Saisies d'un tableau [tb03] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 18 mai 2018

Table des matières

Python - Saisies d'un tableau (Solution)

Mots-Clés Tableau unidimensionnel ■
Utilise Définitions et notations, Tableaux et paramètres, Parcours complet ■
Difficulté ● ● ○

Objectif
Cet exercice met en évidence deux modes de remplissage d'un tableau :

- Cas où l'utilisateur peut donner *a priori* le nombre de valeurs à traiter.
- Cas où l'utilisateur saisit les valeurs jusqu'à ce qu'il n'y en ait plus et donne une valeur fictive pour annoncer la fin de saisie.

1 Saisies d'un tableau / pgsaisies

1.1 Énoncé

Avant tout traitement d'une série de données contenue dans un tableau, il faut en effectuer la saisie. Le premier problème :

- Demande à l'utilisateur combien il y a de valeurs dans la série.
- Remplit un tableau en saisissant les valeurs de cette série.

Quant au deuxième, il :

- Remplit un tableau en saisissant les valeurs d'une série, la fin de saisie étant annoncée par une valeur spéciale *sentinelle* (qui ne fait partie de la série).

Dans ce deuxième problème, nous étudierons plusieurs versions car il faut gérer les questions de débordement.

Types des données
Pour fixer un cadre, nous prenons des valeurs entières.

Définitions Python

```
TMAX = ...
```

1.2 Fonction saisirTab (saisie d'un tableau)

Écrivez le **profil** d'une fonction `saisirTab(t)` qui effectue la saisie du nombre de valeurs dans un entier, puis saisit les valeurs dans un `ITableau t` et renvoie le nombre de valeurs saisies.

Solution Paramètres

Sortants : Un `ITableau t`

Résultat de la fonction : Un entier

Saisie du nombre de valeurs
Le nombre de valeurs doit être compris entre 1 (inutile d'appeler cette fonction pour saisir un tableau vide) et `TMAX` (taille maximale d'un `ITableau`). On utilise donc une répétitive `TantQue` ou `Répéter` selon que l'on souhaite ou non avertir l'utilisateur d'une erreur de saisie.

Écrivez la saisie contrainte du nombre de valeurs entre 1 et `TMAX`.
Affichez l'invite (où `[x]` désigne le contenu de `x`) :

```
Nombre de valeurs dans [1..[TMAX]]?
```

Saisie des valeurs
Soit `n` le nombre de valeurs à saisir. Il faut répéter `n` fois la saisie d'une valeur (parcours complet). Par conséquent, le remplissage du tableau sera piloté par une boucle `Pour`.

Écrivez la saisie des valeurs entières dans `t`.
Affichez une invite spécifiant le rang de l'élément à saisir :

```
t[..]?
```

Validez votre fonction avec la solution.

Solution Python @[UtilsTB.py]

```python
def saisirTab(t):
    """  Saisie de valeurs dans un ITableau

     :param t: un ITableau
     :return: le nombre de valeurs saisies dans [1..TMAX]
    """
    TMAX = len(t)
    n = -1
    while not (1 <= n and n <= TMAX):
        print("Nombre de valeurs dans [1..", TMAX, "]? ", sep="", end="")
        n = int(input())
    for j in range(0, n):
        print("t[", j, "]? ", sep="", end="")
        t[j] = int(input())
    return n
```

1.3 Procédure afficherTab (affichage d'un tableau)

Écrivez le **profil** d'une procédure `afficherTab(t,n)` qui affiche les `n` premières valeurs d'un `ITableau t`.

Affichage des valeurs
L'affichage de `t` est un parcours complet des `n` valeurs.
Il est donc piloté par une boucle `Pour`.

Écrivez le corps de la procédure. Affichez les valeurs à la queue-leu-leu séparés par un espace, le tout entre crochet. Exemple :

```
[45 54... -27]
```

Validez votre procédure avec la solution.

Solution Python @[UtilsTB.py]

```
def afficherTab(t, n):
    """  Affichage d'un ITableau

     :param t: un ITableau
     :param n: nombre de valeurs dans [0..TMAX]
    """
    print("[", end="")
    for j in range(0, n):
        print(t[j], " ", sep="", end="")
    print("]")
```

1.4 Vérification et jeux d'essais (1)

Définissez la constante `TMAX=100`.

