

Algorithmique et Programmation

Examen écrit / Corrigé

G1: monasse(at)imagine.enpc.fr G2: courchay(at)imagine.enpc.fr
G3: eric.bughin(at)gmail.com G4: david.ok(at)imagine.enpc.fr
G5: francois.olivier.c.h(at)gmail.com G6: vialette(at)univ-mlv.fr

26/03/10

1 Enoncé

2 Qu'affiche ce programme ?

1. L'affichage est le suivant

```
C3: +1X2 +3X1 +3X0    // Constructeur appelé l.126
P= +1X2 +3X1 +3X0    // l.127-128
P(-2)=1              // l.129
C2: +2X1 -1X0        // l.130, attention à l'ordre des arguments dans constructeur 2
Q= +2X1 -1X0         // l.131-132
C0: +0X0             // l.97 (appelé par l.133): constructeur sans argument
op+                  // l.106
D: +1X2 +5X1 +2X0    // l.107 destruction de Q après constructeur par recopie
D: +1X2 +5X1 +2X0    // l.133 destruction du résultat de l'addition après copie dans R
R= +1X2 +5X1 +2X0    // l.134-135
C0: +0X0             // l.111 (appelé par l.136)
op-                  // l.120
D: +2X1 -1X0         // l.121 destruction de Q après constructeur par recopie
D: +2X1 -1X0         // l.136 destruction du résultat après copie dans R
R= +2X1 -1X0         // l.137-138
D: +2X1 -1X0         // l.139 destruction de P
D: +2X1 -1X0         // l.139 destruction de Q
D: +1X2 +3X1 +3X0    // l.139 destruction de R
```

Remarque : il y a 7 destructeurs appelés alors que seulement 4 constructeurs semblent appelés. En fait, il y a 3 constructeurs par recopie appelés mais celui-ci est défini par défaut, donc n'affiche rien :

- (a) Recopie de Q dans valeur de retour de l'operator+
- (b) Recopie de Q dans valeur de retour de l'operator-
- (c) Constructeur de R par recopie l.133, n'affiche rien.

3 Programmation : Enrichir la classe polynôme

2. Une possibilité :

```
class Monome {
public:
    float a;
    int k;
    Monome(float coeff, int deg);
};

Monome::Monome(float coeff, int deg) {
```

```

    a = coeff;
    k = deg;
}

```

Remarque : nous laissons les champs `a` et `k` publics pour pouvoir faire la multiplication sans avoir à définir des méthodes pour lire leur valeur.

3. Le plus simple est d'en faire une méthode de `Polynome` de manière à pouvoir lire le degré et les coefficients directement. Ainsi, on insère la ligne

```
Polynome operator*(const Monome& M) const;
```

avant la ligne 24 et on l'implémente de la façon suivante :

```

Polynome Polynome::operator*(const Monome& M) const {
    Polynome Q;
    Q.deg = deg+M.k;
    for(int i=0; i<=deg; i++)
        Q.t[i+M.k] = M.a*t[i];
    for(int i=0; i < M.k; i++)
        Q.t[i] = 0.0f;
    return Q;
}

```

Il faut bien prendre garde de mettre à 0 les coefficients de degré plus petit que `M.k` du résultat.

4. On déclare l'opérateur dans la classe pour accéder aux champs et on peut par exemple le programmer ainsi :

```

Polynome Polynome::operator%(const Polynome& Q) const {
    Polynome R = *this;
    while(R.deg >= Q.deg) {
        Monome M(R.t[R.deg]/Q.t[Q.deg], R.deg-Q.deg);
        R = R-Q*M;
    }
    return R;
}

```

La première ligne effectue une copie du membre de gauche de l'opérateur dans `R`¹. Nous n'avons pas besoin de modifier le degré de `R` puisque l'opérateur de soustraction s'en charge. Attention, cette implémentation ignore les problèmes numériques et suppose que $x - y * (x/y) = 0$ pour faire baisser le degré, ce qui peut être faux à cause de problèmes d'arrondis.

5. Noter que dans la solution proposée ci-dessous, les polynômes sont passés par *valeur*, ce qui permet de les modifier sans toucher aux paramètres.

```

Polynome PGCD(Polynome P, Polynome Q) {
    while(! Q.est_nul()) {
        Polynome R = P%Q;
        P = Q;
        Q = R;
    }
    return P;
}

```

4 Algorithmie : problème de sélection

6. Une possibilité est la suivante :

```

float selection(float t[], int n, int k) {
    float min=t[0], max=t[0];
    for(int i=0; i<n; i++)
        if(t[i]<min) min=t[i];
        else if(t[i]>max) max=t[i];
    int nb=-1;
    do {
        // Incrémente du nombre de valeurs à min
        for(int i=0; i<n; i++)
            if(t[i] == min) ++nb;
    }
}

```

¹`*this` désigne l'objet sur lequel la méthode est appelée, donc le membre de gauche dans le cas d'un opérateur binaire

```

        if(nb >= k) break;
        // Trouve le prochain minimum
        float min2=max;
        for(int i=0; i<n; i++)
            if(min<t[i] && t[i]<min2) min2=t[i];
        min = min2;
    } while(true);
    return min;
}

```

Le type des éléments de `t` n'est pas précisé dans l'énoncé, ici c'est supposé être `float`.

7. On parcourt k fois le tableau, donc $O(kn)$ et comme k peut être de l'ordre de n , $O(n^2)$.
8. Il suffit de trier, par exemple par QuickSort en $O(n \log n)$ et de choisir l'élément d'indice k .
9. Si la fonction pivot a la syntaxe suivante :

```
int pivot(float t[], int debut, int fin);
```

alors il suffit de ne trier que le sous-tableau contenant l'indice k :

```

int debut=0, fin=n-1, i=-1;
while(i != k) {
    i = pivot(t, debut,fin);
    if(i<k) debut=i+1;
    if(i>k) fin=i-1;
}
return t[i];

```

10. Comme pour QuickSort, on suppose qu'en moyenne le pivot se retrouve en position médiane, et on n'a alors qu'à itérer sur un tableau de taille $n/2$:

$$C_n = n + C_{n/2}$$

(le terme n est le coût du pivot).

11. En itérant on trouve

$$C_n = n + n/2 + C_{n/4} = (1 + 1/2 + 1/4 + 1/8 + \dots)n = O(n)$$

(la suite géométrique s'arrête au terme $\log n$, mais est de toute façon majorée par 2).

12. Par exemple pour $k = 0$ (trouver le minimum), on est obligé de tester tous les n éléments, sinon le vrai minimum pourrait être un élément qu'on a ignoré.