



Licence STS

Université Claude Bernard Lyon I

# LIFAP1 : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION



COURS 2 : Bases du langage C



# OBJECTIFS DE LA SÉANCE

- Apprendre la syntaxe du langage C
- Savoir traduire un algorithme en langage C
- Vous permettre de pouvoir débiter les séances de travaux pratiques
  - Environnement Code::Blocks
  - OS : windows



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C



# UN PEU D'HISTOIRE

- 1945, les programmes étaient écrits directement en code machine...
- 1954 : FORTRAN 1
- 1978 : “The C programming language”
  - Brian W. Kernighan et Dennis M. Ritchie
- 1983-1988 : Normalisation ANSI (avec C++)
- 1988 : “The C programming language : 2ème édition”
  - des mêmes auteurs
- Évolutions permanentes



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C

# PREMIER EXEMPLE EN C

## premierexemple.cpp

- #include <iostream.h>

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

```
int main (void)
{
    cout << "Hello world" << endl ;
    return 0 ;
}
```

- Mots clé du langage :
  - Identifiés par une couleur particulière dans l'interface Code::Blocks
- Anatomie d'un programme C :
  - Nom de fichier d'**extension .cpp**
  - Utilisation de bibliothèques (déclarations des fonctions externes) : directive **#include**
    - Pour entrées / sorties
    - Pour opérations mathématiques
    - ...
  - Définitions des **sous-programmes** (fonctions et procédures) → CM 3
  - Définition de la fonction principale **main**

# PREMIER EXEMPLE EN C

premierexemple.cpp

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

int main (void)
{
    cout << "Hello world" << endl ;
    return 0;
}
```

- Fonction principale : main
  - Instruction particulière  
exit ou return (EXIT\_SUCCESS)
    - indique si le programme s'est déroulé et terminé normalement
- Délimitation des blocs par { et }
- Toutes les instructions se terminent par un ";"
- cout : permet d'afficher un message à l'écran (c++) (cout <<)
  - "...": chaîne de caractères
  - On peut avoir plusieurs <<
  - endl : constante C++ permet de passer à la ligne suivante (end-line) après avoir écrit le message



# UN PROGRAMME C

- Suites ordonnées de déclarations ou de définitions
  - de types,
  - de variables,
  - de sous-programmes (CM 3)
- Une fonction particulière : main
  - première fonction appelée lors de l'exécution
  - appelle les autres sous-programmes
- Tout nom doit être déclaré avant d'être utilisé
  - type, variable, fonction
- Préprocesseur : #include
  - directive de compilation pour l'inclusion des déclarations des sous-programmes prédéfinis dans les bibliothèques



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C



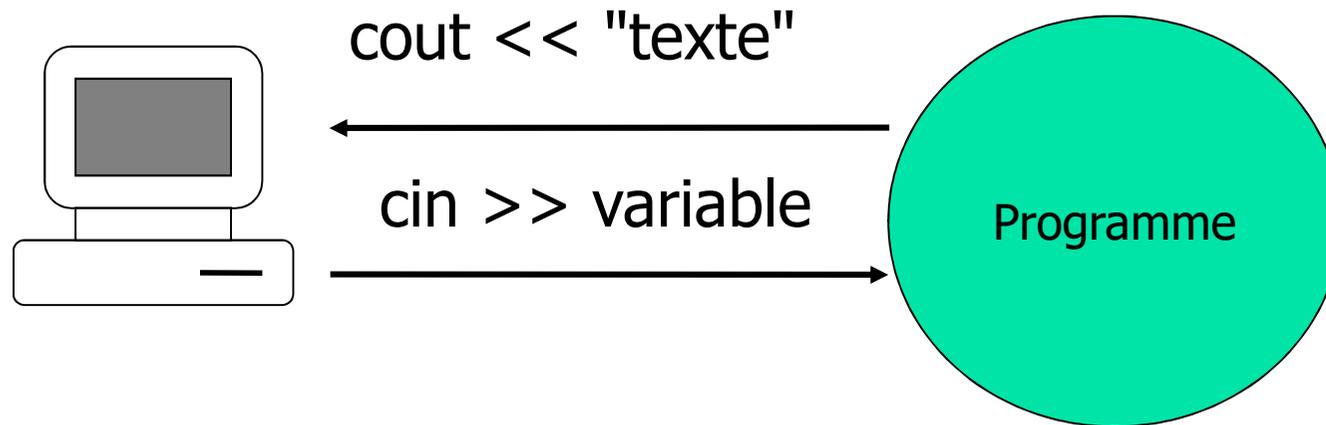
# LES ENTRÉES / SORTIES EN C

- Communication programme / utilisateur
- Traduction de "afficher" et "lire" de l'algorithmique
  - **cout** : permet d'afficher un message à l'écran
    - Exemple : on veut afficher un message de bienvenue à l'utilisateur :  

