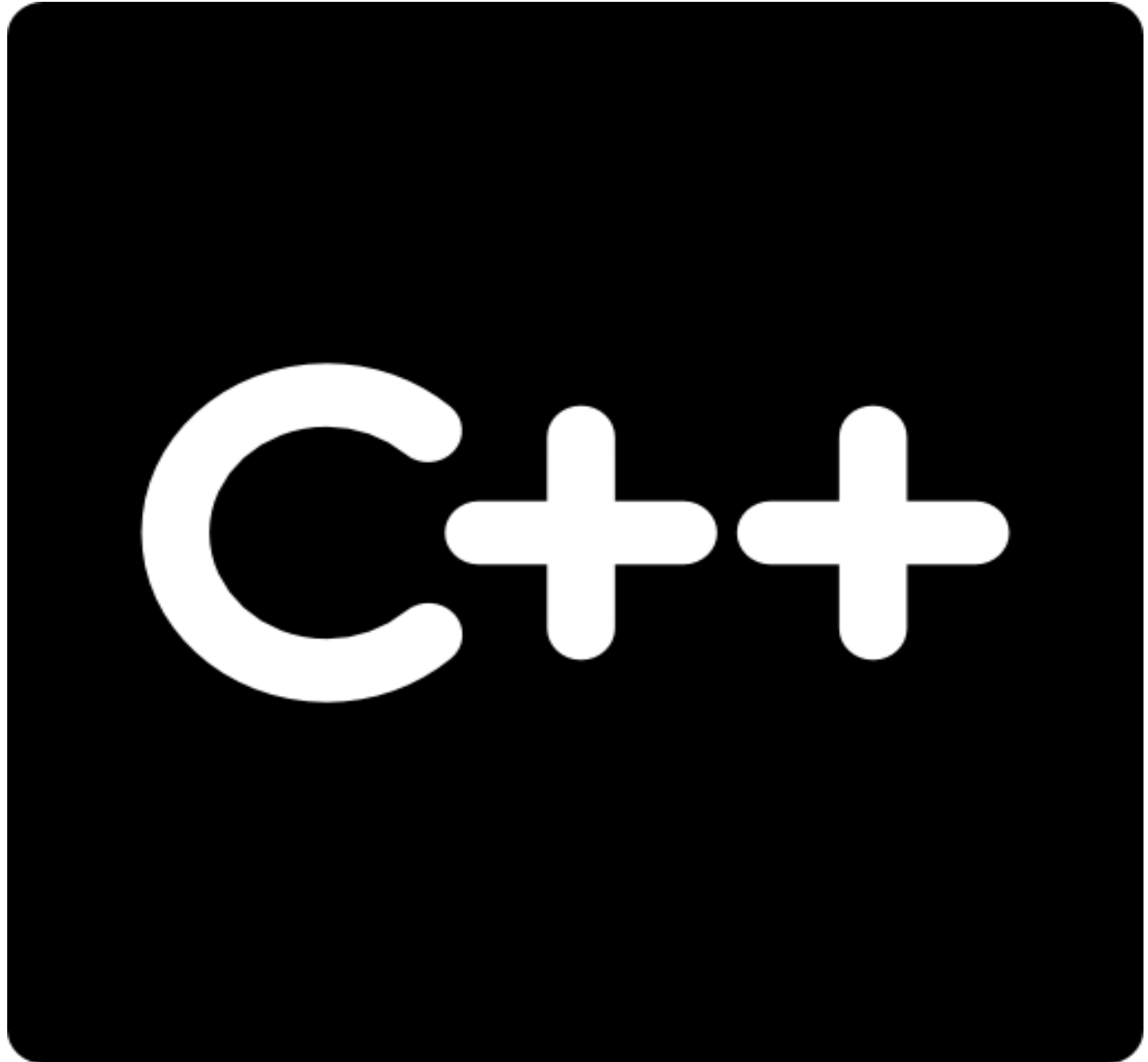


# Cours\_C++

## Table des matières

---

Cours C++ .....	3
Chapitre 1 - Découverte de la programmation C++ .....	3
1 - "Hello World !" .....	4
2 - Les variables .....	5
3 - Interaction et arithmétiques .....	8
4 - Les structures de contrôle .....	11
5 - Découper son programme en fonctions .....	15
6 - Les tableaux .....	18
7 - Lire et modifier des fichiers .....	22
TP1 - Le mot mystère .....	27
8 - Les pointeurs .....	29
Chapitre 2 - La POO (Programmation orienté objet) .....	32
1 - La vérité sur les string .....	32
2 - Les classes .....	33
3 - Les surcharges d'opérateurs .....	41
TP2 - La POO en pratique avec ZFraction .....	44
4 - Les classes et les pointeurs .....	44
5 - L'héritage .....	47
Chapitre 3 - Créer vos propres fenêtres avec Qt .....	48
1 - Compiler votre première fenêtre avec Qt .....	48
Chapitre 4 - Utiliser la bibliothèque standard .....	48
1 - Les conteneurs .....	49
Chapitre 5 - Notions avancés .....	49
1 - Les exeptions .....	50
Annexes .....	50
Logiciel à télécharger (IDE) .....	50
Suppléments Qt .....	50
Utiliser le presse papier avec Qt .....	50
Sécurisation d'un chat avec Qt .....	50
Le hachage avec Qt .....	50



Créé avec HelpNDoc Personal Edition: Produire des livres EPub gratuitement

## Chapitre 1 - Découverte de la programmation C++

# CHAPITRE 1 :

# Découverte de la

# programmation

# C++

---

Cr   avec HelpNDoc Personal Edition: [Cr  er des fichiers d'aide Qt Help multi-plateformes](#)

---

## 1 - "Hello World !"

Voici le programme de notre premier programme, un "Hello World !" :

```
#include <iostream>                //Inclusion de la
biblioth  que d'affichage

using namespace std;              //Espace de nom de la
biblioth  que standard

int main()                        //Fonction principal
{
    cout << "Hello World!" << endl; //Affichage du message
    avec un saut de ligne
    return 0;                    //Retour    0 pour dire
    que tout c'est bien passer
}
```

Ce programme doit nous retourner :

```

C:\Users\Mateo\Projets\test\bin\Debug\test.exe
Hello world!
Process returned 0 (0x0) execution time : 0.004 s
Press any key to continue.

