INFO 2

M. Pétréolle

14 Septembre 2015

Algorithmique

Langages

Programme

Historique et caractéristiques

- Le langage C est un langage évolué et structuré

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...
- Langage d'intérêt général orienté système : peut traiter des aspects de bas niveau : utilisé pour développer des systèmes d'exploitation orienté applications : fonctionnalités de haut niveau

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...
- Langage d'intérêt général orienté système : peut traiter des aspects de bas niveau : utilisé pour développer des systèmes d'exploitation orienté applications : fonctionnalités de haut niveau
- Il est multi-plateformes (unix, dos, windows, ...), avec une bibliothèque très volumineuse

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...
- Langage d'intérêt général orienté système : peut traiter des aspects de bas niveau : utilisé pour développer des systèmes d'exploitation orienté applications : fonctionnalités de haut niveau
- Il est multi-plateformes (unix, dos, windows, ...), avec une bibliothèque très volumineuse
- Riche en opérateurs

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...
- Langage d'intérêt général orienté système : peut traiter des aspects de bas niveau : utilisé pour développer des systèmes d'exploitation orienté applications : fonctionnalités de haut niveau
- Il est multi-plateformes (unix, dos, windows, ...), avec une bibliothèque très volumineuse
- Riche en opérateurs
- Typé mais laxiste

Historique et caractéristiques

- Le langage C est un langage évolué et structuré
- Il a été créé par Kernighan et Ritchie en 1972, normalisé en 1982
- L'évolution du C vers l'objet donne naissance au C++, JAVA ...
- Langage d'intérêt général orienté système : peut traiter des aspects de bas niveau : utilisé pour développer des systèmes d'exploitation orienté applications : fonctionnalités de haut niveau
- Il est multi-plateformes (unix, dos, windows, ...), avec une bibliothèque très volumineuse
- Riche en opérateurs
- Typé mais laxiste

structure d'un programme C++

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

int main
{
int i ;

i=i+1;
cout<< "Bonjour \n"
return 0;
}
```

Directives du préprocessus

Entête du programme

Déclaration des variables

Instructions

Fin du programme

Syntaxe

- L'alphabet est le code ASCII

Syntaxe

- L'alphabet est le code ASCII
- Les identificateurs : ils désignent les variables, constantes, fonctions manipulées par le programme. Au moins 1 caractère alphabétique suivi d'un nombre quelconque de caractères (32 caractères retenus).

Syntaxe

- L'alphabet est le code ASCII
- Les identificateurs : ils désignent les variables, constantes, fonctions manipulées par le programme. Au moins 1 caractère alphabétique suivi d'un nombre quelconque de caractères (32 caractères retenus).
- Attention : minuscule $\neq$ Majuscule
- Les mots clefs : peu nombreux, ils correspondent aux instructions de base. Ils ne peuvent pas être employés comme identificateur et sont écrits en minuscule

Syntaxe

- L'alphabet est le code ASCII
- Les identificateurs : ils désignent les variables, constantes, fonctions manipulées par le programme. Au moins 1 caractère alphabétique suivi d'un nombre quelconque de caractères (32 caractères retenus).
- Attention : minuscule $\neq$ Majuscule
- Les mots clefs : peu nombreux, ils correspondent aux instructions de base. Ils ne peuvent pas être employés comme identificateur et sont écrits en minuscule
- Les commentaires : // commence un commentaire sur une ligne

Les entiers

TYPE	DESCRIPTION	TAILLE MEMOIRE	PLAGE DE VARIATION
signed char	Entier signé	1 octet	- 27 ≤n ≤ 27-1
unsigned char	Caractère ou entier positif	1 octet	0 ≤n ≤ 28-1
short	Entier standard signé	2 octets	- 215 ≤ n ≤215-1
unsigned short	Entier standard positif	2 octets	0 ≤ n ≤ 216 -1
int	Entier signé	2 ou 4 octets suivant les compilateurs	Suivant la taille
unsigned int	Entier positif	2 ou 4 octets suivant les compilateurs	Suivant la taille
long	Entier long signé	4 octets	- 231 ≤n ≤ 231-1
unsigned long	Entier long positif	4 octets	0 ≤n ≤ 232-1

Les réels

réel se dit float en anglais. D'où l'anglicisme "flottant"

TYPE	DESCRIPTION	TAILLE MEMOIRE	PLAGE DE VALEURS POSSIBLES	PRÉCISION
float	réel standard	4 octets	3.4*10-38 à 3.4*1038	6 à 7 chiffres
double	réel double précision	8 octets	1.7*10-308 à 1.7*10308	14 à 15 chiffres
long double	réel quadruple précision	10 octets	3.4*10-4932 à 3.4*104932	16 à 17 chiffres

Autres types

- type void : le type vide, équivalent à pas de type
- type bool : deux valeurs possibles, true et false
- type char : contient un caractère en ASCII
- type tableau : on verra plus tard...

Des exemples

Type numérique

- int a, b, c ;
- int d=-25;
- float x,y;
- float z=-48.59;
- float t=-15.7E+20;

Type caractère :

- char c,f ;
- char g=‘a’;
- char z=65; // char z=65 correspond au code ASCII 65 c’est à dire ”A”

Les constantes

On peut définir des constantes, qui ne peuvent pas varier dans le reste du programme. (Si on essaye, on obtient une erreur lors de la compilation.)

Les constantes

On peut définir des constantes, qui ne peuvent pas varier dans le reste du programme. (Si on essaye, on obtient une erreur lors de la compilation.)

Exemple :

