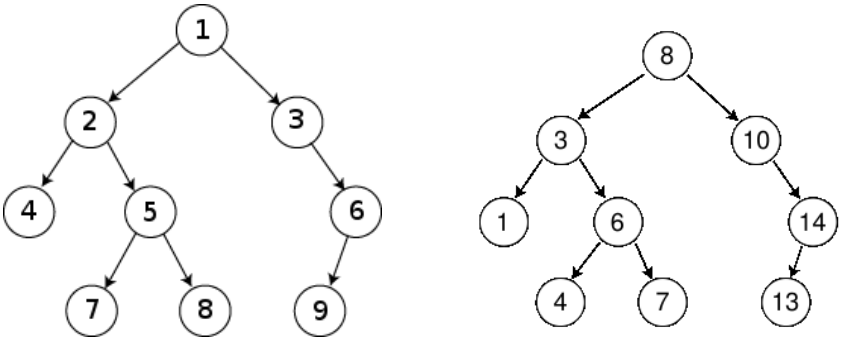
ASI3 Lundi 21 Janvier 2008

Algorithmique et Bases de la programmation

Durée : *3h*
Documents autorisés : **AUCUN**

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.

1 Questions de cours (3,5 points)



(a) Arbre binaire (b) Arbre binaire de recherche

Fig. 1 – Arbres (source Wikipédia)

1.1 Arbre binaire

Soit l'arbre de la figure 1 (a). Donnez le résultat d'un affichage de tous les nœuds de cet arbre par un parcours en profondeur de type :

1. GRD ;
2. RGD ;
3. GDR.

Solution proposée :

1. GRD : 4, 2, 7, 5, 8, 1, 3, 9, 6
2. RGD : 1, 2, 4, 5, 7, 8, 3, 6, 9
3. GDR : 4, 7, 8, 5, 2, 9, 6, 3, 1

1.2 Arbre binaire de recherche

1. En partant d'un arbre binaire de recherche vide, donnez une suite d'insertions de naturels qui permet d'obtenir l'arbre présenté par la figure 1 (b) (sans aucun mécanisme de rééquilibrage).
2. Pourquoi cet arbre n'est pas équilibré ? Quelle action permettrait de l'équilibrer ?

Solution proposée :

1. 8, 3, 10, 1, 6, 4, 7, 10, 14, 13
2. Non car la différence de hauteur des deux fils du nœud "10" est supérieure à 1. Il faudrait faire une simple rotation à gauche au niveau du nœud 10.

2 Jeux des 7 erreurs (3,5 points)

Soit le programme C suivant qui devrait afficher les nombres parfaits compris entre 1 et un nombre saisi par l'utilisateur :

```
1  #include"stdlib.h"
2
3  int saisirBorneMax() {
4    int resultat;
5    do {
6      printf("Borne max : ");
7      scanf("%d",resultat);
8    } while (resultat<=0);
9    return resultat;
10 }
11
12 int estDivisible(int a,int b) {
13   return a%b=0;
14 }
15
16 int sommeDiviseurs() {
17   int i,somme=0;
18   for(i=1;i<=a/2;i++)
19     if (estDivisible(a,i))
20       somme=somme+i;
21   return somme;
22 }
23
24 int estParfait(int a) {
25   return a==sommeDiviseurs(a);
26 }
27
28 void afficherNbsParfaits(int borneMax) {
29   int i;
30   for (i=1;i<=borneMax;i++)
31     if (estParfait(i)) {
32       printf("%d ",i);
33       fflush(stdout);
34     }
35   printf("\n");
36 }
37
38 int main(int nbParam,char* params) {
39   int borneMax;
40   borneMax=saisirBorneMax();
41   afficherNbsParfaits(borneMax);
42 }
```

Ce programme comporte 7 erreurs (qui empêchent de compiler, provoquent des warning avec les options `-pedantic` et `-Wall` , arrêtent brutalement l'exécution du programme ou donnent un mauvais résultat). Identifiez les et proposez pour chacune une correction.