```

Cr    avec HelpNDoc Personal Edition: [Cr   r facilement des fichiers Qt Help](#)

## 2 - Les variables

### Les types de variables :

Type	Explication	Valeurs possible
bool	Bool���an : 2 valeurs possible, true (vrai), false (faux).	true ou false
char	Un caract���re.	'a'
int	Un nombre entier.	-32768 �� 32767
unsigned int	Un nombre positif ou ��gal �� 0.	0 �� 65535
double	Un nombre �� virgule	-1.7*10 <sup>-308</sup> �� 1.7*10 <sup>308</sup>
string	Une cha����ne de caract���re	"Hello World!"

enum	Une énumération de valeurs	Direction{ Nord, Sud}

## Déclaration de variables :

Il existe 2 façon de déclarer des variables :

- **TYPE NOM** ( **VALEUR**);
- **TYPE NOM** = **VALEUR**;

Exemple :

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int ageUtilisateur(16);
7      return 0;
8  }
```

## Le cas des string :

Pour le cas des string, il faut en début de programme, inclure la bibliothèque des string en écrivant ceci : **#include <string>**.

## Déclaration sans initialisation :

Il est aussi possible de déclarer une variable sans initialiser de valeur à l'intérieur comme ceci : **TYPE NOM**;

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string nomJoueur;
8      int nombreJoueurs;
9      bool aGagne;      //Le joueur a-t-il gagné ?
10
11     return 0;
12 }

```

**Attention !**, il ne faut pas en cas de déclaration sans initialisation mettre des parenthèse !

## Afficher la valeur d'une variable :

Pour afficher la valeur d'une variable, on utilise l'instruction `cout` comme dans l'exemple suivant :

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int qiUtilisateur(150);
8      string nomUtilisateur("Albert Einstein");
9
10     cout << "Vous vous appelez " << nomUtilisateur << " et votre QI vaut " << qiUtilisateur
11     << endl;
12     return 0;
13 }

```

## Les références :

Les références permettent de créer un alias à une variable, ce qui permet à une variable d'avoir 2 nom, voici un exemple :

```

1  int ageUtilisateur(16); //Déclaration d'une variable.
2
3  int& maVariable(ageUtilisateur); //Déclaration d'une référence nommée maVariable qui est
   accrochée à la variable ageUtilisateur

```

### 3 - Interaction et arithm  tiques

#### Demander des information    l'utilisateur :

Pour demander des informations    l'utilisateur il faut utiliser la commande `cin` et mettre le r  sultat de cette commande dans une variable du type de l'information demand  .

##### Exemple :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Quel age avez-vous ?" << endl;
7
8     int ageUtilisateur(0); //On prepare une case m  moire pour stocker un entier
9
10    cin >> ageUtilisateur; //On fait entrer un nombre dans cette case
11
12    cout << "Vous avez " << ageUtilisateur << " ans !" << endl; //Et on l'affiche
13
14    return 0;
15 }
```

**Attention !**, Pour les string, il faut utiliser une m  thode particuli  re pour assurer que le programme ne bug pas !

##### Exemple de demande d'information    l'utilisateur avec des string :



```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      cout << "Quel est votre nom ?" << endl;
8      string nomUtilisateur("Sans nom"); //On crée une case mémoire pour contenir une chaîne
      de caractères
9      getline(cin, nomUtilisateur); //On remplit cette case avec toutela ligne que
      l'utilisateur a écrit
10
11     cout << "Combien vaut pi ?" << endl;
12     double piUtilisateur(-1.); //On crée une case mémoire pour stockerun nombre réel
13     cin >> piUtilisateur; //Et on remplit cette case avec ce qu'écritl'utilisateur
14
15     cout << "Vous vous appelez " << nomUtilisateur << " et vous pensez que pivaut " <<
      piUtilisateur << "." << endl;
16
17     return 0;
18 }

```

PS: Si vous vouliez demander d'abord des informations de type nombre et ensuite un string avec la fonction `getline()`, il faut ajouter après le demande des nombres l'instruction : `cin.ignore()`.

## Opération arithmétiques :

Voici un tableau représentant toutes les opérations arithmétiques disponible :

Opération	Symboles	Exemple
Addition	+	resultat = a + b
Soustraction	-	resultat = a - b
Multiplication	*	resultat = a * b
Division	/	resultat = a / b
Modulo	%	resultat = a % b

Addition (raccourci)	<code>+=</code>	<code>resultat += a</code>
Soustraction (raccourci)	<code>-=</code>	<code>resultat -= a</code>
Multiplication (raccourci)	<code>*=</code>	<code>resultat *= a</code>
Division (raccourci)	<code>/=</code>	<code>resultat /= a</code>
Modulo (raccourci)	<code>%=</code>	<code>resultat %= a</code>
Incrémentation	<code>++</code>	<code>a++</code> ou <code>++a</code>
Décrémentatio n	<code>--</code>	<code>a--</code> ou <code>--a</code>

## Les constantes :

Les constantes sont des variables qui une fois déclarer ne peuvent pas changer de valeurs. Cela peut servir pour conserver des nombres qui ne changeront pas comme  $\Pi$ .

Exemple:

- `double const pi(3.14159);`
- `double const g(9.81);`

## Encore plus de math avec `<cmath>` :

Nom de la fonction	Symbole	
Racine carré	$\sqrt{x}$	
Sinus	$\sin(x)$	
Cosinus	$\cos(x)$	
Tangente	$\tan(x)$	
Exponentielle	$e^x$	
Logarithme népérien	$\ln x$	
Logarithme en base 10	$\log_{10} x$	
Valeur absolue	$ x $	
Arrondi vers le bas	$\lfloor x \rfloor$	
Arrondi vers le haut	$\lceil x \rceil$	
Puissance	$\wedge$	

## 4 - Les structures de contrôle

### Les conditions

**Condition if, else :**

Voici ci après l'exemple d'une condition complète :

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int nbEnfants(2);
8
9      if (nbEnfants == 0)
10     {
11         cout << "Eh bien alors, vous n'avez pas d'enfant ?" << endl;
12     }
13     else if (nbEnfants == 1)
14     {
15         cout << "Alors, c'est pour quand le deuxieme ?" << endl;
16     }
17     else if (nbEnfants == 2)
18     {
19         cout << "Quels beaux enfants vous avez la !" << endl;
20     }
21     else
22     {
23         cout << "Bon, il faut arreter de faire des gosses maintenant !" << endl;
24     }
25
26     cout << "Fin du programme" << endl;
27     return 0;
28 }
```

**Condition switch :**

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int nbEnfants(2);
8
9     switch (nbEnfants)
10    {
11        case 0:
12            cout << "Eh bien alors, vous n'avez pas d'enfant ?" << endl;
13            break;
14
15        case 1:
16            cout << "Alors, c'est pour quand le deuxieme ?" << endl;
17            break;
18
19        case 2:
20            cout << "Quels beaux enfants vous avez la !" << endl;
21            break;
22
23        default:
24            cout << "Bon, il faut arreter de faire des gosses maintenant !" << endl;
25            break;
26    }
27
28    return 0;
29 }
```

## Les boucles :

Les boucles permettent de répéter une action suivant si une condition est vrai ou fausse.

