ASI3 Mardi 18 novembre 2014

Algorithmique et bases de la programmation

Durée : *1h30*
Documents autorisés : **AUCUN**

Remarques :
— Veuillez lire attentivement les questions avant de répondre.
— Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
— Rendez une copie propre.
— N'utilisez pas de crayon à papier sur votre copie.

1 QCM (5 points)

Répondez au qcm ci joint (à rendre avec votre copie). La note de chaque question peut varier de -1 à $+1$. La note à l'exercice (0 au minimum) est la moyenne des notes de chaque question multipliée par le nombre de points de l'exercice.

Soit B le nombre de bonnes réponses à une question et soit M le nombre de mauvaises réponses à cette même question ($B + M$ est égale au nombre de réponses à la question). Chaque bonne réponse cochée rapporte $1/B$ points et chaque mauvaise réponse $-1/M$ points.

2 Anagramme (4 points)

Deux mots sont des anagrammes si l'un est une permutation des lettres de l'autre. Par exemple les mots suivants sont des anagrammes :
— aimer et maire
— chien et niche
— ...

Dans cet exercice nous représentons les mots par des chaînes de caractères et par définition on considère que deux chaînes vides sont des anagrammes.

On considère que l'on possède les fonctions suivantes :
— **fonction** longueur (chaine : **Chaine de caracteres) : Naturel**
retourne 0 si `chaine`=""
— **fonction** iemeCaractere (chaine : **Chaine de caracteres**, indice : **Naturel) : Caractere**
⌊**précondition(s)** $1 \leq$ `indice` et `indice` $\leq$ `longueur(chaine)`
— **fonction** positionCaractere (chaine : **Chaine de caracteres**, car : **Caractere) : Naturel**
retourne 0 si `car` non présent dans `chaine`
— **fonction** supprimerCaractere (chaine : **Chaine de caracteres**, position : **Naturel) : Chaine de caracteres**
⌊**précondition(s)** $1 \leq$ `position` et `position` $\leq$ `longueur(chaine)`

Proposez le corps de la fonction récursive suivante qui permet de savoir si deux mots sont des anagrammes :

fonction sontDesAnagrammes (mot1,mot2 : **Chaine de caracteres) : Booleen**

Solution proposée :

fonction sontDesAnagrammes (mot1,mot2) **: Booleen**
Déclaration pos : **Naturel**
debut
si mot1="" et mot2="" **alors**
retourner Vrai
sinon
pos ← positionCaractere(mot2,iemeCaractere(mot1,1))
si pos>0 **alors**
retourner sontDesAnagrammes(supprimerCaractere(mot1,1),
supprimerCaractere(mot2,pos))
sinon

retourner Faux
finsi
finsi
fin

3 Problème : calculer la valeur d'une expression booléenne (11 points)

3.1 TAD Dictionnaire (1 point)

Rappelez le TAD `Dictionnaire` vu en cours (sans les parties axiomes et sémantiques).

Solution proposée :

Nom:	Dictionnaire	
Paramètre:	Clef,Valeur	
Utilise:	**Booleen**,Liste	
Opérations:	dictionnaire:	$\rightarrow$ Dictionnaire
	ajouter:	Dictionnaire $\times$ Clef $\times$ Valeur $\rightarrow$ Dictionnaire
	retirer:	Dictionnaire $\times$ Clef $\rightarrow$ Dictionnaire
	estPresent:	Dictionnaire $\times$ Clef $\rightarrow$ **Booleen**
	obtenirValeur:	Dictionnaire $\times$ Clef $\nrightarrow$ Valeur
	obtenirClefs:	Dictionnaire $\rightarrow$ Liste<Clef>
	obtenirValeurs:	Dictionnaire $\rightarrow$ Liste<Valeur>
Préconditions:	obtenirValeur(d,c):	$estPresent(d, c)$

3.2 Évaluation d'une expression booléenne (10 points)

On considère qu'une expression booléenne est soit une opération unaire (opérateur `non`) soit une opération binaire (opérateur `ou` et `et`). L'opérande d'une opération booléenne est soit une constante (valeur `vrai` ou `faux`), soit une variable, soit une expression booléenne entre parenthèses.

