amU Faculté des sciences Aix Marseille Université

Année 2025-26

Domaine Sciences et Technologies
Master informatique
Fiabilité logicielle : TD 3
Codes UE : SINB34A
Tests en boite noire

1 Calcul somme des n premiers entiers

On donne la spécification suivante : Le programme somme lit deux entiers nommés n et *maxint*, et retourne la somme des n premiers entiers positifs ou nuls (0 est le premier entier) si celle-ci est inférieure ou égale à *maxint* (par définition, on pose $\sum_{i=0}^{n-1} i = 0$ si $n \leq 0$) sinon elle renvoie -1.

Question 1.1 : Déterminer les classes invalides.

Question 1.2 : Déterminer les classes valides.

Question 1.3 : Donner une suite de tests pour la fonction (sous la forme : cas de test, données de test, résultat attendu).

Question 1.4 : Refaire une suite de test en utilisant un arbre de décision.

Question 1.5 : Rajouter les tests correspondant aux cas limites.

2 Triangle

On considère un programme qui doit déterminer le type d'un triangle à partir de la longueur de ses trois côtés. On suppose que les longueurs des côtés sont données dans un fichier texte de nom `filename` et que le programme comporte deux fonctions :

- Une fonction `struct triangle readData(char* filename)` qui renvoie une structure de type triangle (`struct triangle {int a; int b; int c;}`) dont les trois valeurs correspondent aux trois lignes du fichier texte de nom `filename`. Un fichier au bon format doit donc contenir une ligne de texte, chaque ligne correspondant à l'écriture d'un entier positif ou nul. La fonction affectera `-1` à chaque champ de la structure en cas de problème de lecture ou fichier non conforme.
- Une fonction `int typeTriangle(struct triangle t)` qui renvoie le type du triangle `t` dont la longueur des côtés sont les champs entiers `a`, `b`, `c` de la `struct triangle t`. Le type de triangle est codé de la manière suivante :
  - `-1` si les longueurs des côtés ne définissent pas un triangle dont l'aire est strictement positive,
  - `3` si le triangle est équilatéral,
  - `2` s'il est isocèle,
  - `1` s'il est scalène (quelconque).

Question 2.1 : Construire une suite de test pour la fonction `readData` en utilisant un arbre de décision.

Question 2.2 : Construire une suite de test pour la fonction `typeTriangle` en utilisant un arbre de décision.

3 Fichier MS-DOS

Un fichier MS-DOS possède un nom obéissant aux contraintes suivantes :

- le nom est composé d'une première partie constituée de caractères alphanumérique de longueur non nulle et de longueur au plus 8.
- le premier caractère de cette première partie est nécessairement un caractère alphabétique.
- cette première partie est suivie d'un point `.`
- le point est suivi d'une extension obligatoire d'exactement 3 caractères alphabétiques.

Question 3.1 : Identifier tous les concepts intervenant dans la définition d'un nom de fichier valide.

Question 3.2 : Utiliser un arbre de décision pour déterminer une suite de tests pour la fonction de validation d'un nom de fichier.

4 Calendrier grégorien

Le calendrier Grégorien a été introduit en 1582 et est utilisé depuis pour le calendrier usuel. Dans ce calendrier les années bissextiles sont les années multiples de 4 exceptées celles qui sont multiples de 100 sauf si elles sont également multiples de 400. Ainsi 2000 est une année bissextile comme 2020. La fonction `bool est_bissextile(a)` renvoie `true` si a est une année bissextile, `false` sinon.

Question 4.1 : Identifier tous les concepts intervenant dans la définition d'une année bissextile dans le calendrier grégorien.

Question 4.2 : Utiliser un arbre de décision pour déterminer une suite de tests pour la fonction `est_bissextile`.

