Algorithmique et structures de données

Chapitre 1 – Introduction

Samar MOUCHAWRAB, PhD Eng

2A Cycle Préparatoire – Semestre 1
2017/2018

Objectifs du cours

A la fin de ce module, l'étudiant maîtrisera les compétences suivantes :

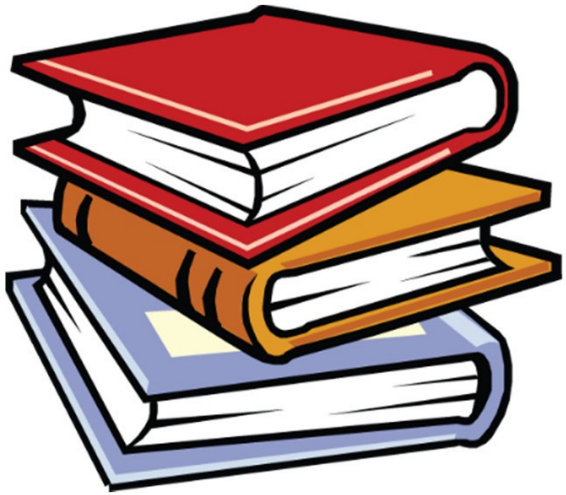
- Spécifier le calcul que l'on attend d'un programme
- Construire un programme efficace et prouver qu'il réalise le calcul attendu
- Evaluer la complexité d'un programme
- Choisir entre différents programmes réalisant le même calcul sur la base de leur complexité et des données à traiter

Plan du cours

1. **Rappel des bases de l'algorithmique** : Variables, opérations d'entrée-sortie, structure conditionnelle, structures répétitives, fonctions …
2. **Structures de données fondamentales** : piles, files, tas, ensembles, dictionnaires, arbres de recherche
3. **Méthodes de résolution** : séquentielle, dichotomique, diviser pour régner, programmation dynamique
4. **Enoncé d'un programme** : invariant de boucle, système formel de Hoare
5. **Complexité d'un programme** : notations en O, Omega et Theta et grandes classes de complexité

Bibliographie

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein, ***Introduction à l'algorithmique***, 2e Édition, Dunod, 2002
- Steven S. Skiena, ***Algorithm Design Manual***, 2e Edition, Springer, 2008
- Jon Kleinberg & Eva Tardos, ***Algorithm design***, Addison Wesley, 2005



Mode d’évaluation

Moyen	Pourcentage
Contrôle continu	25%
Travaux dirigés	15%
Travaux pratiques	
Tests	15%
Assiduité et participation	5%
Examen final	40%

Avant de commencer …

Tour de table:

Présentation en quelques mots …

Test de pré-requis

Test des compétences

Déroulement des séances

1. Déroulement d'une séance de cours

2. Ce qu'il faut avoir en cours

Définition

Un algorithme est une procédure décrivant, étape par étape, une méthode permettant de résoudre un problème.

Exemple:

Problème à résoudre : Recette de la sauce blanche

Algorithme proposé:

1. Faire revenir l'oignon haché fin dans du beurre,
2. Ajouter la farine, bien mélanger;
3. Rajouter le lait chaud, et mélanger pour éviter les grumeaux;
4. Laisser mijoter 15 minutes

Caractéristiques d'un algorithme

- Un algorithme est une suite finie d'instructions
- Chaque étape est décrite d'une façon précise
- Chaque étape est déterministe, ou en d'autres mots produit des résultats uniques
- L'algorithme s'arrête après un nombre fini d'instructions
- L'algorithme peut recevoir des données en entrée
- L'algorithme peut produire des données en sortie

Du problème à l'algorithme, un exemple

Prenons un exemple simple, le rendu de monnaie.

Une instance de ce problème est la donnée de deux valeurs, le prix à payer, disons 230 Dhs, et la somme fournie par le client, disons 250 Dhs. Le résultat attendu est la valeur de la monnaie à rendre, soit 20 Dhs dans notre cas. Naturellement, le résultat s'obtient en retranchant le prix à payer de la somme fournie.

De façon informelle, cet algorithme peut être transcrit ainsi :

```
début de l'algorithme
1. demander le prix à payer
2. demander la somme fournie par le client
3. retrancher le prix à payer de la somme fournie par le client
4. afficher le résultat ainsi obtenu
fin de l'algorithme
```

Les valeurs des données d'entrée sont « récupérées » (lignes 1 et 2), puis le calcul du résultat est effectué (ligne 3) et, enfin, celui-ci est affiché (ligne 4).

