

Cours de C++

(partie 1)

Bibliothèque standard à inclure

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

Bibliothèques usuelles :

```
#include <cmath> → fonctions mathématiques
#include <string> → utiliser les strings
```

Afficher sur console

```
cout << "Hello World" << endl;
```

cout : commande pour afficher sur la console.

endl : fait un retour à la ligne.

Commentaire

On commente notre code avec `//` et un paragraphe avec `/* ... */`

Types de données

int, bool, char, double (nombre avec virgule), string (#include <string>)

Déclarer une variable

```
TYPE NOM(VALEUR);
```

Référence à une variable

On peut faire référence à la même case mémoire d'une variable par **&** :

```
int ageUtilisateur(16);
int& maVariable(ageUtilisateur);
```

Récupérer une donnée de la console

```
cin >> variable_initialisé
```

pour récupérer toute une ligne de string il faut utiliser : `getline(cin, variable_string)`

Si on utilise `cin >> ...` avant `getline(...)`, il faut ajouter `cin.ignore()` entre les deux.

Opérations

= : Affectation

% : Modulo

Déclarer une constante

```
int const nombreNiveaux(10);
```

Incrémentation et décrémentation

`++variable`; On incrémente la variable puis on l'utilise.

`variable++`; On utilise la variable avant de l'incrémenter.

```
--variable;
```

```
variable--;
```

Versions courtes des opérations

```
nombre = nombre + 2; → nombre += 2;
```

nombre = nombre - 2; → nombre -= 2;
 nombre = nombre * 2; → nombre *= 2;
 nombre = nombre / 2; → nombre /= 2;

Fonction de la librairie math

Racine carrée	\sqrt{x}	<code>sqrt()</code>	<code>resultat = sqrt(valeur);</code>
Sinus	$\sin(x)$	<code>sin()</code>	<code>resultat = sin(valeur);</code>
Cosinus	$\cos(x)$	<code>cos()</code>	<code>resultat = cos(valeur);</code>
Tangente	$\tan(x)$	<code>tan()</code>	<code>resultat = tan(valeur);</code>
Exponentielle	e^x	<code>exp()</code>	<code>resultat = exp(valeur);</code>
Logarithme népérien	$\ln x$	<code>log()</code>	<code>resultat = log(valeur);</code>
Logarithme en base 10	$\log_{10} x$	<code>log10()</code>	<code>resultat = log10(valeur);</code>
Valeur absolue	$ x $	<code>fabs()</code>	<code>resultat = fabs(valeur);</code>
Arrondi vers le bas	$\lfloor x \rfloor$	<code>floor()</code>	<code>resultat = floor(valeur);</code>
Arrondi vers le haut	$\lceil x \rceil$	<code>ceil()</code>	<code>resultat = ceil(valeur);</code>

puissance : `pow(a,b) = (a)^b`

Opérateur logique

Symbole	Signification
<code>==</code>	Est égal à
<code>></code>	Est supérieur à
<code><</code>	Est inférieur à
<code>>=</code>	Est supérieur ou égal à
<code><=</code>	Est inférieur ou égal à
<code>!=</code>	Est différent de

Conditionnelle

if (...) { ... } **else if** (...) { ... } **else** { ... }

Switch

```
switch(variable) {
    case 0: ...; break;
    case 1: ...; break;
    default: ...; break;
}
```

Opérateurs conditionnelles

<code>and</code>	ET	<code>&&</code>	ET
<code>or</code>	OU	<code> </code>	OU
<code>not</code>	NON	<code>!</code>	NON

Boucles

while (...) { ... }

do { ... } **while** (...)

for (int compteur(0) ; compteur < 10 ; compteur++)

Les fonctions

int addition (int a, int b) { ... }

void afficher() { ... }

int changer(int& nombre) { ... } → lors de l'utilisation de cette fonction on change aussi le nombre

Les Modules

fichier de spécification (.hpp) :

Il faut ajouter les lignes suivantes :

```
#ifndef MATH_HPP_INCLUDED
#define MATH_HPP_INCLUDED
...
#endif
```

fichier d'implémentation (.cpp) :

Il faut inclure le fichier d'entête avec : #include "nvmodule"

Documentation des fonctions de l'entête

```
/**
 * \brief Fonction qui ajoute 2 au nombre reçu en argument
 * \param nombreRecu Le nombre auquel la fonction ajoute 2
 * \return nombreRecu + 2
 */
```

fonction avec valeur d'argument par défaut

int nombreDeSecondes(int heures, int minutes = 0, int secondes = 0);

→ Appels possible : nombreDeSecondes(1), nombreDeSecondes(1, 2), nombreDeSecondes(1, 2, 3)

!!! Les paramètres par défaut doivent être à droite.

