Algorithmique et Bases de la programmation

Durée : *1h55*
Documents autorisés : **AUCUN**

L'objectif de cet examen pratique est de développer une librairie C proposant des fonctions sur le type `Polynome`.

Informations

Avant de commencer l'examen

Connectez vous à la machine et suivez les instructions suivantes :
- copiez l'archive `C.tgz` qui se trouve dans `/opt/files` dans votre répertoire (commande `cp`) ;
- après l'avoir décompressée (commande `tar` avec les options `zxvf`), renommez le répertoire en lui donnant comme nom : 'EXAM-ALGO-' suivi de votre nom et prenom, sans espace et en minuscule sauf pour les premières lettres du prénom et du nom. Si je devais passer l'examen, je nommerai le répertoire EXAM-ALGO-DelestreNicolas (je taperai donc dans le terminal `mv C EXAM-ALGO-DelestreNicolas`)

À la fin de l'examen

Cinq minutes avant la fin de l'examen :
- assurez vous que votre projet compile ;
- créez une archive de votre répertoire (commande `tar` avec les options `zcvf`), que vous nommerez comme le répertoire (avec le préfixe `.tgz`) ;
- déposez sur moodle votre archive (cours « base de la programmation et algorithmique », section « examens pratiques »).

Attention :
- le lien de dépôt est actif uniquement 10 minutes, si vous dépassez le délai, votre note sera 0 ;
- un seul dépôt est autorisé.

Quelques conseils

Pour réussir votre examen suivez les conseils suivants :
- respectez bien les consignes données ci dessus. **Le fait de ne pas les respecter vous ferra perdre 2 points sur le note finale** ;
- par défaut le projet compile et s'exécute. Après le développement de chaque fonction C, compilez votre projet (`make`) et ne passez à la fonction suivante que lorsque votre projet compile : **le fait de rendre un projet qui ne compile pas, divisera par deux votre note finale** ;
- la note finale sera fonction :
  - du nombre de tests unitaires qui seront valides (exécution du programme `bin/testPolynomeLib_CUnit`) ;
  - de la quantité, de la qualité et de la lisibilité du code (indentation et nom des identifiants).

1 Contexte : Le type `Polynome`

1.1 Analyse

Soit le TAD `Polynome` suivant :

Nom: Polynome

Utilise: **Naturel**,**Reel**

Opérations:
- polynome: → Polynome
- obtenirDegre: Polynome → **Naturel**
- obtenirCoefficient: Polynome × Naturel → **Reel**
- modifierCoefficient: Polynome × **Reel** × **Naturel** → Polynome

Axiomes:
- obtenirCoefficient(polynome(),i)=0
- obtenirDegre(polynome())=0
- obtenirDegre(p)=d tel que $\forall i > d, obtenirCoefficient(p, i) = 0$
- obtenirCoefficient(modifierCoefficient(p,v,i),i)=v

1.2 Conception préliminaire

Dans le paradigme de la programmation structurée, le TAD `Polynome` peut être utilisé à l'aide des fonctions et procédures suivantes : :

- **fonction** polynome **() :** Polynome
- **fonction** obtenirDegre **(**p : Polynome**) : Naturel**
- **fonction** obtenirCoefficient **(**p : Polynome, degre : **Naturel) : Reel**
- **procédure** modifierCoefficient **(E/S** p : Polynome,**E** coef : **Reel**, degre : **Naturel)**

Nous pouvons aussi ajouter les fonctions (liées au choix du paradigme) suivantes :

- **fonction** sontEgaux **(**p1,p2 : Polynome**) : Booleen**
- **fonction** copier **(**p : Polynome**) :** Polynome

1.3 En C

En C, nous pouvons manipuler des variables de type `P_Polynome` en utilisant la librairie *lib/libPolynome.a* (construite à partir des fichiers `Polynome.h` et `Polynome.c`), qui propose entre autres les fonctions C suivantes (extrait du fichier `Polynome.h`)[1] :

```
20 #define P_ERREUR_MEMOIRE 1
21
22 P_Polynome P_polynome_0();
23 int P_obtenirDegre(P_Polynome p);
24 float P_obtenirCoefficient(P_Polynome p, int degre);
25 void P_modifierCoefficient(P_Polynome *p,float coef, int degre); /* errno=
     P_ERREUR_MEMOIRE si pas assez de mémoire */
26 int P_sontEgaux(P_Polynome p1, P_Polynome p2);
27 P_Polynome P_copier(P_Polynome p);
28 void P_supprimer(P_Polynome *p);
```