```
cout << "bienvenue" ;
```
  - **cin** : permet de récupérer une valeur fournie par l'utilisateur
    - Exemple : on veut demander à l'utilisateur une valeur en vue de calculer sa factorielle  

```
cin >> nomvariable ;
```

# LES ENTRÉES / SORTIES EN C



- Attention :
  - << pour cout (du programme vers l'écran)
  - >> pour cin (du clavier vers le programme)

# LES ENTRÉES / SORTIES EN C

- Dans un "cout" on peut mettre
  - Des chaînes de caractères : `cout <<"bienvenue" ;`
  - Le contenu de variables : `cout << a ;`
  - Une constante qui permet de passer à la ligne : `cout << endl ;`
- On peut mélanger ces trois types d'affichage dans un même cout ; il suffit de répéter les "<<" :
  - Ex : `cout <<"bienvenue " << a << endl ;`
  - si la variable a contient "Pierre" alors on affichera sur la même ligne "bienvenue Pierre" puis on passera à la ligne suivante avec endl.
- On peut saisir plusieurs valeurs à la suite dans un "cin" :  
`cin >>a >>b >>c` permettra de saisir les trois variables a, b et c.

`cin >> "bienvenue" ;` est une erreur, pourquoi ?



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C

# CORRESPONDANCE DES TYPES

<b>Algorithmique</b>	<b>Langage C</b>
Entier	int, short
Réel	float, double (différence sur la longueur du codage et donc de la précision)
Booleén	bool
Caractère	char
Chaîne de caractères	char[]
Tableau de 5 entiers	int[5]



# DÉCLARATION DE VARIABLES

- algorithmique
  - v : entier
  - x : réel
  - lx : réel
  - c : caractère
  - tab : tableau [10] d'entiers
- langage C
  - int v ;
  - float x ; (4 octets)
  - double lx ; (8 octets)
  - char c ;
  - int tab[10] ;

Octet : vecteur de huit bits (chiffre binaire)

pouvant représenter  $2^8$  (256) valeurs différentes

# STOCKAGE DES VARIABLES

- Les variables sont stockées dans la mémoire vive de l'ordinateur
- Place occupée fonction du type de la variable
- En mémoire, les variables sont repérées par leur emplacement ou adresse
  - Soient les déclarations suivantes :
    - `int a;`
    - `char toto, indice;`
    - `float fact;`
  - Les variables seront stockées en mémoire de la manière suivante :

Type	Taille (octets)
Char	1
Short, ...	2
Int, long, float	4
Double	8
Long double	10

		a	a	a	a	Toto	indice	fact	fact
...	i-1	i	i+1	i+2	i+3	i+4	i+5	i+6	...



# LES CONSTANTES EN C

- Une constante = nom désignant une valeur non modifiable lors de l'exécution d'un programme.
- Définie grâce à la directive du préprocesseur `#define`, qui permet de remplacer toutes les occurrences du mot qui le suit par la valeur immédiatement derrière elle.
  - `#define PI 3.1415927` remplacera tous les identifiants « PI » (sans les guillemets) par la valeur 3.1415927
- Toutefois, avec cette méthode les constantes ne sont pas typées
  - ➔ utiliser le mot clé `const`, qui permet de déclarer des constantes typées :
    - `const int DIX = 10;`



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C



# IDENTIFICATEURS

- Nom
  - de variable,
  - de constantes.
- Chaîne de caractères
  - commençant par une lettre (majuscule ou minuscule)
  - constituée de lettres [a-zA-Z], de chiffres [0-9], de \_,
  - sans accent, ni espace, ni –
- Attention : MAJUSCULES et minuscules différenciées
- Convention : nom de constante en majuscule
- Un identificateur doit être **évocateur** de ce qu'il représente
  - discriminant : identificateur correct pour variable de calcul du discriminant;  
nom\_etudiants : identificateur correct...
  - Fgmqsdgfk, rapidos, tempo, variable : à éviter !!!



# INSTRUCTION SIMPLE

- Expression ;
- ; : à la fin de chaque instruction simple
- Exécutées séquentiellement de haut en bas (sens de la lecture)
- L'expression est généralement une affectation
  - $x = 3 ;$



# BLOC D'INSTRUCTIONS

- Bloc : séquence d'instructions entre { ... }
- Considéré comme une instruction
- À utiliser systématiquement dans les instructions complexes
- Instructions exécutées du début à la fin du bloc
- Les blocs peuvent être imbriqués



# EXEMPLE : CALCUL DE MOYENNE

- On veut demander à l'utilisateur de donner deux valeurs a et b et on lui affichera la moyenne de ces deux valeurs