### Boucle while :

```

1  int main()
2  {
3      int nbEnfants(-1); // Nombre négatif pour pouvoir entrer dans la boucle
4
5      while (nbEnfants < 0)
6      {
7          cout << "Combien d'enfants avez-vous ?" << endl;
8          cin >> nbEnfants;
9      }
10
11     cout << "Merci d'avoir indique un nombre d'enfants correct. Vous en avez " << nbEnfants
12     << endl;
13
14     return 0;
15 }

```

**Boucle do...while :**

```

1  int main()
2  {
3      int nbEnfants(0);
4
5      do
6      {
7          cout << "Combien d'enfants avez-vous ?" << endl;
8          cin >> nbEnfants;
9      } while (nbEnfants < 0);
10
11     cout << "Merci d'avoir indique un nombre d'enfants correct. Vous en avez " << nbEnfants
12     << endl;
13
14     return 0;
15 }

```

**Boucle for :**

```

1  int main()
2  {
3      int compteur(0);
4
5      for (compteur = 0 ; compteur < 10 ; compteur++)
6      {
7          cout << compteur << endl;
8      }
9
10     return 0;
11 }

```

## 5 - D  couper son programme en fonctions

### Les types de fonction :

Il y a plusieurs type de fonction suivant si la fonction renvoie une variable ou pas.

Si la fonction ne renvoie rien, on   crit ceci :

Type de fonction	Exemple
void	<pre>void maFonction() {     //��l��ments de la fonction }</pre>

Si la fonction renvoie une variable le type de la fonction que vous voulez programmer est le type de la variable que vous voulez retourner.

#### Exemple :

Voici ci-dessous un exemple d'une fonction qui ne renvoie aucune variables mais qui prend 2 param  tre :

```

1  #include <iostream>
2  using namespace std;
3
4  void dessineRectangle(int l, int h)
5  {
6      for(int ligne(0); ligne < h; ligne++)
7      {
8          for(int colonne(0); colonne < l; colonne++)
9          {
10             cout << "**";
11          }
12          cout << endl;
13      }
14  }
15
16  int main()
17  {
18      int largeur, hauteur;
19      cout << "Largeur du rectangle : ";
20      cin >> largeur;
21      cout << "Hauteur du rectangle : ";
22      cin >> hauteur;
23
24      dessineRectangle(largeur, hauteur);
25      return 0;
26  }

```

## Passage par valeur et passage par référence :

Exemple pour les explication : fonction qui ajoute 2 à un nombre passé en paramètre.

### Passage par valeur :

```

1  int ajouteDeux(int a)
2  {
3      a+=2;
4      return a;
5  }

```

Le passage par valeur veut dire que, quand un paramètre est passer à la fonction, la valeurs de la variable passé en paramètre est copié dans une seconde variable qui est à l'intérieur de la fonction. Ce qui veut dire que l'on occupe des cases mémoires supplémentaire. Par contre la variable de départ passé en paramètre ne change pas de valeurs, seule la variable de la fonction change de valeurs

### Passage par référence :



```

1 int ajouteDeux(int& a) //Notez le petit & !!!
2 {
3     a+=2;
4     return a;
5 }

```

Le passage par référence veut dire que, quand un paramètre est passé, l'ont lui attribut un alias qui est le nom de la variable de la fonction. Ce qui veut dire que le minimum de case mémoires sont utilisés. Mais **ATTENTION !** ceci modifie dans le cas de notre exemple la valeur du paramètre.

Exemple d'une fonction `echange()` avec un passage par référence :

```

1 void echange(double& a, double& b)
2 {
3     double temporaire(a); //On sauvegarde la valeur de 'a'
4     a = b;                 //On remplace la valeur de 'a' par celle de 'b'
5     b = temporaire;       //Et on utilise la valeur sauvegardée pour mettre l'ancienne
    valeur de 'a' dans 'b'
6 }
7
8 int main()
9 {
10     double a(1.2), b(4.5);
11
12     cout << "a vaut " << a << " et b vaut " << b << endl;
13
14     echange(a,b); //On utilise la fonction
15
16     cout << "a vaut " << a << " et b vaut " << b << endl;
17     return 0;
18 }

```

Passage par référence constante :

Le passage par référence constante permet d'avoir l'avantage de la référence évitant ainsi la copie de la variable et de ne pas pouvoir modifier l'argument de la fonction. Voici un code d'exemple :

```

1 void f1(string const& texte); //Pas de copie et pas de modification possible
2 {
3 }

```

**Utiliser plusieurs fichier pour fragmenter son programme :**

Nous allons créer notre fonction `ajouteDeux()` dans un fichier séparé.

```
1 int ajouteDeux(int a)
2 {
3     a+=2;
4     return a;
5 }
```

Contenue du fichier `math.h` :

```
1 #ifndef MATH_H_INCLUDED
2 #define MATH_H_INCLUDED
3
4 int ajouteDeux(int nombreRecu);
5
6 #endif // MATH_H_INCLUDED
```

Dans le fichier source, il suffit de mettre le code de notre fonction et tous en haut du fichier d'inclure le prototype : `#include "math.h"`

## Arguments avec des valeurs par défaut :

Pour mettre des valeurs par défaut au arguments d'une fonction, il suffit de mettre `: = valeur`.

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

## 6 - Les tableaux

### Les tableaux statiques :

Les tableaux permettent de regrouper plusieurs informations de même type dans une seule variable et ainsi de gagner du temps sur l'affichage ou le traitement.

#### Déclaration d'un tableau statique :

Un tableau statique se déclare de cette manière : **TYPE NOM [ TAILLE ] ;**

Voici un exemple de déclaration de tableau statique :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int meilleurScore[5];      //Déclare un tableau de 5 int
7
8     double anglesTriangle[3]; //Déclare un tableau de 3 double
9
10    return 0;
11 }
```

**Attention !**, si vous souhaitez déclarer la taille du tableaux avec une variable, il faut **ABSOLUMENT** que la variable soit une constante !

### Accès aux éléments du tableau :

Pour accéder aux éléments du tableau, il suffit d'appeler la variable en rajoutant des crochets et la position désirer que ce soit pour le remplissage ou la lecture de la case que vous souhaitez sélectionner.

Exemple :

```
1 meilleurScore[2] = 5;
```

Pour le parcourir, nous pouvons utiliser une boucle `for` comme ceci :

