

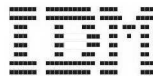
INFORMATIQUE SCIENTIFIQUE

IUT ORSAY

Mesures Physiques - 1^{re} année Algorithmes et Conseils de programmation version 3.0

<http://www.iut-orsay.fr/dptmphy/Pedagogie/Welcome.html>

Bob CORDEAU



Informatique :

Rencontre de la logique formelle et du fer à souder.

Maurice NIVAT

SOMMAIRE

- NOTATIONS ALGORITHMIQUES
- SYNTAXE DU LANGAGE C
- EXEMPLE TYPE D'UN PROGRAMME C
- CORRESPONDANCES ENTRE NOTATIONS ALGORITHMIQUES ET LANGAGE C
- TYPES CLASSIQUES DU C
- CONSEILS ET CONVENTIONS D'ÉCRITURE
- INDENTATION
- PRIORITÉS DES OPÉRATEURS
- ERREURS DE COMPILATION
- EXERCICES CORRIGÉS D'ALGORITHMIQUE

NOTATIONS ALGORITHMIQUES

du cours d'informatique scientifique

Dépouiller la langue jusqu'à la mettre à nu...

Ernest Hemingway

Les types de données de base

- booléen;
- entier;
- flottant;
- caractère.

Commentaires : les souligner

Déclaration des variables : nom + type (on peut les initialiser à la déclaration)

- hauteur : entier
- pi ← 3.14 : flottant

Variables structurées (indexées à partir de 0)

- tab : tableau de 10 flottants tab[0] .. tab[9]
- str : chaîne de 8 caractères str[0] .. str[7]

Affectation

- tab[2] ← 2.36
- index ← index + 3

Opérateurs relationnels

- == : égal
- != : différent
- < : inférieur
- ≤ : inférieur ou égal
- > : supérieur
- ≥ : supérieur ou égal

Entrées/Sorties

- Saisir(index)
- Afficher("Hauteur de la façade : ", hauteur, " mètres")

Sélection

- Elle peut être simple sans le sinon
- Elle peut être complète avec le sinon (cf. alg. 1)

Boucles

- Boucle Faire .. TantQue. (cf. alg. 2)
- Boucle TantQue. (cf. alg. 3)
- Boucle Pour. (cf. alg. 4)

Algorithm 1 Sélection

Si (*condition*)
 < *instructions* >
Sinon
 < *instructions* >
FinSi

Algorithm 2 Boucle Faire .. TantQue

Faire
 < *instructions* >
TantQue(*condition*)

Algorithm 3 Boucle TantQue

TantQue (*condition*)
 < *instructions* >
FinTantQue

Algorithm 4 Boucle Pour

Pour *cpt* **Dans** [*a..b*] **ParPasDe** *n*
 < *instructions* >
FinPour

Algorithm 5 Fonction : calcul de la factorielle (notez que *n* est en entrée)

DébutFonction FACTORIELLE($\rightarrow n$: entier) : entier
 DébutDéclaration
 fact, *cpt* : entier
 FinDéclaration

 fact \leftarrow 1
 Pour *cpt* **Dans** [1..*n*] **ParPasDe** 1
 fact \leftarrow *fact***cpt*
 FinPour

 Retourner *fact*
FinFonction

Fonctions

- Notation : voir exemple (cf. alg. 5).
- Utilisation : toujours dans une expression :
a ← Doubler(3)
Afficher(Doubler(4-x))
c ← 9 + Factorielle(Doubler(n+2))
- Leur exécution produit un résultat, *et un seul*, qui prend la place de la fonction lors de l'évaluation de l'expression.

Procédures

- Notation : voir exemples (cf. alg. 6 et 7).

Algorithm 6 Procédure d'échange (notez que *a* et *b* sont en entrée-sortie)

DébutProcédure ECHANGER(\leftrightarrow a : entier, \leftrightarrow b : entier)

DébutDéclaration

temp ← a : entier variable temporaire

FinDéclaration

a ← b

b ← temp

FinProcédure

Algorithm 7 Procédure de calcul d'heure (notez que *h* et *m* sont en sortie)

DébutProcédure HEURESMINUTES(\rightarrow dureeEnMin : entier, \leftarrow h : entier, \leftarrow m : entier)

h ← dureeEnMin/60

m ← dureeEnMin - 60 * h

FinProcédure

- Utilisation : leur appel constitue à lui seul une instruction :
HeuresMinutes(193, h, m)
Afficher("Valeur après échange")
Echanger(u, v)
- Seules les procédures permettent de modifier la valeur de plusieurs arguments.



Liste des mots réservés

- **Afficher**
- **Dans**
- **DébutDéclarations**
- **DébutFonction**
- **DébutProcédure**
- **DébutProgramme**
- **Faire**
- **FinDéclarations**
- **FinFonction**
- **FinPour**
- **FinProcédure**
- **FinProgramme**
- **FinSi**
- **FinTantQue**
- **ParPasDe**
- **Pour**
- **Retourner**
- **Saisir**
- **Si**
- **Sinon**
- **TantQue**

Remarque

☞ Outre ces mots réservés, on emploiera des fonctions utiles, par exemple :
`Alea (n)`, fonction qui fournit un entier au hasard entre 0 et $n-1$

Par ailleurs, toutes les fonctions mathématiques usuelles seront disponibles. On les écrira simplement comme en math (pourvu que la variable x soit déclarée) :

`sin(x)`, `arccos(x)`, `tanh(x)`, `log(x)`...

SYNTAXE du LANGAGE C

FICHIERS SOURCES _____

```
*.cpp fichiers principaux
*.h fichiers d'en-tête
```

Vos programmes.

Ne pas mettre de code dans les en-têtes.

DIRECTIVES _____

```
#include <iostream.h>
#include "monEnTete.h"
#define ANCIEN NOUVEAU
```

Pas de ';' après une directive.

En-tête système.

En-tête utilisateur.

Remplace partout ANCIEN par NOUVEAU.

```
#define CHEZ_MOI "C:\\InfoScient\\"
```

globales numériques ou textuelles.

DÉCLARATIONS DE VARIABLES _____

```
int i, j=2, cpt;
const double rac2 = 1.414;
```

Toute variable doit être déclarée.

On peut initialiser en déclarant.

Une constante doit être initialisée.

types de base disponibles

```
char bool int short long float double
```

DÉFINIR UN NOUVEAU TYPE _____

```
typedef float TAB[9];
TAB t;
```

typedef CONNU NOUVEAU

TAB est un nouveau type :

t est un tableau de 9 float

NOTATION [] POUR DÉFINIR DES : _____

tableaux

```
double t[7];
int m[2][3];
```

Les indices commencent à 0.

