Chapitre 1. Introduction à la programmation orientée objets (POO)

1. Introduction

1.1. Qu'est-ce qu'un programme ?

Programmer un ordinateur, c'est lui fournir une série d'instructions qu'il doit exécuter. Ces instructions sont généralement écrites dans un langage dit évolué. Le programme est écrit suivant une notation conventionnelle composée d'un vocabulaire, de règles de grammaire et de signification, cette notation est appelée langage de programmation. Avant d'être exécutées, sont traduites en langage machine (qui est le langage du microprocesseur). Cette traduction s'appelle compilation et elle est effectuée automatiquement par un programme appelé compilateur.

Pour le programmeur, cette traduction automatique implique certaines contraintes :

–il doit écrire les instructions selon une syntaxe rigoureuse,

–il doit déclarer les données et fonctions qu'il va utiliser (ainsi le compilateur pourra réserver aux données une zone adéquate en mémoire et pourra vérifier que les fonctions sont correctement employées).

1.2. Quelques exemples de langage

Assembleur, C, C++, C#, Fortran, Java, Pascal, JavaScript, Swift, LOGO, Perl, PHP, Python, Ruby…

2. Classification des langages de programmation

2.1. Génération

- **Langage de la première génération**

Les ordinateurs ont reçu des instructions sous la forme de 1 et de 0. Ce langage est appelé langage machine ou langage de première génération. Un ordinateur a pu le comprendre directement sans aucune conversion. Ce langage est également connu sous le nom de langage machine ou langage binaire. Langage binaire car seulement deux symboles 1 et 0.

- **Langage de la deuxième génération**

Dans ce langage, les instructions ont été remplacées par des termes codés appelés mnémoniques. Pour qu'il devienne un peu plus facile à lire, à comprendre et à corriger mais le programmeur doit connaitre l'architecture du microprocesseur. Un ordinateur ne peut comprendre et travailler que sur le code machine. Le langage d'assemblage avait donc besoin d'un logiciel spécial appelé assembleur qui convertit les mnémoniques en langage machine (exemple d'une instruction de l'assembleur: ADD A, B ; MOV SI,4165h).

- **Langage de la troisième génération**

Les deux générations précédentes de langages étaient comparativement plus faciles à comprendre pour un ordinateur, mais il était difficile pour les humains de les lire, de les comprendre et de les coder. En 1954, John Backus jette les bases d'un nouveau langage: FORTRAN, qui est encore utilisé aujourd'hui. **FORTRAN** était avant tout un langage conçu pour effectuer des calculs scientifiques. Ainsi, est venu l'anglais comme langage de programmation pour donner des instructions aux ordinateurs. Ces langages sont connus sous le nom de langages de haut niveau car ils sont plus faciles à comprendre pour les humains (exemple : C, C++, Java, COBOL Pascal, …etc).

- **Les langages de la quatrième génération**

Les langages de troisième génération nécessitent des procédures détaillées, mais les langages de quatrième génération exigent simplement « que » voulons-nous du code plutôt que « comment faire ». c'est-à-dire la procédure. Ces langages sont similaires aux instructions en langage humain principalement utilisées dans la programmation de bases de données (exemple, Python, Ruby, SQL, …etc).

- **Les langages de la cinquième génération**

Les langages de la cinquième génération sont destinés à résoudre des problèmes à l'aide de contraintes, et non d'algorithmes écrits. Ces langages sont davantage axés sur la mise en œuvre

de l'intelligence artificielle (exemple : Prolog, OPSS Mercury, etc. sont des SGLS. Par exemple Prolog, OPS5, Mecury, … etc).

2.2. Niveaux

- **Langage bas niveau**

Les langages bas niveau sont utilisés pour écrire des programmes relatifs à l'architecture et au matériel spécifiques d'un type d'ordinateur particulier (Exemple: Langage machine, langage assembleur, …etc).