Les tableaux statiques

→ Déclaration : **type** **NOM**[**TAILLE**];

→ Accès aux éléments (commence de 0) : **NOM**[**indice**]

→ Fonction avec tableau en argument : **void** **fonction**(**type** **NOM**[**]**) { ... }

Les tableaux dynamiques

→ Il faut inclure : #include <vector>

→ Déclaration : **vector**<**type**> **NOM**(**TAILLE**);

exemple :

vector<int> tableau(5, 3); //Crée un tableau de 5 entiers valant tous 3

vector<string> listeNoms(12, "nom"); //Crée un tableau de 12 strings valant toutes « nom »

→ Ajouter un élément : **tableau.push_back**(8); //Ajouter l'élément 8 au tableau

→ Supprimer le dernier élément : **tableau.pop_back**();

→ Taille d'un tableau : `tableau.size();`
 → Fonction avec un tableau dynamique en paramètre en utilisant la référence avec & :
`void fonctionAvecTableau(vector<type>& NOM);`

Les tableaux multidimensionnels

→ En utilisant les tableaux statiques : `type NOM[tailleX][tailleY]`
 → En utilisant les vecteurs pour un tableau 2D : `vector<vector<type> > NOM;` (tableau de tableau)

Voir string comme un tableau

```
string texte("Hallo");
texte.push_back('w'); // Hallow
texte.size(); // 6
```

→ On peut faire la concaténation avec le +

Les Fichiers

bibliothèque des fichiers

```
#include <fstream>
```

ouvrir un fichier en écriture

```
string const nomFichier("C:/Nanoc/scores.txt");
ofstream monFlux(nomFichier.c_str()); → ofstream monFlux("C:/Nanoc/scores.txt");
if (monFLux) { // → ( on teste s'il n'y a pas d'erreur )
    monFlux << "Bonjour, je suis une phrase écrite dans un fichier." << endl;
    monFlux << 42.1337 << endl;
} else { cout << "ERREUR: Impossible d'ouvrir le fichier." << endl; }
```

→ Pour écrire dans un fichier sans l'écraser : `ofstream monFlux(nomFichier.c_str(), ios::app);`

ouvrir un fichier en lecture

```
ifstream monFlux("C:/Nanoc/C++/data.txt");
if (monFLux) { ... } else { cout << "ERREUR: Impossible d'ouvrir le fichier." << endl; }
```

→ lecture ligne par ligne : `string ligne; getline(monFlux, ligne);`
 → lecture mot par mot : `string mot; monFlux >> mot;`
 → lecture caractère par caractère : `char a; monFlux.get(a);`
!!! Pour changer de mode de lecture de mot par mot vers un autre il faut utiliser : `monFlux.ignore();`

fermer un fichier

```
monFLux.close();
```

Trouver la position du curseur

Pour ifstream	Pour ofstream
<code>tellg()</code>	<code>tellp()</code>

Déplacer le curseur

Pour ifstream	Pour ofstream
<code>seekg()</code>	<code>seekp()</code>

→ Les fonctions prennent 2 arguments : `flux.seekp(nombreCaracteres, position);`

→ Les positions possibles sont : début (ios::beg), fin (ios::end), position actuel (ios::cur).

Taille d'un fichier

→ On déplace le curseur à la fin et on demande la position du curseur.

Les Pointeurs

Déclaration d'un pointeur

type *pointeur; // le type du pointeur est celui de la variable, le pointeur contient l'adresse de la var

→ Il faut toujours initialiser un pointeur avec la valeur 0, car sans initialisation on ne sait jamais quelle case on est en train de manipuler.

Stocker une adresse

```
int ageUtilisateur(16);
```

```
int *ptr(0);
```

```
ptr = &ageUtilisateur; // pointeur contenant l'adresse de la variable ageUtilisateur
```

Accéder à la valeur pointée

```
cout << "La valeur est : " << *ptr << endl;
```

1. Il va dans la case mémoire nommée ptr.
2. Lit la valeur enregistrée.
3. "Suit la flèche" pour aller à l'adresse pointée.
4. Lit la valeur stockée dans la case.
5. Affiche cette valeur : ici, ce sera 16.

Pour une variable `int nombre` :

- `nombre` permet d'accéder à la **valeur** de la variable ;
- `&nombre` permet d'accéder à l'**adresse** de la variable.

Sur un pointeur `int *pointeur` :

- `pointeur` permet d'accéder à la **valeur du pointeur**, c'est-à-dire à l'adresse de la variable pointée ;
- `*pointeur` permet d'accéder à la **valeur de la variable pointée**.

Allocation dynamique

Allouer de la mémoire

```
int *pointeur(0);
```

```
pointeur = new int; // On demande une case mémoire pour stocker un entier.
```

```
*pointeur = 2; //On accède à la case mémoire pour en modifier la valeur
```

Libérer de la mémoire

```
delete pointeur; //On libère la case mémoire
```

```
pointeur = 0; //On indique que le pointeur ne pointe plus vers rien (!!! nécessaire)
```