Écrivez un script qui demande et stocke des entiers dans un `ITableau` puis l'affiche.

Vérification de la saisie : Il faut vérifier que la boucle de saisie du nombre de valeurs joue correctement son rôle au sein de la procédure `saisirTab`. Donc,

Testez en proposant :

- D'abord un nombre négatif ou nul.
- Puis un nombre strictement supérieur à `TMAX`.
- Puis un entier compris entre 1 et `TMAX`. On le choisira relativement petit (moins de 10) car il faudra faire des saisies (sans erreurs dans le cas d'un programme).

1.5 Fonctions saisirTabSentinelle

Dans ce problème, le nombre de valeurs n'est plus saisi directement.

Écrivez le **profil** d'une fonction `saisirTabSentinelle1(t)` qui saisit des valeurs dans un `ITableau t` et renvoie le nombre de valeurs saisies.

Solution Paramètres

Sortants : Un `ITableau t`

Résultat de la fonction : Un entier

Initialisations
Dans cette fonction, la fin de saisie est indiquée par une valeur spéciale : la **sentinelle**. Elle est donc demandée dans la fonction. Il faut aussi initialiser le nombre de valeurs à zéro car il n'est plus saisi directement : il sera incrémenté à chaque nouvelle saisie.

Écrivez ensuite :

- La déclaration puis la saisie de la variable entière de la sentinelle.
  Affichez l'invite :

  ```
  Valeur de la sentinelle?
  ```

- L'initialisation de `n` (le résultat) à 0.

Boucle de saisie

La valeur de la *sentinelle* **ne doit pas** être rangée dans le tableau. Il faut donc déclarer une variable entière, dédiée à la saisie, et dont la valeur sera affectée à l'élément du tableau sauf lorsqu'elle vaudra la *sentinelle*. Nommons `valeur` cette variable.

Pour construire l'algorithme, considérons que l'utilisateur donne une série **non vide** de valeurs et qu'il termine par la valeur `sentinelle`. Nous examinerons les cas limites par la suite.

Le nombre de répétitions étant **inconnu**, ceci **exclut** les boucles `Pour` et `Itérer`. Si on utilise une boucle `Répéter`, on va perdre le bénéfice de la variable de saisie puisque les deux instructions ci-après vont se suivre dans le corps de la boucle (sauf si on introduit une conditionnelle), alors que la valeur `sentinelle` ne doit pas être rangé dans le tableau.

```
Saisir(valeur)
t[ix] <- valeur
```

Il ne reste donc que la mise en oeuvre avec une boucle `TantQue` sur le modèle [1] :

```
obtenir la première valeur
TantQue la valeur n'est pas la sentinelle Faire
  traiter la valeur
  obtenir la valeur suivante
FinTantQue
```

Le traitement de la valeur comprend la mise-à-jour du nombre de valeurs (une de plus) **et** son rangement dans l'élément de tableau correspondant, à savoir : rangement dans l'élément de tableau d'indice `n` puis incrémentation de `n` (Rappel : Les indices de tableaux commencent à zéro en programmation).

Écrivez enfin :

- La déclaration de la variable entière `valeur`.
- Puis la saisie des valeurs dans le tableau `t`.
  Affichez les invites :

  ```
  Premiere valeur?
  Valeur suivante?
  ```

Validez votre fonction avec la solution.

