

# **GDB c'est quoi ?**

GDB = python tutor sur pc, sans internet !  
(donc tres pratique en examen par exemple)

# Comment le lancer ? (1)

Pour commencer dans le terminal :  
Il faut compiler avec les flags habituels (-Wall -Werror -Wextra)  
+  
**Le flag -g**

```
plerick@c1r3p10 ~/Projet/C 01/intra-uuid-66  
% cc -Wall -Werror -Wextra -g ft_strlen.c [
```

# Comment le lancer ? (2)

Voici la commande :

```
gdb --tui a.out
```

Le --tui sert à avoir un affichage visuel

```
% gdb --tui a.out [
```



# Comment le lancer ? (2)

Vous allez obtenir l'écran suivant :

```
5  /*                                     +:~+ +:~+ +:~+ */
6  /*  By: plerick <marvin@42.fr>          +#+ +#+ +#+ */
7  /*                                     +#+ +#+ +#+ */
8  /*  Created: 2024/08/13 19:24:41 by plerick  ###  ### */
9  /*  Updated: 2024/08/13 19:58:57 by plerick  ###  #####.fr */
10 /*                                     */
11 /* ***** */
12
13 #include <unistd.h>
14 #include <stdio.h>
15
16 int  ft_strlen(char *str)
17 {
18     int  i;
19
20     i = 0;
21     while (str[i] != '\0')
22     {
23         i++;
24     }
25     return (i);
26 }
27
28 int  main(void)
29 {
30     char  c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
37
```

rec No process In:  
gdb) █

# Comment naviguer dans l'outil (1)

Etablir un “break main” pour dire où est ce que l'on veut démarrer (valider avec la touche “enter”)

Pour ça, écrivez :

**b main**

**b main**

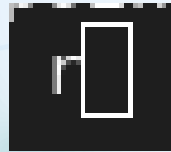


```
10 /*
11 /* *****
12
13 #include <unistd.h>
14 #include <stdio.h>
15
16 int    ft_strlen(char *str)
17 {
18     int    i;
19
20     i = 0;
21     while (str[i] != '\0')
22     {
23         i++;
24     }
25     return (i);
26 }
27
28 int    main(void)
29 {
30     char    c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
```

# Comment naviguer dans l'outil (2)

Il faut maintenant lancer l'outil avec la commande suivante :

**r** (pour run) (valider avec la touche "enter")

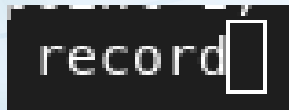




# Comment naviguer dans l'outil (3)

La commande suivante servira pour plus tard (nous y reviendrons) mais il est préférable de la faire maintenant :

**record** (valider avec la touche “enter”)

A screenshot of a terminal window with a black background. The word 'record' is written in white text, followed by a white rectangular cursor box.

```
record
```

# Comment naviguer dans l'outil (4)

Voici la commande pour passer a la ligne suivante et naviguer dans l'outils :

**n** (pour next)



```
B+> 29 {
      30     char c[] = "lol";
      31
      32     ft_strlen(c);
      33     printf ("il y a %d", ft_strlen(c));
      34     return (0);
      35 }
      36
```



```
B+ 29 {
      30     char c[] = "lol";
      31
      32     ft_strlen(c);
      33     printf ("il y a %d", ft_strlen(c));
      34     return (0);
      35 }
      36
```



# Comment naviguer dans l'outil (5)

Une fois sur une ligne appelant une fonction, pour aller voir ce qui se passe dedans, vous devez faire la commande suivante :

**s** (cela va vous emmener dans la fonction)



```
29 {
30     char c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
```

```
12
13 #include <unistd.h>
14 #include <stdio.h>
15
16 int ft_strlen(char *str)
17 {
18     int i;
19
20     i = 0;
21     while (str[i] != '\0')
22     {
23         i++;
24     }
25     return (i);
26 }
27
28 int main(void)
29 {
30     char c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
```

# Comment naviguer dans l'outil (6)

Si vous êtes passés trop vite dans votre code, vous pouvez faire un retour en arrière avec la commande suivante :

**reverse-next** (c'est grâce au "record" de plus tôt)

```
reverse-next
```

# Comment surveiller des variables (1)

Pour afficher des variables et voir comment elles évoluent au fur et à mesure que le code s'exécute il faut faire la commande suivante :

**display** (le nom de ce que vous voulez voir)

```
display i
```



# Comment surveiller des variables (2)

Avec la commande precedente, vous aurez l'evolution a chaque nouvelle etape !

```
12
13 #include <unistd.h>
14 #include <stdio.h>
15
16 int ft_strlen(char *str)
17 {
18     int i;
19
20     i = 0;
21     while (str[i] != '\0')
22     {
23         i++;
24     }
25     return (i);
26 }
27
28 int main(void)
29 {
30     char c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
37
```

```
record-ful Thread 0x7ffff7d787 In: ft_strlen
(gdb) record
(gdb) n
(gdb) reverse-next

No more reverse-execution history.
main () at ft_strlen.c:30
(gdb) n

No more reverse-execution history.
main () at ft_strlen.c:32
(gdb) s
ft_strlen (str=0x7ffffffdc48 "lol") at ft_strlen.c:20
(gdb) display i
1: i = 0
(gdb) n
1: i = 0
1: i = 0
(gdb) □
```

```
12
13 #include <unistd.h>
14 #include <stdio.h>
15
16 int ft_strlen(char *str)
17 {
18     int i;
19
20     i = 0;
21     while (str[i] != '\0')
22     {
23         i++;
24     }
25     return (i);
26 }
27
28 int main(void)
29 {
30     char c[] = "lol";
31
32     ft_strlen(c);
33     printf ("il y a %d", ft_strlen(c));
34     return (0);
35 }
36
37
```

```
record-ful Thread 0x7ffff7d787 In: ft_strlen
(gdb) reverse-next

No more reverse-execution history.
main () at ft_strlen.c:30
(gdb) n

No more reverse-execution history.
main () at ft_strlen.c:32
(gdb) s
ft_strlen (str=0x7ffffffdc48 "lol") at ft_strlen.c:20
(gdb) display i
1: i = 0
(gdb) n
1: i = 0
1: i = 0
(gdb) n
1: i = 1
(gdb) □
```

# Comment quitter

Pour quitter une fois fini, voici la commande :


**q** (pour quit)

```
q
```

Suivi par :

**y** (pour yes)

```
Quit anyway? (y or n) y
```



**Voila c'est fini !  
Maintenant a vous de vous  
entrainer pour l'utiliser !**

plerick – Bloktar sur discord