Activité

Avez-vous utilisé un robot ou vu un ?

Est-ce que le robot entend ce qu'on lui demande ?

Est-ce qu'il comprend ce qu'on lui demande ?

Les robots exécutent des instructions, un ensemble d'actions pour lesquelles ils sont programmés

Activité

On va réaliser une activité ayant pour but de :

- Convertir des activités réelles en instructions d'un algorithme
- Ecrire des instructions avec des symboles (code)
- Comprendre le besoin d'avoir une précision dans le codage
- Débugger un algorithme lors d'une exécution erronée
- Réaliser l'utilité des fonctions et des paramètres

Activité

Présentation de l'activité :

Nous avons un tas de tasses en plastique et un schéma à produire.

Il faut identifier les actions que le robot doit suivre pour produire le schéma, sachant que les actions possibles sont connues et limitées.

↑	Retirer tasse du tas
↓	Mettre tasse en bas
→	Bouger tasse en avant de ½ longueur de tasse
←	Bouger tasse en arrière de ½ longueur de tasse
	Tourner tasse vers la droite de 90°
	Tourner tasse vers la gauche de 90°

Activité

Présentation de l'activité :

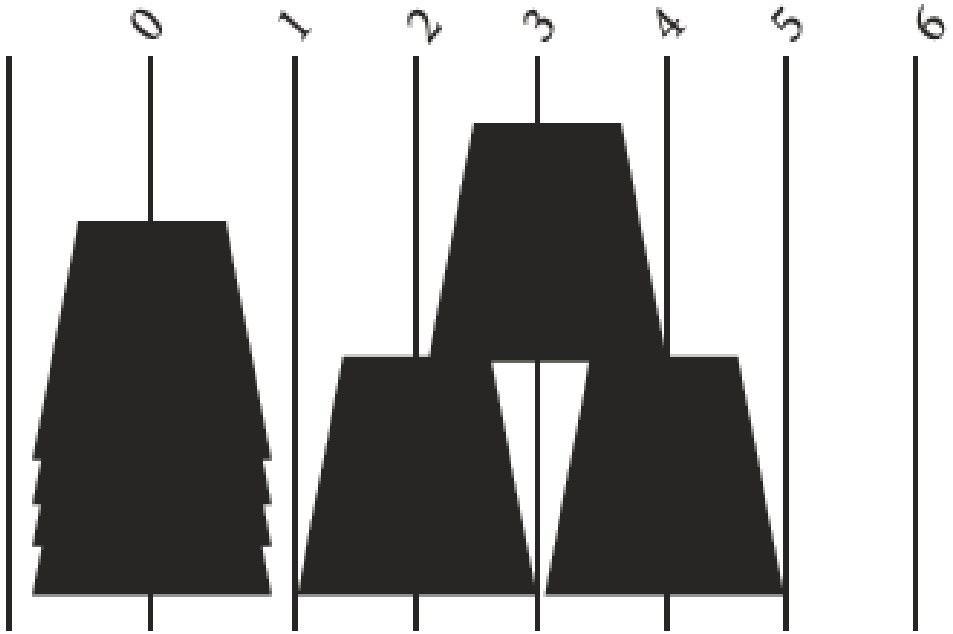
Pour que le robot produise le schéma, il faut lui présenter les instructions à suivre sous format de liste de codes (←→↓↑ …)

Ceci est l'algorithme à exécuter

Quel sera l'algorithme à exécuter si le schéma à produire est :

