ASI3 Mercredi 14 Janvier 2015

Algorithmique et Bases de la programmation

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

Informations

Avant de commencer l'examen

Connectez vous à la machine et suivez les instructions suivantes :
- copiez l'archive `C.tar.gz` 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 : votre nom suivi de votre prénom, sans espace, sans caractères accentués et en minuscule sauf pour les premières lettres du prénom et du nom. Par exemple, si « Paul Du Villaré » devait passer cet examen, il nommerai son répertoire « DuvillarePaul ».

À la fin de l'examen

Cinq minutes avant la fin de l'examen :
- assurez vous que les fichiers sont au format `UTF-8` (c'est par défaut le cas) ;
- 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 suffixe `.tar.gz`) ;
- 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 ;
- on ne peut déposer qu'une seule fois le projet sur moodle.

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 fera perdre 2 points sur la 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 ou ne s'exécute pas (par exemple à cause d'un *segmentation fault*), divisera par deux votre note finale** ;
- la note finale sera fonction :
  - du nombre de tests unitaires qui seront valides ;
  - du nombre de *Warning* (compilation avec l'option `-Wall`) qui seront affichés ;
  - de la quantité, de la qualité et de la lisibilité du code (indentation et nom des identifiants).

1 Des Grands Nombres

L'objectif de cet examen est de commencer le développement d'une bibliothèque C permettant de faire des calculs sur des grands nombres naturels positifs (aussi grand qu'on le veut).

Pour l'instant les seules opérations proposées par cette bibliothèque sont :

- la création d'un grand nombre à partir d'une chaîne de caractères (fonction C `GN_GrandNaturel GN_chaineEnGrandNaturel(char*)`)
- la création d'une chaîne de caractères représentant en base 10 un grand nombre (fonction C `char* GN_grandNaturelEnChaine(GN_GrandNaturel)`) ;
- une fonction permettant de comparer deux grands nombres (fonction C `int GN_compare( GN_GrandNaturel, GN_GrandNaturel)`, tel que cette fonction retourne :
  - -1 lorsque le premier paramètre effectif est plus grand que le deuxième ;
  - 0 lorsque les deux paramètres effectifs sont égaux ;
  - 1 lorsque le deuxième paramètre effectif est plus grand que le premier.
- une procédure d'incrémentation d'un grand nombre (fonction C `void GN_plusUn(GN_GrandNaturel*)`) ;
- une fonction d'addition de deux grands nombres (fonction C `GN_GrandNaturel GN_addition (GN_GrandNaturel,GN_GrandNaturel)`) ;
- une procédure permettant de libérer l'espace mémoire qui est alloué pour stocker un grand nombre (fonction C `void GN_detruire(GN_GrandNaturel*)`).

1.1 Conception détaillée

La conception de ce type de donnée s'appuie sur l'*utilisation* de la structure dynamique de donnée `ListeChainee`.

Type GrandNaturel = **Structure**
  nbChiffres : **NaturelNonNul**
  chiffres : ListeChainee<**Caractere**>
finstructure

Tel que :

- les chiffres du nombre sont représentés en base 10 (grâce aux caractères de `'0'` à `'9'`) ;
- le premier élément de la liste référencée par le champ `chiffres` correspondant au chiffre des unités.

Voici deux exemples d'algorithme des fonctions de la bibliothèque :

fonction chaineEnGrandNaturel (ch : **Chaine de caracteres**) **:** GrandNaturel

⌊**précondition(s)** longueur(ch)>1 et composeUniquementDeChiffres(ch)

Déclaration res : GrandNaturel

debut
  res.chiffres ← listeChainee()
  res.nbChiffres ← 0
  pour i ←1 **à** longueur(ch) **faire**
    si iemeCaractere(ch,i)≠'0' ou res.nbChiffres≠0 ou (iemeCaractere(ch,i)='0' et res.nbChiffres=0 et i=longueur(ch)) **alors**
      res.nbChiffres ← res.nbChiffres+1
      ajouter(res.chiffres,iemeCaractere(ch,i))
    finsi
  finpour

```
   retourner res
fin
procédure plusUn (E/S gn : GrandNaturel)
   Déclaration  l,precedent : ListeChainee
                val, retenue : Naturel
debut
   retenue ← 1
   l ← gn.chiffres
   precedent ← listeChainee()
   tant que non estVide(l) et retenue=1 faire
      val ← caractereEnNaturel(obtenirElement(l))+retenue
      si val>9 alors
         val ← val-10
         retenue ← 1
      sinon
         retenue ← 0
      finsi
      fixerElement(l,naturelAUnChiffreEnCaractere(val))
      precedent ← l
      l ← obtenirListeSuivante(l)
   fintantque
   si retenue>0 alors
      ajouter(l,'1')
      fixerListeSuivante(precedent,l)
      gn.nbChiffres ← gn.nbChiffres+1
   finsi
fin
```

1.2 Développement

Le projet est constitué des fichiers suivants :

- `include/ListeChaineeDeCaracteres.h` qui déclare les fonctions permettant d'utiliser une liste chaînée de caractères ;
- `include/GrandNaturel.h` qui déclare le type GN_GrandNaturel et les fonctions permettant de l'utiliser ;
- `src/ListeChaineeDeCaracteres.c` qui définit entre autres les fonctions déclarées dans `include/ListeChaineeDeCaracteres.h`
- `src/GrandNaturel.c` qui définit entre autres les fonctions déclarées dans `include/GrandNaturel.h`
- `src/gnTU.c` le programme des tests unitaires (boite blanche et boite noire) des fonctions déclarées dans `include/GrandNaturel.h` ;

L'exécution du `make` génère le programme des tests unitaires `test/gnTU`.

2 Travail à réaliser

Complétez le fichier `GrandNaturel.c` (remplacer les parties du code commençant par le commentaire `/* Code a remplacer */` et finissant par le commentaire `/* Fin du code a remplacer */`) de façon à ce que le maximum de tests unitaires fonctionnent (`test/gnTU`). Vous commencerez par

compléter la fonction testée par la suite de tests unitaires « boite blanche » (`GN_chaineEnGrandNaturel`) qui est nécessaire au bon fonctionnement des tests unitaires « boite noire ».