



CNED

B1.2-PARTIE-I-SQ1-S3



MINISTÈRE
DE L'ÉDUCATION
NATIONALE, DE
L'ENSEIGNEMENT
SUPÉRIEUR ET DE
LA RECHERCHE

www.cned.fr

Les cours du CNED sont strictement réservés à l'usage privé de leurs destinataires et ne sont pas destinés à une utilisation collective. Les personnes qui s'en serviraient pour d'autres usages, qui en feraient une reproduction intégrale ou partielle, une traduction sans le consentement du CNED, s'exposeraient à des poursuites judiciaires et aux sanctions pénales prévues par le Code de la propriété intellectuelle. Les reproductions par reprographie de livres et de périodiques protégés contenues dans cet ouvrage sont effectuées par le CNED avec l'autorisation du Centre français d'exploitation du droit de copie (20, rue des Grands Augustins, 75006 Paris)

CNED, BP 60200, 86980 Futuroscope Chasseneuil Cedex, France - © CNED

Table des matières

1	ACCUEIL	3
1.1	OBJECTIFS.....	3
2	SITUATION PROFESSIONNELLE	4
2.1	VOTRE TÂCHE	4
3	PARTIE 1 - APPORTS MÉTHODOLOGIQUES.....	5
3.1	CE QU'IL FAUT SAVOIR	5
3.2	EXEMPLE DE TABLEAUX	5
4	PARTIE 2 - APPLICATION	8
4.1	APPLICATION	8
4.2	VOTRE TÂCHE	9
4.3	ETAPES À SUIVRE	10
4.4	CORRECTION	10
5	EVALUER SES ACQUIS	10
5.1	EVALUER SES ACQUIS : LES TABLEAUX.....	11
5.2	EVALUER SES ACQUIS : USER STORY	11
6	COMPLÉTER LES SAVOIRS	12
6.1	COMPLÉTER LES SAVOIRS.....	12
7	SYNTHÈSE	12
7.1	SYNTHÈSE DE LA SÉANCE	12
8	GLOSSAIRE.....	13
9	DOCUMENTATION	17
9.1	EXEMPLE DE TABLEAUX	17
9.2	SECTION.....	17
9.3	SECTION.....	17
10	VERSION IMPRIMABLE	17

1 Accueil

1.1 Objectifs



Image illustrative d'un développeur devant un écran

1.1.1 Rappel de la mission

Votre travail consiste à rechercher l'origine et les causes de différents bugs dans les applications informatiques de votre client. Ces applications sont développées dans un langage procédural.

1.1.2 Ce que nous allons aborder

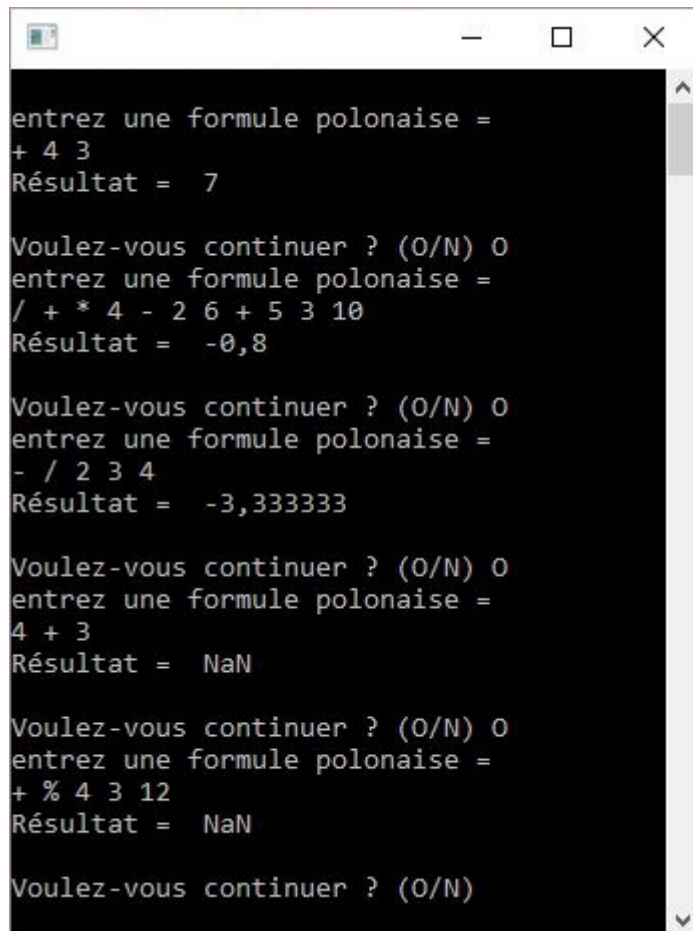
- Comprendre le fonctionnement et l'intérêt des [tableaux](#) en programmation.
- Répondre à une demande formulée dans un récit utilisateur ([user story](#))
- Exploiter un [jeu de données](#) pour tester une fonctionnalité.