```
const float pi=3.14 ;
```

Les commandes graphiques

Constantes caractères utilisées pour contrôler le périphérique de sortie (séquence d'échappement)

Caractère	
\n	Interligne
\t	Tab horizontale
\v	Tab verticale
\r	Retour chariot
\f	Saut de page
\\	Back slash
\'	Cote
\"	Guillemets

Gestion des entrées/sorties

Les entrées

- Ce n'est pas géré en C++ de base. On utilise une bibliothèque pour cela : # include <iostream>

Les entrées

- Ce n'est pas géré en C++ de base. On utilise une bibliothèque pour cela : # include <iostream>
- On utilisera la commande cin.
- Exemple : cin >> *variable* ;
- Cette instruction demande à l'utilisateur de rentrer au clavier la valeur à affecter à la variable *variable*.

Les entrées

- Ce n'est pas géré en C++ de base. On utilise une bibliothèque pour cela : # include <iostream>
- On utilisera la commande cin.
- Exemple : cin >> *variable* ;
- Cette instruction demande à l'utilisateur de rentrer au clavier la valeur à affecter à la variable *variable*.
- >> est un opérateur binaire, dont le premier opérande est cin et le second est une variable à obtenir.

Les entrées

- Ce n'est pas géré en C++ de base. On utilise une bibliothèque pour cela : # include <iostream>
- On utilisera la commande cin.
- Exemple : cin >> *variable* ;
- Cette instruction demande à l'utilisateur de rentrer au clavier la valeur à affecter à la variable *variable*.
- >> est un opérateur binaire, dont le premier opérande est cin et le second est une variable à obtenir.
- >> est un opérateur sur-typé : il est utilisé pour obtenir des variables de n'importe quel type : entier, réel, booléen, ...

Les entrées

- On peut enchainer les demandes d'entrées.
- Exemple : cin >> *variable*1 >> *variable*2;

Les entrées

- On peut enchainer les demandes d'entrées.
- Exemple : cin >> *variable*1 >> *variable*2;
- *variable*1 et *variable*2 n'ont pas forcément le même type.

Les entrées

- On peut enchainer les demandes d'entrées.
- Exemple : cin >> *variable*1 >> *variable*2;
- *variable*1 et *variable*2 n'ont pas forcément le même type.
- Un retour à la ligne automatique a lieu lorsqu'on demande des variables à la suite.

Les sorties

- On utilisera la commande cout.
- Exemple : cout << *variable* ;
- Cette instruction affiche à l'écran la valeur de *variable*.

Les sorties

- On utilisera la commande cout.
- Exemple : cout << *variable* ;
- Cette instruction affiche à l'écran la valeur de *variable*.
- << se comporte comme >>.

Les sorties

- On utilisera la commande cout.
- Exemple : cout << *variable* ;
- Cette instruction affiche à l'écran la valeur de *variable*.
- << se comporte comme >>.
- Pour afficher une chaine de caractère, utiliser ".
- Exemple : cout << "la valeur du perimetre est" << endl << rayon;
- endl est l'instruction de retour à la ligne

Les sorties

- On utilisera la commande cout.
- Exemple : cout << *variable* ;
- Cette instruction affiche à l'écran la valeur de *variable*.
- << se comporte comme >>.
- Pour afficher une chaine de caractère, utiliser ".
- Exemple : cout << "la valeur du perimetre est" << endl << rayon;
- endl est l'instruction de retour à la ligne
- On peut aussi utiliser \n à l'intérieur d'une chaine de caractère pour passer à la ligne

Une remarque

On n'utilise ni printf ni scanf pour gérer les entrées/sorties, ce sont des commandes en C, pas en C++. De plus, elles sont moins flexibles.

Instructions, Expressions et Opérateurs

Instructions

- Une instruction permet de réaliser une action
- Exemple :