```
float a, b, moyenne;  
cout << "Donnez deux valeurs";  
cin >> a >> b;  
moyenne = (a + b) / 2;  
cout << "la moyenne est : " << moyenne;
```



# STRUCTURES DE CONTRÔLE

- Constructions du langage algorithmique
- Alternative :
  - Si-alors-sinon
- Itérations
  - Tant que ... Faire ...FinTantQue
  - Faire ... Tant que ...
  - Pour ... de ... à ... pas de ... Faire ...FinPour
- Sélection
  - Selon... autrement ... Fin Selon



# TRADUCTION DE L'ALTERNATIVE

Si expressionCondition

Alors

    Action(s) 1

Sinon

    Action(s) 2

FinSi

```
if ( expressionCondition )
```

```
{
```

```
    Action(s) 1
```

```
}
```

```
else
```

```
{
```

```
    Action(s) 2
```

```
}
```

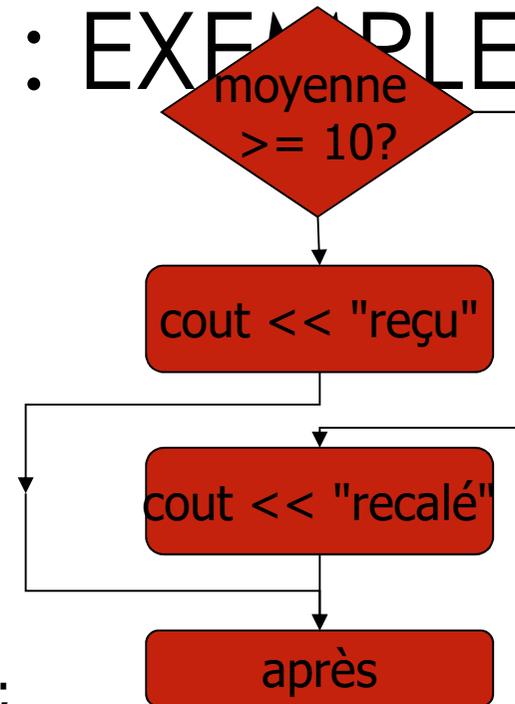
- **Attention** : ne pas mettre de ; après la condition !!
- Comme en algorithmique : partie else (sinon) pas forcément nécessaire

# ALTERNATIVE : EXEMPLE

- Si la moyenne de deux notes est supérieure ou égal à 10 on affiche reçu sinon on affiche recalé.