1.1.3 Temps de travail indicatif

2 heures

2 Situation professionnelle

2.1 Votre tâche



```
entrez une formule polonaise =  
+ 4 3  
Résultat = 7  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
/ + * 4 - 2 6 + 5 3 10  
Résultat = -0,8  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
- / 2 3 4  
Résultat = -3,333333  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
4 + 3  
Résultat = NaN  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
+ % 4 3 12  
Résultat = NaN  
  
Voulez-vous continuer ? (O/N)
```

[Rédigez votre texte alternatif]

2.1.1 Situation professionnelle

2.1.1.1 Présentation du contexte :

L'entreprise cliente veut construire un bouquet d'applications de tests de connaissances en mathématiques de plusieurs niveaux. Elle a déjà fait appel à votre société pour divers développements, afin de fournir ce bouquet.

2.1.1.2 Narrative du récit utilisateur (user story) :

"En tant que responsable de la chaîne de production d'applications pédagogiques, je veux disposer d'une [fonction](#) qui permet de réaliser le calcul d'une formule en notation polonaise, afin de l'intégrer dans le bouquet d'applications de test de connaissances mathématiques de plusieurs niveaux."
La [user story](#) complète, intégrant le [jeu de données](#), est téléchargeable [user_story.pdf](#).

2.1.1.3 Présentation de l'application test fournie :

Elle permet de saisir plusieurs formules de calculs, avec la notation "Polonaise", puis d'afficher le résultat du calcul, après avoir appelé la fonction qui doit être écrite. Elle doit récupérer de la fonction et afficher NaN (Not a Number) si la formule n'est pas correcte.

2.1.1.4 Présentation de la tâche qui vous a été attribuée :

Vous devez récupérer l'application de test (téléchargeable [TestNotationPolonaise.zip](#)), y intégrer la fonction que vous devez coder, tester la fonction avec le jeu de données fourni dans le récit d'utilisateur (user story), puis déposer sur GitHub l'application de test intégrant la fonction.

3 Partie 1 - Apports méthodologiques

3.1 Ce qu'il faut savoir



Pour réaliser cette situation professionnelle, vous devez avoir acquis les savoirs suivants : Découverte des tableaux en programmation, à travers un exemple. Cet exemple est présenté dans les pages suivantes. Savoirs présentés dans les séances précédentes (bases de la programmation et modules).

3.2 Exemple de tableaux

3.2.1 But de l'exemple

3.2.2 Découverte des tableaux quand les variables simples ne suffisent pas

Pour le moment vous ne connaissez que les [types](#) "simples" c'est à dire qui ne prennent qu'une "case" mémoire : numérique, chaîne, booléen. Parfois, ces [variables](#) simples ne suffisent pas. Il existe d'autres types complexes, dont les [tableaux](#) que vous allez découvrir maintenant.



Après la réalisation de la situation professionnelle qui suit ce premier exemple, vous serez invité à approfondir vos connaissances en étudiant la fiche de savoirs : cette étape est indispensable pour aborder les séances suivantes.

