TP 3 - A la découverte des pointeurs

L2 SPI - UE Langage et programmation

Université d'Evry Val d'Essonne

Exercise 1 : Manipulation simple de pointeurs

Soit le programme :

```
int main(void)
{
        int a;
        int *pa;
        int **ppa;

        a = 1;
        pa = &a;
        ppa = &pa;

        return(0);
}
```

1. Que valent les expressions suivantes &a, *pa, &pa, *ppa, **ppa ? Parmi les expressions précédentes, lesquels désignent la même la valeur que celle contenue dans a, pa et ppa ? Vérifier ce que donne l'affichage de ces entités en exécutant le programme et compléter le tableau suivant :

	a	pa	ppa
&a			
*pa			
&pa			
*ppa			
**ppa			

Exercise 2 : Passage par valeur et par adresse

On considère le programme suivant :

```
void echange1(int a, int b)
{
        int tampon;
        tampon = b;
        b = a;
        a = tampon;
}

void echange2(int *a, int *b)
{       int tampon;
        tampon = *b;
        *b = *a;
        *a = tampon;
}

void main()
```

```
{
        int u=3, v=4;
        echange1(u,v);
        echange2(&u,&v);
}
```

1. Comparer les deux fonctions echange1 et echange2. Que sont-elles sensées réaliser ?
2. Laquelle n'a pas atteint son objectif ? Donner les valeurs de u et v dans les deux cas après l'appel de chacune des fonctions.
3. Expliquer le mécanisme de passage des paramètres à une fonction par valeur et par adresse. Donner les avantages et inconvénients de chacun des mécanismes.

Exercise 3 : Opérations sur les pointeurs

Soit le programme :

```
int main()
   {
      int A = 1;
      int B = 2;
      int C = 3;
      int *P1, *P2;
      P1=&A;
      P2=&C;
      *P1=(*P2)++;
      P1=P2;
      P2=&B;
      *P1-=*P2;
      ++*P2;
      *P1*=*P2;
      A=++*P2**P1;
      P1=&A;
      *P2=*P1/=*P2;
      return 0;
   }
```

1. Compléter le tableau ci-dessous pour chaque instruction du programme.

	A	B	C	P1	P2
Init.	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C					
*P1=(*P2)++					
P1=P2					
P2=&B					
*P1-=*P2					
++*P2					
P1=*P2					
A=++*P2**P1					
P1=&A					
*P2=*P1/=*P2					

Exercise 4 : Tableaux et pointeurs

1. Ecrire une fonction *moyenne* qui calcule la moyenne des valeurs entières contenues dans un tableau de taille 5 passé en argument.
2. Modifier votre code pour que la taille du tableau soit choisie par l'utilisateur à l'aide de la fonction **scanf**. Remplacer l'allocation statique du tableau par une allocation dynamique à

l'aide de la fonction **malloc**.

3. Pourquoi la nouvelle version de la fonction *moyenne* a besoin de deux arguments ? Si le premier argument est le pointeur vers le tableau contenant la suite des valeurs, quel est le second ?

Exercise 5 : Tableaux de caractères

1. Donner la taille d'un entier en octets à l'aide de la fonction **sizeof** sur votre machine.
2. Soit les déclarations suivantes. Soit Ai, l'adresse de i et Atab l'adresse de tab.

```
int i[2];
char tab[7]={0x42,0x6C,0x65,0x75,0x00,0x74,0x00};
char *pc ;
int *pi;
```

On exécute un programme et on s'arrête après les instructions suivantes :

```
 pc = tab ;
*i = 0x0602E3FF;
*(i+1)=0x0001000F;
pi = i+1;
```

Quelle est la valeur de la variable pi (en fonction de Ai) ?

3. Donner les valeurs en décimal du contenu du tableau i.
4. On exécute alors l'instruction :

```
 *pi = 255;
```

Donner le nouveau contenu mémoire (de l'adresse Ai à l'adresse Ai+1) en hexadécimal.

5. Donner la taille en octets d'une variable de type char à l'aide de l'opérateur **sizeof**.
6. On s'intéresse à la zone mémoire à partir de l'adresse Atab. On exécute alors l'instruction :

```
printf("\n<%s>\n", tab);
```

Donner le résultat de l'affichage à l'écran. Expliquer cet affichage.

7. On exécute les instructions suivantes :

```
*(pc+4) = tab[2] ;
printf("\n>%s<\n", pc);
```

Donner le résultat de l'affichage à l'écran. Expliquer les étapes qui ont conduit à cet affichage.