1. Nous renvoyons le lecteur sur les répétitives.

Solution C++ @[pgtsaisies.cpp]

```
/**
  Saisie de valeurs dans un ITableau
  @param[out] t - un ITableau
  @return le nombre de valeurs saisies
*/

int saisirTabSentinelle1(ITableau& t)
{
  int sentinelle;
  cout<<"Valeur de la sentinelle? ";
  cin>>sentinelle;
  int n = 0;
  int valeur;
  cout<<"Premiere valeur? ";
  cin>>valeur;
  while (valeur != sentinelle)
  {
    t[n] = valeur;
    ++n;
    cout<<"Valeur suivante? ";
    cin>>valeur;
  }
  return n;
}
```

Solution Python @[pgtsaisies.py]

```
def saisirTabSentinelle1(t):
    """  Saisie de valeurs dans un ITableau

     :param t: un ITableau
     :return: le nombre de valeurs saisies
    """
    sentinelle = int(input("Valeur de la sentinelle? "))
    n = 0
    valeur = int(input("Premiere valeur? "))
    while valeur != sentinelle:
        t[n] = valeur
        n += 1
        valeur = int(input("Valeur suivante? "))
    return n
```

Étude du cas limite : Série vide

Dans le cas où l'utilisateur donne la valeur de la sentinelle en première saisie, la boucle `TantQue` n'est pas exécutée et on obtient bien une série vide puisque `n` vaut 0.

Étude du cas limite : Débordement de tableau

Il faut maintenant éliminé le risque du **débordement** de tableau. En effet, si l'utilisateur essaie de saisir une `TMAX`è valeur, il y aura une erreur au moment de l'affectation dans le tableau car l'élément n'existe pas. Il faut donc **interrompre de force** le traitement

avant qu'il ne tente de la ranger dans le tableau. De ce fait, la boucle `TantQue` aura une double condition d'entrée ce qui la transforme en :

```
TantQue (valeur <> sentinelle Et n < TMAX) Faire
  traiter la valeur
  obtenir la valeur suivante
FinTantQue
```

Mais cette version n'est pas satisfaisante. En effet, examinons la séquence juste après l'entrée dans la boucle avec `n=TMAX-1`. Dans ce cas, `n` passe à `TMAX` après le traitement de la valeur et l'utilisateur est sollicité pour une nouvelle saisie bien que celle-ci ne provoquera pas d'erreur de débordement car il n'y aura pas de nouvelle entrée dans la boucle. Cependant cette demande de dernière saisie n'a pas de sens et ne doit pas avoir lieu. A la place, l'utilisateur doit être informé qu'il a épuisé la capacité de rangement offerte par le programme.

Copiez/collez la fonction `saisirTabSentinelle1` en la fonction `saisirTabSentinelle2` (mêmes paramètres). Modifiez-la de sorte qu'elle traite le débordement du tableau en insérant une conditionnelle **dans** la boucle `TantQue` qui filtre la saisie de la `TMAX`è valeur.

Validez votre fonction avec la solution.

Solution C++ @[pgtsaisies.cpp]

```
/**
  Saisie de valeurs dans un ITableau
  @param[out] t - un ITableau
  @return le nombre de valeurs saisies dans [0..TMAX]
*/

int saisirTabSentinelle2(ITableau& t)
{
  int sentinelle;
  cout<<"Valeur de la sentinelle? ";
  cin>>sentinelle;
  int n = 0;
  int valeur;
  cout<<"Premiere valeur? ";
  cin>>valeur;
  while (valeur != sentinelle && n < TMAX)
  {
    t[n] = valeur;
    ++n;
    if (n == TMAX)
    {
      cout<<"Plus de place"<<endl;
    }
    else
    {
      cout<<"Valeur suivante? ";
      cin>>valeur;
    }
  }
```

```
  if (n == 0)
  {
    cout<<"Serie vide"<<endl;
  }
  return n;
}
```

Solution Python @[pgtsaisies.py]

```
def saisirTabSentinelle2(t):
    """  Saisie de valeurs dans un ITableau

     :param t: un ITableau
     :return: le nombre de valeurs saisies dans [0..TMAX]
    """
    sentinelle = int(input("Valeur de la sentinelle? "))
    n = 0
    valeur = int(input("Premiere valeur? "))
    while valeur != sentinelle and n < TMAX:
        t[n] = valeur
        n += 1
        if n == TMAX:
            print("Plus de place")
        else:
            valeur = int(input("Valeur suivante? "))
    if n == 0:
        print("Serie vide")
    return n
```

Amélioration de la boucle

La modification que nous venons de proposer corrige le défaut décrit ci-dessus mais présente l'inconvénient d'introduire une conditionnelle, donc un calcul booléen, à chaque passage dans la boucle pour filtrer une situation qui a peu de chance de se produire. C'est du « gaspillage de calcul ».