3.2.2.1 But de l'exemple :

Imaginons le problème suivant : "Écrire le programme qui permet de saisir 365 températures puis d'afficher le nombre de températures qui se trouvent au-dessus de la moyenne des températures saisies."

Pour contrôler si chaque température est au-dessus de la moyenne, il faut au préalable avoir calculé la moyenne. Pour calculer la moyenne, il faut au préalable avoir saisi toutes les températures. La seule solution dont nous disposons avec nos connaissances actuelles est l'utilisation de 365 variables pour mémoriser les 365 températures. Voici le code du programme (à droite). Evidemment, il faut remplacer le "..." par de nombreuses instructions, ce qui fait au total un programme d'environ 1500 lignes, sans compter l'immense ligne du calcul de la moyenne !

Vous vous doutez qu'il y a forcément une autre solution.

```

// saisie des 365 températures
Console.Write("Entrez la température n°1 = ");
float t1 = float.Parse(Console.ReadLine());
...
...
...
Console.Write("Entrez la température n°365 = ");
float t365 = float.Parse(Console.ReadLine());
// Calcul de la moyenne
float moyenne = (t1 + ..... + t365) / 365;
// Calcul du nombre de t° au-dessus de la moyenne
int nbsup = 0;
if (t1 > moyenne)
{
    nbsup++;
}
...
...
...
if (t365 > moyenne)
{
    nbsup++;
}
// affichage final
Console.WriteLine("nbre de t° au-dessus de la moyenne = "
                  + nbsup);

```

Affichage et saisie à réaliser 365 fois

365 variables à additionner

Tests à réaliser 365 fois

Code de l'application, montrant la saisie de 365 températures dans 365 variables nommées t1 jusqu'à t365, puis le calcul de la moyenne avec les 365 variables, et enfin 365 tests pour comparer les températures avec la moyenne.

3.2.3 Regrouper les variables

3.2.3.1 Le tableau comme solution :

Une variable de type tableau est un regroupement de **variables** de même **type**, mises dans des cases contigües en mémoire centrale. Pour accéder à une case, il faut utiliser un nom unique (le nom du tableau) suivi d'un indice de case. Voici un exemple de tableau, nommé 't', contenant 10 cases numérotées de 0 à 9 et contenant une température par case :

	0	1	2	3	4	5	6	7	8	9
t	25	10	-5	4	12	21	18	32	34	12

Pour accéder à la 6ème case qui contient la température 21, il faudra écrire `t[5]`.
Créez un nouveau projet et commencez par déclarer une variable qui va contenir le nombre de températures et donc, qui représentera la taille du tableau :

Cette variable évitera l'utilisation de la valeur 365 dans tout le programme.
Maintenant, voici comment déclarer le tableau :

Un tableau à un nom (`t`), un type (`float`) et un nombre de cases (`taille`).
Après cette déclaration, il est maintenant possible de saisir les 365 températures et en même temps, cumuler les valeurs pour le calcul de la moyenne.

`k` variant de 0 à (`taille - 1`), dans la boucle, c'est dans la case `t[0]` que va être mémorisée la 1ère température, dans la case `t[1]` la 2ème température, etc.
Chaque valeur saisie est cumulée dans '`moyenne`'. Après la boucle, le calcul de la moyenne est classique :

Il faut ensuite calculer le nombre de températures au-dessus de cette moyenne :

Il ne reste plus qu'à afficher le résultat :

3.2.4 test

Testez le programme (avec une taille 5) et surtout faites une exécution [pas à pas](#) pour voir comment le tableau est manipulé.

En cas de problème, le code est téléchargeable [Temperatures.zip](#).

4 Partie 2 - Application

4.1 Application



A partir de la situation professionnelle déjà présentée en début de séance, à votre tour désormais de réaliser en autonomie la tâche qui vous a été confiée. Vous trouverez à la page suivante le rappel de la situation professionnelle, puis les étapes à suivre.

