TP5 en Algorithmique

24 février 2009

1. Écrivez un fichier `fraction.h` qui contient : `typedef struct fraction { int num ; int den ; } frac ;` et les fonctions suivantes :
   - une fonction `frac entre()`, qui demande à l'utilisateur deux entiers et rend la fraction correspondente ;
   - une fonction `affiche(frac a)`, qui affiche à l'écran la valeur de `a` ;
   - une fonction `frac reduit(frac a)`, qui réduit `a` à une fraction telle que numérateur et dénominateur n'ont pas de facteurs en commun ;
   - une fonction `frac somme(frac a, frac b)`, qui rend `a+b` (déjà réduite) ;
   - une fonction `frac diff(frac a, frac b)`, qui rend `a-b` (déjà réduite) ;
   - une fonction `frac produit(frac a, frac b)`, qui rend `a*b` (déjà réduite) ;
   - une fonction `frac div(frac a, frac b)`, qui rend `a/b` (déjà réduite) ;
   - une fonction `frac puiss(frac a, int e)`, qui rend a^e.
   - une fonction `grand(frac a, frac b)`, qui rend `1` si `a>=b` et `0` sinon ;

   Créez un programme `prog.c` qui teste toutes les fonctions ainsi définies.
2. Répetez l'exercice ci-dessu en codant les operations sur les nombres complexes.
3. Écrivez un fichier `liste.h` qui contient : `typedef int data ; typedef struct noeud { data x ; struct noeud* next ; } node ;`

   et les fonctions suivantes :
   - une fonction `node* prepend(node* head, data i)`, qui demande la `head` d'une liste de `data` et rend la `head` de la liste une fois ajouté un élément au debout contenant la valeur de `i` ;
   - une fonction `node* rmfirst(node* head)`, qui demande la `head` d'une liste de `data` et rend la `head` de la liste une fois supprimé le premier élément, ou `NULL` si la liste est déjà vide ;
   - une fonction `node* avantlast(node* head)`, qui demande la `head` d'une liste d'entiers non vide et rend le pointeur au avantdernier élément de la liste ;
   - une fonction `node* append(node* head, data i)`, qui demande la `head` d'une liste d'entiers et rend la `head` de la liste une fois ajouté à la fin un élément contenant la valeur de `i` ;
   - une fonction `node* rmlast(node* head)`, qui demande la `head` d'une liste de `data`, et rend la `head` de la liste une fois supprimé le dernier élément, ou `NULL` si la liste est déjà vide ;

   Jusqu'à là le code peut être reutilisé avec d'autres types de donnés, en changeant juste le premier `typedef`.

   Écrire encore
   - une fonction `affichelist(node* head)`, qui affiche à l'écran la valeur des éléments de la liste pointée par `head` ;

– une fonction `node* diviseur(int n)`, qui demande un entier `n` et rend une liste chaînée des diviseurs de `n`, à partir de `1`, jusqu'à `n` ;

Adaptez le code déjà écrit pour gérer des listes de fractions, en remplaçant les noms ; puis écrivez les fonctions suivantes :

– une fonction `node* entrepoly()`, qui demande à l'utilisateur le degree `n` d'un polynôme sur $\mathbb{Q}$ et puis le `n` factions correspondants aux coefficients de ce polynôme et rend la liste de ces `n` coefficients ;
– une fonction `affichepoly(node* headpoly)`, qui affiche à l'écran le polynôme codé par la liste pointé par `headpoly` ;
– une fonction `int racine(node* headpoly, frac rac)`, qui rend `1` si `rac` est la racine du polynôme codé par `headpoly`, `0` sinon ;
– une fonction `node* solutions(node* headpoly)`, qui rend la liste de toutes les fractions différentes qui sont racines du polynôme codé par `headpoly`.