Pointeurs en C - Solutions

Hicham Janati

1 Introduction

1. **Reprendre ce code sur votre machine :**

2. **Créez un pointeur sur un entier et changez sa valeur en passant par le pointeur.**

```c
#include<stdio.h>
int main(){
    int a = 10;
    int* p_a;
    p_a = &a;
    printf("La variable a = %d\n",a);
    *p_a = 1000;
    printf("La variable a = %d\n",a);
    return 0;
}
```

```
La variable a = 10
La variable a = 1000
```

3. **Reprenez cette fonction et devinez pourquoi la ligne 8 n'a aucun effet !**
   En passant la variable a à la fonction *Changer*, la fonction n'opère pas *directement* sur la variable a en mémoire mais crée une copie de a ! à cette copie, elle affecte la valeur 0. Mais comme la copie est une variable *locale* à la fonction, elle est détruite à la sortie de la fonction. La variable a reste inchangée.

4. **Pour y remédier, au lieu de passer à la fonction la variable, on lui donne un pointeur sur a : elle aura ainsi un accès direct à sa place en mémoire et pourra changer sa valeur. Remplacez la fonction Changer par la suivante, quel changement doit être effectué à main pour utiliser cette nouvelle fonction ?** Il faut passer l'adresse de a à la fonction Changer

```c
#include<stdio.h>
void Changer(int* a){
        *a = 0;
    }

int main(){
    int a = 10;
    printf("La variable a = %d\n",a);
    Changer(&a);
    printf("La variable a = %d\n",a);
    return 0;
}
```

5. **Écrire une fonction qui prend deux arguments et permutent leurs valeurs *en mémoire*.**

```c
#include<stdio.h>
void Permut(int* a, int* b){
        int aux;
    aux = *a;
    *a = *b;
    *b = aux;
    }
```

2 Pointeurs et tableaux

1. **Complétez le code suivant pour afficher l'adresse chaque case du tableau T de deux manières différentes :**

```c
int i;
for(i=0;i<5;i++){
        printf("methode 1: &T[i]  =  %p  \n",  &T[i]   );
        printf("methode 2:   T + i     =   %p  \n", T+i     );
}
```

2. **Comme *p est la valeur stockée à l'adresse p, eh ben pour afficher la (i+1)ème valeur d'un tableau T, on peut logiquement écrire *(T+i). À ne pas confondre avec *T+i ! *T+i est tout simplement égale à la valeur T[0]+i alors que *(T+i) est T[i]. (Je dis (i+1) ème car, rappelez-vous, on commence l'indexation à 0). Écrire un programme qui affiche les valeurs du tableau de deux manières différentes.**

```c
int i;
for(i=0;i<5;i++){
        printf("methode 1: T[i] =  %f  \n", T[i]    );
        printf("methode 2:  *(T+i) = %f   \n", *(T+i)    );
}
```

3. **Écrire un programme qui permet à l'utilisateur de remplir un tableau en utilisant la deuxième méthode.**

```c
int main(){
 int T[5];
 int i;
 for(i=0;i<5;i++){
        printf("Saisir la case %d \n", i: );
        scanf("%d",T+i);
 }
```

4. **Écrire un programme qui affiche le tableau en utilisant la deuxième méthode.**

```c
int main(){
 int T[5];
 int i;
 for(i=0;i<5;i++){
        printf("La case %d = %d\n", i, *(T+i));
 }
```

3 Exercices

En utilisant les pointeurs :

1. Écrire une fonction qui prend deux variables passées en adresses et **renvoie** le nombre le plus grand.

```
int PlusGrand(float *a, float *b){
  if (*a>*b) return *a;
  else return *b;
}
```

2. Écrire une fonction qui inverse un tableau. Il suffit de parcourir la moitié du tableau :

```
int Inverser(float *T, int N){
  int i;
  float aux;
  for(i=0;i<N/2;i++) {
  aux = *(T+i)
  *(T+i) = *(T+N-1-i);
  *(T+N-1-i) = aux;
  }
}
```

3. Écrire une fonction qui inverse une chaîne de caractères.

```
int InverserChaine(char* chaine){
  int i=0,j;
  char aux;
  // on calcule la taille de la chaine:
  while(chaine[i]!='\0') i++;
  for(j=0;j<i/2;j++;i--){
        aux = chaine[i];
    chaine[i] = chaine[j];
    chaine[j] = aux;}
  }
}
```

4. Écrire une fonction qui prend deux tableaux de même taille et permutent leurs valeurs.

```
int Permutation(int* A, int *B, in N){
  int aux,i;
  for(i=0;i<N;j++){
        aux = A[i];
    A[i] = B[i];
    B[i] = aux;
  }
}
```

5. Vous avez appris qu'une fonction à qui on passe une variable en valeur en fait une copie locale et ne peut donc pas accéder à son adresse en mémoire. Créez un programme qui permet de le vérifier.
   Il suffit d'afficher l'adresse de la variable dans main et dans la fonction pour vérifier qu'elles sont bien différentes : la fonction crée bien une autre variable locale.

```
#include<stdio.h>
void F(int a){
      printf(" Adresse de a dans F: %p ", &a);
   }

int main(){
```

```
7       int a = 10;
8       printf(" Adresse de a dans main: %p ", &a);
9       F(a);
10          return 0;
11  }
```

```
1   Adresse de a dans main: 0x7ffdb893c8ac
2   Adresse de a dans F:    0x7ffdb893c88c
```

L'avant dernier caractère n'est pas le même.