```
si (moyenne >=10)
alors
    afficher (reçu)
sinon
    afficher (recalé)
Fin si
```

```
if (moyenne >=10)
{
    cout << "reçu";
} else {
    cout << "recalé";
}
```





# TRADUCTION DE TANTQUE...FAIRE

TantQue ExpressionCondition Faire  
Action(s)  
FinTantQue

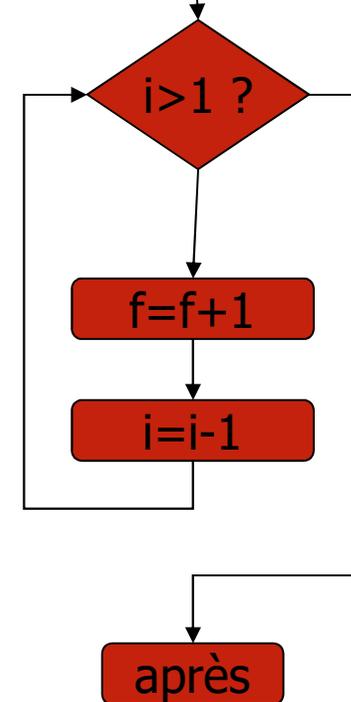
## **Traduction en C :**

```
while (expressionCondition)
{
    Action(s)
}
```

# TRADUCTION DE TANTQUE...FAIRE

TantQue  $i > 1$  Faire  
     $f \leftarrow f+i$   
     $i \leftarrow i-1$   
FinTantQue

```
while (i>1)
{
    f=f+i;
    i=i-1;
}
```





# TRADUCTION DE FAIRE ... TANTQUE

Faire

Action(s)

TantQue

expressionCondition

do

{

Action(s)

}

while(expressionCondition)

;

La séquence d'instructions "Action(s)" est effectuée au minimum une fois puisque l'évaluation de la condition est effectué au sortir de la boucle.

# BOUCLE À NOMBRE D'ITÉRATIONS CONNU

```
for ( instruction1 ;  
      expressionCondition ;  
      instruction2 )  
{  
    Action(s)  
}
```

```
/* initialisation */  
instruction 1 ;  
/* boucle */  
while ( expressionCondition )  
{  
    Action(s)  
}  
  
instruction 2 ; /* pas de  
l'itération */  
}
```

Deux méthodes pour écrire la même chose

Nombre d'itérations connu au départ

# EXEMPLE POUR ÉCRIRE LES NOMBRES DE 1 À 10

```
#include <iostream.h>
#include <stdlib.h>
int main()
{
    int i;
    for (i = 1; i <=10; i=i+1)
    {
        cout << i << " ";
    }
    return 0 ;
}
```

```
#include <iostream.h>
#include <stdlib.h>
int main()
{
    int i;
    i = 1;
    while ( i <=10)
    {
        cout << i <<" " ;
        i = i + 1 ;
    }
    return 0 ;
}
```



# SELON CHOIX

```
switch ( expression )
{
    case e1 :
        Action(s) 1
        break ;
    case e2 :
        Action(s) 2
        break ;
    ....
    default :
        Action(s) par défaut
}
```

- En algorithmique : selon choix
- **expression** est une expression entière quelconque
- **break** permet de sortir du **switch** ; sinon on continue à exécuter la ligne suivantes
- **default** : au cas où l'expression ne prendrait aucune des valeurs définies

# SELON CHOIX : EXEMPLE

## **selon** jour

- 1 : afficher('Lundi')
- 2 : afficher('Mardi')
- 3 : afficher('Mercredi')
- 4 : afficher('Jeudi')
- 5 : afficher('Vendredi')
- 6 : afficher('Samedi')
- 7 : afficher('Dimanche')

## **autrement :**

afficher('Erreur')

## **fin selon**

```
switch (jour)
{
    case 1 : cout<<"lundi";
            break;
    case 2 : cout<<"mardi";
            break;
    case 3 : cout<<"mercredi";
            break;
    case 4 : cout<<"jeudi";
            break;
    case 5 : cout<<"vendredi";
            break;
    case 6 : cout<<"samedi";
            break;
    case 7 : cout<<"dimanche";
            break;
    default : cout<<"erreur";
            break;
}
```



# LES COMMENTAIRES EN C

- Objectifs
  - Expliquer comment fonctionne le programme
  - Justifier les choix qui ont été faits
  - S'y retrouver quand on reprend un programme
- Bloc de commentaires sur plusieurs lignes délimités par `/* et */` : exemple