On suppose que :
- la valeur booléenne d'une variable est donnée par un dictionnaire dont la clé est l'identifiant de la variable ;
- l'on possède les types énumérés :
  - **Type** OperateurUnaire = {not}
  - **Type** OperateurBinaire = {and,or}

L'objectif est de concevoir l'opération `evaluerExpressionBooleenne` qui calcule la valeur d'une expression booléenne représentée par une chaîne de caractères (par exemple `"a et (c ou (vrai et d))"`). Un booléen permet de savoir si cette chaîne représente ou pas une expression booléenne valide. L'opération `evaluerExpressionBooleenne` est insensible à la casse. Les constantes, opérateurs, variables sont séparés par des espaces. Enfin :
- les constantes sont identifiées par les chaînes de caractères `"vrai"` et `"faux"` ;
- les opérateurs booléens sont identifiés par les chaînes de caractères `"non"`, `"et"` et `"ou"` ;
- les variables sont identifiées par une suite de caractères parmi les caractères suivants : `'a'` à `'z'`, `'0'` à `'9'` et `'_'`. Il y a au moins un caractère et le premier sera obligatoirement une lettre ou un `'_'`.

3.2.1 Analyse (2 points)

Complétez l'analyse descendante donnée par la figure 1. Pourquoi y a-t-il un lien de dépendance entre l'opération `reconnaitreOperande` et `reconnaitreExpressionBooleenne` ?

Solution proposée :

