université Paris Ouest Nanterre La Défense

TD 4 : les fonctions, les tableaux et les pointeurs

Exercice 1. – Passage de tableau en paramètre

1. Qu'affiche le programme suivant ? Justifiez.

```
#include <stdio.h>

/* Cette fonction fait certainement */
/* quelque chose... */
void transformation(int tab[], int taille) {
  int i;
  for (i=0; i<taille; i++)
    if (tab[i]%2 == 0)
      tab[i] = tab[i]/2;
}

/* Le programme principal a pour but de */
/* montrer ce que fait la fonction. */
int main() {
  int tabf[5] = {1,2,6,3,7};
  int i;

  /* On affiche le tableau avant */
  for (i=0; i<5; i++) {
    printf("%d ",tabf[i]);
  }
  printf("\n ");

  /* On appelle la fonction */
  transformation(tabf,5);
  /* On affiche le tableau après */
   for (i=0; i<5; i++) {
    printf("%d ",tabf[i]);
  }
  printf("\n ");

  return 0;
}
```


Exercice 2. – Fonctions et pointeurs

Soit le programme suivant :

```
#include <stdio.h>

void ma_fonction(int *param1, int param2) {
  int var_loc = 3;
  *param1 = var_loc * param2;
  param2 = var_loc + 1;
}

int main() {
  int v1, v2;

  v1 = 10;
  v2 = 3;

  ma_fonction(&v2,v1);
  return 0;
}
```

1. Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme précédent. Nous vous rappelons que les paramètres sont implantés sous la forme de variables locales à la fonction.

Exercice 3. – Passage de valeur et d'adresse

On considère une fonction `f` ayant deux paramètres `i1` et `i2` de type entier et renvoyant un résultat de type entier. La fonction calcule la somme des deux paramètres, **l'enregistre dans une variable locale `i3`** (de type entier), puis retourne la valeur de cette variable.

1. Écrivez la fonction `f`.

2. Les instructions suivantes d'appel de `f` provoquent-elles une erreur de typage ? Pourquoi ?

   1. `int a; a = f(1,2);`
   2. `int a,b,c; a = f(b,c);`
   3. `int a,b,c; a = f(&b,c);`

3. Quel sens cela a-t-il d'ajouter l'instruction `i2 = i3` au code de la fonction `f` (juste avant l'instruction **`return`**) ? Expliquez votre réponse.

4. Dans le programme suivant, la fonction `f` est celle obtenue après la modification de la question précédente. Donnez :
   – l'instruction d'appel à la fonction `f` dans la fonction `main`,
   – la valeur des variables `i1`, `i2` et `res` avant l'appel à `f`, au début de `f`, à la fin de `f` et après l'appel à `f`.
   Attention, pour les variables de nom identique dans les fonctions `f` et `main` vous devrez préciser de quelle variable vous donnez la valeur.

```
#include <stdio.h>

/* fonction f*/
.... f(.............){
  ...............
}
```



```
int main() {
  int i1=1, i2=2, res;

  /* res prend la valeur retournee par la fonction f */
  /* appelee avec i1 et i2 comme parametres */
  ..........................

  return  0;
}
```

5. Maintenant, nous voudrions que la modification du paramètre `i2` soit répercutée également à l'extérieur de la fonction.

   Modifiez le prototype et le corps de le fonction `f` en conséquence. Donnez le contenu des variables `i1` et `i2` et `res` avant l'appel à la fonction `f`, au début de `f`, à la fin de `f` et après l'appel à la fonction `f`.

6. Avec cette nouvelle version de `f`, quelles instructions d'appel provoquent des erreurs de typage ?

   1. `int a; a = f(1,2);`
   2. `int a,b,c; a = f(b,c);`
   3. `int a,b,c; a = f(b,&c);`

7. Donnez les états successifs de la mémoire (pile d'exécution) lorsque l'exécution du programme suivant atteint les lignes 17, 6, 9, 11 et 19.

```
1  include <stdio.h>
2
3  int f(int i1, int *p_i2) {
4    int i3;
5    int i4 = *p_i2;
6
7    i3 = i1 + i4;
8    i4 = i3;
9
10   *p_i2 = i4;
11
12   return i3;
13 }
14
15 int main(int arv, char * arg[]) {
16   int i1=1, i2=2, res=0;
17
18   res=f(i1, &i2);
19
20   return 0;
21 }
```

Exercice 4. – Calcul des Racines d'un polynôme du 2nd degré

Soit le programme suivant qui doit permettre d'afficher le nombre de racines d'un polynôme du second degré.

```
#include <stdio.h>

/* fonction racine */
.......... racine (..............................){
  ............................
```

```
}

int main(){
  int a,b,c;
  int nb_racines;

  printf("Saisie du polynome a*x*x + b*x +c \n");
  printf("Veuillez saisir a :");
  scanf("%d", &a);

  printf("Veuillez saisir b : ");
  scanf("%d", &b);

  printf("Veuillez saisir c : ");
  scanf("%d", &c);

  /* appel a la fonction racine pour determiner */
  /* la valeur du nombre de racines */
  ..............................

  if (nb_racines==2)
    printf("Le polynome a 2 racines. \n");

  if (nb_racines==1)
    printf("Le polynome a 1 racine double. \n");

  if (nb_racines==0)
    printf("Le polynome n'a pas de racine reelle. \n");

  return 0;
}
```

1. Donnez le prototype de la fonction `racine` qui doit retourner le nombre de racines du polynôme $ax^2 + bx + c$ et ajoutez dans la fonction `main` qui vous est données l'appel à la fonction `racine`.
2. Écrivez la fonction `racine`.
3. Nous souhaitons maintenant, qu'en plus de retourner le nombre de racines, la fonction `racine` permette de récupérer les valeurs des racines. Donnez le prototype de la fonction `racine` modifiée et modifiez la fonction `main` si nécessaire.
4. Écrivez la nouvelle fonction `racine`.

