

```
cd .. revenir au répertoire précédent dans l'arborescence.
cd / revenir à la racine de l'arborescence.
cd (tout seul) revenir au répertoire "maison" (home directory) de l'utilisateur.
```

```
// Commentaire orienté ligne (délimité à gauche, sur une ligne)
/*
Commentaire orienté bloc (délimité à gauche et à droite, sur plusieurs lignes)
*/
```

```
int a=1, b=-20118; // on peut préciser : unsigned int, short int, long int
double c=1.2E3; // 1.2E3 signifie 1.2 * 1000
char d='A', e='R'; // pour avoir plusieurs caractères, utiliser string
bool f=true, g=false; // 0 est considéré comme false, donc 16 % 2 est false
const double pi(3.1415926535); // si 2 est un double, il faut l'écrire 2.0
```

```
if ( «condition» ) { «actions si» } else { «actions sinon» } // il n'y a pas de point-virgule
Attention à ne pas confondre l'opérateur d'affectation = avec l'opérateur relationnel de test d'égalité ==
utilisé dans les conditions logiques.
```

```
int a(5), b(1);
switch (a-b) { case 0 :
case 1 : cout << "a est proche de b" << endl;
break ;
case 3 :
case 5 : cout << "a-b n'est pas une valeur binaire..." << endl;
break;
default : cout << "a-b est pair ou est plus grand que 6" << endl;} // il n'y a pas de point-virgule
```

```
do { cout << "Entrez une valeur entre 1 et 3:";
cin >> n; } while ( (n < 1) || (n > 3) ); // il y a un point-virgule
```

```
continue // Dans le bloc d'une boucle. Le programme passe directement à la fin de ce bloc, sans exécuter
// les instructions qui le suivent. Cependant, il ne sort pas de la boucle, contrairement à break.
```

```
for (int i(0); i < 8; ++i) {
int j=i*2+3;
if (j < 10) continue; // si j < 10: on passe directement à l'itération suivante (prochaine valeur de i)
cout << i << ", " << j << endl; } // sinon, on passe ici.
```

```
(a && b && c && ...) // Si a est faux, cela assure que toute l'expression est fautive, et les autres arguments ne
// sont pas évalués (les arguments sont évalués jusqu'au 1er argument faux).
((x != 0) && (4/x > 3)) // L'expression 4/x produirait une erreur si x était égal à zéro au moment de son
// évaluation. Mais du fait de l'évaluation paresseuse (exécutée de gauche à droite), cette
// expression n'est pas calculée quand x=0, et ne produit pas d'erreur.
(ch=='e' || ch=='a' || ch=='i' || ch=='o' || ch=='u' || ch=='y') // teste si le caractère ch est une voyelle.
```

**Portée des variables** (exemple :)

```
int var(1); // variable globale
void main () { // début du bloc 1
int i(2); // variable locale au bloc 1
{ int i(3); // début du bloc 2, variable locale au bloc 2
const int var(4); // autre variable locale au bloc 2
cout << "La variable locale i vaut : " << i << "\n" << "La constante locale var vaut : " << var << endl; } // fin du bloc 2
cout << "La variable locale i vaut : " << i << "\n" << "La variable globale var vaut : " << var << endl; } // fin du bloc 1
```

Résultat :  
La variable locale i vaut : 3  
La constante locale var vaut : 4  
La variable locale i vaut : 2  
La variable globale var vaut : 1

**Prototypage de fonction** : type nom ( type<sub>1</sub> arg<sub>1</sub>, ..., type<sub>n</sub> arg<sub>n</sub>);

**Définition de fonction** : type nom ( type<sub>1</sub> arg<sub>1</sub>, ..., type<sub>n</sub> arg<sub>n</sub>){... // corps de la fonction  
return valeur;}

**Appel de fonction** : nom ( var<sub>1</sub>, ..., var<sub>n</sub> );

Exemple de fonction : bool divisible (const int a, const int b) {return ((b != 0) && (a % b == 0)); }

Appel de cette fonction : cout << (divisible(4,2) ? "oui" : "non") << endl;