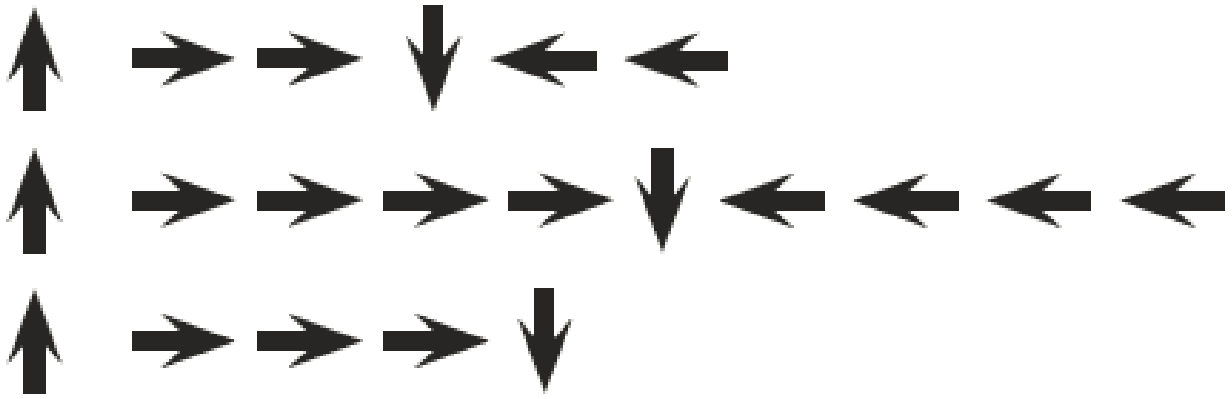
Activité

Les actions à suivre, étape par étape



Activité

Et l'algorithme à fournir au robot



Activité

A vous de coder ☺

Activité

Qu’est-ce que vous en tirer de cette activité ?

Langage à utiliser

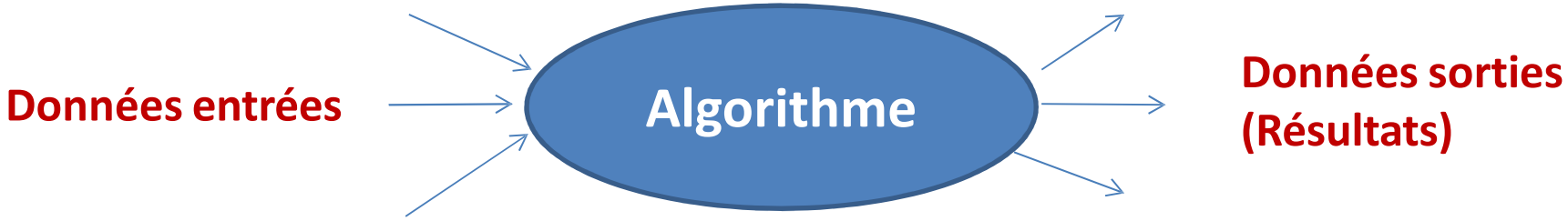
Souvent les algorithmes sont écrits avec un pseudocode.

Toutefois et pour une meilleure optimisation du temps d'apprentissage et pour combiner la théorie des algorithmes à la pratique de la programmation, nous utiliserons le langage C pour décrire les algorithmes dans ce cours.

Le langage C étant un langage simple mais puissant, il était à la base d'autres langages : C++ et Java

Les variables

Un algorithme agit sur des **données** dans le but d'obtenir un **résultat**. Pour cela, il manipule un certain nombre d'objets.



Les variables

Un objet est caractérisé par :

- un **identificateur**, c'est-à-dire un nom utilisé pour le désigner (rappelons que ce nom devra être « parlant » et distinct des mots-clés du langage de programmation ou d'algorithmique).
- un **type**, correspondant à la nature de l'objet (entier naturel, entier relatif ou chaîne de caractères par exemple). Le type détermine en fait l'ensemble des valeurs possibles de l'objet, et par conséquence l'espace mémoire nécessaire à leur représentation en machine, ainsi que les opérations (appelées primitives) que l'on peut lui appliquer.
- une **valeur** (ou contenu de l'objet). Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre (l'objet est alors une **variable**), ou être défini une fois pour toutes (on parle alors de **constante**).

Les variables

Exemple

Division entière par soustractions successives.

Le problème consiste à déterminer q et r, quotient et reste de la division entière de a par b, en n'utilisant que des opérations d'addition ou de soustraction.

Sur un exemple (a=25, b=6) le principe intuitif est le suivant :

```
                    q = 0
25 - 6 = 19         q = 0 + 1 = 1
19 - 6 = 13         q = 1 + 1 = 2
13 - 6 = 7          q = 2 + 1 = 3
7 - 6 = 1           q = 3 + 1 = 4

Résultat : q = 4 et r = 1.
```

Les variables

Quelles sont les variables utilisées dans cet exemple ?

Entrées : a et b

Sorties : q et r

Quels sont les types de ces variables ?

Tous ont le même type, entier (int)

Les variables

Programme en C :

```
#include <stdio.h>

int main(){
   int a, b, q, r;

   printf ("Entrer a : ");
   scanf ("%d", &a);
   printf ("Entrer b non nul : ");

   do {
         scanf ("%d", &b);
         if (b == 0) printf ("b ne doit pas etre nul. Reessayer\n");
   } while (b == 0);

   q = 0;
   r = a;
   while ( r >= b) {
            r =  r  - b ;
            q++;
   }
   printf ("Le quotient de la division entiere de %d par %d est %d. Et le
reste est %d\n", a, b, q, r);

   return 0;
}
```

Les types de variables

Les entiers

Il y a cinq types de variables entières (« integer » en anglais) :

- char ;
- short int, ou plus simplement short ;
- int ;
- long int, ou long ;
- long long int, ou long long

Les types entiers peuvent prendre les modificateurs signed et unsigned qui permettent respectivement d'obtenir un type signé ou non signé.

