ASI3 20 Janvier 2021

Algorithmique avancée et Programmation C

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, avec uniquement des lettres, sans espace, sans caractères accentués et en CamelCase (minuscule sauf pour les premières lettres de chaque mot du prénom et du nom). Par exemple, si « François-Xavier Du Villaré » devait passer cet examen, il nommerai son répertoire « DuVillareFrancoisXavier ».

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 « Algorithmique avancée et programmation C », section « Annales des 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 Un type Chaine pour le C

Pour rappel, il n'y a pas nativement de type de chaîne de caractères en C. Lorsque l'on veut représenter une chaîne de longueur l, on utilise une zone mémoire (référencée par un pointeur de type `char*`) de taille $l+1$ octets, tels que les n premiers octets représentent les n premiers caractères de la chaîne et le dernier octet de valeur 0 (ou `'\0'`) représente la fin de la chaîne. Les constantes chaînes de caractères sont entourées de double guillemets. Par exemple la constante `"bonjour"` de sept caractères utilise huit octets.

L'objectif de cet examen est de commencer le développement d'une bibliothèque C permettant de disposer d'un type Chaîne (pour chaîne de caractères) et des opérations associées.

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

CH_chaine qui permet de créer une chaîne à partir d'un `char*` ;

CH_longueur qui permet d'obtenir la longueur d'une chaîne ;

CH_copier qui permet d'obtenir une copie d'une chaîne ;

CH_concatenerP qui permet de concaténer deux chaînes (version procédurale telle que la seconde chaîne est concaténée à la première, avec un passage en entrée sortie pour la première chaîne) ;

CH_concatenerF qui permet de concaténer deux chaînes (version fonctionnelle qui retourne une nouvelle chaîne qui est la concaténation des deux chaînes données) ;

CH_inverser qui permet d'inverser une chaîne ;

CH_enCharEtoile qui permet d'obtenir une représentation de la chaîne en `char*`.

1.1 Conception détaillée

La conception de ce type de donnée s'appuie sur l'*utilisation* de la structure dynamique de donnée `ListeChainee`, tel que le premier caractère de la chaîne se trouve en tête de liste.

Type Chaine = **Structure**
  longueur : **Naturel**
  caracteres : ListeChainee<**Caractere**>
finstructure

Voici quelques algorithmes sur les listes chaînées qui sont utiles pour concevoir les opérations données ci-dessus :

procédure concatener **(E/S** l1 : ListeChainee,**E** l2 : ListeChainee**)**
  Déclaration temp : ListeChainee
debut
  si estVide(l1) **alors**
    l1 ← l2
  sinon
    si non estVide(l2) **alors**
      temp ← obtenirListeSuivante(l1)
      concatener(temp,l2)
      si estVide(obtenirListeSuivante(l1)) **alors**
        fixerListeSuivante(l1, temp)
      finsi
    finsi

finsi
fin
procédure inverser **(E/S** l : ListeChainee**)**
Déclaration resultat,temp : ListeChainee
debut
resultat ← listeVide()
tant que non estVide(l) **faire**
temp ← l
l ← obtenirListeSuivante(l)
fixerListeSuivante(temp,resultat)
resultat ← temp
fintantque
l ← resultat
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/Chaine.h` qui déclare le type `CH_Chaine` et les fonctions permettant de l'utiliser ;
— `src/ListeChaineeDeCaracteres.c` qui définit entre autres les fonctions déclarées dans `include/ListeChaineeDeCaracteres.h`
— `src/Chaine.c` qui définit entre autres les fonctions déclarées dans `include/Chaine.h`
— `src/chaineTU.c` le programme des tests unitaires des fonctions déclarées dans `include/Chaine.h` ;

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

2 Travail à réaliser

L'ensemble des sources (.c) compilent sans erreur et sans *warning*, le programme `test/chaineTU` est créé sans erreur et sans *warning*, mais aucun tests n'est validé.

Complétez le fichier `Chaine.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/chaineTU`).