La syntaxe : (condition ? action1 : action2) est une forme abrégée de la structure conditionnelle : if (condition) action1; else action2;

#include <cmath>

- sqrt(x) : calcule la racine carrée de x.
- log(x) : calcule le logarithme népérien de x.
- exp(x) : calcule l'exponentielle de x.
- sin(x) : calcule le Sinus de x (x en radians)
- cos(x) : calcule le Cosinus de x (x en radians)
- acos(x) : calcule la réciproque du cosinus (arcCosinus, noté parfois cos<sup>-1</sup>)
- asin(x) : calcule la réciproque du sinus (arcSinus, noté parfois sin<sup>-1</sup>)
- pow(x, y) : calcule x puissance y (si y n'est pas entier, x doit être strictement positif)

#include <string>

```
string c,d;
c = "Ceci est une chaîne de caractères" ; //affectation de Ceci est une chaîne de caractères à c
cin >> d ; // Utilisateur entre un mot, stocké dans d
cout << c[3] << endl ; //affiche le i final du mot Ceci
cout << c.size() << endl ; //affiche 32, le nombre de signes dans c
cout << c.substr(5,3) << endl ; //affiche les 3 lettres du mot est dont le e est à la position 5 dans c
cout << c.find("chaîne") << endl ; //affiche la position du mot chaîne dans c, à savoir 13
cout << c.rfind("e") << endl ; //affiche la position du e le plus à droite dans c, à savoir 31
c.insert(2, "ce") ; // insère ce à la position 2, ce qui donne Cececi est
c.replace(7,3, "n est pas") ; // remplace 3 caractères dès la position 7 : Cececi nest pas une
c.replace(0,2, "") ; // supprime 2 caractères de c dès la position 0 : Ceci nest pas
```

```
string reponse("solution"); if (n > 1) {reponse = reponse + 's';} // Ajout d'un 's' final au pluriel
```

```
enum Type { valeur1, valeur2, ... };
enum CantonRomand { Vaud, Valais, Geneve, Fribourg, Neuchatel, Jura };
CantonRomand moncanton(Vaud);
int const NB_CANTONS_ROMANDS(Jura+1); // la première valeur énumérée correspond à 0
for (unsigned int i(Vaud); i<= Jura; ++i) ... // les valeurs énumérées se comportent comme des entiers
population[moncanton] = 616; // ou comme des éléments de tableau
```

```
Initialisation dun tableau de taille fixe N: type identificateur[N] = { val1, ..., valN };
Appel du i-ème élément de ce tableau : identificateur[i-1];
```

```
Exemples : double matrice[3][2] = { {4,3}, {2,1}, {3,2} };
statistique[Vaud][population] = 616000;
```

Pour le passage d'un tableau en argument d'une fonction, on peut omettre de spécifier la taille : int f(double tableau[]);  
Veiller néanmoins à ce que la taille du tableau soit connue de la fonction : int f(double tableau[], int const taille);  
Le passage dun tableau en argument d'une fonction se fait toujours par référence : ajouter **const** pour ne pas le modifier.

```
#include <vector>
vector < type > identificateur(N, valeur); // Initialisation dun tableau dynamique de taille N
vector < vector <int> > tab(5, vector<int>(6)); // correspond à une matrice à 5 lignes et 6 colonnes.
vector <int> tab(3, 0); // déclaration et initialisation à 0 d'un vecteur d'entiers de dimension 3
tab[2] = -45; // affecte la valeur -45 à l'élément de position 2 du vecteur tab
cout << tab[1] << endl; // affiche l'élément de position 1 du tableau tab, à savoir 0
cout << tab.size() << endl; // affiche la taille du tableau tab, à savoir 3
tab.push_back(20); // ajoute à la fin du tableau un élément valant 20
cout << tab.size() << endl; // affiche à nouveau la taille de tab, cette fois-ci 4
tab.pop_back(); // supprime le dernier élément
cout << tab.back() << endl; // affiche le dernier élément, à savoir -45
tab.clear(); // vide le tableau
```

```
void saisie(vector<int>& vect, int const TAILLE = 4) { // Voici un bout de code qui
int val; // initialise un vecteur d'entiers
vect.clear(); // que l'on suppose strictement
cout << "Saisie de " << TAILLE << " valeurs :" << endl; // positifs. Lors de la saisie,
while (vect.size() < TAILLE) { // l'utilisateur peut de plus
cout << "Entrez le coef. " << vect.size() << " : " << flush; // recommencer en
cin >> val; // entrant zéro ou effacer
if (val < 0) vect.pop_back(); // le dernier élément en
else if (val == 0) vect.clear(); // entrant un nombre
else vect.push_back(val); // négatif.*/
```

