Pointeurs

Exercice 1. Mémoire simplifiée.

Remplissez le tableau ci-après indiquant la valeur de chaque variable à la fin de la ligne de code correspondante. Les premières lignes vous sont données en exemple (`NDF` pour non défini, `?` pour valeur inconnue).

adresse	@1	@2	@3	@4	@5	@6
variable	a	b	c	p1	p2	p3
`int a=3, b;`	3	?	NDF	NDF	NDF	NDF
`int *p1 = &a, *p2 = NULL;`	3	?	NDF	@1	NULL	NDF
`char c = 'z';`	3	?	z	@1	NULL	NDF
`char *p3 = &c;`	3	?	z	@1	NULL	@c
`*p3 = 'r';`	3	?	r	@1	NULL	@c
`p2 = &b;`	3	?	r	@1	@2	@c
`*p2 = 9;`	3	9	r	@1	@2	@c
`*p2 = *p1;`	3	3	r	@1	@2	@c
`*p1 = 8;`	8	3	r	@1	@2	@c
`p1 = p2;`	8	3	r	@2		@c
`*p1 = 8;`	8	8	r			@c
`*p1 = *p2 = *p3+5;`	8					@c

Exercice 2. Manipulation de pointeurs.

Cette série de questions a pour unique but de vous faire manipuler les bases des pointeurs. Mêmes s'ils sont quelque peu rébarbatifs, sollicitez votre chargé de TD afin de vérifier l'exactitude de votre code pour chacun d'entre eux.

1) Déclarez une variable `a` et un pointeur `p_a` de types entier. Affectez à la variable `a` la valeur de votre choix.
2) A l'aide de courtes instructions, affichez la valeur de votre variable de deux façons différentes (toujours en utilisant la fonction `printf()` bien entendu) ;
3) Affectez à votre pointeur la valeur NULL puis rédigez une instruction permettant de mettre la valeur 10 à l'endroit où il pointe. Expliquez le résultat.
4) Déclarez une variable `f` de type `float` et un pointeur `p2` de type `double`. A l'aide d'un schéma de la mémoire, expliquez pourquoi il est interdit de faire pointer `p2` sur `f`.

1)
```
int a = 1;
int* p_a = &a;
```
2)
```
printf("%d\n", a);
printf("%d\n", *p_a);
```
3)
```
p_a = NULL;
*p_a = 10;
```

```
celin@LAPTOP-B7O5D89I:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 5/EXERCICES/echange$ gcc *.c *.h
celin@LAPTOP-B7O5D89I:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 5/EXERCICES/echange$ ./a.out
Segmentation fault
celin@LAPTOP-B7O5D89I:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 5/EXERCICES/echange$
```

→ on envoie SO "nulle part".

4) double prend plus d'espace mémoire qu'un float

1 Utilité des pointeurs

Exercice 3. Mémoire simplifiée.

Complétez le tableau ci-dessous représentant l'état de la mémoire **à la fin de l'exécution du code** présenté dans ce même exercice.

```
void func (short *pa, int b)
{
  int i;
  for (i=0; i<4; i++)
    *pa+=b;
}

int main()
{
  short a=1337, *pa = &a;
  int b = 7, *pb = NULL;
  func (pa, b);

  pb = &b;
  *pb *= *(pa)/100;

  return 0;
}
```

Adresse	Nom de var.	Valeur de var.
@1	a	1358
@2	pa	@1
@3	b	1336
@4	pb	@3
@5		
@6		
@7		
@8		

Exercice 4. Mémoire complète.

Complétez le tableau ci-dessous représentant l'état de la mémoire **à la fin de l'exécution du code** présenté ci-dessous.

```
int i;
short *pa=NULL,*pb=NULL;
short  a=1337, b=7;
pa = &a;
pb = &b;
for (i=0; i<4; i++, a+=i)
  *pb += *pa / 10;
```

Adresse	Nom de var.	Valeur de var.
0x0160		
0x0161		
0x0162		
0x0163		
0x0164		
0x0165		
0x0166		
0x0167	pa	NULL
0x0168		
0x0169		
0x016A		
0x016B		
0x016C		
0x016D		
0x016E		
0x016F	i	
0x0170		
0x0171		
0x0172		

2 Résumé

Exercice 5. Synthèse du cours.

Complétez les phrases suivantes. Notez que celles-ci forment un bon résumé du cours que vous venez de voir !

1) Un pointeur est une valeur qui contient une adresse
2) Une adresse fait référence à un _____ _____ .
3) On accède à l'adresse d'une variable à l'aide du symbole & que l'on place avant la variable.
4) L'étoile permet d'accéder à la valeur _____ pointée par le pointeur. On dit plus simplement que l'on accède au _____ d'un pointeur.
5) L'étoile permet aussi de _____ un pointeur .
6) La fonction `toto (int a)` réalise un passage par variable
7) La fonction `titi (int *a)` réalise un passage par pointeur

8) Si je veux faire en sorte que la modification d'une zone mémoire soit effective même après la fin de ma fonction, je dois réaliser un passage par pointeur

Exercice 6. Création de fonctions (Moodle).

Pour chacune des questions suivantes, vous devez rédiger le code C permettant de réaliser la ou les actions demandées.

1) Écrire le prototype de la fonction `f()` qui prend en paramètres d'entrée l'entier a, le pointeur sur flottant b et le caractère c et qui retourne l'adresse d'un entier.
2) Écrire la fonction `puissancePointeur()` qui prend en paramètres un pointeur sur un entier `p_a` et un entier `n` et qui ne retourne rien. Cette fonction doit stocker à l'adresse donnée par `p_a` le résultat du contenu initial de `p_a` à la puissance `n`.
3) Rédigez les lignes de code permettant l'appel de la fonction précédente depuis le `main()` afin de calculer 4^3.
4) Rédigez une fonction `echange()` prenant en paramètres les adresses de deux caractères et qui échange leurs contenus. Rédigez également l'appel de cette fonction sur les variables `c1 = 'r'` et `c2 = 'k'`. Vérifiez votre résultat à l'aide de `printf()` avant, pendant et après l'appel de la fonction.
5) Appelez la fonction précédente avec un pointeur à NULL. Si votre programme vous retourne un segfault, corrigez votre fonction pour éviter cette mésaventure. Sinon, c'est que vous avez déjà des réflexes d'un codeur chevronné !

3 Pour aller plus loin

Exercice 7. Toujours plus loin.

Pour les plus rapides d'entre vous, nous vous proposons une petite récréation calculatoire. Considérons un entier positif inférieur à 1000 et calculons son miroir. Ajoutons ces deux entiers puis réalisons le même processus jusqu'à ce que le nombre obtenu soit un palindrome. Par exemple, 37 ajouté à 73 forme 110. Ajoutons 011 à 110 pour former 121 ; un palindrome est trouvé.

Environ 90% des entiers inférieurs à 1000 forment un palindrome en moins de 7 itérations. Votre objectif est de calculer le nombre moyen d'itérations à réaliser pour trouver un palindrome à partir d'un entier à 3 chiffres.