```
1 double moyenne(0);
2 for(int i(0); i<nombreNotes; ++i)
3 {
4     moyenne += notes[i]; //On additionne toutes les notes
5 }
6 //En arrivant ici, la variable moyenne contient la somme des notes (79.5)
7 //Il ne reste donc qu'à diviser par le nombre de notes
8 moyenne /= nombreNotes;
```

### Parcourir son tableau avec une fonction :

Pour parcourir son tableau avec une fonction, il nous faut savoir qu'il y a quelques restrictions :

- Une fonction ne peut pas renvoyer de tableau statique.
- Un tableau statique ne peut passer en argument de fonction qu'en temps que référence.

Bien sûr pour cela il n'est pas nécessaire de mettre le `&` après le type de variable, ceci est fait automatiquement. Par contre il faut rajouter un second argument qui contiendra la taille du tableau.

Voici un exemple pour illustrer mes propos :

```

1  /*
2  *  Fonction qui calcule la moyenne des éléments d'un tableau
3  *  - tableau : Le tableau dont on veut la moyenne
4  *  - tailleTableau : La taille du tableau
5  */
6  double moyenne(double tableau[], int tailleTableau)
7  {
8      double moyenne(0);
9      for(int i(0); i<tailleTableau; ++i)
10     {
11         moyenne += tableau[i];    //On additionne toutes les valeurs
12     }
13     moyenne /= tailleTableau;
14
15     return moyenne;
16 }

```

## Les tableaux dynamiques :

Les tableaux dynamique ce base sur le même principe que les tableaux statiques mais eux peuvent prendre toutes les tailles. Il peuvent par exemple contenir que 3 valeurs, ou bien 8, ou 19...

### Déclaration d'un tableau dynamique :

Pour déclarer un tableau dynamique, il faut commencer par importer une bibliothèque de cette manière : `#include <vector>`.

Ensuite pour la déclaration à proprement dite, il faut faire comme ceci : `vector<TYPE> NOM ( TAILLE )` ;

Voici un exemple de déclaration de tableau dynamique :

```

1  #include <iostream>
2  #include <vector> //Ne pas oublier !
3  using namespace std;
4
5  int main()
6  {
7      vector<int> tableau(5);
8
9      return 0;
10 }

```

REMARQUE : Avec ce type de tableaux, on peut directement remplir toutes les cases du tableau avec une donnée.

Voici un exemple :

```
1 vector<int> tableau(5, 3); //Crée un tableau de 5 entiers valant tous 3
2 vector<string> listeNoms(12, "Sans nom");
3 //Crée un tableau de 12 strings valant toutes « Sans nom »
```

### Accès aux éléments du tableau dynamique :

Pour accéder aux éléments d'un tableaux dynamique, la méthode est la même que pour accéder aux éléments d'un tableau statique.

### Changer la taille :

Pour changer la taille du tableau on peut faire plusieurs chose.

Commençons par ajouter une valeur en fin de tableau :

```
1 vector<int> tableau(3,2); //Un tableau de 3 entiers valant tous 2
2 tableau.push_back(8);
3 //On ajoute une 4ème case au tableau qui contient la valeur 8
```

Maintenant supprimons la dernière valeur du tableau :

```
1 vector<int> tableau(3,2); //Un tableau de 3 entiers valant tous 2
2 tableau.pop_back(); //Et hop ! Plus que 2 cases
3 tableau.pop_back(); //Et hop ! Plus que 1 case
```

Pour finir, voici la dernière méthode que je vous présenterai, il s'agit de la méthode permettant de savoir la taille du tableau :

```
1 vector<int> tableau(5,4); //Un tableau de 5 entiers valant tous 4
2 int const taille(tableau.size());
3 //Une variable qui contient la taille du tableau
4 //La taille peut varier mais la valeur de cette variable ne changera pas
5 //On utilise donc une constante
6 //À partir d'ici, la constante 'taille' vaut donc 5
```

### Les tableaux dynamiques et les fonctions :

Dans le cas d'un tableau dynamique, le traitement pour une fonction est plus simple, il suffit de passer le tableau en passage par référence constante comme ceci :

```
1 //Une fonction recevant un tableau d'entiers en argument
2 void fonction(vector<int> const& a)
3 {
4     //...
5 }
```

## Les tableaux multi-dimensionnel :

Les tableaux multi-dimensionnel se déclare de la même manière que les tableaux statique, ce sont des tableaux statique mais avec 2 dimension.

Il se déclare de la manière suivante :

```
int tableau[5][4];
//tableau[tailleX][tailleY];
```

Ce tableau contiendra en résumé 4 tableaux statique de 5 cases.

## 7 - Lire et modifier des fichiers

### Écrire dans un fichier :

Dans un premier temps, pour manipuler les fichiers, il faut importer un module. Ce module s'importe comme d'habitude avec un `#include` et s'intitule `<fstream>`.

Pour l'importer, il faut faire comme ceci : `#include <fstream>`.

### Ouvrir un fichier en écriture :

Pour ouvrir un fichier en écriture, il suffit de faire comme dans l'exemple ci-dessous :

```

1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 int main()
6 {
7     ofstream monFlux("C:/Nanoc/scores.txt");
8     //Déclaration d'un flux permettant d'écrire dans le fichier
9     // C:/Nanoc/scores.txt
10    return 0;
11 }

```

Mais avec cette technique qui est la base, il peut y avoir des problèmes, alors on teste si le fichier est bien ouvert avant de faire quoi que ce soit :

```

1  ofstream monFlux("C:/Nanoc/scores.txt"); //On essaye d'ouvrir le fichier
2
3  if(monFlux) //On teste si tout est OK
4  {
5      //Tout est OK, on peut utiliser le fichier
6  }
7  else
8  {
9      cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
10 }

```

### Écrire dans un flux :

Maintenant que notre fichier est ouvert, nous pouvons écrire à l'intérieur. La technique est la même que pour afficher du texte dans la console. Il faut utiliser les chevrons :

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      string const nomFichier("C:/Nanoc/scores.txt");
9      ofstream monFlux(nomFichier.c_str());
10
11     if(monFlux)
12     {
13         monFlux << "Bonjour, je suis une phrase écrite dans un fichier." << endl;
14         monFlux << 42.1337 << endl;
15         int age(23);
16         monFlux << "J'ai " << age << " ans." << endl;
17     }
18     else
19     {
20         cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
21     }
22     return 0;
23 }

```

**Attention !**, si vous souhaitez ajouter des informations à la fin du documents s'il existe, il faut écrire dans la déclaration:  
`ofstream monFlux("Chemin", ios::app); !`

## Lire dans un fichier :

### Déclaration du flux :

Pour lire dans un fichier la méthode est la même concernant la déclaration du flux. Sauf que dans le cas d'une ouverture, la classe à appeler est : `ifstream`.

### Lecture dans le flux :

Pour lire le contenu d'un fichier, il y a 3 manière de le lire :

- Ligne par ligne, avec `getline()`;
- Mot par mot, avec les chevrons `>>`;
- Caractère par caractère, avec `get()`.