Son nom est un pointeur sur son premier élément.

matrice de 2 lignes et 3 colonnes.

chaînes de caractères

```
char s[9];
```

s[0] .. s[8] terminée par '\0'

AFFECTATION _____

```
t[4] = 12.5;
```

De même type : choux ≠ carottes

Ne pas confondre avec ==

RELATIONS _____

```
== != > < >= <=
```

Attention à égal (==) et à différent (!=)

ALTERNATIVE _____

```
if(j == 2) { cout << "j est pair"; }
if(Premier(k)) {
    cout << "k est premier";
} else {
    cout << "k est composé";
}
```

La condition est toujours booléenne.

Signifie Premier(k) est vrai

SÉLECTION PAR ÉTIQUETTES _____

```
switch(i) {
    case 1 : a = s[j]; break;
    case 2 : c = b+5; break;
    default : // tous les autres cas
}
```

Ne pas oublier les break !

Étiquettes de type discret :

int, short, char, long...

BOUCLES

for

```
for(i = 0; i < N; i++) {  
    som = som + t[i];  
}
```

Les bases de la répétition.

La plus utile quand on connaît les valeurs de début et de fin de l'indice.

do .. while

```
do {  
    cout << "Entrez un entier > 0 : ";  
    cin >> n;  
} while(n < 1);
```

Condition en fin de boucle.

while

```
while((n%i) == 0) {  
    n = n/2;  
    cpt++;  
}
```

Condition en début de boucle.

ENTRÉES / SORTIES ("à la C++")

Les communications.

affichage

```
cout << p << "\% de " << t;
```

saisie

```
cin >> val;
```

ENTRÉES / SORTIES ("à la C")

Les communications.

formats courants

```
%d entier  
%i entier  
%f flottant  
%c caractère  
%s chaîne de caractères  
%o octal  
%x hexadécimal
```

modificateurs :

```
%li entier long  
%hi entier court  
%lf réel double  
%2.3f nn.nnn
```

affichage

```
printf("%.2lf \% de %d\n", p, t);
```

13.95 % de 148

saisie

```
scanf("%li", &numLong);  
scanf("%s", string);
```

Ne pas oublier le &...

... sauf pour les chaînes!

FONCTIONS et PROCÉDURES

Prototype + appel + définition

fonction

```
int Factorielle(int n); // prototype  
int Factorielle(int n) { // définition  
    int i, fact = 1;  
    for(i = 0; i <= n; i++) {  
        fact = fact + 1;  
    }  
    return fact;  
}
```

Toujours dans une expression.

Donne un seul résultat typé.

Déclaration de variables locales.

procédure

```
void InitTab(double t[]); // proto.  
void InitTab(double t[]) { // déf.  
    int i;  
    for(i = 0; i < N; i++) {  
        t[i] = 0;  
    }  
}
```

Pas de type (void), donc pas d'affectation lors de l'appel.

Déclaration de variables locales.

Pas de return

<p>OPÉRATEURS BOOLÉENS _____ && !</p>	<p><i>Opérateurs logiques. ET / OU / NON</i></p>
<p>OPÉRATEURS BINAIRES _____ & ^ ~ << >></p>	<p><i>Opérateurs bit à bit (masque). ET / OU / XOR / complément Décalage des bits gauche/droite</i></p>
<p>POINTEURS et RÉFÉRENCES : variables typées — opérateur & int x, &y = x; opérateur * int *pa; // *pa : valeur de pa déclaration et manipulation int x = 3, *pa; pa = &x; // donc *pa vaut 3 références et procédures void RAZ(int& a); void main(void) { int i = 15; RAZ(i); } void RAZ(int& a) { a = 0; } pointeurs et procédures void RAZ(int *pa); void main(void) { int i = 15; RAZ(&i); } void RAZ(int *pa) { *pa = 0; } passage des tableaux en arguments int f(int tab[]); // un prototype int f(int *tab); // autre écriture int j, m, n, t[5]; j = f(t); // un appel m = f(&t[3]); // autre appel n = f(t+3); // même résultat</p>	<p><i>Contient l'adresse d'une autre variable de même type. y est un alias de x</i></p> <p><i>pa est un pointeur (une adresse).</i></p> <p><i>*pa est un int et pa un (int *) Car pa reçoit l'adresse de x. Exemple à bien connaître! Prototype.</i></p> <p><i>Ici i = 15 Après l'appel : i = 0</i></p> <p><i>Définition. Exemple à bien connaître! Prototype.</i></p> <p><i>Ici i = 15 Après l'appel : i = 0</i></p> <p><i>Définition.</i></p> <p><i>Aspect tableau. Aspect pointeur équivalent.</i></p> <p><i>On passe l'adresse dans les 3 cas. On passe une partie du tableau. Écriture équivalente par pointeur.</i></p>
<p>STRUCTURES _____ déclaration de structures struct individu { char nom[40]; int age; char sexe; } déclaration de variables structurées struct individu Luc, Bob; struct individu *Aude; utilisations Luc.sexe = 'M'; Luc.age = 17; Bob.age = 3 * Luc.age - 4; Aude->sexe = 'F';</p>	<p><i>Paquet d'informations hétérogènes. Ou mieux avec un typedef :</i></p> <pre>typedef struct { char nom[40]; int age; char sexe; } INDIVIDU;</pre> <p><i>Ou mieux avec un typedef :</i> INDIVIDU Luc, Bob; INDIVIDU *Aude; <i>Syntaxes pointée ou fléchée :</i> <i>Notation pointée.</i> <i>Notation pointée.</i> <i>Notation pointée.</i> <i>Notation fléchée pour les pointeurs de structure.</i></p>

EXEMPLE DE STRUCTURE TYPE d'un programme C en quatre parties

1 - Identification du programme

```
// TP747e2.c  
// Dennis Ritchie, 29/02/2013
```

2 - En-tête (directives, nouveaux types et prototypes globaux)

```
#include <iostream>  
using namespace std;  
  
const int N = 25;  
typedef int TAB[N];
```

3 - Fonction principale : reflet de l'algorithme général

```
int main(void)  
{  
    int iMin, tmp;  
    TAB t;  
    // prototype local  
    int IndiceDuMin(TAB, // tableau d'entiers  
                    int); // taille du tableau  
  
    srand(time(NULL));  
    for(int i = 0; i < N; i++)  
    {  
        t[i] = rand() % N;  
    }  
    iMin = IndiceDuMin(t, N); // appel de fonction  
    cout << " Avant : t[0] = " << t[0] << "\t t[iMin] = " << t[iMin] << "\n";  
    // inversion de t[0] et de t[iMin]  
    tmp = t[0];  
    t[0] = t[iMin];  
    t[iMin] = tmp;  
    cout << "\n Apres : t[0] = " << t[0] << "\t t[iMin] = " << t[iMin] << "\n";  
  
    cout << endl;  
    system("PAUSE");  
    return 0;  
}
```

4 - Définition des fonctions et procédures : reflets des algorithmes particuliers

```
int IndiceDuMin(TAB t, int n)  
{  
    int u = t[0], iMin = 0;  
  
    for(int i = 1; i < n; i = i+1)  
    {  
        if(t[i] < u)  
        {  
            u = t[i];  
            iMin = i;  
        }  
    }  
    return iMin;  
}
```

CORRESPONDANCES

Algo ↔ C

Le style est une façon très simple de dire des choses compliquées.