```
/* blablabla
   ici on calcule ...*/
```
- Commentaire en fin de ligne

```
// commentaire
```
- À utiliser sans modération !!



# EXPRESSIONS ENTIÈRES

- Opérations réalisables sur les entiers
- Types : int, short
- Opérations arithmétiques :
  - Opérations mathématiques standards : +, -, \*, /
  - Modulo = reste de la division entière (%)
    - $(12 \% 5) = 2$
    - $(5 \% 12) = 5$
- Résultat entier si opérandes entiers :  $1 / 2 == 0$



# EXPRESSIONS RÉELLES

- Opérations réalisables sur les réels
- Types : float, double
  - float : stockages
  - double : calcul (plus de précision)
- Opérations arithmétiques : +, -, \*, /



# TYPAGE ET CONVERSION IMPLICITE

- Calcul :
  - 2 opérandes int : résultat int
  - 2 opérandes float ou double : résultat double
  - opérande entier et opérande réel : résultat double
- Affectation :
  - entier dans une variable réelle : conversion
    - $a=2$
    - si a réel alors  $a=2.0$
  - réel dans une variable entière : on enlève la partie décimal
    - $ent = 2,245$
    - si ent entier alors  $ent = 2 !!!$



# OPÉRATEURS RELATIONNELS

- égal : `==`
- différent : `!=`
- inférieur ou égal : `<=`
- inférieur strictement : `<`
- supérieur ou égal : `>=`
- supérieur strictement : `>`
  
- attention : ne pas utiliser `==` avec les réels
  - Ex  $((1/3)^3)/3 \dots$  égal? 1 ...
  - problème de précision dans le codage des réels (cf LIFAP3)



# OPÉRATEURS LOGIQUES

- et : `&&` (et commercial, perluète ou esperluette)
- ou : `||` (pipe, 2 barres verticales) (Alt GR + 6)
- non : `!`
  
- Utiliser les parenthèses pour respecter les priorités des opérateurs



# BOOLÉEN

- Pas vraiment de booléen en C (même si le type bool existe)
- Codage des valeurs booléennes dans les entiers :
  - 0 : faux
  - autres valeurs, souvent 1 : vrai ( !0 == 1)
- Stockage dans un entier :
  - int b ;
  - b = ( (a < 2) && (i > 10) )
  - b aura pour valeur soit 0 ou soit 1



# PLAN

- Historique du C
- Un programme C
- Les entrées / sorties en C
- Types des données algorithmique / C
- Éléments syntaxiques du langage C
  - Structures de contrôles
- Traduction d'algorithmes simples en langage C



# EXEMPLE 1

- Ecrire un programme qui demande à l'utilisateur de taper un entier et qui affiche GAGNE si l'entier est entre 56 et 78 bornes incluses PERDU sinon

```
#include<iostream.h>
```

```
int main()
```

```
{
```

```
    int a;
```

```
    cout << "Tapez un entier : ";
```

```
    cin >> a;
```

```
    if ( (a>=56) && (a<=78) )
```

```
        cout << "GAGNE"<< endl;
```

```
    else cout << "PERDU"<< endl;
```

```
    return 0;
```

```
}
```



## EXEMPLE 2

- Ecrire un programme qui affiche tous les entiers de 8 jusqu'à 23 (bornes incluses) en utilisant un for

```
int main()
{
    int i;
    for(i=8;i<=23;i++)
        cout<<i<<endl;
    return 0;
}
```



## EXEMPLE 3

- Même exercice que précédemment en utilisant un while