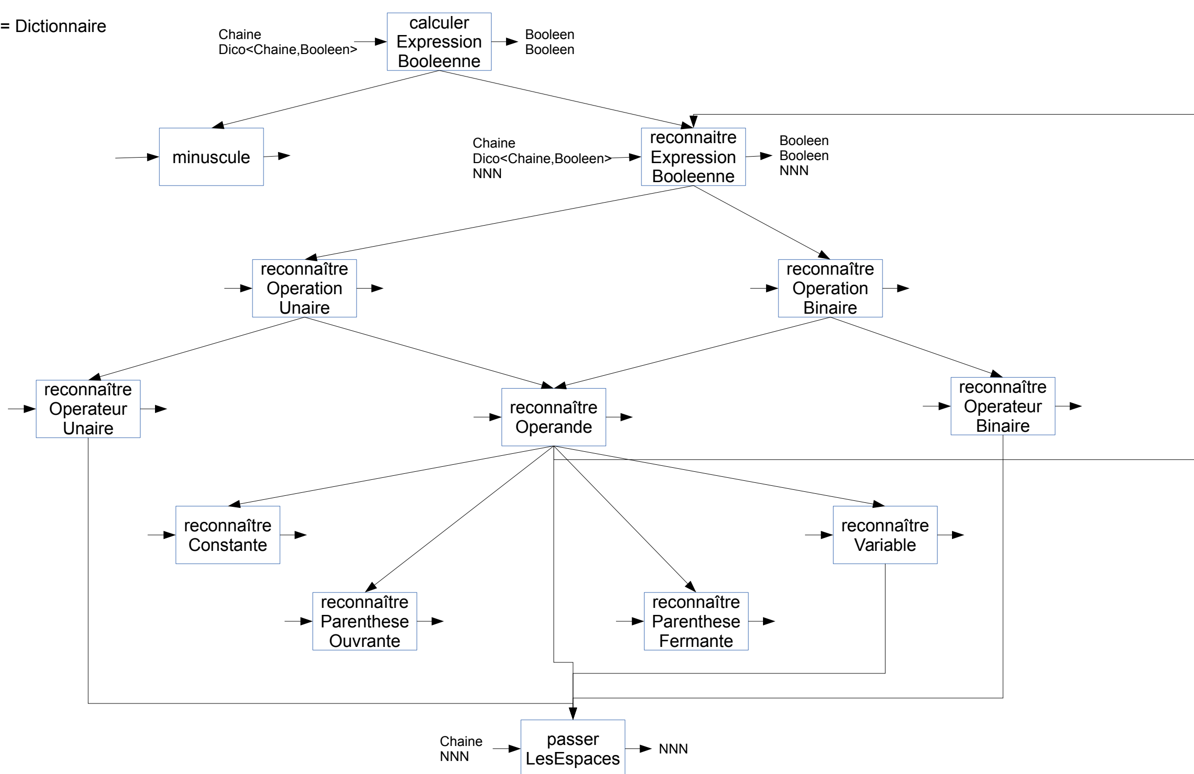
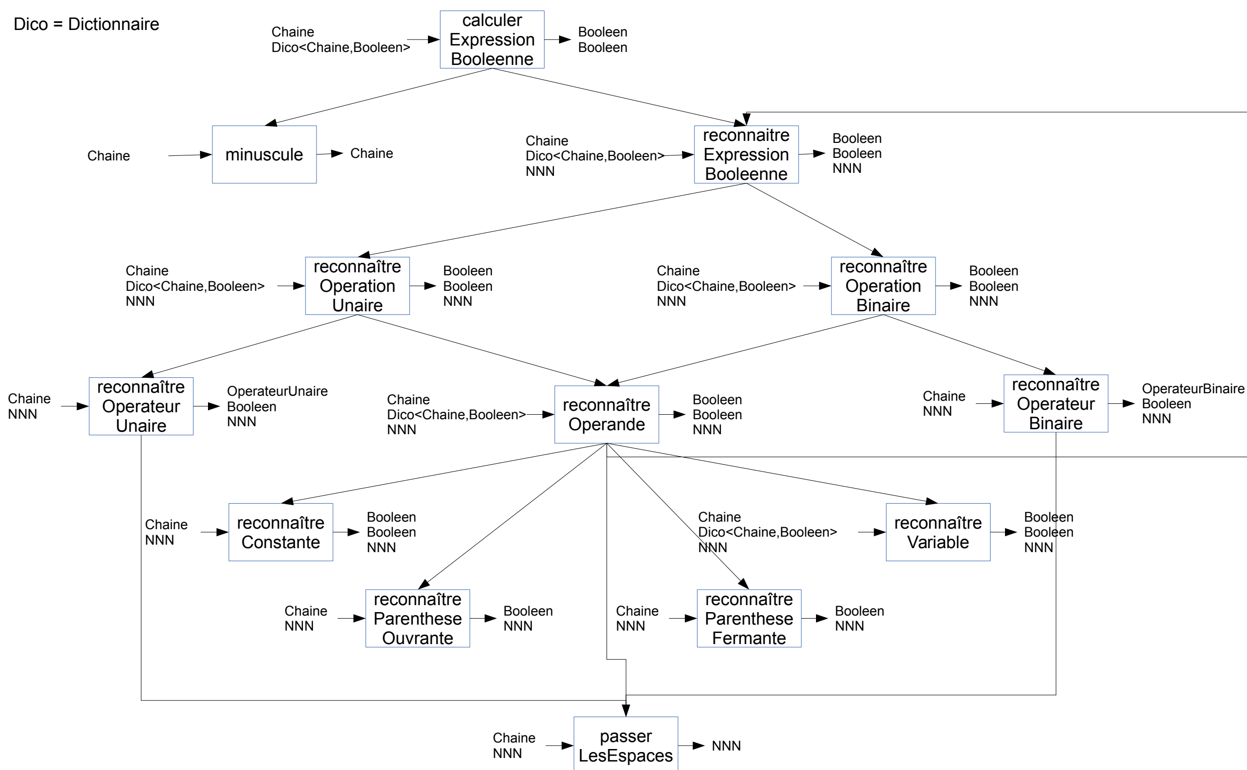


Figure 1 – Analyse descendante



3.2.2 Conception préliminaire (1 point)

Donnez les signatures des procédures ou fonctions des opérations `reconnaitreOperationBinaire`, `reconnaitreOperande`, `reconnaitreConstante`, `reconnaitreParentheseOuvrante`, `reconnaitreParentheseFermante`, `reconnaitreVariable`,

`passerLesEspaces`.