Jean Cocteau

Déclarations et affectations

Langage algorithmique	Langage C
<code>z : flottant</code>	<code>double z ;</code>
<code>z ← 16.24</code>	<code>z = 16.24 ;</code>
<code>i, j : entier</code>	<code>int i, j ;</code>
<code>n ← 6 : entier</code>	<code>int n = 6 ;</code>
<code>t, tab : tableau de n entier</code>	<code>int t[n], tab[n] ;</code>

Instructions et structures de contrôle

Langage algorithmique	Langage C
<code>t[i] ← t[i]+1</code>	<code>t[i] = t[i]+1 ;</code>
<code>Afficher("Impédance : ", z, " ohm")</code>	<code>cout << "Impedance : " << z << " ohm" ;</code>
<code>Saisir(z)</code>	<code>cin >> z ;</code>
<code>Pour i Dans [0..n[ParPasDe 1 ... FinPour</code>	<code>for(int i = 0; i < n; i = i + 1) { ... }</code>
<code>Faire ... TantQue (i < n)</code>	<code>do { ... } while(i < n) ;</code>
<code>TantQue (i < n) ... FinTantQue</code>	<code>while(i < n) { ... }</code>
<code>Si (i == j) ... Sinon ... FinSi</code>	<code>if(i == j) { ... } else { ... }</code>
<code>Retourner(z)</code>	<code>return z ;</code>
<code><u>commentaire</u></code>	<code>// commentaire</code>
<code>i ← Minimum(t, tab)</code>	<code>i = Minimum(t, tab) ;</code>

Fonctions : *signatures* algorithmiques / *prototypes* C

algo	DébutFonction Factorielle(→n : entier) : entier
C	<code>int Factorielle(int n) ;</code>
algo	DébutFonction Next(→t : tableau de flottant, →n : entier) : flottant
C	<code>float Next(float t[], int n) ;</code>
algo	DébutProcédure Echanger(↔x : flottant, ↔y : flottant)
C	<code>void Echanger(double& x, double& y) ;</code>
algo	DébutProcédure HeuresMinutes(→t : entier, ←h : entier, ←m : entier)
C	<code>void HeuresMinutes(int t, int& h, int& m) ;</code>

Taille et amplitude des types classiques

En utilisant l'instruction `sizeof`, ainsi que les constantes prédéfinies dans l'en-tête `limits.h`, on trouve les valeurs suivantes :

```
Dev-C++ / windows
-----

taille d'un <char>      : 1 octet(s)   [-128 .. 127]
taille d'un <unsigned char> : 1 octet(s)   [0 .. 255]
taille d'un <bool>     : 1 octet(s)
taille d'un <short>    : 2 octet(s)   [-32768 .. 32767]
taille d'un <unsigned short> : 2 octet(s)   [0 .. 65535]
taille d'un <int>      : 4 octet(s)   [-2147483648 .. 2147483647]
taille d'un <unsigned int> : 4 octet(s)   [0 .. 4294967295]
taille d'un <long>     : 4 octet(s)   [-2147483648 .. 2147483647]
taille d'un <unsigned long> : 4 octet(s)   [0 .. 4294967295]
taille d'un <float>    : 4 octet(s)   [1.E-36 .. 1.E+36]
taille d'un <double>   : 8 octet(s)   [1.E-303 .. 1.E+303]
```

FIG. 1 – Taille et amplitude des types classiques

Conseils et conventions d'écriture

Ce que l'on conçoit bien s'énonce clairement.

Nicolas Boileau

Mots-Clés du langage C

La description du langage C est basée sur la norme internationale ISO/IEC 9899 qui a été adoptée en 1990 et amendée en 1995 et 1999. La norme de 1995 est reconnue par tous les compilateurs courants. Les nouvelles extensions définies dans la version de 1999 (appelée ANSI C99) ne sont pas implantées dans beaucoup de compilateurs, et sont indiquées dans le tableau ci-dessus par un astérisque.

Les 37 mots-clés suivants sont réservés et ne doivent pas être utilisés comme identificateurs. Voici une liste complète :

auto	enum	restrict*	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	_Bool*
continue	if	static	_Complex*
default	inline*	struct	_Imaginary*
do	int	switch	
double	long	typedef	
else	register	union	

Identificateurs

Un identificateur est un nom (de variable, de constante, de type, de fonction...) **choisi** par le programmeur.

- Il se compose de *lettres* (a à z et A à Z), du caractère *soulignement* (_) ou de *chiffres* (0 à 9) ;
- le premier caractère doit être une lettre (ou un caractère *soulignement*, mais c'est une convention utilisée par certains compilateurs, donc à éviter) ;
- éviter les caractères accentués (peu portable) ;
- avec tout compilateur, se limiter à des identificateurs de 32 caractères ;
- ne pas oublier que le C distingue les majuscules des minuscules : `tab`, `Tab` et `TAB` sont trois identificateurs différents ;
- enfin il est interdit d'utiliser les mots clés réservés du langage C.

Voici quelques exemples d'identificateurs :

Valides :

```
a, DM, FLOAT, _var1, HautDeFenetre;
```

Non valides :

```
do, 586cpu, z\%hler, nl-flag, US_$.
```

Dans ce cadre, on reste libre de choisir ses identificateurs, à ceci près qu'il est commode et plus clair, pour la lecture du code, de s'imposer quelques conventions.

Choix des identificateurs : conseils et conventions

1. Constantes et nouveaux types

Ils sont conventionnellement écrits en majuscules :

```
const int MAX_ELEM = 5;
typedef double PRESSION;
```

2. Variables

(a) Variables à courte portée (compteurs de boucle, indices de tableau...). Les choisir courts et conventionnels (le code y gagne en clarté) : *i*, *j*, *k*, *cpt*...

(b) Variables à portée étendue (les autres).

- Choisir des noms expressifs ;
- commencer le nom par une minuscule ;
- si le nom est *composé*, adopter un style (par exemple : *jour_du_mois* ou bien *jourDuMois*), mais gardez-le : le choix du style est moins important pour la lisibilité que sa **cohérence**.

3. Fonctions

- Commencer le nom par une majuscule ;
- préférer des mots d'action, souvent des verbes.
Par exemple : `LireFichier()`, `RangeValeur()`, `AfficheListe()`...

L'indentation

On nomme ainsi les retraits en début de ligne destinés à faciliter la lecture des programmes et des algorithmes.