Solution proposée :

1. ligne 1 : " à la place de < et >
2. ligne 1 : manque l'inclusion de stdio.h
3. ligne 7 : manque le &
4. ligne 13 : manque un =
5. ligne 16 : manque int a
6. ligne 38 : manque un * à params
7. ligne 42 manque return EXIT_SUCCESS ;

3 Polynôme (6 points)

Soit le TAD *Polynome* suivant :

Nom:	Polynome	
Utilise:	**Reel**,**Naturel**	
Opérations:	monome:	**Reel** × **Naturel** → Polynome
	addition:	Polynome × Polynome → Polynome
	degre:	Polynome → **Naturel**
	coefficient:	Polynome × **Naturel** → **Reel**
Sémantiques:	monome:	qui permet d'obtenir un polynôme composé d'un seul monome (un coefficient et un degré)
	addition:	qui permet de calculer l'addition de deux polynômes
	degre:	qui permet d'obtenir le degré d'un polynôme (monome de degré le plus élevé ayant un coefficient différent de 0)
	coefficient:	qui permet d'obtenir le coefficient d'un monome identifié par son degré

3.1 Conception préliminaire

Donnez les signatures des fonctions des opérations du TAD *Polynome*.

Solution proposée :
fonction monome (a : **Reel**, n : **Naturel**) **:** Polynome
fonction addition (p1,p2 : Polynome) **:** Polynome
fonction degre (p : Polynome) **: Naturel**
fonction coefficient (p : Polynome, n : **Naturel**) **: Reel**

3.2 Utilisation

Donnez le corps des fonctions suivantes :

1. **fonction** multiplicationParScalaire (p : Polynome, a : **Reel**) **:** Polynome
2. **fonction** soustraction (p1,p2 : Polynome) **:** Polynome
3. **fonction** multiplication (p1,p2 : Polynome) **:** Polynome

4. **fonction** derivation (p : Polynome) **:** Polynome

Solution proposée :

```
fonction multiplicationParScalaire (p : Polynome, a : Reel) : Polynome
   Déclaration  resultat : Polynome
                i : Naturel
                coef : Reel
debut
   resultat ← monome(0,0)
   pour i ←0 à degre(p) faire
      resultat ← addition(resultat,monome(a*coefficient(p,i),i))
   finpour
   retourner resultat
fin

fonction soustraction (p1,p2 : Polynome) : Polynome
   Déclaration  temp : Polynome
debut
   temp ← multiplicationParScalaire(p2,-1)
   retourner addition(p1,temp)
fin

fonction multiplication (p1,p2 : Polynome) : Polynome
   Déclaration  resultat : Polynome
                i,j : Naturel
debut
   resultat ← monome(0,0)
   pour i ←0 à degre(p1) faire
      pour j ←0 à degre(p2) faire
         resultat ← addition(resultat,monome(coefficient(p1,i)*coefficient(p2,j),i+j))
      finpour
   finpour
   retourner resultat
fin

fonction derivation (p : Polynome) : Polynome
   Déclaration  resultat : Polynome
                i : Naturel
                coef : Reel
debut
   resultat ← monome(0,0)
   pour i ←1 à degre(p) faire
      resultat ← addition(resultat,monome(coefficient(p,i)*naturelEnReel(i),i-1))
   finpour
   retourner resultat
fin
```

4 T9 (7 points)

Le T9, sigle de « text on 9 keys », est un mode de saisie de texte, mis au point par la société Tegic Communications[1], à l'aide d'un clavier numérique. Il est aujourd'hui très utilisé sur les téléphones portables pour saisir des SMS.

Ce système de saisie permet de s'affranchir de multiples frappes : une seule pression par lettre suffi. Ainsi supposons qu'un utilisateur souhaite taper le mot « SAGE », un téléphone qui en est équipé permettra de ne presser que :

- 1 fois la touche 7 pour sélectionner le groupe PQRS,
- 1 fois la touche 2 pour sélectionner le groupe ABC,
- 1 fois la touche 4 pour sélectionner le groupe GHI,
- 1 fois la touche 3 pour sélectionner le groupe DEF,

c'est-à-dire 7243, soit seulement 4 pressions au total (au lieu de 8 avec le mode de saisie classique).

Fig. 2 – Clavier de téléphone (source Wikipédia)

Bien entendu des ambiguïtés peuvent apparaître. Cette même suite de chiffres permet aussi de saisir les mots « page », « paie », « rage », etc. L'utilisateur peut donc faire défiler tous les mots correspondant à une suite de chiffres afin de choisir celui qu'il veut.

4.1 Analyse

On se propose de créer un TAD *DictionnaireT9* qui aurait les fonctionnalités suivantes :

- obtenir un dictionnaire T9 vide,
- ajouter un mot (chaîne de caractères) non vide à un dictionnaire T9,
- obtenir une liste ordonnée de mots correspondant à une suite de chiffres représentée à l'aide d'une chaîne de caractères. Si la suite de chiffres ne référence aucune liste, c'est une liste vide qui est retournée.

