

Les pointeurs et l'évaluation d'expressions :

Rappels :

Si on place un symbole & devant un nom de variable, on obtient son adresse au lieu de sa valeur.

Si on place un symbole * devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.

Lien utile :

Voici un lien vers un outils de visualisation des pointeurs donné par monsieur Donnet sur le forum ecampus : <http://pythontutor.com/c.html#mode=edit> .

Exercices :

(Exercices provenant du Challenge 5 de 2020-2021.)

Soit l'état de la mémoire suivant :

Adresse : Valeur :		
y :	504	??
	500	316
s[0] :		...
		...
		...
	316	1917
x :	312	1789
	308	1521
	304	1099
	300	500

La première colonne donne le nom des variables, la deuxième donne des adresses mémoires (nous les exprimerons dans ce challenge en base 10) et, enfin, la troisième donne la valeur stockée à cette adresse (?? signifie que la valeur est indéterminée). De plus, voici comment ont été déclarées les variables (on suppose que les entiers (int) sont représentés sur 4 octets) :

```

1 #include<stdio.h>
2 int x, y, *p = x + 2;
3 char *s = "Challenge";
4 char *t = s + 4, *u = t + 2;
```

Vous devez indiquer la valeur des expressions suivantes. Considérez qu'entre chaque expression, la mémoire est réinitialisée telle que présentée dans le schéma. Si un accès mémoire est demandé à une adresse hors de l'intervalle [300, 504], mentionnez l'erreur de segmentation par la valeur SF. Les adresses sont aussi représentées sur 4 octets.

1. x

On a

x : 300

500

Dans ce cas, on veut la valeur ce qui nous donne x = 500

2. *(int*)x

x : 300

500

On a que la valeur de x = 500


(int*) transforme x en pointeur qui prend pour adresse 500

x : 300

500

(int*)x : 500

316



Et *(int*)x regarde la valeur pointée par l'adresse contenue dans x

Ça signifie que je veux accéder à l'objet à l'adresse "x" donc accédé à l'objet ce trouvant à l'adresse mémoire "500" donc 316

3. &x

On a

x : 300

500

Dans ce cas, au vu de l'opérateur & on veut l'adresse de x ce qui nous donne &x = 300

4. $\&*p$

D'abords calculons $*p = x + 2$

$*p$ va prendre une adresse comme résultat

On a l'adresse de $x = 300$

Etant donné que l'on fait la somme d'une adresse et d'une valeur, on doit multiplier la valeur par 4 (car dans cet exercice, on suppose que les entiers – int – sont représentés sur 4 octets) pour pouvoir sommer.

Autrement dit, $*p = 300 + 2*4 = 308$

$*p :$	308	1521
$+2*4$	304	1099
$x :$	300	500

Ce qui nous donne :

	504	??
$y :$	500	316
		...
$s[0] :$...
		...
	316	1917
	312	1789
$*p :$	308	1521
	304	1099
$x :$	300	500

Et donc $\&*p = 308$

5. $*\&y$

On a

$y :$	500	316
-------	-----	-----

L'opération est équivalente à $*(\&y)$

Par la priorité des opérations, on s'occupe d'abord de ce qui se passe entre parenthèse ($\&y$) puis on s'occupe du reste $*$

On a $(\&y) = 500$ qui donne l'adresse de y et donc $*\&y = *(500)$ qui donne la valeur à l'adresse

		$*\&y$
	$\&y$	↓
$y :$	500	316

Donc $*(\&y) = *(500) = 316$

6. &y - p

On a,

y :	<u>500</u>	316
*p :	<u>308</u>	1521

Commençons par calculer &y, au vu de l'opérateur &, on prend l'adresse de y → &y = 500

Etant donné que p est un pointeur, on sait que *p nous donne la valeur → *p = 1521 et que p sans * nous donne son adresse → p = 308

Donc &y - p = 500 - 308 = 192 mais attention la valeur attendue est un entier et non la valeur d'une adresse c'est à dire qu'il faut diviser le résultat par 4 (car dans cet exercice, les variables que l'on utilise sont des entiers représentés sur 4 octets).