  cout <<("Entrer un caractère : ");

Instructions

- Une instruction permet de réaliser une action
- Exemple :
  cout <<("Entrer un caractère : ");
- Instruction composée :
  {
  instruction ;
  instruction ;
  ...
  }

Expressions

- Une expression est évaluée, elle retourne une valeur : $2 \times \cos(a)/(y - 2)$

Expressions

- Une expression est évaluée, elle retourne une valeur : $2 \times \cos(a)/(y - 2)$
- Cette valeur peut être stockée dans une variable : result $= 2 \times \cos(a)/(y - 2)$;

Expressions

- Une expression est évaluée, elle retourne une valeur : $2 \times \cos(a)/(y - 2)$
- Cette valeur peut être stockée dans une variable : result $= 2 \times \cos(a)/(y - 2)$;
- ou servir dans la suite du calcul : $2 \times \cos(a)/(y - 2) + 4$

Expressions

- Une expression est évaluée, elle retourne une valeur : $2 \times \cos(a)/(y-2)$
- Cette valeur peut être stockée dans une variable : result $= 2 \times \cos(a)/(y-2)$;
- ou servir dans la suite du calcul : $2 \times \cos(a)/(y-2)+4$
- ou être écrite sur un flux de sortie : cout $<<$ $2 \times i$;

Opérateurs

Les opérateurs permettent de construire des expressions.

Opérateurs

Les opérateurs permettent de construire des expressions.

- Le langage C dispose de beaucoup d'opérateurs

Opérateurs

Les opérateurs permettent de construire des expressions.

- Le langage C dispose de beaucoup d'opérateurs
- Attention à l'ordre de priorité des opérateurs

Opérateurs unaires

+	signe plus	+5
-	signe moins	-5

Opérateurs unaires

+	signe plus	+5
-	signe moins	-5
++ préfixé	incrémentation	++i
++ postfixé	incrémentation	i++

Opérateurs unaires

+	signe plus	+5
-	signe moins	-5
++ préfixé	incrémentation	++i
++ postfixé	incrémentation	i++
- - préfixé	décrémentation	- -i
- - postfixé	décrémentation	i- -

Opérateurs unaires

+	signe plus	+5
-	signe moins	-5
++ préfixé	incrémentation	++i
++ postfixé	incrémentation	i++
- - préfixé	décrémentation	- -i
- - postfixé	décrémentation	i- -
!	négation	!true

Opérateurs binaires

=	Affectation	x=4

Opérateurs binaires

$=$	Affectation	x=4
$+$	addition	3+5
-	soustraction	3-5

Opérateurs binaires

=	Affectation	x=4
+	addition	3+5
-	soustraction	3-5
*	multiplication	3*5

Opérateurs binaires

=	Affectation	x=4
+	addition	3+5
-	soustraction	3-5
*	multiplication	3*5
\	division euclidienne	5/2
\	division	5.0/2.0

Opérateurs binaires

=	Affectation	x=4
+	addition	3+5
-	soustraction	3-5
*	multiplication	3*5
\	division euclidienne	5/2
\	division	5.0/2.0

Opérateurs binaires

==	comparaison logique	x==4

Opérateurs binaires

$==$	comparaison logique	x==4
$>$	comparaison logique	$3 > 5$
$\leq$	comparaison logique	$3 \leq 3$

Opérateurs binaires

$==$	comparaison logique	x==4
$>$	comparaison logique	$3 > 5$
$\leq$	comparaison logique	$3 \leq 3$
& &	et logique	Vrai & & Faux

Opérateurs binaires

==	comparaison logique	x==4
>	comparaison logique	3 > 5
≤	comparaison logique	3 ≤ 3
& &	et logique	Vrai & & Faux
||	ou logique	Vrai || Faux
\	division	5.0/2.0

Opérateurs binaires

==	comparaison logique	x==4
>	comparaison logique	3 > 5
≤	comparaison logique	3 ≤ 3
& &	et logique	Vrai & & Faux
||	ou logique	Vrai || Faux
\	division	5.0/2.0

Structures de contrôle

Les tests sans alternative

- if (expression booléenne) instruction simple ou composée ;

Les tests sans alternative

- if (expression booléenne) instruction simple ou composée ;
- Exemple 1 :
  if (x==4) y=5.2;

Les tests sans alternative

- if (expression booléenne) instruction simple ou composée ;
- Exemple 1 :
  if (x==4) y=5.2;
- Exemple 2 :