Pour éviter ceci, on peut reprendre la première version et interrompre la saisie après le `(TMAX-1)`è élément au lieu du `TMAX`è. Il faut alors introduire une conditionnelle après la répétitive pour autoriser l'éventuelle saisie de la `TMAX`è valeur. Contrairement à la conditionnelle introduite dans la boucle, celle-ci ne s'exécutera qu'une seule fois.

Copiez/collez la fonction `saisirTabSentinelle1` en la procédure `saisirTabSentinelle` (mêmes paramètres). Modifiez-la de sorte qu'elle traite le débordement du tableau en insérant une conditionnelle **après** la boucle `TantQue` qui enregistre dans le tableau l'éventuelle `TMAX`è valeur.

Validez votre fonction avec la solution.

Solution Python @[UtilsTB.py]

```python
def saisirTabSentinelle(t):
    """  Saisie de valeurs dans un ITableau

     :param t: un ITableau
     :return: le nombre de valeurs saisies dans [0..TMAX]
    """
    sentinelle = int(input("Valeur de la sentinelle? "))
    tmax1 = len(t) - 1
    n = 0
    valeur = int(input("Premiere valeur? "))
    while valeur != sentinelle and n < tmax1:
        t[n] = valeur
        n += 1
        valeur = int(input("Valeur suivante? "))
    if valeur != sentinelle:
        t[n] = valeur
        n += 1
        print("Plus de place")
    if n == 0:
        print("Serie vide")
    return n
```

1.6 Vérification et jeux d'essais (2)

Dans votre script, modifiez l'appel de `saisirTab` en celui de `saisirTabSentinelle`.

Vérification 1 : Testez avec une série normale, c.-à-d. comportant un nombre de valeurs inférieur à la taille du tableau.

Vérification 2 : Testez avec la série vide en entrant directement la valeur de la sentinelle.

Vérification 3 : Fixez la constante `TMAX=5` puis testez avec une série complète qui remplit complètement le tableau.

Commentaires

Si l'utilisateur fournit exactement `TMAX` valeurs, la saisie s'arrête d'elle-même et le message de « Capacité atteinte » s'affiche.

2 Que retenir de cet exercice?

Les **tableaux** (variables structurées) et les **répétitives** (instructions structurées) sont faits pour « travailler » ensemble. En effet, la plupart du temps tous les éléments non vides d'un tableau, qui sont de même type, sont destinés à être traité de la même manière. C'est pourquoi la mise en place d'un tableau dans un algorithme s'accompagne le plus souvent de parcours de celui-ci à l'aide de répétitives, soit pour le remplir, soit pour l'exploiter.

Un algorithme plus long n'est pas forcément plus coûteux en temps de calcul. Nous en avions un exemple dans les versions 2 et 3 de la procédure `saisirTabSS`. La version 2, plus courte, génère autant d'exécutions de conditionnelles qu'il y a de passages dans la boucle. La version 3, plus longue, ne génère qu'une seule exécution de cette conditionnelle.

Chaque fois qu'un tableau est rempli sans connaissance préalable du nombre de valeurs qui vont y être affectées, il y a risque de débordement puisque les tableaux sont des **variables statiques** qui ne peuvent être redimensionnées en cours d'exécution [2]. Deux types de solutions peuvent être apportés : contrôler le débordement comme nous l'avons fait dans cet exercice ou... ne pas utiliser de tableaux mais des **variables dynamiques** comme, par exemple, des **vecteurs** ou des **listes chaînées**.

Nous avons utilisé exactement le même modèle d'algorithme pour contrôler l'arrêt de la saisie d'une série de valeurs : l'utilisation d'une répétitive `TantQue` qui s'arrête après saisie d'une sentinelle.

3 Références générales

Comprend [Tartier-AL1 :c8 :ex30] ■

2. Les langages de programmation autorisent le redimensionnement des tableaux qui ne sont plus tout à fait des tableaux au sens algorithmique du terme.