**Corrigé TD2**

# Exercice I :

**====A===**
**0x7fffffffde94**
**1**
**0x7fffffffde94**
**0x7fffffffde94**

**====B===**
**a = 6, b = 5, *ptr1 = 6, *ptr2 = 6**

**====C===**
**warning: ‘p’ is used uninitialized [-Wuninitialized]**
**Segmentation fault**
***p = 4 - aa = 4**

**====D===**
**1 2 3**

**====E===**
**[ 5 3 5 ]**

**====F===**
**2 4 5**
**2**

**====G===**
**0x7fffffffdea0 => Adresse de nombre => 0x5000**

# Exercice III

```
// Somme P1(x) + P2(X) et eval P(x)

void p_sum(float *R, const float *P1 , const float *P2 , unsigned int degree) ;

float p_eval_horner(float x, const float *P, unsigned int degree);

float p_eval(float x, const float *P, unsigned int degree);

float xpuissancei(float x, unsigned int degree);


//*****************************************************
int main () {
   // P1 : x^2 + 2x + 3
   // P2 : 3x^3 + x^2 + 4
   // P3 : 3x^3 + 2x^2 + 2x + 7 (c'est-à-dire P1+P2)
   float P1[4] = { 3., 2., 1., 0. } ;
   float P2[4] = { 4., 0., 1., 3. } ;
   float P3[4] ;
   float x;

   //somme
   p_sum (P3 , P1 , P2 , 3) ;

   //affichage somme
   for ( int i = 0 ; i <= 3 ; ++i) {
      printf ("c%d = %.2f\n", i, P3[i]) ;
   }

   // Calcul P(x)
   printf("Entrez x :");
   scanf("%f", &x);
   printf ("\n P1(%.2f) = %.2f \n", x, p_eval_horner(x,P1,3)) ;
   printf ("\n P1(%.2f) = %.2f \n", x, p_eval(x,P1,3)) ;
   printf ("\n (%.2f)^3 = %.2f \n", x, xpuissancei(x,3)) ;
   return 0 ;
}


//*****************************************************

// Somme P1(x) + P2(X)
void p_sum (float * R, const float * P1 , const float * P2 , unsigned int degree) {
   for ( unsigned int i = 0; i <= degree ; ++i) {
      R[i] = P1[i] + P2[i];
   }
}
```

```
//*****************************************************

// Evaluation P(x) par Methode de Horner
float p_eval_horner(float x, const float *P, unsigned int degree) {
   unsigned int i = degree ;
   float result = P[i];
   while (i >0) {
      result = (x* result ) + P[i -1];
      --i;
   }
   return result ;
}



//*****************************************************

// Evaluation P(x) par Methode brutale
float p_eval(float x, const float *P, unsigned int degree) {
   unsigned int i = degree ;
   float result = P[0];
   while (i >0) {
      result = result + P[i] * xpuissancei(x,i);
      --i;
   }
   return result ;
}

float xpuissancei(float x, unsigned int degree){
   unsigned int i = degree ;
   float result = 1;
   while (i > 0) {
      result = x*result;
      --i;
   }
   return result ;
}
```