C'est une habitude qu'il faut s'imposer avec rigueur, elle permet d'extraire rapidement la structure (le *squelette*) du programme, la logique d'une boucle ou d'un test, et de détecter bien des oublis de fermeture de bloc.

Parmi plusieurs styles d'indentation on propose le suivant.

Choisissez votre style et... gardez-le !

```
DébutProgramme   Combinaisons à deux dés
  DébutDéclaration
     $i, j, n, s$  : entier
  FinDéclaration

   $n \leftarrow 0$ 

  Faire
    Afficher("Entrez un entier [2 .. 12] : ")
    Saisir( $n$ )
  TantQue(( $n < 2$ ) ET ( $n > 12$ ))

  Pour  $i$  Dans [1..6] ParPasDe 1
    Pour  $j$  Dans [1..6] ParPasDe 1
      Si ( $i + j == n$ )
         $s \leftarrow s + 1$ 
      FinSi
    FinPour
  FinPour
  Afficher("Il y a",  $s$ , " façons de faire ",  $n$ , " avec 2 dés.")
FinProgramme
```

```
#include <iostream>
using namespace std;

int main(void)
{
    int n, s = 0;

    do
    {
        cout << " Entrez un entier [2 .. 12] : ";
        cin >> n;
    }
    while(n < 2 || n > 12);

    for(int i = 1; i < 7; i++)
    {
        for(int j = 1; j < 7; j++)
        {
            if(i+j == n)
            {
                s++;
            }
        }
    }

    cout << "\n Il y a " << s << " facons de faire " << n << " avec 2 des.\n";

    system("PAUSE");
    return 0;
}
```

Ordre de priorité des opérateurs

Il est sage d'utiliser la **règle pratique** suivante :

Niveau de priorité	Opérateurs	arithmétiques
1	*	/ %
2	+	-

Entourer tout le reste de parenthèses !

Vous pouvez aussi vous servir du « truc » mnémotechnique suivant, l'acronyme **PEMDAS** :

P pour **parenthèses**. Elles ont la plus haute priorité ;

E pour **exposants**. Ils sont évalués ensuite, avant les autres opérations ;

M et **D** pour **multiplication** et **division**, qui ont la même priorité ;

A et **S** pour **addition** et **soustraction**, de même priorité et évaluées en dernier.

Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite.

Que faire en cas d'erreurs de compilation ?

Quand le compilateur indique des erreurs, *il a raison!* Lisez soigneusement les messages du compilateur (oui, même en anglais !), et pensez tout de suite aux points suivants :

- il manque souvent un point-virgule (à la ligne au-dessus, ou proche...), ou bien une parenthèse, une accolade... Utilisez l'option « Aide parenthèses/accolades relatives » (case à cocher dans les « Propriétés de l'Editeur » de Dev-C++);
- l'instruction `for (... ; ... ; ...)` comporte trois parties séparés par des points virgules ;
- quand il manque un `#include`, les noms correspondants sont inconnus. Voir la documentation intégrée de l'IDE ou consulter le fichier « libC.pdf » ;
- un nom inconnu est souvent mal orthographié (vérifier sa casse) ;
- confusions fréquentes entre = et == ou bien entre & et &&...

EXERCICES CORRIGÉS D'ALGORITHMIQUE

1. ÉNONCÉS

Remarque

Les exercices suivants vous sont fournis à titre d'exemples et de modèle. Utilisez-les pour vous entraîner.

Ils sont soit simples, soit moins simples (notés \triangleright dans la marge) soit difficiles (notés $\triangleright\triangleright$).

1. Écrire la trace de l'algorithme suivant. Que fait-t-il ?

Algorithm 8 Algorithme inconnu

DébutProgramme Trace d'un algorithme

DébutDéclaration

a, b, c : entier

FinDéclaration

a ← A

b ← B

c ← C

a ← a + b + c

b ← b + c

c ← a - c

a ← a - c

b ← c - b + a

c ← c - b

FinProgramme

2. Écrire un algorithme qui, à partir de la saisie d'un rayon et d'une hauteur, calcule le volume d'un cône droit.
3. En utilisant les opérateurs booléens ET, OU et NON, écrire un programme qui affiche la table de vérité du *ou exclusif*. Vérifiez que :
 $a \text{ XOR } b = (a \text{ OU } b) \text{ ET } (\text{NON}(a) \text{ OU } \text{NON}(b))$.
4. L'utilisateur donne un entier positif n et le programme affiche PAIR s'il est divisible par 2 et IMPAIR sinon.
5. L'utilisateur donne un entier positif et le programme annonce combien de fois de suite cet entier est divisible par 2.

- ▷ 6. L'utilisateur donne un entier supérieur à 1 et le programme affiche, s'il y en a, tous ses diviseurs propres *sans répétition* ainsi que leur nombre. Si il n'y en a pas, il indique qu'il est premier. Par exemple :
- ```
Entrez un entier > 1 : 12
Diviseurs propres (sans répétition) de 12 : 2 3 4 6
soit 4 diviseurs propres.
```
- ```
Entrez un entier > 1 : 13
Diviseurs propres (sans répétition) de 13 :
Il est premier.
```
7. Il s'agit d'écrire, d'une part, un programme principal, et d'autre part, une fonction utilisée dans le programme principal.
- L'utilisateur remplit un tableau de $N = 100$ entiers avec des entiers aléatoires en utilisant une fonction `AleaEntier(N)` qui retourne un entier entre 0 et $N - 1$. Une fonction nommée `IndiceDuMin()` reçoit ce tableau et retourne l'indice de la case qui contient le minimum.
- Écrire l'algorithme de la fonction `IndiceDuMin()`.
- Écrire l'algorithme d'un programme qui échange le premier élément du tableau avec le minimum de ce tableau.
8. Un tableau *tab* comporte $N = 100$ variables flottantes dont les n premières ($n < 100$) sont utilisées.
- Écrire une fonction `IndiceDuMax()` qui retourne l'indice du plus grand flottant parmi ces n , et une autre `IndiceDuMin()` qui retourne l'indice du plus petit.
- Écrire ensuite un programme principal effectuant les actions suivantes :
- saisie *filtrée* de n (vous devez faire en sorte que n ne puisse pas être saisi hors de ses limites) ;
 - remplissage aléatoire des n premières valeurs de *tab* (on utilisera une fonction `Aléa()`, sans argument, qui retourne un flottant au hasard entre -1.0 et $+1.0$) ;
 - affichage de *l'amplitude* du tableau (écart entre sa plus grande et sa plus petite valeur) ;
 - affichage de la moyenne des n premières valeurs de *tab*.
9. Saisir un entier entre 1 et 3999 (pourquoi cette limitation ?). L'afficher en nombre romain.
- ▷ 10. Un tableau contient n entiers ($2 < n < 100$), tous compris entre 0 et 500. Vérifier qu'ils sont tous différents.
11. L'utilisateur donne un entier n entre 2 et 12, le programme donne le nombre de façons de faire n en lançant deux dés.
12. Même problème que le précédent mais avec n entre 3 et 18 et trois dés.