4.2 Votre tâche

```
entrez une formule polonaise =  
+ 4 3  
Résultat = 7  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
/ + * 4 - 2 6 + 5 3 10  
Résultat = -0,8  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
- / 2 3 4  
Résultat = -3,333333  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
4 + 3  
Résultat = NaN  
  
Voulez-vous continuer ? (O/N) O  
entrez une formule polonaise =  
+ % 4 3 12  
Résultat = NaN  
  
Voulez-vous continuer ? (O/N)
```

[Rédigez votre texte alternatif]

4.2.1 Rappel de la situation professionnelle :

4.2.1.1 Présentation du contexte :

L'entreprise cliente veut construire un bouquet d'applications de tests de connaissances en mathématiques de plusieurs niveaux. Elle a déjà fait appel à votre société pour divers développements, afin de fournir ce bouquet.

4.2.1.2 Narrative du récit utilisateur (user story) :

"En tant que *responsable de la chaîne de production d'applications pédagogiques*, je veux disposer d'une [fonction](#) qui permet de réaliser le calcul d'une formule en notation polonaise, afin de l'intégrer dans le bouquet d'applications de test de connaissances mathématiques de plusieurs niveaux."

La [user story](#) complète, intégrant le [jeu de données](#), est téléchargeable [user_story.pdf](#).

4.2.1.3 Présentation de l'application test fournie :

Elle permet de saisir plusieurs formules de calculs, avec la notation "Polonaise", puis d'afficher le résultat du calcul, après avoir appelé la fonction qui doit être écrite. Elle doit récupérer de la fonction et afficher NaN (Not a Number) si la formule n'est pas correcte.

4.2.1.4 Présentation de la tâche qui vous a été attribuée :

Vous devez récupérer l'application de test (téléchargeable [TestNotationPolonaise.zip](#)), y intégrer la fonction que vous devez coder, tester la fonction avec le jeu de données fourni dans le récit d'utilisateur (user story), puis déposer sur GitHub l'application de test intégrant la fonction.

4.3 Etapes à suivre

1. Si ce n'est pas déjà fait, récupérez le code du projet test [TestNotationPolonaise.zip](#).
2. Ouvrez le sous [Visual Studio](#) et publiez-le sur [GitHub](#) (inutile de créer une seconde branche pour travailler).
3. Dans le projet test, ajoutez la [fonction](#) que vous devez créer, en respectant les attentes du [user story](#) (si vous ne l'avez pas encore récupéré, il est téléchargeable [user_story.pdf](#)).
4. Testez la fonction avec le [jeu de données](#) présent dans le récit d'utilisateur. Pensez à utiliser le [débugueur](#) pour chercher l'origine des éventuels dysfonctionnements.
5. Si tout fonctionne correctement, mettez à jour la version sur GitHub.



Votre responsable met à votre disposition une fiche d'aide (téléchargeable [ici](#)), qui explique le principe de la notation polonaise et apporte quelques outils supplémentaires qui pourront vous être utiles. Il vous conseille aussi de découper la chaîne reçue en paramètre pour ranger chaque élément dans une case d'un tableau puis, de partir de la fin du tableau et à chaque fois de s'arrêter sur une case qui contient un signe d'opération (case d'indice n). Il suffit alors de faire l'opération avec les 2 cases suivantes (d'indices n+1 et n+2), de ranger le résultat à la place du signe et de supprimer les 2 cases suivantes (en décalant toutes les cases de 2 crans vers la gauche, donc la case n+3, si elle existe, est copiée en n+1, etc). Enfin il est recommandé de remplacer le contenu des 2 dernières cases du tableau, par un espace.

4.4 Correction



Vous pouvez désormais télécharger : Le code de l'application test avec la fonction Polonaise intégrée

5 Evaluer ses acquis

5.1 Evaluer ses acquis : les tableaux

Exercice 1 - Corrigé à la page 18

Pour chaque situation, précisez si l'utilisation d'un tableau est indispensable ou non.