```
if (x==y)
        {
          y=5.2;
          x=4.2;
        }
```

Les tests avec alternatives

- if (expression booléenne) instruction simple ou composée
  else
  instruction simple ou composée

Les tests avec alternatives

- if (expression booléenne) instruction simple ou composée
  else
  instruction simple ou composée
- Exemple :
  if ($x > y$) cout << x ;
  else
  cout << y ;

Les boucles for

La syntaxe est la suivante :

for (instruction1 ; instruction2 ; instruction 3)
instruction4 simple ou composée

Les boucles for

La syntaxe est la suivante :

for (instruction1 ; instruction2 ; instruction 3)
instruction4 simple ou composée

- instruction1 regroupe les compteurs et initialisation

Les boucles for

La syntaxe est la suivante :

for (instruction1 ; instruction2 ; instruction 3)
instruction4 simple ou composée

- instruction1 regroupe les compteurs et initialisation
- instruction2 est la condition d'arrêt de la boucle, testée avant l'exécution

Les boucles for

La syntaxe est la suivante :

for (instruction1 ; instruction2 ; instruction 3)
instruction4 simple ou composée

- instruction1 regroupe les compteurs et initialisation
- instruction2 est la condition d'arrêt de la boucle, testée avant l'exécution
- instruction3 est l'incrémentation

Les boucles for

La syntaxe est la suivante :

for (instruction1 ; instruction2 ; instruction 3)
instruction4 simple ou composée

- instruction1 regroupe les compteurs et initialisation
- instruction2 est la condition d'arrêt de la boucle, testée avant l'exécution
- instruction3 est l'incrémentation
- attention ! L'incrémentation est effectuée après l'exécution de instruction4.

Les boucles for (2)

Exemple 1 :

