre à votre responsable de TP par email impérativement à la fin de la séance. Le sujet de l'email sera [2012][Architecture][TP10] Nom Prénom .

Le TP comporte plus d'exercices que vous ne pourrez normalement en faire pendant la séance. Ces exercices supplémentaires-vous sont donnés à titre d'entraînement et votre responsable de TP pourra vous les corriger si vous les lui rendez. Attention, corriger vos exercices ne veut pas dire qu'il vous fournira une correction type des exercices ! Il vous indiquera des pistes pour résoudre vos problèmes et pour vous aider à trouver une solution correcte qui fonctionne.

3 Exercice 1 : Etude d'un programme C (Obligatoire)

Voici un programme test.c que vous trouverez également sur mon site web. Récupérez ce fichier et exécutez les commandes demandées.

```
/*  
***** Programme test.c *****  
*/  
#include <stdio.h>  
#define MAX 12  
  
int foo = 3;  
const int cte = 5;  
  
extern int ext1;  
  
int main()  
{  
    int x,y,z;  
  
    x = MAX;  
    y = cte;  
    z = ext1;  
    printf("foo: %d, cte=%d, ext1=%d, x=%d\n, y=%d, z=%d", foo,cte,ext1,x,y,z);  
  
    return 0;  
}
```

Architecture (ASR1 et ASR2)

TP N° 10

3.1 Préprocesseur

Le préprocesseur prend un fichier source C et traite tout ce qui commence par un # en début de ligne. Dans le fichier résultat (suffixe .i) ces commandes ont été exécutées et il ne reste que du C pur. Pour obtenir le fichier .i, faites:

```
gcc -E test.c > test.i
```

Comparez le contenu des fichiers `test.c` et `test.i`. En particulier

1. Combien de lignes comporte chacun des fichiers (utiliser la commande `wc -l`) ?
2. Qu'est devenue la ligne `#include <stdio.h>` ?
3. Que sont devenus les commentaires ?
4. Quel traitement a été réalisé sur la ligne `x = MAX` ?
5. Quel est finalement le travail du préprocesseur ?

3.2 Génération de code assembleur

Générez le code assembleur de votre programme avec l'option `-S` du compilateur `gcc`. Cette option va vous donner un fichier avec l'extension .s qui contiendra le code assembleur du programme `test`.

Tapez dans le terminal :

```
gcc -S test.c
```

Dans ce code assembleur, identifiez les différentes sections `.data`, `.rodata`, et `.text`.

Quels symboles sont associés à la section `.data` ? À quoi correspond cette section ?

Quels symboles et valeurs sont associés à la section `.rodata` ? À quoi correspond cette section ?

Vérifiez vos réponses aux questions 1 et 2 en ajoutant des déclarations au programme `test.c`.

Quel est le point d'entrée de ce programme ?

3.3 Génération du fichier objet

Tapez la commande `nm` sur votre fichier pour obtenir la liste des symboles du fichier `test.c` :

```
nm test.o
```

1. Comparer les résultats de la commande `nm` sur ce fichier `.o` avec ce que vous aviez identifié comme section dans le code assembleur.
2. Qu'indique `nm` pour les symboles `printf`, et `ext1` ? Que veulent-ils dire ? Tapez man `nm` pour vérifier.
3. Où ces symboles sont-ils définis ? Quel programme est en charge de rechercher ces symboles ?
4. Qu'est ce qui change dans le résultat de `nm` si on supprime l'utilisation de `ext1` dans l'instruction `printf` du fichier `test.c` ? Reprendre les étapes ci-dessus en supprimant la ligne `extern int ext1;` et son affichage dans `printf`.

3.4 Compilation et exécution

Générez le code objet de votre programme en tapant :

```
gcc -o test test.o
```

Quels problèmes se posent ? Remédiez-y en créant un autre fichier `externes.c` qui contient les définitions manquantes et créez le programme `test`.

Lancez la commande `nm test`. Que contient de plus la table de symboles ? D'après-vous, de quoi s'agit-il ?

Architecture (ASR1 et ASR2)

TP N° 10

4 Exercice 2 : Transformation d'un exécutable. (Obligatoire)

Sur mon site web, vous trouverez un fichier exécutable `motdepasse_simple` et `motdepasse`. Lancez ce programme. Si vous rentrez le bon mot de passe, le programme vous indiquera que vous avez réussi.

Comme vous n'avez pas assez de temps pour tester toutes les possibilités de mot de passe, réfléchissez à une autre solution pour cracker le programme.

Cherchez à récupérer le code assembleur de cet exécutable (décompilation) pour observer les instructions. Ensuite, vous modifierez le programme exécutable directement pour qu'il affiche que vous avez réussi, peu importe le mot de passe rentré.

Finalement, quel est le mot de passe ?

5 Exercice 3 : A vous de programmer (Conseillé).

Ecrivez en C un programme récursif qui calcule la somme des n premiers nombres entiers naturels (codés sur 16 bits) d'un tableau. On supposera disposer d'un tableau sous la forme d'une variable globale. Ce tableau sera initialisé à l'avance de 20 valeurs différentes. Le nombre de caractères à sommer est lu au clavier, et le résultat est affiché à l'écran.

L'algorithme récursif est le suivant :

```
Int16 SomNb (unsigned int16 n) {  
    if ( n > 1 ) then  
        return ( SomNb(n-1) + TAB(n-1) ) ;  
    else  
        return TAB(0) ;  
}
```

Générez le code assembleur de votre programme ainsi que la table des symboles.

Comparez ce code avec le code assembleur que vous auriez écrit pour ce programme.