5 Adresse email

La RFC 2822 décrit ce qu'est une adresse email valide. Pour cet exercice, on utilise une version très simplifiée :

```
addr-spec = local-part "@" domain
local-part = dot-atom
domain = dot-atom

ALPHA = x41-5A / %x61-7A #lettres (A-Z, a-z)
DIGIT = %x30-39 #chiffres (0-9)
atext = ALPHA / DIGIT / "*" / "-" / "+" / "_"
atom = 1*atext # une séquence d'un ou plusieurs caractères atext
dot-atom = 1*atext *("." 1*atext)
```

Notation : Pour indiquer la répétition d'un élément, on utilise la notation `a*b(élément)`. Le paramètre optionnel `a` donne le nombre minimal d'éléments à inclure (par défaut 0). Le paramètre optionnel `b` donne le nombre maximal d'éléments à inclure (par défaut l'infini).

Question 5.1 : Identifier tous les concepts intervenant dans la définition d'une adresse email valide.

Question 5.2 : Utiliser un arbre de décision pour déterminer une suite de tests pour la fonction de validation d'une adresse email.

6 Date courante

Certains systèmes informatiques peuvent calculer la date courante dans le calendrier grégorien à partir de leur date et heure d'installation, représentées par 4 nombres `day`, `month`, `year`, `seconds` avec `seconds` donnée en secondes écoulées depuis 0h00s, et du nombre `elapsedTime` représentant le temps en seconde écoulé depuis cette date. La fonction `String date(int day, int month, int year, int seconds, int elapsedTime)` renvoie la date courante calculée à partir de la date d'installation donnée par les 4 premiers arguments et du temps écoulé depuis donné par le cinquième argument sous la forme `jj/mm/aaaa-hh:mm:ss` par exemple `30/01/2020-20:50:45` comme une chaine de caractère. Toutes les dates correspondent au calendrier grégorien. Il s'agit d'écrire une suite de tests pour cette fonction.

Question 6.1 : Identifier les concepts importants pour les calculs effectués par la fonction.

Question 6.2 : Identifier les classes de données invalides et construire un arbre de décision permettant de définir les cas de tests correspondants.

Question 6.3 : Construire une suite de tests pour les données valides en utilisant un arbre de décision un arbre de décision.

7 Commandes système

Une commande système peut avoir 4 options possibles qui peuvent se cumuler comme la commande `ls` peut avoir les options `-a`, `-r`, ... et on peut invoquer `ls -a -r`. L'ordre des options n'a pas d'importance, c'est-à-dire que `ls -a -r` et `ls -r -a` sont identiques. On veut écrire une suite de tests permettant de vérifier la bonne programmation de la commande.

Question 7.1 : Quel est le nombre total de manières distinctes d'invoquer la commande avec des options ? Est-il raisonnable de tester toutes les possibilités si on a 10 options et non pas 4 ?

Question 7.2 : On veut tester pour chaque paire d'option `op1`, `op2` toutes les possibilités d'invoquer la commande : avec les deux options, sans aucune des deux, avec une seule d'elle ce qui donne 4 possibilités pour chaque paire. Calculer le nombre total de cas à tester.

Question 7.3 : Donner une suite de tests permettant de tester toutes les paires d'options. Noter qu'un cas de test peut invoquer d'autres options en même temps que `op1` ou `op2`. Vérifier que le nombre de données de tests est inférieur au nombre total de manière d'invoquer la commande.

8 Règlement d'obtention de diplôme

On donne le règlement d'obtention de diplôme master qui comporte 10 modules $M_1, \ldots, M_{10}$ avec $M_1, \ldots, M_4$ modules prioritaires : si un étudiant a tous les modules avec au moins 10/20 à chaque module et au moins 12/20 de moyenne générale, il obtient son diplôme. Sinon, il peut passer des rattrapages s'il a au moins 8/20 de moyenne générale, sinon il redouble. De plus, certains modules sont prioritaires, et moins de 8/20 de moyenne à un de ces modules empêche de passer les rattrapages et fait redoubler. Un étudiant qui réussit ses rattrapages obtient son diplôme sinon il n'obtient pas son diplôme et redouble.

Question 8.1 : Des informations pouvant être nécessaires manquent. Lesquelles ?

Question 8.2 : Définir les causes et les effets contenus dans la spécification.

Question 8.3 : Dessiner le graphe Cause/Effet.

Question 8.4 : Donner un système de contraintes propositionnelles décrivant le graphe.

Question 8.5 : Définir une suite de tests pour cette application.