ASI3 Mardi 16 Janvier 2018

# Algorithmique avancée et Programmation C

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

L'objectif de cet examen pratique est de développer une file de priorité d'entiers.

## 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 : 'EXAM-ALGO-' (les tirets sont ceux du signe moins) suivi de vos nom et 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 nommerait son répertoire EXAM-ALGO-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 qu'il n'y a aucun accent dans les commentaires que vous auriez pu ajouter** ;

— 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 « 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 File de priorité d'entiers : FDPDE

## 1.1 Analyse

Soit le TAD `FileDePriorite` suivant :

**Nom**: FileDeProriete
**Paramètre**: Element (possédant un ordre total, un élément $e1$ est considéré comme plus prioritaire qu'un élément $e2$ si et seulement si $e1 > e2$)
**Utilise**: **Booleen**, **Naturel**, **NaturelNonNul**
**Opérations**:
- fileDePriorite: $\rightarrow$ FileDeProriete
- estVide: FileDeProriete $\rightarrow$ **Booleen**
- longueur: FileDeProriete $\rightarrow$ **Naturel**
- enfiler: FileDeProriete $\times$ Element $\rightarrow$ FileDeProriete
- défiler: FileDeProriete $\nrightarrow$ FileDeProriete
- obtenirElement: FileDeProriete $\nrightarrow$ Element
- obtenirIemeElement: FileDeProriete $\times$ **NaturelNonNul** $\nrightarrow$ Element

**Sémantiques**:
- fileDePriorite(): permet d'obtenir une file de priorité vide
- estVide(f): permet de savoir si $f$ est vide
- longueur(f): permet d'obtenir le nombre d'éléments de $f$
- enfiler(f,e): permet d'ajouter un élément $e$ à $f$
- defiler(f): permet d'enlever de $f$ l'élément le plus prioritaire
- obtenirElement(f): permet d'obtenir l'élément le plus prioritaire de $f$
- obtenirIemeElement(f,i): permet d'obtenir le $i$ème élément de $f$

**Préconditions**:
- defiler(f): $non(estVide(f))$
- obtenirElement(f): $non(estVide(f))$
- obtenirIemeElement(f,i): $i \leq longueur(f)$

## 1.2 Conception

### 1.2.1 Conception préliminaire

Dans le paradigme de la programmation structurée, les opérations de ce TAD correspondent aux signatures de fonctions et procédures suivantes :

— **fonction** fileDePriorite () **:** FileDeProriete
— **fonction** estVide (f : FileDeProriete) **: Booleen**
— **fonction** longueur (f : FileDeProriete) **: Naturel**

— **procédure** enfiler **(E/S** f : FileDeProriete, **E** element : Element**)**
— **procédure** défiler **(E/S** f : FileDeProriete**)**

⌊**précondition(s)** $non(estVide(f))$

— **fonction** obtenirElement **(**f : FileDeProriete**) :** Element

⌊**précondition(s)** $non(estVide(f))$

— **fonction** obtenirIemeElement **(**f : FileDeProriete, i : **NaturelNonNul) :** Element

⌊**précondition(s)** $i \leq longueur(f)$

À ces fonctions et procédures, nous pouvons ajouter la procédure qui supprime tous les éléments d'une file de priorité :
— **procédure** vider **(E/S** f : FileDeProriete**)**

### 1.2.2 Conception détaillée

Nous avons vu en cours que l'on peut concevoir une file à l'aide de la SDD `ListeChainee`. Nous allons nous en inspirer pour concevoir le type `FileDePrioriteDEntiers` de la façon suivante :

**Type** FileDePrioriteDEntiers **= Structure**
nb : **Naturel**
premierASortir : ListeChainee<Entier>
dernierASortir : ListeChainee<Entier>
**finstructure**

La figure 1 représente une file de priorité possédant trois entiers : 5, 2 et 1.

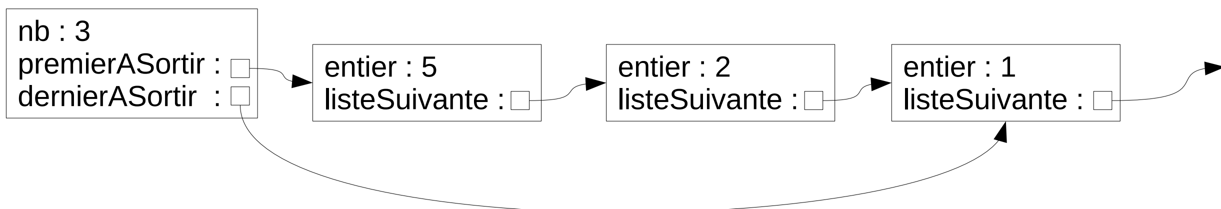


FIGURE 1 – Une file de priorité avec trois entiers

Le seul algorithme réellement difficile à concevoir est celui de la procédure `enfiler` qui insère un entier à la bonne position :

**procédure** enfiler **(E/S** f : FileDePrioriteDEntiers, **E** e : **Entier)**
**debut**
f.nb ← f.nb+1
insererDansListe(f.premierASortir,e)
**si** estVide(f.dernierASortir) **alors**
f.dernierASortir ← f.premierASortir
**sinon**
**si** non estVide(obtenirListeSuivante(f.dernierASortir)) **alors**
f.dernierASortir ← obtenirListeSuivante(f.dernierASortir)
**finsi**
**finsi**
**fin**

Cette procédure utilise la procédure suivante :

**procédure** insererDansListe **(E/S** l : ListeChainee<**Entier**>, e :**Entier)**

**Déclaration** temp : ListeChainee<Entier>

**debut**
  **si** estVide(l) **alors**
    ajouter(l,e)
  **sinon**
    **si** obtenirElement(l)<e **alors**
      ajouter(l,e)
    **sinon**
      temp ← obtenirListeSuivante(l)
      insererDansListe(temp,e)
      fixerListeSuivante(l,temp)
    **finsi**
  **finsi**
**fin**

### 1.3 Développement

Nous décidons d'implanter le type `FileDePrioriteDEntiers` en C en utilisant le type `ListeChaineeDEntiers`. Pour cela, nous disposons des fichiers suivants :

— `include/ListeChaineeDEntiers.h` qui déclare les fonctions permettant d'utiliser une liste chaînée d'entiers ;
— `include/FileDePrioriteDEntiers.h` qui déclare les fonctions permettant d'utiliser une file de priorité d'entiers ;
— `src/ListeChaineeDEntiers.c` qui définit les fonctions déclarées dans `include/ListeChaineeDEntiers.h`
— `src/FileDePrioriteDEntiers.c` qui définit les fonctions déclarées dans `include/FileDePrioriteDEntiers.h`
— `src/fdpdeTU.c` le programme des tests unitaires des fonctions déclarées dans `include/FileDePrioriteDEntiers.h` ;

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

## 2 Travail à réaliser

Complétez le fichier `FileDePrioriteDEntiers.c` de façon à ce que le maximum de tests unitaires fonctionnent (`test/fdpdeTU`).