TD1

# Exercice 1

## Regardez le contenu de argc et argv. Est-ce cohérent avec le contenu de launch.json ? Quel est le lien entre le caractère / et 47 ?

Oui c’est cohérent : *\*prog* contient le chemin du fichier executé, *\*arg* contient 456

/ est le caractère 47 de la table ASCCII

# Exercice 2

## Avancez pas à pas jusqu’à la ligne 11. Observez les valeurs de prog et arg. Notez qu’une info bulle affiche leur contenu quand vous passez sur la variable dans l’éditeur.

J’ai vu

# Exercice 3

## On remarque que \*w = 2 ainsi que z, ce qui est normal puisque \*w=z. On aimerait voir le contenu de &w. On peut pour cela ajouter un espion. Cliquer + dans la zone ESPION et tapez &w. Dépliez la valeur de &w. Comprenez-vous les relations entre : &w, \*&w et \*\*&w

&w = adresse mémoire de w   
\*&w = valeur de w  
\*\*&w = valeur pointée par l’adresse de w 🡺 donc valeur de z

# Exercice 4

## Avancez jusqu’au dernier point d’arrêt et regardez la sortie dans le terminal.

J’ai vu « bonjour »

# Exercice 5

## Enlevez tous les points d’arrêt. Placez un point d’arrêt conditionnel (clic droit) sur la ligne 10 à condition que z == 2. Lancez le debug, vous devez constater que vous vous arrêtez bien

Oui

# Exercice 6

## Placez un espion sur \*argv@3.

Une image contenant texte

Description générée automatiquement

# Exercice 7

## En ligne de commande du debugger, taper « -exec info registers », cela affiche le contenu des registres de la machine

Une image contenant texte

Description générée automatiquement

# Exercice 8

### Placez un point d’arrêt sur chacune des lignes du programme.

oui

### Placez les espions

oui

### Tracez pas à pas jusqu’à avoir exécuté l’instruction char \*ci=tab;. Que déduisez-vous de l’organisation mémoire ? Comment sont placés les octets à l’intérieur d’un int ?

Les int sont stockés en l’ordre inverse de ses bits

### Y a-t-il une différence entre \*ci@16 et \*si@8 ?

Oui les adresses pointées ne sont pas les memes

Dans si on affiche les données en int contrairement a ci qui affiche les données en char

### Tracez pas à pas jusqu’à la fin du programme et observez bien ii, si et ci. Pour quelle raison ii++ et si++ n’ont pas le même effet sur la valeur de ii et si ? De même, pourquoi ii++ et ci++ n’ont pas le même effet sur la valeur de ii et ci ?

Les incrémentations sont différentes pour les différents types de données

# Exercice 9

## Tentez de découvrir quel est le bug de ce programme.

Le problème est que l’on va trop loin dans la boucle for, on override la valeur de la constante sig

# Exercice 10

## Quel algorithme de tri implémente la fonction mysort ?

Tri shell

# Exercice 11

## Placer des points d’arrêts et des observateurs d’expression.

Oui

# Exercice 12

## Corrigez maintenant le problème et recompilez avec ctrl-shift-b

Oui

# Exercice 13

## Dans les programmes fournis dans l’archive, vous avec le programme list.c. Tentez de le déboguer le plus rapidement possible.

Pas trouvé

# Exercice 14

## Exécutez votre programme de tri corrigé avec ltrace. Quelles est (sont) la (les) bibliothèque(s) partagées que votre programme tri utilise. Quelles sont les fonctions de cette (ces) bibliothèques(s) qui sont utilisées ?

Une image contenant texte

Description générée automatiquement

# Exercice 15

## Utilisez le programme *strace* pour trouver où se trouve(nt) la (les) bibliothèques(s) qui sont chargées

Une image contenant texte

Description générée automatiquement

# Exercice 16

## Enfin, voici un dernier programme rechercheBinaire.c pour vous entrainer au debug d’un programme C dans l’environnement Visual Code.