```
int main()
{
    int i=8;
    while(i<=23)
    {
        cout<<i<<endl;
        i++;
    }
    return 0;
}
```

# OPÉRATIONS EXOTIQUES SUR LES ENTIERS

- Opérations qui modifient la valeur stockée
  - ++ et -- (incrémentatation et décrémentation automatique)
  - souvent utilisé sous la forme : `i++` ;
  - équivalent à `i = i + 1`;
- `a = 0; i = 1; a = i++` ;
  - que valent a et i après exécution ?  
`a == 1, i == 2`
- `a = 0; i = 1; a = ++i` ;
  - que valent a et i après exécution ?  
`a == 2, i == 2`

<code>a = i++</code>	équivalent
	<code>a = i</code>
	<code>i = i + 1</code>

<code>a = ++i</code>	équivalent
	<code>i = i + 1</code>
	<code>a = i</code>



# CONCLUSION

- Petit tour d'horizon des éléments syntaxiques de base du langage C
  - Types et variables
  - Structures de contrôle
  - Conditions / expressions
- À enrichir durant les prochaines séances
- De quoi débiter les travaux pratiques
- Présentation rapide de l'outil utilisé



# LIFAP1 : LA PLATEFORME ASKER POUR S'ENTRAINER ENTRE LE COURS ET LES TDS

Contact : [marie.lefevre@liris.cnrs.fr](mailto:marie.lefevre@liris.cnrs.fr)

LIFAP1 - Plateforme d'entrainement ASKER



# PLATEFORME ASKER : HTTPS://ASKER.UNIV- LYON1.FR/

Merci de vous authentifier

 m.lefevre

 .....

Remember me

**Authentification**

**Connectez-vous avec vos identifiants Lyon 1**



# LISTE DES UES



**Une fois connecté, vous voyez, en haut à gauche, un dossier par UE.**

**Cliquez sur le nom de l'UE pour l'ouvrir**

# LISTE DES EXERCICES

LIFAP2 LIFAP1 m.lefevre

 Bases de l'algorithmique

	LIFAP1 - Les structures de contrôle	MARIE LEFEVRE	
	LIFAP1 - La syntaxe en algorithmique	MARIE LEFEVRE	
	LIFAP1 - La boucle POUR pour afficher	MARIE LEFEVRE	
	LIFAP1 - Les étapes de conception	MARIE LEFEVRE	

 Bases du C

**Dans le dossier LIFAP1  
vous trouverez des dossiers thématiques  
contenant chacun plusieurs exercices**

# POUR FAIRE UN EXERCICE

The screenshot shows a list of exercises under the heading 'Bases de l'algorithmique'. The exercises are listed in a table-like format with columns for a checkmark icon, the exercise title, the author name 'MARIE LEFEVRE', and a refresh icon. Two refresh icons are circled in red. Red text annotations are placed over the interface: 'Demander un nouvel exercice de ce type' is positioned over the second row, and 'Refaire cet exercice précisément' is positioned over the 'Exercice 3' row.

Icon	Exercise Title	Author	Refresh Icon
✓	LIFAP1 - Les structures de contrôle	MARIE LEFEVRE	↻
✓	LIFAP1 - La syntaxe en algorithmique	MARIE LEFEVRE	↻
	Exercice 1		↻
	Exercice 2		↻
	Exercice 3		↻
✓	LIFAP1 - La boucle POUR pour afficher	MARIE LEFEVRE	↻
↕	LIFAP1 - Les étapes de conception	MARIE LEFEVRE	↻

**Demander un nouvel exercice de ce type**

**Refaire cet exercice précisément**

# VOIR SES RÉPONSES

Exercice 5	<b>Voir les réponses précédentes</b>	<b>Répondre à l'exercice</b>
Résultats : 0% 100%  Aucune réponse postée pour cette tentative		

# LES EXERCICES

LIFAP2 LIFAP1 m.lefevre

LIFAP1 - La syntaxe en algorithmique

Consigne : Répondez aux questions suivantes

Cochez les types valides en algorithmique :

- réel
- caractère
- booléen
- bloc
- multiplication
- entier

**L'exercice**

**Plusieurs questions :  
valider chaque  
question  
indépendamment**

**Valider & Refaire**

Retour Valider C

1 2

# LA CORRECTION D'UNE QUESTION

LIFAP2 LIFAP1

m.lefevre

LIFAP1 - La syntaxe en algorithmique

Consigne : Répondez aux questions suivantes :

Cochez les types valides en algorithmique :

- réel
- caractère
- booléen
- bloc
- multiplication
- entier

Commentaire :

Votre score : 0%

Solution

- réel
- caractère
- booléen
- bloc
- multiplication
- entier

Retour



1

2