- ▷▷ 13. Généralisation des deux questions précédentes. L'utilisateur saisie deux entrées, d'une part le nombre de dés, nb (que l'on limitera pratiquement à 10), et d'autre part la somme, s , comprise entre nb et $6.nb$. Le programme calcule et affiche le nombre de façons de faire s avec les nb dés.
14. On peut, par analogie avec les *tableaux*, définir une notation de manipulation de *matrice* : si m est une matrice, on note $m[i, j]$ la valeur qui se trouve à l'intersection de la i^e ligne et de la j^e colonne.
- On déclare trois matrices carrées de dimension N (pratiquement, on se limitera à $N_{max} = 10$) : $m1$, $m2$ et $m3$ contenant des entiers. On affecte $m1$, ligne à ligne, par les N^2 premiers entiers pairs commençant à 2 ; $m2$ est la matrice unité, c'est-à-dire qu'elle contient des 1 sur la diagonale principale (NW-SE) et des 0 partout ailleurs.
- Écrire l'algorithme du calcul de $m3 = m1 - m2$.



2. SOLUTIONS

N°	a	b	c
1	A		
2		B	
3			C
4	$A + B + C$		
5		$B + C$	
6			$A + B$
7	C		
8		A	
9			B

L'algorithme effectue une permutation circulaire à droite des valeurs d'entrée.

Algorithm 9 Volume d'un cône droit (Ex. n° 2)

DébutProgramme *Volume d'un cône droit*

DébutDéclaration

$\pi \leftarrow 3.141593$: flottant

rayon, hauteur, volume : flottant

FinDéclaration

Afficher("Rayon du cône : ")

Saisir(*rayon*)

Afficher("Hauteur du cône : ")

Saisir(*hauteur*)

$volume \leftarrow (\pi * rayon * rayon * hauteur) / 3$

Afficher("Volume du cône = ", *volume*)

FinProgramme

Algorithm 10 Table de vérité du *ou exclusif* : XOR (Ex. n° 3)

DébutProgramme Table de vérité de XOR
DébutDéclaration
 a, b : booléen
FinDéclaration

Afficher(" *a b* *a XOR b*")
a ← FAUX
b ← FAUX
Afficher(*a, b*, " ", (*a OU b*) ET (NON(*a*) OU NON(*b*)))
b ← VRAI
Afficher(*a, b*, " ", (*a OU b*) ET (NON(*a*) OU NON(*b*)))
a ← VRAI
b ← FAUX
Afficher(*a, b*, " ", (*a OU b*) ET (NON(*a*) OU NON(*b*)))
b ← VRAI
Afficher(*a, b*, " ", (*a OU b*) ET (NON(*a*) OU NON(*b*)))
FinProgramme

Algorithm 11 Pair ou impair ? (Ex. n° 4)

DébutProgramme Pair ou impair ?
DébutDéclaration
 n : entier entier à tester
FinDéclaration

Faire
 Afficher("Entrez un entier strictement positif")
 Saisir(*n*)
 TantQue(*n* < 1)

Si (*n*%2 == 0)
 Afficher("PAIR")
 Sinon
 Afficher("IMPAIR")
 FinSi
FinProgramme

Algorithm 12 Nombre de divisions par 2 (Ex. n° 5)

DébutProgramme Nombre de divisions par 2
DébutDéclaration
 n : entier entier à tester
 cpt : entier nombre de diviseurs par 2 de *n*
FinDéclaration

Faire
 Afficher("Entrez un entier strictement positif")
 Saisir(*n*)
TantQue(*n* < 1)

cpt ← 0 initialise le compteur

TantQue (*n*%2 == 0)
 n ← *n*/2
 cpt ← *cpt* + 1
FinTantQue

Afficher("Il est ", *cpt*, " fois divisible par 2.")
FinProgramme

Algorithm 13 Diviseurs propres sans répétition (Ex. n° 6)

DébutProgramme Diviseurs propres sans répétition
DébutDéclaration
 n, i, cpt : entier
FinDéclaration

Faire
 Afficher("Entrez un entier > 1")
 Saisir(*n*)
TantQue(*n* < 2)

i ← 2 plus petit diviseur propre possible de *n*
 cpt ← 0 initialise le compteur de diviseurs
 Afficher("Diviseurs propres (sans répétition) de ", *n*, " :")

Faire
 Si (*n*%*i* == 0)
 cpt ← *cpt* + 1
 Afficher("", *i*)
 FinSi
 i ← *i* + 1
TantQue(*i* ≤ *n*/2)

Si (*cpt* ≠ 0)
 Afficher("soit ", *cpt*, " diviseurs propres")
 Sinon
 Afficher("Il est premier")
 FinSi
FinProgramme

Algorithm 14 Fonction `IndiceDuMin()` d'un tableau d'entiers (Ex. n° 7)

DébutFonction `INDICEDUMIN`($\leftrightarrow t$: entier[N], $\rightarrow n$: entier) : entier

DébutDéclaration

$min, i, iMin$: entier

FinDéclaration

$min \leftarrow t[0]$

$iMin \leftarrow 0$

Pour i **Dans** $[1..n[$ **ParPasDe** 1

Si ($t[i] < min$)

$min \leftarrow t[i]$

$iMin \leftarrow i$

FinSi

FinPour

Retourner ($iMin$)

FinFonction

Algorithm 15 Échange de $t[0]$ avec $t[iMin]$ (Ex. n° 7)

DébutProgramme Échange de $t[0]$ avec $t[iMin]$

DébutDéclaration

$N \leftarrow 100$: entier

$i, iMin, tmp$: entier

t : entier[N]

FinDéclaration

Pour i **Dans** $[0..N[$ **ParPasDe** 1

$t[i] \leftarrow \text{AleaEntier}(N)$

FinPour

$iMin \leftarrow \text{IndiceDuMin}(t, N)$

$tmp \leftarrow t[0]$

$t[0] \leftarrow t[iMin]$

$t[iMin] \leftarrow tmp$

FinProgramme

Algorithm 16 Fonction `IndiceDuMin()` d'un tableau de flottant (Ex. n° 8)

DébutFonction `INDICEDUMIN`($\leftrightarrow t : \text{flottant}[N], \rightarrow n : \text{entier}$) : entier

DébutDéclaration

$min : \text{flottant}$

$i, iMin : \text{entier}$

FinDéclaration

$min \leftarrow t[0]$

$iMin \leftarrow 0$

Pour i **Dans** $[1..n[$ **ParPasDe** 1

Si ($t[i] < min$)

$min \leftarrow t[i]$

$iMin \leftarrow i$

FinSi

FinPour

Retourner ($iMin$)