Les types de variables

Les entiers

Type	Taille	Borne inférieure	Borne supérieure
signed char	≥ 8 bits	-128	127
unsigned char	≥ 8 bits	0	255
short	≥ 16 bits	-32 767	+32 767
unsigned short	≥ 16 bits	0	+65 535
int	≥ 16 bits	-32 767	+32 767
unsigned int	≥ 16 bits	0	+65 535
long	≥ 32 bits	-2 147 483 647	+2 147 483 647
unsigned long	≥ 32 bits	0	+4 294 967 295
long long *(C99)*	≥ 64 bits	-9 223 372 036 854 775 807	+9 223 372 036 854 775 807
unsigned long long *(C99)*	≥ 64 bits	0	+18 446 744 073 709 551 615

« int » peut être de 2 ou de 4 octets dépendamment du compilateur utilisé

Les types de variables

Les nombres réels

Les nombres réels ne pouvant tous être représentés, sont approximés par des nombres à virgule flottante. Comme dans le cas des entiers, il existe plusieurs types de nombre à virgule flottante. En voici la liste triée par précision croissante :

- float
- double
- long double

Les types de variables

Les caractères

À l'origine, le type permettant de représenter un caractère est char.

Une constante représentant un caractère (de type char) est délimitée par des apostrophes, comme par exemple 'a'.

Certains caractères ne sont pas graphiques. Par définition, on ne peut pas donc les écrire dans un code source de manière visible. Le langage C adopte la convention ci-contre pour désigner certains d'entre eux de manière littérale.

Constante	Caractère
`'\''`	une apostrophe
`'\"'`	un guillemet
`'\?'`	un point d'interrogation
`'\\'`	un backslash
`'\a'`	un signal sonore (ou visuel)
`'\b'`	un espace arrière
`'\f'`	saut au début de la page suivante
`'\n'`	saut de ligne
`'\r'`	un retour chariot
`'\t'`	une tabulation
`'\v'`	une tabulation verticale

Les types de variables

Les chaînes de caractères

Une chaîne de caractères, comme son nom l'indique, est une suite de caractères avec la particularité d'avoir un caractère nul ('\0') à sa fin. Une chaîne de caractères est en fait implémentée en C avec un tableau de type char.

On appelle une chaîne littérale la manière dont est définie une constante chaîne dans un code source. Sous sa forme la plus simple, on déclare une chaîne comme une suite de caractères entre guillemets (double quote) :

```
"Ceci est une chaîne de caractère";
""; /* Chaîne vide */
```

Si la chaîne est trop longue, on peut aussi la couper sur plusieurs lignes :

```
"Ceci est une chaîne de caractère, " /* pas de ; */
"déclarée sur plusieurs lignes.";
```

Les types de variables

Les booléens

Le langage (jusqu'à la norme C99) ne fournit pas de type booléen. La valeur entière 0 prend la valeur de vérité faux et toutes les autres valeurs entières prennent la valeur de vérité vrai.

La norme C99 a introduit le type _Bool, qui peut contenir les valeurs 0 et 1. Elle a aussi ajouté l'en-tête <stdbool.h>, qui définit le type bool qui est un raccourci pour _Bool, et les valeurs true et false. Néanmoins, ces nouveautés du C99 ne sont pas très utilisées, les habitudes ayant été prises d'utiliser 0 et différent de zéro pour les booléens en C.

A noter que toute expression utilisant des opérateurs booléens, retourne 1 si l'expression est vraie et 0 si elle est fausse, ce qui rend quasiment inutile l'usage du type booléen.

Les types de variables

Le vide

Le langage C fournit un autre type, void qui représente rien, le vide.

Il n'est pas possible de déclarer une variable de type void mais ce type est utilisé avec les fonctions et les pointeurs.