Définir le TAD *DictionnaireT9*

Solution proposée :

Nom:	DictionnaireT9	
Utilise:	**Chaine de caracteres**,ListeOrdonnee	
Opérations:	dictionnaireT9:	→ DictionnaireT9
	ajouter:	DictionnaireT9 × **Chaine de caracteres** → DictionnaireT9
	listeMots:	DictionnaireT9 × **Chaine de caracteres** → ListeOrdonnee<**Chaine de caracteres**>

[1]Texte de présentation extrait de Wikipédia

Sémantiques: dictionnaireT9: permet d'obtenir un dictionnaire T9 vide
ajouter: permet d'ajouter un mot non vide à un dictionnaire T9
listeMots: permet d'obtenir une liste ordonnée de mots correspondant à une suite de chiffres

4.2 Conception prélimnaire

Donnez les signatures des fonctions et procédures des opérations du TAD *DictionnaireT9*

Solution proposée :

- **fonction** dictionnaireT9 **() :** DictionnaireT9
- **procédure** ajouter **(E/S** d : DictionnaireT9 , **E** m : **Chaine de caracteres)**
  ⌊**precondition(s)** m≠""
- **fonction** listeMots (d : DictionnaireT9, lesChiffres : **Chaine de caracteres) :** ListeOrdonnee<**Chaine de caracteres**>

4.3 Conception détaillée

On se propose de représenter le TAD *DictionnaireT9* à l'aide d'une structure dynamique représentant un arbre d'arité 8 :
Type DictionnaireT9 = ^ Noeud
Type Noeud = **Structure**
lesMots : ListeOrdonnee<**Chaine de caracteres**>
lesFils : **Tableau**['2'..'9'] **de** DictionnaireT9
finstructure

On suppose posséder les fonctions :

- *obtenirTouche* qui permet d'obtenir pour une lettre donnée (un caractère) le chiffre correspondant (un caractère). Par exemple `obtenirTouche('K')` donne `'5'` :
  - **fonction** obtenirTouche (c : **Caractere) : Caractere**
    ⌊**precondition(s)** c ∈ ['a'..'z'] ou c ∈ ['A'..'Z']
- *longueur* qui permet d'obtenir la longueur d'une chaîne de caractères (avec `longueur("")=0`) :
  - **fonction** longueur (c : **Chaine de caracteres) : Naturel**
- *iemeCaractere* qui permet d'obtenir le ième caractère d'une chaîne de caractères :
  - **fonction** iemeCaractere (c : **Chaine de caracteres**, i : **NaturelNonNul) : Caractere**
    ⌊**precondition(s)** 0<i≤longueur(c)
- *sousChaine* qui permet d'obtenir une sous-chaîne d'une chaîne de caractères :
  - **fonction** sousChaine (c : **Chaine de caracteres**, debut,fin : **NaturelNonNul) : Caractere**
    ⌊**precondition(s)** 0<debut<fin≤longueur(c)

Donnez le corps des fonctions et procédures du TAD *DictionnaireT9*.

Solution proposée :
fonction dictionnaireT9 **() :** DictionnaireT9
debut
retourner NULL

fin

procédure ajouter **(E/S** d : DictionnaireT9 , **E** m : **Chaine de caracteres)**

⌊**precondition(s)** m≠""

debut

ajouterRecursif(d,m,m)

fin

procédure initDictionnaireT9 **(E/S** d : DictionnaireT9 **)**

Déclaration i : **Caractere**

debut

d.lesMots ← listeOrdonnee()

pour i ←'1' **à** '9' **faire**

d.lesFils[i] ← NULL

finpour

fin

procédure ajouterRecusrif **(E/S** d : DictionnaireT9 , **E** lesLettres,m : **Chaine de caracteres)**

Déclaration leChiffre : **Caractere**

debut

si d==NULL **alors**

d ← allouer(Noeud)

initDictionnaireT9(d)

finsi

si lesLettres="" **alors**

inserer(d.lesMots,m)

sinon

leChiffre ← obtenirTouche(iemeCaractere(lesLettres,1))

ajouterRecusrif(d.lesFils[leChiffre],sousChaine(lesLettres,2,longueur(lesLettres)),m)

finsi

fin

fonction listeMots **(**d : DictionnaireT9, lesChiffres : **Chaine de caracteres) :** ListeOrdonnees<**Chaine de caracteres**>

debut

si d==NULL **alors**

retourner listeOrdonnee()

sinon

si lesChiffres="" **alors**

retourner d.lesMots

sinon

retourner listeMots(d.lesFils[iemeCaractere(lesChiffres,1)],sousChaine(lesChiffres, 2,longueur(lesChiffres)))

finsi

finsi

fin