FinFonction

Algorithm 17 Fonction `IndiceDuMax()` d'un tableau de flottant (Ex. n° 8)

DébutFonction `INDICEDUMAX`($\leftrightarrow t : \text{flottant}[N], \rightarrow n : \text{entier}$) : entier

DébutDéclaration

$max : \text{flottant}$

$i, iMax : \text{entier}$

FinDéclaration

$max \leftarrow t[0]$

$iMax \leftarrow 0$

Pour i **Dans** $[1..n[$ **ParPasDe** 1

Si ($t[i] > max$)

$max \leftarrow t[i]$

$iMax \leftarrow i$

FinSi

FinPour

Retourner ($iMax$)

FinFonction

Algorithm 18 Amplitude et moyenne d'un tableau de flottant (Ex. n° 8)

DébutProgramme *Amplitude et moyenne d'un tableau de flottant*

DébutDéclaration

$N \leftarrow 100$: entier
 $i, n, iMin, iMax$: entier
 tab : flottant[N]
 som : flottant

FinDéclaration

Faire

Afficher("Entrez un entier [2 .. 100]")

Saisir(n)

TantQue($n < 1$) OU ($n > N$)

$som \leftarrow 0$

Pour i **Dans** $[0..n[$ **ParPasDe** 1

$tab[i] \leftarrow Alea()$

$som \leftarrow som + tab[i]$

FinPour

$iMin \leftarrow IndiceDuMin(tab, n)$

$iMax \leftarrow IndiceDuMax(tab, n)$

Afficher("Amplitude de tab : ", $tab[iMax] - tab[iMin]$)

Afficher("Moyenne des ", n , " premières valeurs de tab : ", som/n)

FinProgramme

Algorithm 19 Numération romaine (Ex. n° 9)

DébutProgramme *Numération romaine*
DébutDéclaration
 n : entier
FinDéclaration

Faire
 Afficher("Entrez un entier [1..4000]")
 Saisir(n)
TantQue($n < 1$ OU $n > 3999$)

TantQue ($n \geq 1000$)
 Afficher("M")
 $n \leftarrow n - 1000$
 FinTantQue

Si ($n \geq 900$)
 Afficher("CM")
 $n \leftarrow n - 900$
 FinSi

Si ($n \geq 500$)
 Afficher("D")
 $n \leftarrow n - 500$
 FinSi

Si ($n \geq 400$)
 Afficher("CD")
 $n \leftarrow n - 400$
 FinSi

TantQue ($n \geq 100$)
 Afficher("C")
 $n \leftarrow n - 100$
 FinTantQue

 ...

TantQue ($n \geq 1$)
 Afficher("I")
 $n \leftarrow n - 1$
 FinTantQue
FinProgramme

Algorithm 20 Tableau d'éléments différents (Ex. n° 10)

DébutProgramme *Tableau d'éléments différents***DébutDéclaration** $N \leftarrow 100$: entier n : entier nombre de valeurs utilisées T : entier[N] on suppose T déjà garni $Verif$: booléen[N] i : entier $stop$: booléen**FinDéclaration****Pour** i **Dans** $[0..N[$ **ParPasDe** 1 $Verif[i] \leftarrow$ FAUX**FinPour** $stop \leftarrow$ FAUX $i \leftarrow 0$ **TantQue** $((stop ==$ FAUX) ET $(i < n))$ **Si** $(Verif[T[i]])$ $stop \leftarrow$ VRAI**Sinon** $Verif[T[i]] \leftarrow$ VRAI**FinSi** $i \leftarrow i + 1$ **FinTantQue****Si** $(stop)$ **Afficher**("Au moins une valeur est répétée")**Sinon****Afficher**("Tous les éléments sont distincts")**FinSi****FinProgramme**

Algorithm 21 Combinaisons à 2 dés (Ex. n° 11)

DébutProgramme Combinaisons à 2 dés
DébutDéclaration
 i, j, n : entier
 $s \leftarrow 0$: entier somme cherchée
FinDéclaration

Faire
 Afficher("Entrez un entier [2 .. 12]")
 Saisir(n)
TantQue($n < 2$) OU ($n > 12$)

 Pour i **Dans** [1..6] **ParPasDe** 1
 Pour j **Dans** [1..6] **ParPasDe** 1
 Si ($i + j == n$)
 $s \leftarrow s + 1$
 FinSi
 FinPour
 FinPour

 Afficher("Il y a ", s , " façons de faire ", n , " avec 2 dés")
FinProgramme

Algorithm 22 Combinaisons à 3 dés (Ex. n° 12)

DébutProgramme Combinaisons à 3 dés
DébutDéclaration
 i, j, k, n : entier
 $s \leftarrow 0$: entier somme cherchée
FinDéclaration

Faire
 Afficher("Entrez un entier [3 .. 18]")
 Saisir(n)
TantQue($n < 3$) OU ($n > 18$)

 Pour i **Dans** [1..6] **ParPasDe** 1
 Pour j **Dans** [1..6] **ParPasDe** 1
 Pour k **Dans** [1..6] **ParPasDe** 1
 Si ($i + j + k == n$)
 $s \leftarrow s + 1$
 FinSi
 FinPour
 FinPour
 FinPour

 Afficher("Il y a ", s , " façons de faire ", n , " avec 3 dés")
FinProgramme

Algorithm 23 Combinaisons à n dés (Ex. n° 13)

DébutProgramme *Combinaisons à n dés***DébutDéclaration** $MAX \leftarrow 10$: entier $M \leftarrow 6$: entier nb , s , j , k , sum , cpt : entier I : entier[MAX]**FinDéclaration****Faire****Afficher**("Nombre de dés entre 2 et ", MAX)**Saisir**(nb)**TantQue**(($nb < 2$)OU($nb > MAX$))**Faire****Afficher**("Entrez un entier entre", nb , " et ", $M * nb$)**Saisir**(s)**TantQue**(($s < nb$)OU($s > M * nb$))**Pour** j **Dans** $[0..nb[$ **ParPasDe** 1 $I[j] \leftarrow 1$ **FinPour** $cpt \leftarrow 0$ **Faire** $sum \leftarrow 0$ **Pour** k **Dans** $[0..nb[$ **ParPasDe** 1 $sum \leftarrow sum + I[k]$ **FinPour****Si** ($sum == s$) $cpt \leftarrow cpt + 1$ **FinSi** $j \leftarrow 0$ **Si** ($I[j] < M$) $I[j] \leftarrow I[j] + 1$ **Sinon****TantQue** ($I[j] == M$) $I[j] \leftarrow 1; j \leftarrow j + 1$ **FinTantQue** $I[j] \leftarrow I[j] + 1$ **FinSi****TantQue**($j < nb$)**Afficher**("Il y a ", cpt , " façons de faire ", s , " avec ", nb , " dés")**FinProgramme**