Ce qui nous donne, 192 / 4 = 48

Au final, on a &y - p = 48

Ce calcul est similaire à un tableau d'entier :

int intervalle = 48;

L'écart entre la première valeur et la dernière est de 48

int tab[intervalle];

Si les int sont bien représentés sur 4 octets, on a

tab [47] :					→ 48 int = 192 octets
					+
tab[...] :					...
					+
tab[0] :					→ 1 int = 4 octets

L'espace total occupé par le tableau est de 192 octets et pour avoir l'intervalle entre la première et la dernière valeur on divise par 4 ce qui nous donne bien 48

7. &y + x

y :	<u>500</u>	316
x :	300	<u>500</u>

On a &y = 500 et x = 500

Etant donné que l'on fait la somme d'une adresse et d'une valeur, on doit multiplier la valeur par 4 (car dans cet exercice, y est un entier représenté sur 4 octets) pour pouvoir sommer.

Autrement dit, &y + x = 500 + 4*500 = 2500

8. $y \& +x$

y :	500	<u>316</u>
x :	300	<u>500</u>

On a $y = 316$ et $x = 500$

Dans ce cas, l'opérateur $\&$ est l'opérateur de comparaison bit-à-bit

Pour effectuer une telle opération, il suffit de convertir 316 et 500 en binaire et ensuite de faire la table de vérité d'une conjonction $= \wedge = \&$.

$$(316)_{10} = (100111100)_2$$

$$(500)_{10} = (111110100)_2$$

$$\begin{array}{r} 100111100 \\ \& 111110100 \\ \hline 100110100 \end{array}$$

Le résultat de la table de vérité est 100110100

En convertissant 100110100 en base décimal, on obtient $(100110100)_2 = 308$

On a donc $y \& + x = 308$

9. $*p * y$

y :	500	<u>316</u>
*p :	308	<u>1521</u>

On a $*p = 1521$ et dans ce cas le deuxième opérateur est l'opérateur de la multiplication étant donné que y n'est pas un pointeur. La variable $y = 316$

En remplaçant les valeurs dans le calcul, on a $*p * y = 1521 * 316 = 480636$

10. $++*p$

*p :	308	<u>1521</u>
------	-----	-------------

On peut noter l'expression sous la forme $++(*p)$

On a $*p = 1521$

Ce qui nous donne $++ 1521$

Puisque le $++$ est à gauche, 1521 sera directement incrémenté $\rightarrow ++*p = ++ 1521 = 1522$

11. (*p)++

*p : 308 1521

On a *p = 1521

Ce qui nous donne 1521 ++

Puisque le ++ est à droite, 1521 sera incrémenté après avoir été affiché

→ *p++ = 1521++ = 1521

12. *p--

*p : 308 1521

On a *p = 1521

Ce qui nous donne 1521 --

Puisque le -- est à droite, 1521 sera décrémenté après avoir été affiché

→ *p-- = 1521-- = 1521

13. *--p

*p : 308 1521

On peut noter l'expression sous la forme *(--p)

On a p = 308 ce qui nous donne *(--308)

Puisque le -- est à gauche, on décrémente 308 MAIS attention étant donné que 308 est une adresse, on ne décrémente pas de 1 mais de 1*4 (car dans cet exercice, on suppose que les entiers sont représentés sur 4 octets)

Autrement dit, *(--p) = *(--308) = *(308 - 1*4) = *(304)

Et le pointeur sur l'adresse 304 est égal à 1099

Au final, *--p = 1099

14. (short *)p + y

y :	500	<u>316</u>
*p :	<u>308</u>	<u>1521</u>

On a p = 308 et y = 316

(short*) convertie le pointeur vers entier p en pointeur vers short. Ça ne change rien à son adresse MAIS ça change la taille qui est accessible depuis ce pointeur



Ce qui veut dire que p n'est plus représenté sur 4 octets mais 2 octets

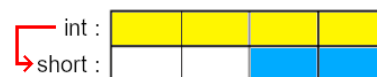
L'opération est donc $(\text{short}^*)p + y = 308 + 316 * 2 = 940$

15. y + (short)*p

y :	500	<u>316</u>
*p :	308	<u>1521</u>

On a y = 316 et *p = 1521

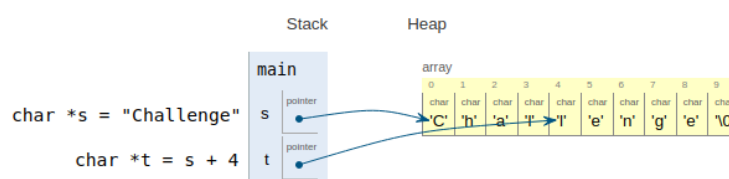
Le short à l'effet d'un cast sur *p



Autrement dit, celui-ci devient un short mais nous n'avons pas de perte de précision, donc *p reste égal à 1521

