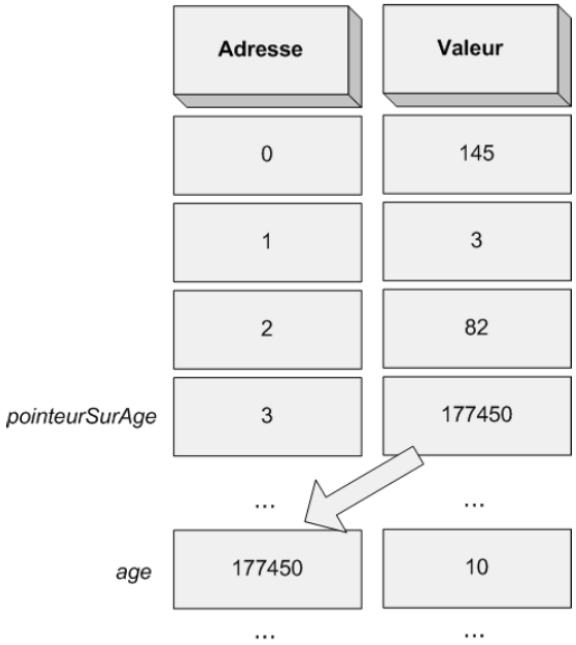
Tp n° 1 : Pointeurs

Intro



En résumé

1. Chaque variable est stockée à une adresse précise en mémoire.
2. Les pointeurs sont semblables aux variables, à ceci près qu'au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
3. Si on place un symbole & devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : &age).
4. 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.
5.

Fonction malloc

void * malloc(size_t memorySize);
Cette fonction permet d'allouer un bloc de mémoire dans le tas (le heap en anglais). Attention : la mémoire allouée dynamiquement n'est pas automatiquement relachée. Il faudra donc, après utilisation, libérer ce bloc de mémoire via un appel à la fonction free.

Paramètres

memorySize : permet de spécifier la taille (en octets) du bloc de mémoire à allouer.

Valeur de retour

Si tout se passe bien, la fonction vous renvoie un pointeur sur la zone nouvellement allouée. Attention, ce pointeur est de type pointeur générique (void *) : si nécessaire, il est donc de votre responsabilité de "caster" votre pointeur vers un autre type (du moins si vous souhaitez ne pas avoir de warning affiché par le compilateur, ce que je vous recommande vivement).
Dans le cas où le bloc de mémoire ne pourrait vous être réservé (plus de mémoire disponible, par exemple), le pointer NULL vous sera retourné

6. voir la fonction calloc
   https://koor.fr/C/cstdlib/calloc.wp

Partie 1 : Manipulation des pointeurs

EXERCICE 1

Taper ces lignes de code et afficher et interpréter les valeurs de A, B, C, *p1 et de *p2 pour chaque instruction :

```
…
int A, B, C ;
int *p1, *p2;
A=1 ;
B=2 ;
C=3 ;
p1 =&A ;
p2=&C ;
*p1=(*p2)++ ;
p1=p2 ;
p2=&B ;
*p1-=*p2 ;
++*p2 ;
*p1*=*p2 ;
A=++*p2**p1 ;
p1=&A ;
*p2=*p1/=*p2 ;
…
```

EXERCICE 2

P1 et P2 sont des pointeurs pointant sur des réels et alloués dynamiquement avec la fonction *malloc*. Ecrire un programme qui affecte au contenu de P1 la valeur -45,78 et au contenu de P2 la valeur 678,89, affiche les valeurs de P1, P2 et de leurs contenus.

Partie 2 : Pointeurs et tableaux statiques

EXERCICE 3

Soit P un pointeur qui 'pointe' sur un tableau A :

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Ecrire un programme permettant d'afficher les valeurs fournies par ces expressions :

```
*P+2
*(P+2)
&P+1
*(&A[4]-3)
*(A+3)
&A[7]-P
P+(*P-10)
*(P+*(P+8)-A[7])
```

EXERCICE 4

Ecrire un programme qui lit deux tableaux d'entiers A et B, et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A.

Utiliser le formalisme pointeur à chaque fois que cela est possible.

EXERCICE 5

Ecrire un programme qui remplit 2 matrices A et B de taille (M,N), calcule leur somme dans une 3ème matrice C de taille (M,N) et l'affiche.

Partie 3 : Pointeurs et tableaux dynamiques

EXERCICE 6

Ecrire un programme qui range les éléments d'un tableau d'entier A, de taille N, dans l'ordre inverse. Le programme utilisera des pointeurs P1 et P2 et une variable numérique AIDE pour la permutation des éléments. Le programme doit afficher le nouveau tableau.

Partie 4 : Pointeurs et chaines de caractères

EXERCICE 7

Ecrire un programme qui lit une chaîne de caractères CH et détermine la longueur de la chaîne à l'aide d'un pointeur P. Le programme n'utilisera pas de variables numériques.

EXERCICE 8

Ecrire de deux façons différentes, un programme qui vérifie si une chaîne CH introduite au clavier est un palindrome (sans utiliser la bibliothèque *<string.h>)*

Rappel : Un palindrome est un mot qui reste le même qu'on le lise de gauche à droite ou de droite à gauche.
Exemple : otto est un palindrome.