

# **Erreurs fréquentes**

## à l'examen intermédiaire 08/09

Vincent Lepetit  
vincent.lepetit@epfl.ch

# Question 3.a

Ecrivez la fonction d'en-tête

```
void additionne_polynomes(float * P1, float * P2, float * R)
```

qui permet de calculer la somme de deux polynômes  $P1$  et  $P2$ , passés en paramètre. Le résultat devra être stocké dans le polynôme  $R$ . On suppose que ce polynôme  $R$  est déjà alloué correctement.

Il y a 3 erreurs dans cette fonction:

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < 5; i++)
        *R[i] = *P1[i] + *P2[i];
    return R;
}
```

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < 5; i++)
        *R[i] = *P1[i] + *P2[i];
    return R;
}
```

Le type de la fonction et le type de l'expression après `return` doivent être les mêmes.

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < 5; i++)
        *R[i] = *P1[i] + *P2[i];
    return R;
}
```

Si le type de la fonction est `void`, la fonction ne doit pas avoir de d'instruction `return` suivie d'une expression.

Elle *peut* néanmoins avoir une instruction `return` seule:  
`return ;`

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < 5; i++)
        *R[i] = *P1[i] + *P2[i];
}
```

**Pas d'étoiles ici. On peut écrire soit:**

`R[i] = P1[i] + P2[i];`

**ou:**

`*(R + i) = *(P1 + i) + *(P2 + i);`

***mais pas:***

`*R[i] = *P1[i] + *P2[i];`

Pas de constantes numériques!  
Utilisez la constante prédéfinie (`degre_max`).  
Cependant, cette erreur est bien moins grave que les deux précédentes.

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < 5; i++)
        R[i] = P1[i] + P2[i];
}
```

Cette fonction est correcte:

```
void additionne_polynome(float * P1, float * P2, float * R)
{
    for(int i = 0; i < degre_max; i++)
        R[i] = P1[i] + P2[i];
}
```

# Question 3.b

Écrivez la fonction d'en-tête

```
float evaluer_polynome(float * P, float a)
```

qui permet d'évaluer le polynôme en a.

La fonction ci-dessous fonctionne mais est trop compliquée:

```
float evaluer_polynome(float * P, float a)
{
    float U[degre_max];

    U[0] = P[degre_max];
    for(int i = 1; i < degre_max; i++)
        U[i] = P[degre_max - i - 1] + a * U[i - 1];

    return U[degre_max - 1];
}
```

Le tableau  $U$  n'est pas nécessaire. On peut utiliser une seule variable:

```
float evaluer_polynome(float * P, float a)
{
    float u = P[degre_max];

    for(int i = 1; i < degre_max; i++)
        u = P[degre_max - i - 1] + a * u;

    return u;
}
```

# Question 3.c

Ecrivez la fonction d'en-tête

```
float * cree_polynome_nul(void)
```

qui permet de créer un tableau correspondant au polynôme nul.

La fonction suivante a une erreur:

```
float * cree_polynome_nul(void)
{
    int * T = new int[degre_max];

    for(int i = 0; i < degre_max; i++)
        T[i] = 0;

    return T;
}
```

```
float * cree_polynome_nul(void)
{
    int * T = new int[degre_max];

    for(int i = 0; i < degre_max; i++)
        T[i] = 0;

    return T;
}
```

Le type de `T` et le type de la fonction doivent être les mêmes.

Le type de la fonction est pointeur sur `float`, et `T` a été déclaré en pointeur sur `int`. Les deux types ne correspondent donc pas.

```
float * cree_polynome_nul(void)
{
    float * T = new float[degre_max];

    for(int i = 0; i < degre_max; i++)
        T[i] = 0;

    return T;
}
```

**Il n'est pas nécessaire d'initialiser le tableau à 0.  
new le fait déjà.**

**Cependant, c'est un problème mineur (je n'ai pas enlevé  
de points à ceux qui ont écrit cette boucle).**

Cette fonction est correcte:

```
float * cree_polynome_nul(void)
{
    return new float[degre_max];
}
```

# Question 3.e

Ecrivez la fonction d'en-tête

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)
```

qui suppose que  $P$  est un polynôme de degré 2, et qui permet de trouver les racines de l'équation  $P = 0$ . On suppose qu'on dispose de la fonction

```
void calcule_racines_abc(float a, float b, float c,  
                        float * racine1, float * racine2)
```

qui calcule les 2 racines de l'équation  $ax^2 + bx + c = 0$ . La fonction `calcule_racines` devra utiliser cette fonction `calcule_racines_abc`.

La fonction suivante n'est pas correcte:

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(float P[2], float P[1], float P[0],  
                       float * racine1, float * racine2);  
}
```

Ne mettez pas les types des paramètres à l'appel de la fonction !!!

Les types sont nécessaires seulement à la déclaration de la fonction !!!

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(float P[2], float P[1], float P[0],  
                       float * racine1, float * racine2);  
}
```

**Autres versions incorrectes:**

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(P[2], P[1], P[0], *racine1, *racine2);  
}
```

**et**

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(P[2], P[1], P[0], &racine1, &racine2);  
}
```

```
void calcule_racines_abc(float a, float b, float c,  
                        float * racine1, float * racine2)  
{  
    ...  
}
```

Les types des paramètres formels et les types des paramètres effectifs doivent être les mêmes.

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(P[2], P[1], P[0], &racine1, &racine2);  
}
```

`racine1` est de type pointeur sur `float` (`float *`).  
Le paramètre effectif `&racine1` est donc de type pointeur sur pointeur sur `float`!  
Le paramètre formel correspondant est de type pointeur sur `float`, donc les types ne correspondent pas.

```
void calcule_racines_abc(float a, float b, float c,  
                        float * racine1, float * racine2)  
{  
    ...  
}
```

Dans la fonction `calcule_racines`, `racine1` est déjà de type pointeur sur `float`.

Dans la fonction ci-dessous, les types correspondent donc:

```
void calcule_racines(float * P,  
                    float * racine1, float * racine2)  
{  
    calcule_racines_abc(P[2], P[1], P[0], racine1, racine2);  
}
```

Vérifiez donc:

- quand vous écrivez une fonction, que le type de la fonction et que le type de l'expression dans l'instruction `return` soient les mêmes.
- quand vous appelez une fonction, que les types des paramètres formels et les paramètres effectifs soient identiques.

Ca ne garantit pas que votre code soit correct, mais ça vous permet d'éviter les erreurs les plus grossières.