**Ligne par ligne :**

```
1 string ligne;
2 getline(monFlux, ligne); //On lit une ligne complète
```

**Mot par mot :**

```
1 double nombre;
2 monFlux >> nombre; //Lit un nombre à virgule depuis le fichier
3 string mot;
4 monFlux >> mot;    //Lit un mot depuis le fichier
```

**Caractère par caractère :**

```
1 char a;
2 monFlux.get(a);
```

**Si plusieurs méthode de lecture sont employé :**

```
1 ifstream monFlux("C:/Nanoc/C++/data.txt");
2
3 string mot;
4 monFlux >> mot;          //On lit un mot depuis le fichier
5
6 monFlux.ignore();        //On change de mode
7
8 string ligne;
9 getline(monFlux, ligne); //On lit une ligne complète
```

**Lire un fichier dans sa globalité :**

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      ifstream fichier("C:/Nanoc/fichier.txt");
9
10     if(fichier)
11     {
12         //L'ouverture s'est bien passée, on peut donc lire
13
14         string ligne; //Une variable pour stocker les lignes lues
15
16         while(getline(fichier, ligne)) //Tant qu'on n'est pas à la fin, on lit
17         {
18             cout << ligne << endl;
19             //Et on l'affiche dans la console
20             //Ou alors on fait quelque chose avec cette ligne
21             //À vous de voir
22         }
23     }
24     else
25     {
26         cout << "ERREUR: Impossible d'ouvrir le fichier en lecture." << endl;
27     }
28
29     return 0;
30 }

```

## Quelques astuces :

### Fermer prématurément un fichier :

Pour refermer un fichier prématurément, il suffit d'écrire : `monFlux.close()`.

## TP1 - Le mot mystère

```

1  #include <iostream>
2  #include <string>
3  #include <ctime>
4  #include <cstdlib>
5
6  using namespace std;
7
8  string melangerLettres(string mot)
9  {
10     string melange;
11     int position(0);
12
13     //Tant qu'on n'a pas extrait toutes les lettres du mot
14     while (mot.size() != 0)
15     {
16         //On choisit un numéro de lettre au hasard dans le mot
17         position = rand() % mot.size();
18         //On ajoute la lettre dans le mot mélangé
19         melange += mot[position];
20         //On retire cette lettre du mot mystère
21         //Pour ne pas la prendre une deuxième fois
22         mot.erase(position, 1);
23     }
24
25     //On renvoie le mot mélangé
26     return melange;
27 }
28
29 int main()
30 {
31     string motMystere, motMelange, motUtilisateur;
32
33     //Initialisation des nombres aléatoires
34     srand(time(0));
35
36     //1 : On demande de saisir un mot
37     cout << "Saisissez un mot" << endl;
38     cin >> motMystere;
39
40     //2 : On récupère le mot avec les lettres mélangées dans motMelange
41     motMelange = melangerLettres(motMystere);
42
43     //3 : On demande à l'utilisateur quel est le mot mystère
44     do
45     {
46         cout << endl << "Quel est ce mot ? " << motMelange << endl;
47         cin >> motUtilisateur;
48
49         if (motUtilisateur == motMystere)
50         {
51             cout << "Bravo !" << endl;
52         }
53         else
54         {
55             cout << "Ce n'est pas le mot !" << endl;
56         }
57     }while (motUtilisateur != motMystere);
58     //On recommence tant qu'il n'a pas trouvé
59
60     return 0;
61 }

```

## 8 - Les pointeurs

Les pointeurs sont des variables qui sont capable de contenir que des adresse de variables d  clar  .

### Afficher l'adresse d'une variable :

Pour afficher l'adresse d'une variable, il suffit de rajouter le pr  fixe & au nom de la variable. Voici ci-apr  s un exemple :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int ageUtilisateur(16);
7     cout << "L'adresse est : " << &ageUtilisateur << endl;
8     //Affichage de l'adresse de la variable
9     return 0;
10 }
```

L'adresse retourn   sera cod   en hexad  cimal sous le format suivant : 0x22ff1c.

### Les pointeurs :

#### D  clarer un pointeur :

Pour d  clarer un pointeur, il faut ajouter le pr  fixe \* au nom du pointeur. Pour savoir quelle type mettre au pointeur, il faut mettre le m  me type que l'adresse de la variable pointer. Voici un exemple :

**Attention !**, il ne faut JAMAIS d  clarer un pointeur vide, il faut toujours le d  clarer avec la valeur 0 !

```

1 int *pointeur(0);
2 double *pointeurA(0);
3 unsigned int *pointeurB(0);
4 string *pointeurC(0);
5 vector<int> *pointeurD(0);
6 int const *pointeurE(0);

```

### Stocker une adresse dans un pointeur :

Pour stocker une adresse dans un pointeur, il faut procéder comme ceci :

```

1 int main()
2 {
3     int ageUtilisateur(16);
4     //Une variable de type int
5     int *ptr(0);
6     //Un pointeur pouvant contenir l'adresse d'un nombre entier
7
8     ptr = &ageUtilisateur;
9     //On met l'adresse de 'ageUtilisateur' dans le pointeur 'ptr'
10
11     return 0;
12 }

```

### Afficher l'adresse du pointeur :

Pour visualiser l'adresse du pointer par le pointeur, il suffit de la montrer dans le flux de sortie standard comme ci-dessous :

```

1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int ageUtilisateur(16);
7     int *ptr(0);
8
9     ptr = &ageUtilisateur;
10
11     cout << "L'adresse de 'ageUtilisateur' est : " << &ageUtilisateur << endl;
12     cout << "La valeur de pointeur est : " << ptr << endl;
13
14     return 0;
15 }

```

### Accéder à la valeur pointer :

Pour accéder à la valeur pointer, rien de plus simple, il suffit de mettre le pointeur dans le flux de sortie standard en rajoutant le préfixe \* comme ci-après :

```
1 int main()
2 {
3     int ageUtilisateur(16);
4     int *ptr(0);
5
6     ptr= &ageUtilisateur;
7
8     cout << "La valeur est : " << *ptr << endl;
9
10    return 0;
11 }
```