	Oui	Non
a) Saisir 20 notes et afficher la moyenne	<input type="radio"/>	<input type="radio"/>
b) Saisir 20 notes et afficher le % des notes supérieures à la moyenne des notes saisies	<input type="radio"/>	<input type="radio"/>
c) Saisir 20 notes et afficher le % des notes supérieures à 10	<input type="radio"/>	<input type="radio"/>
d) Saisir 20 notes et les afficher dans l'ordre inverse de la saisie	<input type="radio"/>	<input type="radio"/>
e) Saisir 20 notes et les afficher triées dans l'ordre croissant	<input type="radio"/>	<input type="radio"/>
f) Saisir 20 notes et afficher la note la plus basse et la note la plus haute	<input type="radio"/>	<input type="radio"/>

5.2 Evaluer ses acquis : User Story

Exercice 2 - Corrigé à la page 18

Complétez les phrases en choisissant les termes corrects:

Un récit d'utilisateur, appelé aussi (1) , permet de présenter une demande d'un utilisateur. Elle comporte généralement 3 parties :
Une partie (2) qui se présente sous la forme d'une phrase mentionnant, dans l'ordre "..... (3) ", "..... (4) " et "..... (5) ".
Une partie présentant plus en détail la description de la demande.
Une partie "..... (6) " qui généralement comporte un (7) pour vérifier que la production correspond bien aux attentes.
Lorsque le (8) ne donne pas satisfaction, il faut corriger l'application puis relancer l'ensemble complet du (9) .

Solutions proposées:

1:	Sélectionner, "User Case", "User Story", "User Narrative"
2:	Sélectionner, narrative, indépendante, dépendante, concise
3:	Sélectionner, Pourquoi, Comment, Qui, Quoi,
4:	Sélectionner, Pourquoi, Comment, Qui, Quoi
5:	Sélectionner, Pourquoi, Comment, Qui, Quoi,

6:	Sélectionner, Critère d'intégration, Critère d'acceptation, Critère de tests, Critère de demande
7:	Sélectionner, test de production, jeu de données, critère d'acceptation
8:	Sélectionner, test de production, jeu de données, critère d'acceptation
9:	Sélectionner, test de production, jeu de données, critère d'acceptation

6 Compléter les savoirs

6.1 Compléter les savoirs



Pour compléter les savoirs abordés dans cette séance, vous pouvez maintenant consulter la fiche de savoirs sur les tableaux. L'étude de cette fiche est indispensable pour la suite de la formation. Consultez la Fiche savoirs - Les tableaux. Téléchargez les codes de corrections des exercices de la fiche de savoirs (consultables aussi à la fin de la Fiche). Vous pouvez aussi retrouver la correction en vidéo de la plupart des exercices, sur Youtube. Allez dans la playlist "Bases de la programmation (C#)" et suivez les exercices 31 à 42 (juste les tableaux) et 56 à 59 (tableaux et modules).

7 Synthèse

7.1 Synthèse de la séance

7.1.1 Ce qu'il faut retenir

Pour gérer une [User Story](#) vous devez :

- comprendre les attentes;
- décomposer la tâche à réaliser;
- soumettre la fonctionnalité créée, aux [jeux de données](#) fourni par l'utilisateur;
- en cas de dysfonctionnement, penser à utiliser le [débugueur](#)
- après chaque correction, soumettre à nouveau tout le jeu de données.

Les [tableaux](#), abordés dans cette séance, ne sont pas directement liés à la notion de User Story. Ils représentent juste une connaissance technique supplémentaire pour résoudre certains problèmes.

7.1.2 Notions clés

- **Récit d'utilisateur (user story)** contient la demande de l'utilisateur ainsi que ses attentes finales.
- **Jeu de données** pour réaliser un ensemble de tests sur une fonctionnalité (ou une application complète : il est souvent intégré au récit d'utilisateur.
- **Type Tableau** pour résoudre certains problèmes qui nécessitent le regroupement de plusieurs valeurs accessibles par un même nom et un indice de case.