Puisqu'on additionne deux valeurs, on a $y + (\text{short}) * p = 316 + 1521 = 1837$

16. t - s



On sait que t est l'adresse mémoire du caractère 'l' faisant partie de la chaîne de caractère s "Challenge" et s est l'adresse mémoire du caractère 'C' faisant partie de la chaîne de caractère s ("Challenge")

t est donc séparé de s par 3 autres adresses mémoires (celles de 'h', 'a', 'l') ce qui nous donne un intervalle égal à 4

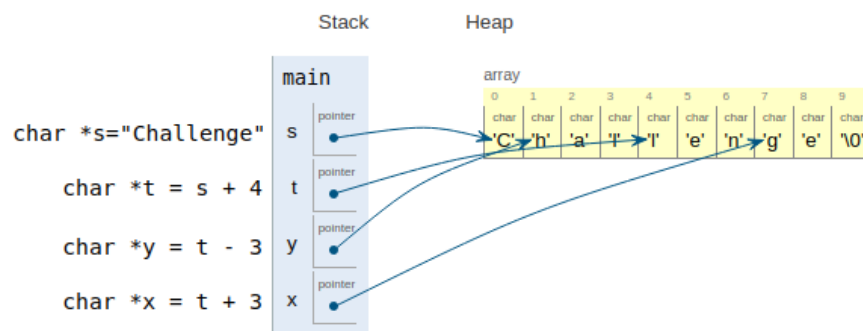
Par exemple, imaginons que l'adresse s vaut 1000, t vaudra alors 1004

Donc à partir de l'exemple, on a $t - s = 1004 - 1000 = 4$

On aurait pu faire le même calcul en partant des données de l'image, s = 0 et t = 4

Ce qui nous donne $t - s = 4 - 0 = 4$

17. $t[-3] - t[3]$



En partant de l'image, on a $t[-3] = y[0] = 'h'$ et $t[3] = x[0] = 'g'$

Ici nous faisons une opération arithmétique sur deux chars

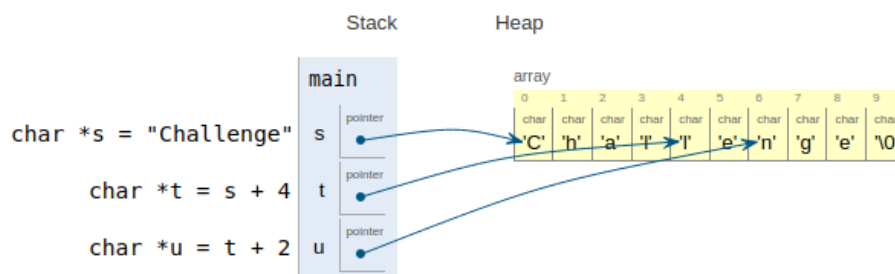
En faisant une opération arithmétique sur des chars, une promotion de type entier se produit, les caractères sont remplacés par leur valeur dans la table ASCII