## L'allocation dynamique de la mémoire :

L'allocation dynamique de la mémoire permet d'avoir le contrôle complet de la mémoire dans le programme. Voici un exemple pour vous faire comprendre le fonctionnement :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int* pointeur(0);
7     pointeur = new int;
8
9     cout << "Quel est votre age ? ";
10    cin >> *pointeur;
11    //On écrit dans la case mémoire pointée par le pointeur 'pointeur'
12
13    cout << "Vous avez " << *pointeur << " ans." << endl;
14    //On utilise à nouveau *pointeur
15    delete pointeur;    //Ne pas oublier de libérer la mémoire
16    pointeur = 0;        //Et de faire pointer le pointeur vers rien
17
18    return 0;
19 }
```

## Chapitre 2 - La POO (Programmation orienté objet)

# CHAPITRE 2 : La programmation orienté objet

---

Créé avec HelpNDoc Personal Edition: [Créer des aides HTML, DOC, PDF et des manuels depuis une même source](#)

---

### 1 - La vérité sur les string

#### Les méthode de l'objet `string` :

##### Méthode `.size()` :

La méthode `.size()` permet de connaître la longueur de la chaîne de caractère.

##### Méthode `.erase()` :

Permet de supprimer tout le contenu de la chaîne de caractère.



**Méthode `.c_str()` :**

Permet de renvoyer un pointeur vers le tableau de *char* que contient l'objet `string`.

**2 - Les classes****Cr   er une classe :**

Pour cr   er une classe, il faut d   j    lui trouver un nom comme pour une variable, mais ATTENTION, le nom d'une classe doit toujours commencer par une majuscule. Dans cette classe nous allons mettre des attribut qui eux on comme particularit    de toujours commencer par "m\_". Ensuite vous pourrez mettre vos m   thodes qui elle reprennent tous les concepts des fonctions.

Voici un exemple de prototype de classe :

```

1  class Personnage
2  {
3      // Méthodes
4      void recevoirDegats(int nbDegats)
5      {
6
7      }
8
9      void attaquer(Personnage &cible)
10     {
11
12     }
13
14     void boirePotionDeVie(int quantitePotion)
15     {
16
17     }
18
19     void changerArme(string nomNouvelleArme, int degatsNouvelleArme)
20     {
21
22     }
23
24     bool estVivant()
25     {
26
27     }
28
29     // Attributs
30     int m_vie;
31     int m_mana;
32     string m_nomArme;
33     int m_degatsArme;
34 };

```

**Attention !**, il faut penser à mettre un point virgule après la class !!!!

## Droits d'accès et encapsulation :

Dans la POO, il existe deux type de droits d'accès :

- public : les méthodes et attributs peuvent être appelés de l'extérieur;
- private : les méthodes et attributs ne peuvent être appelés de nulle part, par défaut tout ce qui est dans une classe est private.

Maintenant voici le code précédent mais en tenant compte des droits d'accès :

```

1  class Personnage
2  {
3      // Tout ce qui suit est public (accessible depuis l'extérieur)
4      public:
5
6      void recevoirDegats(int nbDegats)
7      {
8
9      }
10
11     void attaquer(Personnage &cible)
12     {
13
14     }
15
16     void boirePotionDeVie(int quantitePotion)
17     {
18
19     }
20
21     void changerArme(string nomNouvelleArme, int degatsNouvelleArme)
22     {
23
24     }
25
26     bool estVivant()
27     {
28
29     }
30
31     // Tout ce qui suit est privé (inaccessible depuis l'extérieur)
32     private:
33
34     int m_vie;
35     int mMana;
36     string m_nomArme;
37     int m_degatsArme;
38 };

```

## Séparer prototype et définitions :

Le but de séparer prototype et définitions est pour les mêmes raisons que les fonction pour que le programmes soit découper en plusieurs fichier et ainsi portable.

**Personnage.h :**

```

1  #ifndef DEF_PERSONNAGE
2  #define DEF_PERSONNAGE
3
4  #include <string>
5
6  class Personnage
7  {
8      public:
9
10     void recevoirDegats(int nbDegats);
11     void attaquer(Personnage &cible);
12     void boirePotionDeVie(int quantitePotion);
13     void changerArme(std::string nomNouvelleArme, int degatsNouvelleArme);
14     bool estVivant();
15
16     private:
17
18     int m_vie;
19     int mMana;
20     std::string m_nomArme; //Pas de using namespace std, il faut donc mettre std:: devant
    string
21     int m_degatsArme;
22 };
23
24 #endif

```

**Personnage.cpp :**

```

1  #include "Personnage.h"
2
3  using namespace std;
4
5  void Personnage::recevoirDegats(int nbDegats)
6  {
7      m_vie -= nbDegats;
8      //On enlève le nombre de dégâts reçus à la vie du personnage
9
10     if (m_vie < 0) //Pour éviter d'avoir une vie négative
11     {
12         m_vie = 0; //On met la vie à 0 (cela veut dire mort)
13     }
14 }
15
16 void Personnage::attaquer(Personnage &cible)
17 {
18     cible.recevoirDegats(m_degatsArme);
19     //On inflige à la cible les dégâts que cause notre arme
20 }
21
22 void Personnage::boirePotionDeVie(int quantitePotion)
23 {
24     m_vie += quantitePotion;
25
26     if (m_vie > 100) //Interdiction de dépasser 100 de vie
27     {
28         m_vie = 100;
29     }
30 }
31
32 void Personnage::changerArme(string nomNouvelleArme, int degatsNouvelleArme)
33 {
34     m_nomArme = nomNouvelleArme;
35     m_degatsArme = degatsNouvelleArme;
36 }
37
38 bool Personnage::estVivant()
39 {
40     return m_vie > 0;
41 }

```

## Constructeurs et destructeurs :

Le rôle du constructeur est d'initialiser les attributs de la classe et le rôle du destructeurs est de supprimer les attribut allouer dynamiquement.

### Constructeur :

```

1 #include <string>
2
3 class Personnage
4 {
5     public:
6
7     Personnage(); //Constructeur
8     void recevoirDegats(int nbDegats);
9     void attaquer(Personnage &cible);
10    void boirePotionDeVie(int quantitePotion);
11    void changerArme(std::string nomNouvelleArme, int degatsNouvelleArme);
12    bool estVivant();
13
14
15    private:
16
17    int m_vie;
18    int mMana;
19    std::string m_nomArme;
20    int m_degatsArme;
21 };

```

```

1 Personnage::Personnage()
2 {
3     m_vie = 100;
4     mMana = 100;
5     m_nomArme = "Épée rouillée";
6     m_degatsArme = 10;
7 }

```

OU

```

1 Personnage::Personnage() : m_vie(100), mMana(100), m_nomArme("Épée rouillée"),
  m_degatsArme(10)
2 {
3     //Rien à mettre dans le corps du constructeur, tout a déjà été fait !
4 }

```

**Surcharger le constructeur :**

```

1 Personnage(std::string nomArme, int degatsArme);

```

```

1 Personnage::Personnage(string nomArme, int degatsArme) : m_vie(100),
  mMana(100),m_nomArme(nomArme), m_degatsArme(degatsArme)
2 {
3
4 }

```

**Le destructeur :**