7.1.3 Pour approfondir

- [User Story \(Wikipedia\)](#)
- [User Story \(Oeil du coach\)](#)

8 Glossaire

Affectation

Transfert d'une valeur, variable ou calcul dans une variable (avec compatibilité de type).

En C#, le symbole de l'affectation est '='.

Exemples :

correct = true;

nbre = nbre + 1; // raccourci d'écriture : nbre++;

variable1 = variable2;

Alternative

Ensemble d'instructions qui s'exécutent sous condition (synonymes : condition, test)

Exemple :

if (essai > valeur)

```
{  
    Console.WriteLine(" --> trop grand !");  
}
```

else

```
{  
    Console.WriteLine(" --> trop petit !");  
}
```

Application

Ensemble d'instructions formant un tout et pouvant s'exécuter pour répondre à une demande (synonyme : programme).

Autocompletion

Aide en cours de frappe qui affiche les différentes possibilités (mot du langage, variable...) par rapport à ce qui a commencé à être saisi.

Bloc de code

Ensemble d'instructions entourées par délimiteurs (en C#, des accolades).

Les différentes structures (programme, alternative, itération, ...) contiennent un bloc de code.

Commentaire

Information normalement ignorée par l'ordinateur, mais qui apporte des explications en clair sur les fonctionnalités du programme.

Les commentaires peuvent être mis à n'importe quel endroit du code.

Il existe 3 types de commentaires. Les voici avec les syntaxes en C# :

Commentaire d'une ligne :

// ce qui suit le double slash est un commentaire, jusqu'à la fin de la ligne

Commentaire de plusieurs lignes :

/* tout ce qui est entre le slash-étoile

et le étoile-slash, même sur plusieurs lignes

est un commentaire */

Commentaire de type cartouche :

C'est le seul interprété par l'ordinateur et il ne se positionne pas n'importe où : il est en début de programme (ou avant des modules : notion qui sera vue plus tard).

```
/**  
 * Jeu du nombre caché  
 * author : Emds  
 * date : 23/05/2020  
 */
```

Compilation

Analyse du code pour repérer les erreurs syntaxiques (par exemple une variable utilisée mais non déclarée) puis, en l'absence d'erreur, traduction du code en langage "machine" directement compréhensible par l'ordinateur.

Concaténation

Construction d'une chaîne en ajoutant, les unes à la suite des autres, plusieurs chaînes (valeurs entre guillemets et/ou variables).

Condition

Prédicat qui est soit vrai, soit faux.

Une alternative (if, switch) contient une condition. Une itération (while...) contient une condition.

Débogage

Recherche de dysfonctionnements dans le programme.

La recherche doit se faire à 3 niveaux :

- tests des possibilités classiques (fonctionnement de base de l'application)
- tests des cas particuliers
- tests des comportements inattendus

Déclaration

Déclarer une variable consiste à lui donner un nom et un type. Cette étape est obligatoire en C# (et dans plusieurs langages) avant de pouvoir utiliser la variable.

Fonction

Bloc de code isolé, indépendant, qui peut être sollicité par un programme ou un autre module.

Il existe 2 catégories de modules :

- la procédure : elle permet juste d'isoler un bloc de code et peut être appelée au moment où ce bloc de code doit être exécuté
- la fonction : elle se manipule comme une valeur car elle retourne une valeur (on peut donc l'affecter à une variable, l'utiliser dans un calcul, dans un test, l'afficher...)

Les modules peuvent recevoir des paramètres, donc des informations du programme appelant.

GitHub

Site proposant l'hébergement d'applications basé sur le logiciel de versionning Git qui offre de nombreux outils dont la mémorisation des différentes versions d'une application et le travail collaboratif.