```
int compteur ;
for (compteur=0; compteur ==10; compteur++)
        cout << compteur ;
```

Les boucles for (2)

Exemple 1 :

```
int compteur ;
for (compteur=0; compteur ==10; compteur++)
        cout << compteur ;
```

Exemple 2 :

```
int compteur ;
for (compteur=0; compteur <9;
compteur=compteur+2)
        cout << compteur ;
```

Les boucles for (3)

Exemple 3 :

```
int compteur ;
for (compteur=1; compteur <>10;
compteur=compteur+2)
        cout << compteur ;
```

Les boucles while

La syntaxe est :

```
while (expression booléenne)
    instruction simple ou composée
```

Les boucles while

La syntaxe est :

```
while (expression booléenne)
    instruction simple ou composée
```

- Le test est effectué avant l'instruction. La boucle peut ne rien faire.

Les boucles while (2)

- Exemple 1:
  int compteur ;
  while (compteur <10) {cout << compteur ;
  compteur++;}

Les boucles while (2)

- Exemple 1:
  int compteur ;
  while (compteur <10) {cout << compteur ;
  compteur++;}
- Exemple 2:
  int compteur ;
  while (compteur <10) cout << compteur ;

Les boucles do while

La syntaxe est

do instruction simple ou composée
while (expression booléenne);

Les boucles do while

La syntaxe est

do instruction simple ou composée
while (expression booléenne);

- Exemple 1 :
  int compteur=0 ; do { cout << compteur;
  compteur = compteur +1 }
  while (compteur <> 10) ;

Les tableaux

- Un tableau est une suite de cases contenant toutes un objet de même type.

Les tableaux

- Un tableau est une suite de cases contenant toutes un objet de même type.
- Les cases sont indexées par des entiers.

Les tableaux

- Un tableau est une suite de cases contenant toutes un objet de même type.
- Les cases sont indexées par des entiers.
- En C++, la première case a pour indice 0 et la dernière case a pour indice $N - 1$, si le tableau est de taille N

Les tableaux

- Un tableau est une suite de cases contenant toutes un objet de même type.
- Les cases sont indexées par des entiers.
- En C++, la première case a pour indice 0 et la dernière case a pour indice $N - 1$, si le tableau est de taille N
- Une tentative d'accéder à la case d'indice N NE FERA PAS planter le programme. Le résultat pourra être n'importe quoi...

Les tableaux

- Un tableau est une suite de cases contenant toutes un objet de même type.
- Les cases sont indexées par des entiers.
- En C++, la première case a pour indice 0 et la dernière case a pour indice $N - 1$, si le tableau est de taille N
- Une tentative d'accéder à la case d'indice N NE FERA PAS planter le programme. Le résultat pourra être n'importe quoi...
- Les boucles Pour sont très adaptées pour travailler sur un tableau.

Les tableaux (2)

- La syntaxe est la suivante :
  type nom_du_tableau[taille_du_tableau];

Les tableaux (2)

- La syntaxe est la suivante :
  type nom_du_tableau[taille_du_tableau];
- Exemple 1 :
  int notes[24];

Les tableaux (2)

- La syntaxe est la suivante :
  type nom_du_tableau[taille_du_tableau];
- Exemple 1 :
  int notes[24];
- La taille du tableau doit être entrée ou bien en dur, ou bien via une constante. Ce ne DOIT PAS être une variable.

Les tableaux (2)

- La syntaxe est la suivante :
  type nom_du_tableau[taille_du_tableau];
- Exemple 1 :
  int notes[24];
- La taille du tableau doit être entrée ou bien en dur, ou bien via une constante. Ce ne DOIT PAS être une variable.
- Exemple 2 :
  int const taille =24;

Les tableaux (2)

- La syntaxe est la suivante :
  type nom_du_tableau[taille_du_tableau];
- Exemple 1 :
  int notes[24];
- La taille du tableau doit être entrée ou bien en dur, ou bien via une constante. Ce ne DOIT PAS être une variable.
- Exemple 2 :
  int const taille =24;
  int notes[taille];
- Il est toujours préférable de donner la taille via une constante, cela permet de modifier plus facilement le programme.

Les tableaux (3)

- Comme pour les autres variables, on peut initialiser un tableau lors de sa création.

Les tableaux (3)

- Comme pour les autres variables, on peut initialiser un tableau lors de sa création.
- Exemple 1 :
  int tableau[5]= $\{0, 2, 4, 6, 8\}$;

Les tableaux (3)

- Comme pour les autres variables, on peut initialiser un tableau lors de sa création.
- Exemple 1 :
  int tableau[5]= $\{0, 2, 4, 6, 8\}$;
- Exemple 2 :
  int tableau[5]=$\{-4, 2\}$; n'initialise que les deux premières cases.

Les tableaux (3)

- Comme pour les autres variables, on peut initialiser un tableau lors de sa création.
- Exemple 1 :
  int tableau[5]= $\{0, 2, 4, 6, 8\}$;
- Exemple 2 :
  int tableau[5]=$\{-4, 2\}$; n'initialise que les deux premières cases.
- On accède à la i ème case d'un tableau via la commande suivante
  nom_du_tableau[i]

Les tableaux (3)

- Comme pour les autres variables, on peut initialiser un tableau lors de sa création.
- Exemple 1 :
  int tableau[5]= $\{0, 2, 4, 6, 8\}$;
- Exemple 2 :
  int tableau[5]=$\{-4, 2\}$; n'initialise que les deux premières cases.
- On accède à la i ème case d'un tableau via la commande suivante
  nom_du_tableau[i]
- On peut modifier la i ème case d'un tableau comme si c'était une variable :
  nom_du_tableau[i]= nom_du_tableau[i]+4;

Les fonctions

- Une fonction est un bloc d'instructions, nommé, qui retourne un résultat ou effectue des opérations.

Les fonctions

- Une fonction est un bloc d'instructions, nommé, qui retourne un résultat ou effectue des opérations.
- Une fonction peut prendre des paramètres (exemple :longueur, largeur) en arguments.

Les fonctions

- Une fonction est un bloc d'instructions, nommé, qui retourne un résultat ou effectue des opérations.
- Une fonction peut prendre des paramètres (exemple :longueur, largeur) en arguments.
- Une fonction peut ensuite être appelée dans le reste de l'algorithme.

Les fonctions

- Une fonction est un bloc d'instructions, nommé, qui retourne un résultat ou effectue des opérations.
- Une fonction peut prendre des paramètres (exemple :longueur, largeur) en arguments.
- Une fonction peut ensuite être appelée dans le reste de l'algorithme.
- L'intérêt de définir une fonction est que l'on peut faire appel à elle plusieurs fois dans un algorithme, ou l'utiliser dans plusieurs algorithmes. Cela évite d'avoir à coder plusieurs fois la même chose.

Les fonctions (2)

- En C++, on déclare une fonction après les *include* et avant le *main*().

Les fonctions (2)

- En C++, on déclare une fonction après les *include* et avant le *main*().
- On précise : le type de la variable qu'elle renvoie, le nom de la fonction, ainsi que la liste de paramètres dont on précise le type. L'instruction return permet de renvoyer une variable qui pourra être utilisée par le *main*().

Les fonctions (2)

- En C++, on déclare une fonction après les *include* et avant le *main*().
- On précise : le type de la variable qu'elle renvoie, le nom de la fonction, ainsi que la liste de paramètres dont on précise le type. L'instruction return permet de renvoyer une variable qui pourra être utilisée par le *main*().
- Exemple 1 :

