Architecture des ordinateurs
TP stack

Regarder le fichier `TP_stack.c`, puis le compiler et l'exécuter avec la commande `gcc -Wall -Wextra -o TP_statck -g TP_stack.c && ./TP_stack`. Le résultat n'est sans doute pas celui qui était attendu. Changer la valeur de retour de la fonction et vérifier que c'est bien cette nouvelle valeur qui est affichée. Nous allons étudier ce programme à l'aide de l'utilitaire GDB.

Lancer GDB avec la commande `gdb -q TP_stack`. Taper deux fois `list` pour obtenir le code source (grâce à l'option `-g` utilisée lors de la compilation). Désassembler ensuite les fonctions `main` et `fonction` à l'aide des commandes `disass main` et `disass fonction`.

```
(gdb) list
1       #include <stdio.h>
2
3       int fonction(void){
4         char ensi[4] = "ensi"; //656E7369
5         char *retour;
6         retour = ensi + 20;
7         (*retour) += 10;
8         return 5;
9       }
(gdb) list
11      int main(void){
12        int i;
13        fonction();
14        i = 1;
15        printf ("%d\n", i);
16        return 0;
17      }
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000000701 <+0>: push   %rbp
   0x0000000000000702 <+1>: mov    %rsp,%rbp
   0x0000000000000705 <+4>: sub    $0x10,%rsp
   0x0000000000000709 <+8>: callq  0x6aa <fonction>
   0x000000000000070e <+13>: movl   $0x1,-0x4(%rbp)
   0x0000000000000715 <+20>: mov    -0x4(%rbp),%eax
   0x0000000000000718 <+23>: mov    %eax,%esi
   0x000000000000071a <+25>: lea    0xa3(%rip),%rdi        # 0x7c4
   0x0000000000000721 <+32>: mov    $0x0,%eax
   0x0000000000000726 <+37>: callq  0x580 <printf@plt>
   0x000000000000072b <+42>: mov    $0x0,%eax
   0x0000000000000730 <+47>: leaveq
   0x0000000000000731 <+48>: retq
End of assembler dump.
```

```
(gdb) disass fonction
   0x00000000000006aa <+0>: push   %rbp
   0x00000000000006ab <+1>: mov    %rsp,%rbp
   0x00000000000006ae <+4>: sub    $0x20,%rsp
   0x00000000000006b2 <+8>: mov    %fs:0x28,%rax
   0x00000000000006bb <+17>: mov    %rax,-0x8(%rbp)
   0x00000000000006bf <+21>: xor    %eax,%eax
   0x00000000000006c1 <+23>: movl   $0x69736e65,-0xc(%rbp)
   0x00000000000006c8 <+30>: lea    -0xc(%rbp),%rax
   0x00000000000006cc <+34>: add    $0x14,%rax
   0x00000000000006d0 <+38>: mov    %rax,-0x18(%rbp)
   0x00000000000006d4 <+42>: mov    -0x18(%rbp),%rax
   0x00000000000006d8 <+46>: movzbl (%rax),%eax
   0x00000000000006db <+49>: add    $0xa,%eax
   0x00000000000006de <+52>: mov    %eax,%edx
   0x00000000000006e0 <+54>: mov    -0x18(%rbp),%rax
   0x00000000000006e4 <+58>: mov    %dl,(%rax)
   0x00000000000006e6 <+60>: mov    $0x5,%eax
   0x00000000000006eb <+65>: mov    -0x8(%rbp),%rcx
   0x00000000000006ef <+69>: xor    %fs:0x28,%rcx
   0x00000000000006f8 <+78>: je     0x6ff <fonction+85>
   0x00000000000006fa <+80>: callq  0x570 <__stack_chk_fail@plt>
   0x00000000000006ff <+85>: leaveq
   0x0000000000000700 <+86>: retq
End of assembler dump.
```

On place un break à la ligne 12 (du code C), avant l'appel de la fonction (`break 12`), un second à la ligne 5 après être entré dans la fonction (`break 5`), puis un troisième avant de quitter la fonction (`break 8`). On lance alors le programme en tapant `r` (pour run) qui va s'arréter au premier break.

On vérifie que l'on est bien juste avant l'instruction `callq 0x6aa <fonction>` en affichant le contenu du registre `rip` (pointeur d'instruction) avec la commande `x $rip`. On affiche de la même manière le contenu des registres `rbp` et `rsp`, puis on continue en tapant `c`, en s'arrétant au second break.

Afficher de nouveau le contenu des trois registres précédents, en vérifiant que l'on est bien placé juste après l'instruction qui place la chaine `ensi` dans la pile (variable locale). Nous voyons que la fonction déclare un pointeur `retour` qui pointe à l'adresse `ensi+20`. On affiche son contenu avec `x ensi+20`.

Vérifier dans le code assembleur de `main` que le contenu précédent est justement l'adresse de retour de la fonction (i.e. celle de l'instruction qui suit le `callq`). Regarder ensuite à quelle instruction correspond l'adresse de retour augmentée de 10 (qui est réalisée à la ligne 7 du code C). Vérifier que l'adresse de retour a bien été augmentée de 10 en tapant `c`, puis `x ensi+20`.

Regarder dans le code assembleur quels sont les instructions qui n'ont pas été réalisées à cause du changement d'adresse du retour de la fonction. Normalement vous avez compris pourquoi la variable i ne vaut pas 1 avant l'appel à la fonction `printf`. Maintenant il reste à voir pourquoi le programme affiche la valeur de retour de la fonction. Regarder pour cela dans quel registre était stocké l'argument de `printf` et regarder quelle valeur y est stockée.