Decimal	Hexadecimal	Binary	Octal	Char
103	67	1100111	147	g
104	68	1101000	150	h

On a $'h' = 104$ et $'g' = 103$

Au final, on a $t[-3] - t[3] = 'h' - 'g' = 104 - 103 = 1$

18. $*(s + (*u - t['u' - 't']))$



Procédons étape par étape,

Commençons par calculer $t['u' - 't']$

Decimal	Hexadecimal	Binary	Octal	Char
116	74	1110100	164	t
117	75	1110101	165	u

On a $'u' = 117$ et $'t' = 116$

De là $t['u' - 't'] = t[117 - 116] = t[1] = 'e' = 101$

Decimal	Hexadecimal	Binary	Octal	Char
101	65	1100101	145	e

Ensuite calculons $*u - t['u' - 't']$

Decimal	Hexadecimal	Binary	Octal	Char
110	6E	1101110	156	n

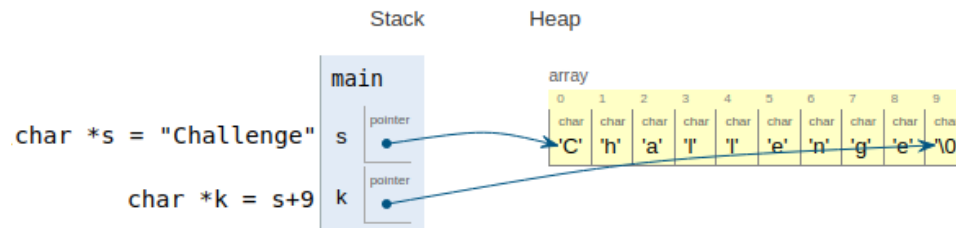
On a $*u = 'n' = 110$

Donc $*u - t['u' - 't'] = 110 - 101 = 9$

Après on fait $s + (*u - t['u' - 't'])$

Pour rappel s vaut "Challenge" mais il ne faut pas oublier que les chaînes de caractères ont automatiquement le caractère de fin de chaîne `"\0"` donc s vaut en réalité "Challenge\0" (où `\0` est un seul caractère)

Donc $s + (*u - t['u' - 't']) = s + (9) = '\0'$



Enfin on fait le pointeur sur ce caractère `'\0'`. `'\0'` permet (à `printf()` par exemple) de déterminer où se trouve la fin de la chaîne de caractère, il ne peut y en avoir qu'un par chaîne de caractère. `'\0'` est équivalent à 0, attention `'\0'` et 0 sont synonymes mais `'0'` n'a rien avoir ! Dans la table ascii `'\0'` et 0 ont la valeur 0 mais `'0'` lui a la valeur 48, ce qui n'a évidemment rien avoir

Pour finir, on a $*(s + (*u - t['u' - 't'])) = *('\0')$

On accède au pointeur en adresse mémoire 0 $\rightarrow *('\0') == *(0)$

Et le pointeur vers 0 est toujours NULL (qui est = 0)

Donc si on fait `*(0)` en c, on réfère vers NULL et si on l'affiche, on aura la valeur 0

Au final, on a $*(s + (*u - t['u' - 't'])) = *(s + ('n' - t[117 - 116])) = *(s + (110 - t[1]))$

$= *(s + (110 - 'e')) = *(s + (110 - 101)) = *(s + (9)) = *('\0') = 0$

19. p++ + ++x

*p :	<u>308</u>	1521
x :	300	<u>500</u>

On a $p = 308$ et $x = 500$

Etant donné qu'on additionne une adresse et une valeur, on doit multiplier la valeur par 4 (car dans cet exercice, x est un entier représenté sur 4 octets)

On obtient donc $x = 500 * 4 = 2000$

Ensuite regardons les opérateurs d'incrémentement,

On a $p = 308$

Ce qui nous donne $p++ = 308 ++$

Puisque le $++$ est à droite, 308 sera incrémenté après l'opération

→ $p++ = 308++ = 308$

On a aussi $x = 2000$

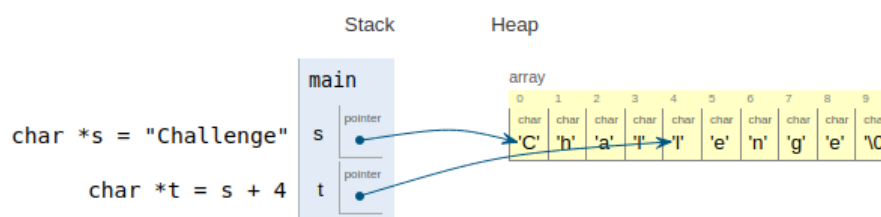
Ce qui nous donne $++x = ++2000$

Puisque le $++$ est à gauche, on incrémente 2000 MAIS attention étant donné que 2000 est une adresse, on n'incrémente pas de 1 mais de $1*4$ (car dans cet exercice, on suppose que les entiers sont représentés sur 4 octets)

→ $++x = ++2000 = 1*4 + 2000 = 2004$

Au final, $p++ + ++x = 308++ + ++2000 = 308 + 2004 = 2312$

20. printf("%s", t)



On demande la valeur de cette expression, pas le comportement de printf. Le comportement de printf est d'afficher dans une console le contenu passé en paramètres.

MAIS ce n'est pas ce que printf retourne, printf retourne la longueur de la chaîne de caractères affichée dans la console. Dans notre cas la longueur de "leng" vaut 5.