Algorithm 24 Différence de deux matrices carrées (Ex. n° 14)

DébutProgramme *Différence de deux matrices carrées***DébutDéclaration** $N_{MAX} \leftarrow 10$: entier N, i, j, k : entier $m1, m2, m3$: matrice de $N_{MAX} \times N_{MAX}$ entier**FinDéclaration****Faire** **Afficher**("Dimension des matrices : ") **Saisir**(N)**TantQue**($N < 2$) *OU* ($N > N_{MAX}$) $k \leftarrow 2$ **Pour** i **Dans** $[0..N[$ **ParPasDe** 1 **Pour** j **Dans** $[0..N[$ **ParPasDe** 1 $m1[i, j] \leftarrow k$ $k \leftarrow k + 2$ **Si** ($i == j$) $m2[i, j] \leftarrow 1$ **Sinon** $m2[i, j] \leftarrow 0$ **FinSi** $m3[i, j] \leftarrow m1[i, j] - m2[i, j]$ **Afficher**("m3[" i ," j "]=" $m3[i, j]$ ") **FinPour****FinPour****FinProgramme**

PROGRAMMES

```
// Bob - algo_01.cpp

#include <iostream>
using namespace std;

int main(void)
{
    char a='A', b='B', c='C';

    cout << " a = " << a
         << "\t b = " << b
         << "\t c = " << c << endl;

    a = a + b + c;
    b = b + c;
    c = a - c;
    a = a - c;
    b = c - b + a;
    c = c - b;

    cout << "\n a = " << a
         << "\t b = " << b
         << "\t c = " << c << endl;

    cout << endl;
    system("PAUSE");
    return 0;
}
```

```
// Bob - algo_02.cpp

#include <iostream>
using namespace std;
#include <math.h>

using namespace std;

int main(void)
{
    double rayon, hauteur, volume;

    cout << " Rayon du cone (m) : ";
    cin >> rayon;
    cout << " Hauteur du cone (m) : ";
    cin >> hauteur;

    volume = (M_PI*rayon*rayon*hauteur)/3.0;

    cout << "\n volume du cone = " << volume << " m*m" << endl;

    cout << endl;
    system("PAUSE");
    return 0;
}
```

```
// Bob - algo_03.cpp

#include <iostream>
using namespace std;

int main(void)
{
    bool a, b;

    cout << " a\tb\ta XOR b\n-----\n";

    a = b = false;
    cout << " " << a << "\t" << b << "\t " << ((a || b) && (!a || !b)) << endl;
    b = true;
    cout << " " << a << "\t" << b << "\t " << ((a || b) && (!a || !b)) << endl;
}
```

```

a = true; b = false;
cout << " " << a << "\t" << b << "\t " << ((a || b) && (!a || !b)) << endl;
b = true;
cout << " " << a << "\t" << b << "\t " << ((a || b) && (!a || !b)) << endl;

cout << endl;
system("PAUSE");
return 0;
}

```

```

// Bob - algo_04.cpp

#include <iostream>
using namespace std;

int main(void)
{
    int n;

    do
    {
        cout << " Entrez un entier strictement positif : ";
        cin >> n;
    } while(n < 1);

    if(n % 2)
    {
        cout << "\n IMPAIR\n";
    }
    else
    {
        cout << "\n PAIR\n";
    }

    cout << endl;
    system("PAUSE");
    return 0;
}

```

```

// Bob - algo_05.cpp

#include <iostream>
using namespace std;

int main(void)
{
    int n, cpt=0;

    do
    {
        cout << " Entrez un entier strictement positif : ";
        cin >> n;
    } while(n < 1);

    while(!(n % 2))
    {
        n = n / 2;
        cpt = cpt + 1;
    }
    cout << "\n Il est " << cpt << " fois divisible par 2.\n";

    cout << endl;
    system("PAUSE");
    return 0;
}

```

```

// Bob - algo_06.cpp

#include <iostream>
using namespace std;

int main(void)
{
    int n, i=2, ns2, cpt=0;

```

```

do
{
    cout << " Entrez un entier strictement positif : ";
    cin >> n;
} while(n < 1);

ns2 = n/2;
cout << "\n Diviseurs propres (sans repetition) de " << n << " :";

do
{
    if(!(n % i))
    {
        cpt = cpt + 1;
        cout << " " << i;
    }
    i = i + 1;
} while(i <= ns2);

if(cpt)
    cout << "\n soit " << cpt << " diviseurs propres.\n";
else
    cout << "\n Il est premier.\n";

cout << endl;
system("PAUSE");
return 0;
}

```

```

// Bob - algo_07.cpp

#include <iostream>
using namespace std;

int main(void)
{
    const int N = 100;
    int i, iMin, tmp, t[N];
    // prototype local
    int IndiceDuMin(int [], // tableau d'entier (entrée/sortie)
                    int); // taille du tableau (entrée)

    srand(time(NULL));

    for(i = 0; i < N; i = i + 1)
    {
        t[i] = rand() % N;
    }

    iMin = IndiceDuMin(t, N);
    cout << " Avant : t[0] = " << t[0] << "\t t[iMin] = " << t[iMin] << endl;
    tmp = t[0];
    t[0] = t[iMin];
    t[iMin] = tmp;
    cout << " Apres : t[0] = " << t[0] << "\t t[iMin] = " << t[iMin] << endl;

    cout << endl;
    system("PAUSE");
    return 0;
}

int IndiceDuMin(int t[], int n)
{
    int u = t[0], iMin = 0;

    for(int i = 1; i < n; i = i + 1)
    {
        if(t[i] < u)
        {
            u = t[i];
            iMin = i;
        }
    }
    return iMin;
}

```

```

// Bob - algo_08.cpp

#include <iostream>
using namespace std;

int main(void)
{
    const int N = 100;
    int n, iMin, iMax;
    float tab[N], som = 0.0;
    int IndiceDuMin(float [], // tableau de flottant (entrée/sortie)
                    int); // taille du tableau (entrée)
    int IndiceDuMax(float [], // tableau de flottant (entrée/sortie)
                    int); // taille du tableau (entrée)
    float Alea(int); // (entree)

    srand(time(NULL));
    do
    {
        cout << " Entrez un entier [2 .. 100] : ";
        cin >> n;
    } while(n < 1 || n > N);

    for(int i = 0; i < N; i = i + 1)
    {
        tab[i] = Alea(n);
        som = som + tab[i];
    }

    iMin = IndiceDuMin(tab, n);
    iMax = IndiceDuMax(tab, n);
    cout << "\n Amplitude de tab = " << (tab[iMax]-tab[iMin]) << endl;
    cout << " Moyenne des " << n << " premieres valeurs de tab = "
         << (som/n) << endl;

    cout << endl;
    system("PAUSE");
    return 0;
}

int IndiceDuMin(float t[], int n)
{
    int iMin = 0;
    float u = t[0];

    for(int i = 1; i < n; i = i + 1)
    {
        if(t[i] < u)
        {
            u = t[i];
            iMin = i;
        }
    }
    return iMin;
}

int IndiceDuMax(float t[], int n)
{
    int iMax = 0;
    float u = t[0];

    for(int i = 1; i < n; i = i + 1)
    {
        if(t[i] > u)
        {
            u = t[i];
            iMax = i;
        }
    }
    return iMax;
}

float Alea(int k)
{
    return (float)(rand() % k);
}