**typedef** type alias; //type est le type à redéfinir, et **alias** est un nom supplémentaire pour désigner le type **type**.

Exemple : typedef vector< double > Matrice;

```
struct Complexe {double x; double y;}; // Attention aux points-virgules !
Complexe z = { 0.0, 1.0 }; // exemple d'initialisation de structure
z.x = -3; // la partie réelle du nombre complexe z vaut -3
z.x++; // la partie réelle de z vaut à présent -2
```

**Pointeurs** : pour **déclarer** une variable (nommée **ptr**) pointant sur un entier on écrira : **int\* ptr;**

Pour **accéder** au contenu pointé par un pointeur, il faut écrire : **\*ptr**

L'opérateur d'**adresse &** renvoie l'adresse de la variable à laquelle il s'applique.

```
int i(4), j(5);
int* ptr(&i); // ptr pointe sur i
cout << *ptr << endl; // affiche 4
ptr = &j; // ptr pointe maintenant sur j
j++; // j vaut maintenant 6
cout << *ptr << endl; // affiche 6
*ptr = *ptr + 2; // augmente la valeur pointée (donc celle de j)
cout << j << endl; // affiche 8
int f(int, double); // prototype d'une fonction (prennant un int et un double et retournant un entier)
int (*ptr)(int, double); // pointeur sur une telle fonction
ptr = f; // ici ptr pointe sur f, c'est la même chose que ptr = &f; (dans le cas des fonctions)
ptr = new int(5); // pour faire une initialisation en même temps que l'allocation mémoire
delete ptr; // on n'a plus besoin de ptr : on libère la zone mémoire qu'on lui avait allouée.
```

```
#include <fstream> (variables de deux types: ifstream ou ofstream) Exemple :
ifstream fichier; // déclaration du flot fichier en lecture
string nom_fichier("test"); // initialisation du nom du fichier
fichier.open(nom_fichier.c_str()); // le flot est lié au fichier "test". Equivalut à: fichier.open("test");
if (! fichier.fail()) { // s'exécute tant qu'il ny a pas d'erreur
string mot;
while (!fichier.eof()) { // tant que la lecture du fichier nest pas finie
fichier >> mot; // lecture d'un mot dans le fichier, et affectation à la variable "mot"
if (!fichier.fail()) // on a effectivement pu lire qqchose
cout << mot;
fichier.close (); // fermeture du flot */
}
} else {cout << "Ouverture du fichier " << nom_fichier << " impossible." << endl;}}
```

```
#include <iomanip>
cout << manipulateur << expression << ... << manipulateur << expression ...;
setprecision(int dg) indique le nombre de chiffres après la virgule pour l'affichage de réels (dg chiffres).
setw(int size) indique la largeur de la chaîne de caractère en entrée/sortie (largeur size).
Utilisé avec cout la chaîne en sortie sera affichée sur size caractères
Utilisé avec cin, permet de lire size caractères à chaque fois.
setfill(char c) sur cout, utilise le caractère c pour effectuer les alignements avec setw.
ws (white space) lors d'une opération d'entrée, saute les espaces (espace, tabulation, retour de ligne, ...) : positionne le flot d'entrée sur le prochain caractère non blanc.
dec affiche les nombres en décimal.
hex affiche les nombres en hexadécimal (base 16).
```