Exercice 1

Ecrire un algorithme en C qui vérifie si un entier est pair ou impair

Solutions

Exercice 1

```
#include <stdio.h>

void main() {

    int ival, reste;

    printf("Entrer un entier : ");
       scanf("%d", &ival);

       reste = ival % 2;

       if (reste == 0)
              printf("%d est un entier pair\n", ival);

       else
              printf("%d est un entier impair\n", ival);
}
```

Solutions

Exercice 1 – avec test si nombre entier

```
#include <stdio.h>
#include <math.h>
int isInteger(float x) {
        int E = floor(x);
        if (E - x != 0)
                return 0;
        else
                return 1;
}
int main() {
        float x;
        do {
                printf("Veuillez saisir un nombre entier: ");
                scanf("%f", &x);
        } while ( isInteger(x) == 0 );
        int E = floor(x);
        if (E%2==0)
                printf("Le nombre est pair");
        else
                printf("Le nombre est impair");
        return 0;
}
```

Solutions

Exercice 1 – autre implémentation de la fonction isInteger

```
int isInteger (float x) {
    int q;
    q=(int)x;

    if((q - x) != 0)
        return 0;
    else
        return 1;
}
```

Exercice 2

Ecrire un algorithme en C qui calcule la somme des nombres impairs et des nombres pairs entre 1 et N (N à demander à l'utilisateur)

Solutions

Exercice 2

```
#include <stdio.h>

void main() {
	int i, num, odd_sum = 0, even_sum = 0;

	printf("Entrer la valeur de N : ");
	scanf("%d", &num);

	for (i = 1; i <= num; i++) {
		if (i % 2 == 0)
			even_sum = even_sum + i; //somme des pairs
		else
			odd_sum = odd_sum + i; //somme des impairs
	}

	printf("Sum of all  odd numbers = %d\n", odd_sum);
	printf("Sum of all even numbers = %d\n", even_sum);

}
```

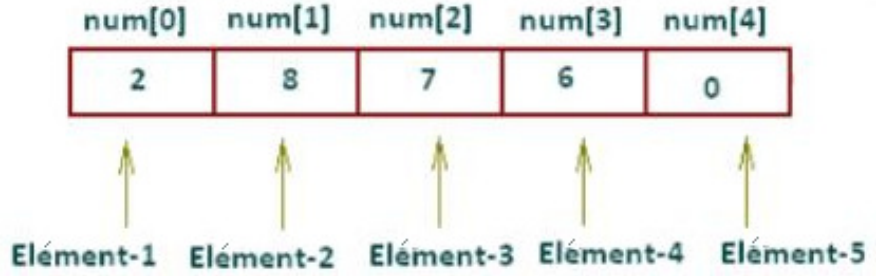
Les tableaux

On appelle tableau une variable composée de données de même type, stockée de manière contiguë en mémoire (les unes à la suite des autres).

Un tableau est donc une suite de cases (espace mémoire) de même taille. La taille de chacune des cases est conditionnée par le type de donnée que le tableau contient.

Les éléments du tableau peuvent être :

- des données de type simple : int, char, float, long, double... (la taille d'une case du tableau est alors le nombre d'octets sur lequel la donnée est codée)
- des pointeurs (objets contenant une adresse mémoire).
- des tableaux
- des structures



Unidimensionnels vs. multidimensionnels

Lorsque le tableau est composé de données de type simple, on parle de tableau unidimensionnel (vecteur).

Lorsque celui-ci contient lui-même d'autres tableaux on parle alors de tableaux multidimensionnels (aussi matrice ou table).

Exemple: Tableau de 3 lignes et 4 colonnes.

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Déclaration de tableau

En langage C, la syntaxe de la définition d'un tableau unidimensionnel est la suivante :

```
type Nom_du_tableau [Nombre d'éléments]
```

Exemple la définition d'un tableau qui doit contenir 8 éléments de type char :

```
char T [8]
```

Pour connaître la **taille d'un tableau**, c'est-à-dire déterminer le nombre d'octets que celui-ci occupe en mémoire, on peut utiliser l'opérateur `sizeof()`

Accès aux éléments

Pour accéder à un élément du tableau, il suffit de donner le nom du tableau, suivi de l'indice de l'élément entre crochets :

```
Nom_du_tableau[indice]
```

- L'indice du premier élément du tableau est 0
- Un indice est toujours positif
- L'indice du dernier élément du tableau est égal au nombre d'éléments – 1

Initialisation des éléments

Lorsque l'on définit un tableau, les valeurs des éléments qu'il contient ne sont pas définies, il faut donc les initialiser, c'est-à-dire leur affecter une valeur.