```
1 ~Personnage();
```

```
1 Personnage::~~Personnage()
2 {
3     /* Rien à mettre ici car on ne fait pas d'allocation dynamique
4     dans la classe Personnage. Le destructeur est donc inutile mais
5     je le mets pour montrer à quoi cela ressemble.
6     En temps normal, un destructeur fait souvent des delete et quelques
7     autres vérifications si nécessaire avant la destruction de l'objet. */
8 }
```

## Les méthodes constantes :

Les méthodes constantes sont des méthodes qui sont que en "lecture seule". Exemple : si vous avez une méthode qui se contente juste d'afficher quelque chose à l'écran, c'est une méthode constante. Les méthodes constantes sont, en résumé, les méthodes qui ne modifient aucun attribut.

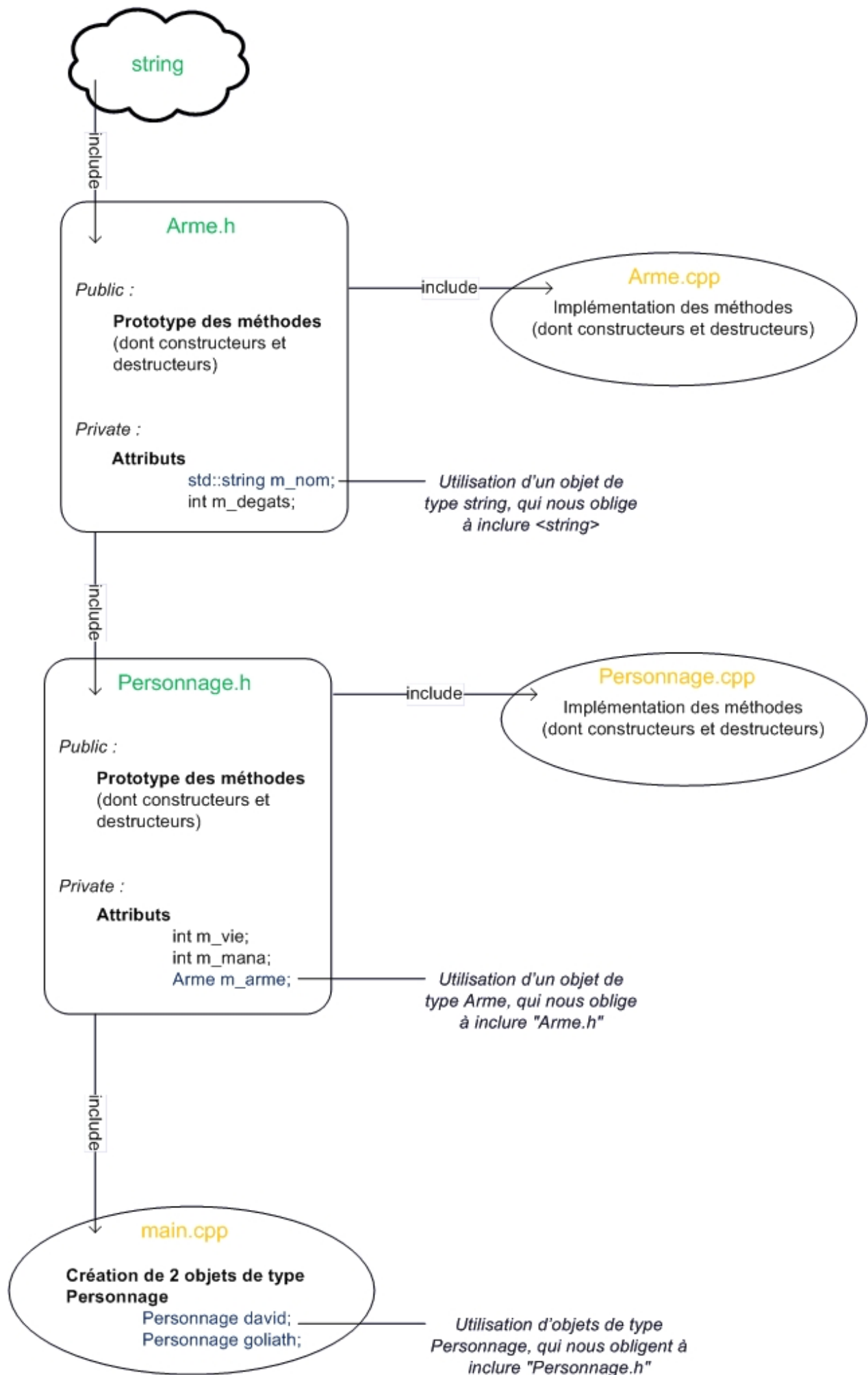
Exemple de méthode constante :

```
1 //Prototype de la méthode (dans le .h) :
2 void maMethode(int parametre) const;
3
4
5 //Déclaration de la méthode (dans le .cpp) :
6 void MaClasse::maMethode(int parametre) const
7 {
8
9 }
```

## Associer des classes entre elles :

Pour faire cela, il suffit d'appeler une autre classe dans notre classe et de mettre un attribut avec notre classe.

## MEGA RESUME :





### 3 - Les surcharges d'op  rateurs

**Notre classe type pour ce chapitre :**

**Dur  e.h :**

```
1  #ifndef DEF_DUREE
2  #define DEF_DUREE
3
4  class Duree
5  {
6      public:
7
8      Duree(int heures = 0, int minutes = 0, int secondes = 0);
9
10     private:
11
12     int m_heures;
13     int m_minutes;
14     int m_secondes;
15 };
16
17 #endif
```

**Dur  e.cpp:**

```
1  #include "Duree.h"
2
3  Duree::Duree(int heures, int minutes, int secondes) : m_heures(heures), m_minutes(minutes),
4      m_secondes(secondes)
5  {
6  }
```

Les surcharge d'op  rateur permettent dans la cas de notre classe dur  e de d  finir comment notre objet va r  agir aux op  rateur de comparaison et arithm  tique.

## Les opérateurs de comparaisons :

Exemple avec l'opérateur == :

```
1 bool operator==(Duree const& a, Duree const& b);
```

Cette fonction est à mettre dans le fichier de la classe pour plus de lisibilité mais pas dans la classe car ce n'est pas une méthode de la classe.

Dans le fichier .cpp, il faut ajouter une méthode car cette surcharge ne peut être directement faite qu'elle est hors de la classe et que les attributs de la classe sont privés.