Solution proposée :

- **procédure** reconnaitreOperationBinaire **(E** ch : **Chaine de caracteres**, mem : Dico<**Chaine de caracteres**,**Booleen**>, **E/S** pos :**NaturelNonNul**, **S** valeur : **Booleen**, valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)
- **procédure** reconnaitreOperande **(E** ch : **Chaine de caracteres**, mem : Dico<**Chaine de caracteres**,**Booleen**>, **E/S** pos :**NaturelNonNul**, **S** valeur : **Booleen**, valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)
- **procédure** reconnaitreConstante **(E** ch : **Chaine de caracteres**, **E/S** pos :**NaturelNonNul**, **S** valeur : **Booleen**, valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)
- **procédure** reconnaitreVariable **(E** ch : **Chaine de caracteres**, mem : Dico<**Chaine de caracteres**,**Booleen**>, **E/S** pos :**NaturelNonNul**, **S** valeur : **Booleen**, valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)
- **procédure** reconnaitreParentheseOuvrante **(E** ch : **Chaine de caracteres**, **E/S** pos :**NaturelNonNul**, **S** valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)
- **procédure** reconnaitreParentheseFermante **(E** ch : **Chaine de caracteres**, **E/S** pos :**NaturelNonNul**, **S** valide : **Booleen)**

  ⌊**précondition(s)** pos<longueur(ch)

3.2.3 Conception détaillée (7 points)

Donnez les algorithmes des procédures ou fonctions des opérations :

1. `reconnaitreOperationBinaire`
2. `reconnaitreOperande`
3. `reconnaitreVariable`

Solution proposée :

procédure reconnaitreOperationBinaire **(E** ch : **Chaine de caracteres**, mem : Dico<**Chaine de caracteres**,**Booleen**>, **E/S** pos :**NaturelNonNul**, **S** valeur : **Booleen**, valide : **Booleen)**

⌊**précondition(s)** pos<longueur(ch)

```
Déclaration  posDebut : NaturelNonNul
             valeurG, valeurD : Booleen
             op : OperateurBinaire
debut
   posDebut ← pos
   reconnaitreOperande(ch,pos,valeurG,valide)
   si valide alors
      reconnaitreOperateurBinaire(ch,pos,op,valide)
      si valide alors
         reconnaitreOperande(ch,pos,valeurD,valide)
         si valide alors
            cas où op vaut
               and:
                  valeur ← valeurG et ValeurD
               or:
                  valeur ← valeurG ou ValeurD
            fincas
         finsi
      finsi
   finsi
   si non valide alors
      pos ← posDebut
```

```
    finsi
fin
procédure reconnaitreOperande (E ch : Chaine de caracteres, mem : Dico<Chaine de caracteres,Booleen>,
E/S pos :NaturelNonNul, S valeur : Booleen, valide : Booleen)
    ⌊précondition(s)   pos<longueur(ch)
debut
    posDebut ← pos
    passerLesEspaces(ch,pos)
    reconnaitreConstante(ch,pos,valeur,valide)
    si non valide alors
        reconnaitreVariable(ch,mem,pos,valeur,valide)
        si non valide alors
            reconnaitreParentheseOuvrante(ch,pos,valide)
            si valide alors
                passerLesEspaces(ch,pos)
                reconnaitreExpressionBooleenne(ch,pos,valeur,valide)
                si valide alors
                    passerLesEspaces(ch,pos)
                    reconnaitreParentheseFermante(ch,pos,valide)
                finsi
            finsi
        finsi
    finsi
    si non valide alors
        pos ← posDebut
    finsi
fin
procédure reconnaitreVariable (E ch : Chaine de caracteres, mem : Dico<Chaine de caracteres,Booleen>,
E/S pos :NaturelNonNul, S valeur : Booleen, valide : Booleen)
    ⌊précondition(s)   pos<longueur(ch)
debut
    posDebut ← pos
    si non((iemeCaractere(ch,pos)≥'a' et iemeCaractere(ch,pos)≤'z') ou iemeCaractere(ch,pos)='_') alors
        valide ← FAUX
    sinon
        id ← caractereEnChaine(iemeCaractere(ch,pos))
        pos ← pos+1
        tant que (iemeCaractere(ch,pos)≥'a' et iemeCaractere(ch,pos)≤'z') ou (iemeCaractere(ch,pos)≥'0' et
        iemeCaractere(ch,pos)≤'9') ou iemeCaractere(ch,pos)='_' faire
            id ← id+caractereEnChaine(iemeCaractere(ch,pos))
            pos ← pos+1
        fintantque
        si estPresent(dico,id) alors
            valeur ← obtenirValeur(dico,id)
        sinon
            valide ← FAUX
        finsi
    finsi
    si non valide alors
        pos ← posDebut
    finsi
fin
```