- Affectation des valeurs aux éléments un par un :

```
T[0] = T[1] = T[2] = 0;
```

- **Utiliser une boucle** qui va permettre d'initialiser successivement chacun des éléments grâce à un compteur qui servira d'indice :

```
int T [10]; int Indice;
for (Indice = 0; Indice <= 9; Indice++) {
      T [Indice] = 0;
}
```

Initialisation des éléments

Pour initialiser un tableau avec des valeurs spécifiques, il est possible d'initialiser le tableau à la définition en plaçant entre accolades les valeurs, séparées par des virgules :

```
int T [10] = {1, 2, 6, 5, 2, 1, 9, 8, 1, 5};
```

- Le nombre de valeurs entre accolades ne doit pas être supérieur au nombre d'éléments du tableau.
- Les valeurs entre accolades doivent être des constantes.
- Si le nombre de valeurs entre accolades est inférieur au nombre d'éléments du tableau, les derniers éléments sont initialisés à 0.
- Il doit y avoir au moins une valeur entre accolades

Ainsi, l'instruction suivante permet d'initialiser tous les éléments du tableau à zéro :

```
int T [10] = {0};
```

Les tableaux multidimensionnels

Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.

Un tableau multidimensionnel se définit de la manière suivante :

```
type Nom_du_tableau [a1][a2][a3] ... [aN]
```

Chaque élément entre crochets désigne le nombre d'éléments dans chaque dimension. Le nombre de dimensions n'est pas limité.

Exemple, un tableau d'entiers positifs à deux dimensions (3 lignes, 4 colonnes) se définira avec la syntaxe suivante :

```
int a[3][4]
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Exercices

Exercice 3

Ecrire un algorithme en C qui calcule le produit scalaire de deux vecteurs d'entiers

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

Solutions

Exercice 3 – solution

```
#include <stdio.h>
void main() {
        /* Déclarations */
        int n;      /* dimension         */
        int U[50], V[50]; /* tableaux donnés */
        int i;      /* indice courant    */
        int PS = 0;   /* produit scalaire */

        /* Saisie des données */
        printf("Dimension des tableaux (max 50): ");
        scanf("%d", &n );
        printf("** Premier tableau **\n");
        for (i=0; i<n; i++) {
                printf("Element %d : ", i);
                scanf("%d", &U[i]);
        }
```

Solutions

Exercice 3 – solution

```
        printf("** Deuxième tableau **\n");
        for (i=0; i<n; i++) {
                printf("Element %d : ", i);
                scanf("%d", &V[i]);
        }

        /* Calcul du produit scalaire */
        for (i=0; i<n; i++) {
                PS += U[i]*V[i];
                printf("PS : %d * %d ", U[i],V[i]);
        }
        /* Edition du résultat */
        printf("Produit scalaire : %d\n", PS);
}
```

Exercices

Exercice 4

Ecrire un algorithme en C qui additionne deux matrices.

Exemple:

$$\begin{pmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{pmatrix}$$

Solutions

Exercice 4 – solution

```
#include <stdio.h>
void main()  {
        /* Déclarations */
        int A[50][50]; /* matrice donnée   */
        int B[50][50]; /* matrice donnée   */
        int C[50][50]; /* matrice résultat */
        int N, M;      /* dimensions des matrices */
        int I, J;      /* indices courants */

        /* Saisie des données */
        printf("Nombre de lignes   (max.50) : ");
        scanf("%d", &N );
        printf("Nombre de colonnes (max.50) : ");
        scanf("%d", &M );
```

Solutions

```
    printf("*** Matrice A ***\n");
    for (I=0; I<N; I++)
            for (J=0; J<M; J++)    {
                    printf("Element[%d][%d] : ",I,J);
                    scanf("%d", &A[I][J]);
     }
    printf("*** Matrice B ***\n");
    for (I=0; I<N; I++)
            for (J=0; J<M; J++) {
                    printf("Element[%d][%d] : ",I,J);
                    scanf("%d", &B[I][J]);
     }
    /* Affichage des matrices */
    printf("Matrice donnee A :\n");
    for (I=0; I<N; I++) {
            for (J=0; J<M; J++)
                    printf("%7d", A[I][J]);
            printf("\n");
}
```

Solutions

```
        printf("Matrice donnee B :\n");
        for (I=0; I<N; I++) {
                for (J=0; J<M; J++)
                        printf("%7d", B[I][J]);
                printf("\n");
        }
        /* Affectation du résultat de l'addition à C */
        for (I=0; I<N; I++)
                for (J=0; J<M; J++)
                        C[I][J] = A[I][J]+B[I][J];