Quelques termes en relation avec GitHub :

- **dépôt (repository)** : zone de stockage des fichiers du projet
- **branche** : zone de travail indépendante, dans laquelle il est possible d'enregistrer différentes modifications, sans affecter les autres branches du même projet
- **branche principale (master)** : contient le projet d'origine et final, cette branche reçoit les modifications des autres branches lorsque le responsable de la branche le décide.
- **valider (commit)** : enregistrement des modifications dans le dépôt actuel local
- **pousser (push)** : envoyer les modifications validées, vers le serveur distant, dans la branche concernée
- **requête de tirage (pull request)** : requête pour demander la prise en compte des modifications proposées

- **fusionner (merge)** : réunion de 2 branches (par exemple, récupération d'une branche secondaire pour mettre à jour la branche principale)

IDE

Environnement de développement intégré (Integrated Development Environment).

Logiciel permettant de coder une application, avec un ensemble d'outils d'aide au codage (colorisation du code, aide en cours de frappe, débogage...).

Visual studio est un IDE.

Indentation

Décalage dans le code pour mettre en évidence les différents blocs.

Exemple :

```
if (essai > valeur)
{
    Console.WriteLine("Trop grand !")
}
```

Initialisation

Première affectation d'une valeur dans une variable

Instruction

Ordre que l'ordinateur va exécuter : affichage, saisie, affectation, test...

Itération

Ensemble d'instructions qui peuvent s'exécuter plusieurs fois, tant qu'une condition est respectée (synonyme : boucle).

Exemple :

```
while (essai != valeur)
{
    // instructions qui se répètent tant que la condition est vraie.
}
```

Jeu de données

Ensemble de données à fournir à la fonctionnalité pour contrôler si on obtient bien au final les résultats attendus.

Synonyme : jeu d'essais

main

Module principal qui s'exécute automatiquement dès l'exécution de l'application.

Vous verrez par la suite qu'il existe d'autres types de modules (un module est un regroupement d'instructions qui représente une unité indépendante).

Pas à pas

Le débogueur permet une exécution "pas à pas", c'est à dire en s'arrêtant après chaque ligne (instruction) de code exécutée, pour montrer l'ordre d'exécution des instructions et l'évolution du contenu des variables.

Point d'arrêt

Marque placée sur une ligne du programme, repérée par le débogueur qui arrêtera l'exécution au niveau de cette ligne, permettant ensuite d'avancer pas à pas dans l'exécution du code.

Programme

Ensemble d'instructions formant un tout et pouvant s'exécuter pour répondre à une demande (synonyme : application).

Sensibilité à la casse

Distinction entre majuscule et minuscule.

Exemple : les variables 'total' et 'Total' sont différentes.

Tableaux

Type complexe de données, regroupant plusieurs cases mémoires, chacune accessible par le nom du tableau et un indice.

Toutes les cases sont de même type.

Test

Ensemble d'instructions qui s'exécutent sous condition (synonymes : alternative, condition)

Exemple :

```
if (essai > valeur)
{
    Console.WriteLine("--> trop grand !");
}
else
{
    Console.WriteLine("--> trop petit !");
}
```

Ticket d'incident

(synonyme : rapport d'incident)

Document qui recense des informations sur un dysfonctionnement repéré (application concernée, demandeur, date, catégorie, niveau de priorité, description du dysfonctionnement...).

Le ticket d'incident est alors confié à un technicien qui a la responsabilité de trouver l'origine du dysfonctionnement et de le corriger. Il complète alors le ticket d'incident et le clôture après avoir rendu une version opérationnelle de l'application.

Transtypage

Changement de type d'une variable ou valeur (synonymes : parse, caste)

Exemples :

```
int essai = int.Parse(Console.ReadLine())
int essai = int.Parse("23")
```

Type

Format d'une case mémoire pouvant mémoriser une variable.

Types simples (avec équivalence en C#) :

- entier (int)
- réel (float)
- chaîne (string)
- booléen (bool)

User story

(récit d'utilisateur)

Présentation d'une demande utilisateur, généralement en 3 parties :

- phrase narrative sous la forme "qui demande", "demande quoi", "demande pourquoi"
- présentation plus précise de la demande
- descriptions des critères d'acceptations (qui doivent être satisfaits) souvent sous forme de jeu de données

Variable

Zone mémoire nommée et typée, permettant de stocker une information utilisable et modifiable dans le programmes.