2 La librairie sur les polynômes : `PolynomeLib`

On se propose de développer une librairie C de manipulation de polynômes. Cette librairie permettra de :

[1] La dernière fonction est liée au langage C

- calculer la multiplication d'un polynôme par un réel ;
- calculer l'addition de deux polynômes ;
- calculer la soustraction de deux polynômes ;
- calculer la multiplication de deux polynômes ;
- calculer la dérivée d'un polynôme ;
- calculer la représentation en chaîne de caractères d'un polynôme, tel que :
  - le polynôme nul sera représentée par le chaîne `"0.00"` ;
  - les coefficients des monômes seront représentées par des réels avec une précision de deux chiffres après la virgule ;
  - l'opération puissance sera représentée par le caractère `'^'` ;
  - les monômes seront organisés suivant l'ordre décroissant de leur degrés (par exemple `"x^2+2.00x-3.50"`) ;
  - si le monôme de plus haut degré est positif, il n'y aura pas le caractère `'+'` devant ;
  - l'opérateur de multiplication dans un monôme entre le coefficient et l'inconnue n'est pas représenté ;
  - les monômes de coefficient 0 n'apparaissent pas dans la chaîne (par exemple `"-3.00x^4+x^2-2.00x-3.50"`) ;
  - le coefficient d'un monôme de valeur 1 n'apparaît pas dans la chaîne ;
  - la puissance du monôme de degré 1 n'apparaît pas dans la chaîne ;
  - l'inconnue du monôme de degré 0 n'apparaît pas dans la chaîne.

2.1 Conception préliminaire

Dans le paradigme de la programmation structurée, ces opérations se concrétisent par les fonctions suivantes :

- **fonction** multiplicationParReel (p : Polynome, r : **Reel**) **:** Polynome
- **fonction** addition (p1,p2 : Polynome) **:** Polynome
- **fonction** soustraction (p1,p2 : Polynome) **:** Polynome
- **fonction** multiplication (p1,p2 : Polynome) **:** Polynome
- **fonction** derivee (p : Polynome) **:** Polynome
- **fonction** polynomeEnChaine (p : Polynome) **: Chaine de caracteres**

2.2 Conception détaillée

Les algorithmes des ces fonctions étant très simples, nous n'allons en détailler ici que deux :

fonction multiplicationParReel (p : Polynome, r : **Reel**) **:** Polynome

Déclaration resultat : Polynome
i : **Naturel**

debut
resultat ← polynome()
pour i ←0 **à** degres(p) **faire**
modifierCoefficient(resultat,r*obtenirCoefficient(p,i),i)
finpour
retourner resultat
fin

fonction addition (p1,p2 : Polynome) **:** Polynome

Déclaration resultat : Polynome
i : **Naturel**

debut

resultat ← polynome()
pour i ←0 **à** max(degres(p1),degre(p2) **faire**
modifierCoefficient(resultat,obtenirCoefficient(p1,i)+obtenirCoefficient(p2,i),i)
finpour
retourner resultat
fin

3 Développement

Le projet C contient les fichiers suivants :
- `include/Polynome.h` : partie publique du type `P_Polynome`
- `include/PolynomeLib.h` : partie publique de la librairie `PolynomeLib`
- `src/Polynome.c` : partie privée du type `P_Polynome`
- `src/PolynomeLib.c` : partie privée de la librairie `PolynomeLib`
- `src/testPolynome_CUnit.c` : code source d'un test unitaire du type `P_Polynome`
- `src/testPolynomeLib_CUnit.c` : code source du test unitaire de la librairie `PolynomeLib`

L'exécution du `make` produit :
- `lib/libPolynome.a` : la librairie C proposant le type `P_Polynome`
- `lib/libPolynomeLib.a` : la librairie C proposant `PolynomeLib`
- `bin/testPolynome_CUnit` : test unitaire du type `P_Polynome`
- `bin/testPolynomeLib_CUnit` : test unitaire de la librairie `PolynomeLib`

Travail à réaliser Complétez le fichier `src/PolynomeLib.c`[2].

[2]Pour l'implantation de la fonction `polynomeEnChaine` nous vous invitons à utiliser entre autres la fonction C sprintf (cf. les exemples du `man sprintf`)