        /* Edition du résultat */
        printf("Matrice resultat C :\n");
        for (I=0; I<N; I++) {
                for (J=0; J<M; J++)
                        printf("%7d", C[I][J]);
                printf("\n");
    }
}
```

Initialisation d'un tableau sans taille précisée

Il est possible de déclarer une variable de type tableau, sans préciser sa taille, si on procède à l'initialiser dans la même instruction:

```
int tab1[] = { 1, 2, 3 };
int tab2[];
```

tab1 est créé correctement avec taille de tableau = 12 octets
Erreur de compilation lors de la déclaration de tab2 : `array size missing`

De même on peut déclarer la taille comme étant une variable définie avant l'instruction de déclaration du tableau. Toutefois, la réservation de mémoire pour le tableau lors de l'exécution pourra rencontrer des problèmes surtout dans le cas d'un tableau multidimensionnel.

```
int n, m;
int tab[n][m];
```

Les chaînes de caractères

Il n'existe pas de type spécial chaîne ou string en C. Une chaîne de caractères est traitée comme un tableau à une dimension de caractères (vecteur de caractères).

Il existe quand même des notations particulières et une bonne quantité de fonctions spéciales pour le traitement de tableaux de caractères.

Déclaration de chaînes de caractères en C:

```
char <NomVariable> [<Longueur>];
```

Exemples:

```
char NOM [20];
char PRENOM [20];
char PHRASE [300];
```

Les chaînes de caractères constantes

Les chaînes de caractères constantes (string literals) sont indiquées entre guillemets. La chaîne de caractères vide est alors: ""

Dans les chaînes de caractères, nous pouvons utiliser toutes les séquences d'échappement définies comme caractères constants:

```
"Ce \ntexte \nsera réparti sur 3 lignes."
```

Le symbole " peut être représenté à l'intérieur d'une chaîne par la séquence d'échappement \":

```
"Affichage de \"guillemets\" \n"
```

Le symbole ' peut être représenté à l'intérieur d'une liste de caractères par la séquence d'échappement \' :

```
{'L','\'','a','s','t','u','c','e','\0'}
```

Les chaînes de caractères constantes

Plusieurs chaînes de caractères constantes qui sont séparées par des signes d'espacement (espaces, tabulateurs ou interlignes) dans le texte du programme seront réunies en une seule chaîne constante lors de la compilation. Exemple:

```
"un "  "deux"
         " trois"
```

sera évalué à `"un deux trois"`

Ainsi il est possible de définir de très longues chaînes de caractères constantes en utilisant plusieurs lignes dans le texte du programme.

L'accès à un élément d'une chaîne de caractères peut se faire de la même façon que l'accès à un élément d'un tableau.

Initialisation des chaînes de caractères

En général, les tableaux sont initialisés par l'indication de la liste des éléments du tableau entre accolades:

```
char CHAINE[] = {'H','e','l','l','o','\0'};
```

Pour le cas spécial des tableaux de caractères, nous pouvons utiliser une initialisation plus simple en indiquant une chaîne de caractère constante:

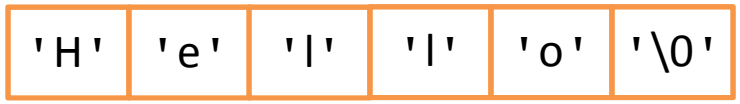
```
char CHAINE[] = "Hello";
```

Lors de l'initialisation par {}, l'ordinateur réserve automatiquement le nombre d'octets nécessaires pour la chaîne, c.-à-d.: le nombre de caractères + 1 (le dernier caractère est le caractère NUL \0 indiquant la fin de la chaîne). Nous pouvons aussi indiquer explicitement le nombre d'octets à réserver, si celui-ci est supérieur ou égal à la longueur de la chaîne d'initialisation.

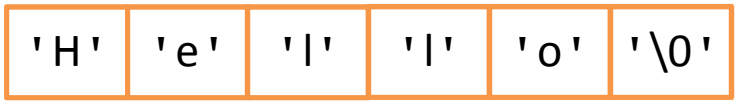
Initialisation des chaînes de caractères

Exemples:

```
char s[] = "Hello"
```




```
char s[6] = "Hello"
```




```
char s[8] = "Hello"
```


```
char s[5] = "Hello"
```

Erreur

```
char s[4] = "Hello"
```

Erreur

Initialisation des chaînes de caractères

Exercice 5:

Lesquelles des chaînes suivantes sont initialisées correctement ?