```
float fonction (float x, int y)
{
return x+y;
}
```

Les fonctions (3)

- Une fois la fonction déclarée et décrite, on peut faire appel à elle depuis le programme principal (main) en utilisant son nom et en lui passant des paramètres compatibles avec les types prévus dans la déclaration.

Les fonctions (3)

- Une fois la fonction déclarée et décrite, on peut faire appel à elle depuis le programme principal (main) en utilisant son nom et en lui passant des paramètres compatibles avec les types prévus dans la déclaration.
- Exemple 2 :

```
int carre (int x)
{
return x*x;
}
void main()
{ cin >> x>>y;
cout<<carre( x)+carre( y); }
```

Les fonctions (4)

- Une fonction qui ne renvoie rien aura pour type void.

Les fonctions (4)

- Une fonction qui ne renvoie rien aura pour type void.
- Si un tableau est utilisé comme argument d'une fonction, il sera souvent utile de mettre aussi la taille du tableau en argument, pour pouvoir l'utiliser dans la fonction.

Les fonctions (4)

- Une fonction qui ne renvoie rien aura pour type void.
- Si un tableau est utilisé comme argument d'une fonction, il sera souvent utile de mettre aussi la taille du tableau en argument, pour pouvoir l'utiliser dans la fonction.
- Pour utiliser un tableau en argument, voici la syntaxe

```
int somme (int tableau[], int taille) {
int s=0, i;
for (i=0;i<taille;i++)
{s = s + tableau[i]; }
return s;
}
```