Visual Studio

Suite de logiciels de développement pour Windows et mac OS conçue par Microsoft.

9 Documentation

9.1 Exemple de tableaux

Temperatures.zip

9.2 Section

fiche_aide.pdf

user_story.pdf

TestNotationPolonaise.zip

TestNotationPolonaiseCorrection.zip

Fiche savoirs

9.3 Section

corrections.zip

10 Version imprimable

Titre de la ressource

Solutions

Exercice 1 - Page 11

Pour chaque situation, précisez si l'utilisation d'un tableau est indispensable ou non.

	Oui	Non
a) Saisir 20 notes et afficher la moyenne	<input type="radio"/>	<input type="radio"/>
b) Saisir 20 notes et afficher le % des notes supérieures à la moyenne des notes saisies	<input type="radio"/>	<input type="radio"/>
c) Saisir 20 notes et afficher le % des notes supérieures à 10	<input type="radio"/>	<input type="radio"/>
d) Saisir 20 notes et les afficher dans l'ordre inverse de la saisie	<input type="radio"/>	<input type="radio"/>
e) Saisir 20 notes et les afficher triées dans l'ordre croissant	<input type="radio"/>	<input type="radio"/>
f) Saisir 20 notes et afficher la note la plus basse et la note la plus haute	<input type="radio"/>	<input type="radio"/>

a) Saisir 20 notes et afficher la moyenne

NON. Pour calculer la moyenne, il suffit de cumuler et de compter les notes.

b) Saisir 20 notes et afficher le % des notes supérieures à la moyenne des notes saisies

OUI. Comme pour les températures, pour calculer ce pourcentage il va falloir comparer chaque note avec la moyenne des notes qui ne peut être calculée qu'après avoir saisi toutes les notes.

c) Saisir 20 notes et afficher le % des notes supérieures à 10

NON. Il suffit, au fur et à mesure de la saisie, de compter le nombre de notes au-dessus de 10 et le nombre de notes au total.

d) Saisir 20 notes et les afficher dans l'ordre inverse de la saisie

OUI. Pour afficher dans l'ordre inverse, il faut au préalable les avoir mémorisées. Il y a cependant une autre solution un peu spéciale qui évite la mémorisation dans 20 variables : en mémorisant les notes dans une seule variable chaîne, avec une concaténation inversée, pour ensuite afficher la variable chaîne en une seule fois. Ce n'est cependant pas forcément très joli.

e) Saisir 20 notes et les afficher triées dans l'ordre croissant

OUI. Pour trier, il faut faire des comparaisons entre les notes, donc il faut les avoir toutes mémorisées.

f) Saisir 20 notes et afficher la note la plus basse et la note la plus haute

NON. Cet exercice a déjà été fait : il suffit de comparer les notes à chaque fois avec un min et un max.

Exercice 2 - Page 11

Complétez les phrases en choisissant les termes corrects:

Un récit d'utilisateur, appelé aussi "**User Story**" (1), permet de présenter une demande d'un utilisateur. Elle comporte généralement 3 parties :

Une partie **narrative** (2) qui se présente sous la forme d'une phrase mentionnant, dans l'ordre "**Qui** (3)", "**Quoi** (4)" et "**Pourquoi** (5)".

Une partie présentant plus en détail la description de la demande.

Une partie "**Critère d'acceptation** (6)" qui généralement comporte un **jeu de données** (7) pour vérifier que la production correspond bien aux attentes.

Lorsque le **jeu de données (8)** ne donne pas satisfaction, il faut corriger l'application puis relancer l'ensemble complet du **jeu de données (9)** .

Le vocabulaire utilisé est assez courant : autant savoir le reconnaître et le comprendre.