```
a)    char a[]   = "un\ndeux\ntrois\n";
b)    char b[12] = "un deux trois";
c)    char c[]   = 'abcdefg';
d)    char d[10] = 'x';
e)    char e[5]  = "cinq";
f)    char f[]   = "Cette "   "phrase"   "est coupée";
g)    char g[2]  = {'a', '\0'};
h)    char h[4]  = {'a', 'b', 'c'};
i)    char i[4]  = "'o'";
```

a, e, f, g, h, i

Fonctions sur les chaînes de caractères

Les fonctions de <stdio.h>

1- Affichage de chaînes de caractères avec printf

printf avec le spécificateur de format %s permet d'intégrer une chaîne de caractères dans une phrase.

En plus, le spécificateur %s permet l'indication de la largeur minimale du champ d'affichage. Dans ce champ, les données sont justifiées à droite. Si on indique une largeur minimale négative, la chaîne sera justifiée à gauche. Un nombre suivant un point indique la largeur maximale pour l'affichage.

```
char NOM[] = "hello, world";
printf(":%s:", NOM);                ->:hello, world:
printf(":%5s:", NOM);               ->:hello, world:
printf(":%15s:", NOM);              ->:   hello, world:
printf(":%-15s:", NOM);             ->:hello, world   :
printf(":%.5s:", NOM);              ->:hello:
```

Fonctions sur les chaînes de caractères

2- Affichage de chaîne de caractère avec puts

`puts( <Chaîne> )`

puts écrit la chaîne de caractères désignée par <Chaîne> sur stdout et provoque un retour à la ligne.

En pratique, `puts(TXT);` est équivalent à `printf("%s\n",TXT);`

Exemples

```
char TEXTE[] = "Voici une première ligne.";
puts(TEXTE);
puts("Voici une deuxième ligne.");
```

Fonctions sur les chaînes de caractères

3- Lecture de chaînes de caractères avec scanf

La fonction scanf a besoin des adresses de ses arguments. Les noms des variables numériques (int, char, long, float, ...) doivent être marqués par le symbole '&'. Mais comme le nom d'une chaîne de caractères est le représentant de l'adresse du premier caractère de la chaîne, il ne doit pas être précédé de l'opérateur adresse '&'.

Exemple: `char a [30] ; scanf ("%s", a);`

4- Lecture de chaînes de caractères avec gets

`gets( <Chaîne> )`

gets lit une chaîne de caractères désignée par <Chaîne> du stdin et la retourne si la lecture se passe correctement, ou retourne NULL si erreur dans la lecture.

En pratique, `gets(TXT);` est équivalent à `scanf("%s",TXT);`

Fonctions sur les chaînes de caractères

Les fonctions de <string.h>

Cette bibliothèque fournit un ensemble de fonctions qui permettent de manipuler des chaînes de caractères. Exemples:
`strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok`

Les noms de ces fonctions commencent toujours par "str" pour string (chaîne en anglais).

Voici des détails sur quelques unes de ces fonctions:

`char * strcpy (chaîne1, chaîne2)` : copie le contenu de «chaîne2» dans « chaîne1 ». L'ancienne valeur de chaîne1 est perdue. Retourne un « pointeur sur » chaîne1.

Fonctions sur les chaînes de caractères

`int strlen (chaîne)` : renvoie le nombre de caractères de la chaîne. Cette fonction compte les caractères jusqu'à trouver le '\0'. La valeur renvoyée est le nombre de caractères avant le '\0'

`char * strcat (chaîne1, chaîne2)` : copie le contenu de chaîne2 à la fin de chaîne1. Retourne un « pointeur sur » chaîne1.

`int strcmp ( chaîne1 , chaîne2 )` : compare les deux chaînes, retourne 0 en cas d'égalité, une valeur négative si « chaîne1 » est "avant" « chaîne2 » dans l'ordre lexicographique, une valeur strictement positive sinon.

`char * strncat ( chaine1 , chaine2 , n )` : concatène au plus les « n » premiers caractères de « chaine2 » à « chaine1 ».

Exercices

Exercice 6

Ecrire un programme qui lit 5 mots, séparés par des espaces et qui les affiche ensuite dans une ligne, mais dans l'ordre inverse. Les mots sont mémorisés dans 5 variables M1, ... ,M5.

```
#include <stdio.h>
void main() {
        char M1[30], M2[30], M3[30], M4[30], M5[30];
        printf("Entrez 5 mots, separes par des espaces :\n");
        scanf ("%s %s %s %s %s", M1, M2, M3, M4, M5);
        printf("%s %s %s %s %s\n",M5, M4, M3, M2, M1);
}
```

Exercices

Exercice 7

Ecrire un programme qui calcule la fréquence d'un caractère dans une chaîne de caractères.