```
1 bool Duree::estEgal(Duree const& b) const
2 {
3     //Teste si a.m_heure == b.m_heure etc.
4     if (m_heures == b.m_heures && m_minutes == b.m_minutes && m_secondes == b.m_secondes)
5         return true;
6     else
7         return false;
8 }
```

```
1 bool operator==(Duree const& a, Duree const& b)
2 {
3     return a.estEgal(b);
4 }
```

Bien sûr ne pas oublier dans la surcharge de passer les arguments en références constantes pour ne pas qu'il y ait de copie, ni de modification.

Tableau des opérateurs de comparaison et arithmétiques :

==	operator==
!=	operator!=
<	operator<
>	operator>

<=	operator<=
>=	operator>=
+	operator+
+=	operator+=
-	operator-
--	operator--
*	operator*
*=	operator*=
/	operator/
/=	operator/=
%	operator%
%=	operator%=

**Les opérateurs de flux :**

```
1 bool operator==(Duree const& a, Duree const& b);
```

```

1 void Duree::afficher(ostream &flux) const
2 {
3     flux << m_heures << "h" << m_minutes << "m" << m_secondes << "s";
4 }

```

```

1 bool operator==(Duree const& a, Duree const& b);

```

---

Créé avec HelpNDoc Personal Edition: [Produire facilement des livres électroniques Kindle](#)

---

## TP2 - La POO en pratique avec ZFraction

---

Créé avec HelpNDoc Personal Edition: [Générateur complet d'aides multi-formats](#)

---

### 4 - Les classes et les pointeurs

#### Pointeur d'une classe vers une autre :

Ici, nous allons voir comment un personnage peut changer d'arme.

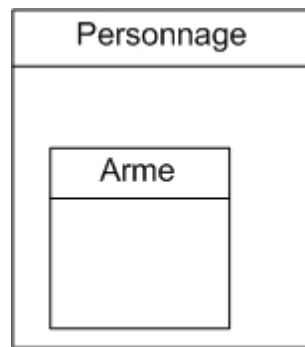
Maintenant, notre code ressemble à celui-ci :

```

1 class Personnage
2 {
3     public:
4
5     //Quelques méthodes...
6
7     private:
8
9     Arme m_arme; // L'Arme est "contenue" dans le Personnage
10    //...
11 };

```

Voici le schéma que l'on peut associer à ce code:

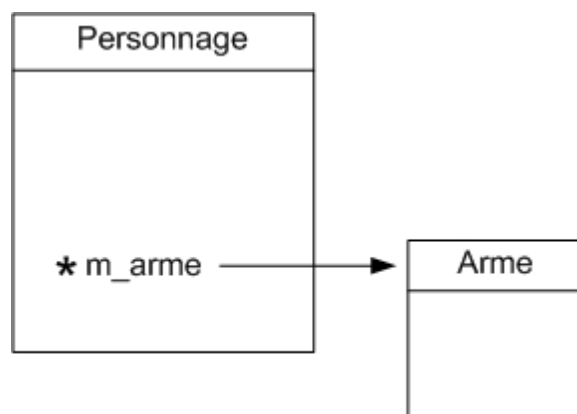


Maintenant, mettons un pointeur à la place d'une instance de classe classique :

```

1  class Personnage
2  {
3      public:
4
5      //Quelques méthodes...
6
7      private:
8
9      Arme *m_arme;
10     //L'Arme est un pointeur, l'objet n'est plus contenu dans le Personnage
11     //...
12 };
  
```

Voici son schéma :



Maintenant, le personnage peut changer d'arme ou ne plus en avoir car l'objet **Arme** ne lui appartient pas, il pointe vers lui.

## Gestion de l'allocation dynamique :

### Allocation de la mémoire pour l'objet :

Maintenant, il faut déclarer le pointeur pour qu'il pointe vers quelque chose, sinon ça ne sert à rien. Il faudra écrire le code suivant dans le constructeur de notre classe `Personnage` :

```
1 Personnage::Personnage(string nomArme, int degatsArme) : m_arme(0), m_vie(100), m_mana(100)
2 {
3     m_arme = new Arme(nomArme, degatsArme);
4 }
```

Ensuite, on a le choix de mettre ou non des arguments dans la déclaration suivant quelle constructeur on souhaite appeler.

**Attention !**, par sécurité, il faut penser à initialiser le pointeur à 0 !!!!

### Désallocation de la mémoire pour l'objet :

Il faut penser à supprimer notre pointeur lors de la destruction de `Personnage`, sinon il occupera de la mémoire pour rien. Il faut mettre le code suivant :

```
1 Personnage::~~Personnage()
2 {
3     delete m_arme;
4 }
```

Avec ce code dans le destructeur de la classe `Personnage`, tout sera supprimé correctement.

### Penser que `m_arme` est un pointeur ! :

Maintenant, il faut penser que `m_arme` est un pointeur, donc remplacer les éléments en rapport avec `m_arme` par `->`.

```
1 void Personnage::attaquer(Personnage &cible)
2 {
3     cible.recevoirDegats(m_arme->getDegats());
4 }
```

## Le pointeur `this` :

Le pointeur `this`, sert à pointer la classe vers elle même. Utiliser comme ça on peut avoir l'adresse de la classe et utiliser avec `*`, on peut pointer vers la classe elle même.

## Le constructeur de copie :

Le constructeur de copie sert quand vous voulez créer un objet à partir d'un objet.

Exemple de code de constructeur de copie :

```
1 Objet(Objet const& objetACopier);
```

```
1 Personnage::Personnage(Personnage const& personnageACopier)
2   : m_vie(personnageACopier.m_vie), m_mana(personnageACopier.m_mana), m_arme(0)
3 {
4
5 }
```

## 5 - L'héritage

## Chapitre 3 - Cr   er vos propres fen   tres avec Qt

# CHAPITRE 3 : Cr   er vos propres fen   tres avec Qt

## 1 - Compiler votre premi   re fen   tre avec Qt

## Chapitre 4 - Utiliser la biblioth   que standard



# CHAPITRE 4 :

# Utiliser la

# bibliothèque

# standard

---

Créé avec HelpNDoc Personal Edition: [Générateur facile de livres électroniques et documentation](#)

---

## 1 - Les conteneurs

---

Créé avec HelpNDoc Personal Edition: [Créer facilement des fichiers Qt Help](#)

---

## Chapitre 5 - Notions avancés

# CHAPITRE 5 :

# Notions avancés

---

Créé avec HelpNDoc Personal Edition: [Produire des aides en ligne pour les applications Qt](#)

---

## 1 - Les exeptions

---

Créé avec HelpNDoc Personal Edition: [Produire des livres Kindle gratuitement](#)

---

## Annexes

---

Créé avec HelpNDoc Personal Edition: [Créer de la documentation iPhone facilement](#)

---

## Logiciel à télécharger (IDE)

---

Créé avec HelpNDoc Personal Edition: [Créer de la documentation iPhone facilement](#)

---

## Suppléments Qt

---

Créé avec HelpNDoc Personal Edition: [Créer des documents d'aide PDF facilement](#)

---

## Utiliser le presse papier avec Qt

---

Créé avec HelpNDoc Personal Edition: [Produire facilement des livres électroniques Kindle](#)

---

## Sécurisation d'un chat avec Qt

---

Créé avec HelpNDoc Personal Edition: [Avantages d'un outil de création d'aide](#)

---

## Le hachage avec Qt

---

Créé avec HelpNDoc Personal Edition: [Créer des documentations web iPhone](#)

---