```

```

// Bob - algo_09.cpp

```

```

#include <iostream>
using namespace std;

#define M 1000
#define CM 900
#define D 500
#define CD 400
#define C 100
#define XC 90
#define L 50
#define XL 40
#define X 10
#define IX 9
#define V 5
#define IV 4
#define I 1

int main(void)
{
    int n;

    do
    {
        cout << " Entrez un entier [1 .. 4000] : ";
        cin >> n;
    } while(n < 1 || n > 3999);

    cout << "\n En Romain : ";

    while( n >= M) { cout << "M"; n -= M; }
    if( n >= CM) { cout << "CM"; n -= CM; }
    if( n >= D) { cout << "D"; n -= D; }
    if( n >= CD) { cout << "CD"; n -= CD; }
    while( n >= C) { cout << "C"; n -= C; }
    if( n >= XC) { cout << "XC"; n -= XC; }
    if( n >= L) { cout << "L"; n -= L; }
    if( n >= XL) { cout << "XL"; n -= XL; }
    while( n >= X) { cout << "X"; n -= X; }
    if( n >= IX) { cout << "IX"; n -= IX; }
    if( n >= V) { cout << "V"; n -= V; }
    if( n >= IV) { cout << "IV"; n -= IV; }
    while( n >= I) { cout << "I"; n -= I; }

    cout << endl << endl;
    system("PAUSE");
    return 0;
}

```

```

// Bob - algo_10.cpp

#include <iostream>
using namespace std;

int main(void)
{
    const int N = 100;
    int i, j, n, T[N];
    bool tousDiff = true;

    do
    {
        cout << " Entrez un entier [1 .. 100] : ";
        cin >> n;
    } while(n < 1 || n > N);

    srand(time(NULL));
    for(int k = 0; k < n; k = k + 1)
    {
        T[k] = rand() % 501; // dans [0 .. 500]
    }

    i = 0;
    while(tousDiff && (i < n-1))
    {
        j = i + 1;
        while(tousDiff && (j < n))
        {
            if(T[i] == T[j])

```

```

    {
        tousDiff = false;
    }
    else
    {
        j = j + 1;
    }
}
i= i + 1;
}

if(tousDiff)
{
    cout << "\n Tous les elements sont distincts.\n";
}
else
{
    cout << "\n Au moins une valeur est repetee.\n";
}

cout << endl;
system("PAUSE");
return 0;
}

```

```

// Bob - algo_11.cpp

#include <iostream>
using namespace std;

int main(void)
{
    int n, s=0;

    do
    {
        cout << " Entrez un entier [2 .. 12] : ";
        cin >> n;
    } while(n < 2 || n > 12);

    for(int i = 1; i < 7; i = i + 1)
    {
        for(int j = 1; j < 7; j = j + 1)
        {
            if(i+j == n)
            {
                s = s + 1;
            }
        }
    }

    cout << "\n Il y a " << s << " facons de faire " << n << " avec 2 des.\n";

    cout << endl;
    system("PAUSE");
    return 0;
}

```

```

// Bob - algo_12.cpp

#include <iostream>
using namespace std;

int main(void)
{
    int n, s=0;

    do
    {
        cout << " Entrez un entier [3 .. 18] : ";
        cin >> n;
    } while(n < 3 || n > 18);

    for(int i = 1; i < 7; i = i + 1)
    {
        for(int j = 1; j < 7; j = j + 1)
        {

```

```

    for(int k = 1; k < 7; k = k + 1)
    {
        if(i+j+k == n)
        {
            s = s + 1;
        }
    }
}

cout << "\n Il y a " << s << " facons de faire " << n << " avec 3 des.\n";

cout << endl;
system("PAUSE");
return 0;
}

```

```

// Bob - algo_13.cpp

#include <iostream>
using namespace std;

int main(void)
{
    const long MAX = 10;
    const long M = 6;
    long nbd, s, j, sum;
    long cpt = 0;
    long I[MAX];

    do
    {
        cout << "\n Nombre de des entre 2 et " << MAX << " ?\t\t";
        cin >> nbd;
    } while(nbd < 2 || nbd > MAX);

    do
    {
        cout << "\n Entrez un entier entre " << nbd << " et " << M*nbd << " ?\t";
        cin >> s;
    } while(s < nbd || s > M*nbd);

    if((s == nbd) || (s == M*nbd))
        cpt = 1;
    else
    {
        for(long k = 0; k < nbd; ++k)
            I[k] = 1;

        do
        {
            sum = 0;
            for(long k = 0; k < nbd; k = k + 1)
                sum = sum + I[k];

            if(sum == s)
                cpt = cpt + 1;

            j = 0;
            if( I[j] < M)
                I[j] = I[j] + 1;
            else
            {
                while(I[j] == M)
                {
                    I[j] = 1;
                    j = j + 1;
                }
                I[j] = I[j] + 1;
            }
        } while(j < nbd);
    }

    cout << "\n Il y a " << cpt << " facons de faire " << s
        << " avec " << nbd << " des.\n";

    cout << endl;
    system("PAUSE");
    return 0;
}

```

```
}
```

```
// Bob - algo_14.cpp
#include <iostream>
using namespace std;

int main(void)
{
    const int N_MAX = 10;
    int N;
    int m1[N_MAX][N_MAX], m2[N_MAX][N_MAX], m3[N_MAX][N_MAX];

    do
    {
        cout << " Dimension des matrices carrees [2 .. " << N_MAX << "] : ";
        cin >> N;
    } while(N < 2 || N > N_MAX);

    for(int i = 0, k = 2; i < N; i = i + 1)
    {
        for(int j = 0; j < N; j = j + 1)
        {
            m1[i][j] = k;
            k = k + 2;
            if(i == j)
            {
                m2[i][j] = 1;
            }
            else
            {
                m2[i][j] = 0;
            }
            m3[i][j] = m1[i][j] - m2[i][j];
        }
    }

    cout << "\n\n m3 = m1 - m2 :\n\n";
    for(int i = 0; i < N; i = i + 1)
    {
        for(int j = 0; j < N; j = j + 1)
        {
            cout << "\t" << m3[i][j];
            if(j == N-1)
            {
                cout << "\n";
            }
        }
        cout << "\n\n";
    }

    cout << endl;
    system("PAUSE");
    return 0;
}
```