- **Langage haut niveau**

Proche du langage humain(anglais), existent dans le but de rendre simple l'écriture d'un programme informatique. Ils utilisent par exemple des symboles mathématiques et des expressions connus (Exemple: Java, PHP, Python, FORTRAN, …etc).

3. Définition de paradigmes de langages de programmation

Un paradigme de programmation est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation (à comparer à la méthodologie, qui est une manière de résoudre des problèmes spécifiques de génie logiciel).

3.1. Quelques paradigmes courants

Les premiers programmes informatiques étaient généralement constitués d'une suite d'instructions s'exécutant de façon linéaire (l'exécution commence de la première instruction du fichier source et se poursuivait ligne après ligne jusqu'à la dernière instruction du programme).

3.2. Programmation impérative

C'est le paradigme originel et le plus ancien. L'implémentation de la quasi-totalité des processeurs qui équipent les ordinateurs est de nature impérative. Ils sont faits pour exécuter du code écrit sous forme d'opcodes (opération codes), qui sont des instructions élémentaires exécutables par le processeur. L'ensemble des opcodes forme le langage machine spécifique au processeur et à son architecture. L'état du programme à un instant donné est défini par le contenu de la mémoire centrale à cet instant, et le programme lui-même est écrit en style impératif en langage machine, ou le plus souvent dans une traduction lisible par les humains du langage machine, dénommée assembleur.

Les langages de plus haut niveau utilisent des variables et des opérations plus complexes, mais suivent le même paradigme. Les recettes de cuisine et les vérifications de processus industriel sont deux exemples de concepts familiers qui s'apparentent à de la programmation impérative ; de ce point de vue, chaque étape est une instruction, et le monde physique constitue l'état modifiable. Puisque les idées basiques de la programmation impérative sont à la fois conceptuellement familières et directement intégrées dans l'architecture des microprocesseurs, la grande majorité des langages de programmation sont impératifs et comportent cinq types d'instructions principales impératives :

- Une séquence d'instruction ;
- Une assignation ou une affectation ;
- Une instruction conditionnelle ;
- Une boucle ;
- Des branchements sans condition.

La programmation impérative présente quelques inconvénients majeurs :

- La programmation impérative est très difficile à organiser ;
- Il y a beaucoup de répétitifs dans le code et le système de lecture de haut en bas est souvent très handicapant sur les gros programmes.

3.3. Programmation procédurale

La programmation procédurale est un paradigme de programmation basé sur le concept d'appel procédural. Une procédure, aussi appelée routine, sous-routine ou fonction (à ne pas confondre avec les fonctions de la programmation fonctionnelle reposant sur des fonctions mathématiques) contient simplement une série d'étapes à réaliser. N'importe quelle procédure peut être appelée à n'importe quelle étape de l'exécution du programme, incluant d'autres procédures voire la procédure elle-même (récursivité). La programmation procédurale est

souvent un meilleur choix qu'une simple programmation séquentielle ou programmation non-structurée.

3.3.1. Avantages

- La possibilité de réutiliser le même code à différents emplacements dans le programme sans avoir à le retaper ;
- Une façon plus simple de suivre l'évolution du programme. La programmation procédurale permet de se passer d'instructions telles que "GOTO" et "JUMP" évitant ainsi bien souvent de se retrouver avec un programme compliqué qui part dans toutes les directions (appelé souvent « programmation spaghetti ») ;
- La création d'un code plus modulaire et structuré.
- Les langages de programmation procédurale facilitent la tâche du programmeur en permettant de privilégier une approche procédurale.

La modularité est généralement une caractéristique souhaitable pour un programme informatique. C'est particulièrement le cas pour les programmes complexes et/ou importants en taille, que l'on peut ainsi découper en modules indépendants les uns des autres. Dans ce cadre, on parle d'arguments pour les entrées et de « valeurs de retour » pour les sorties.

La portée des variables est l'un des mécanismes qui permet la modularité des procédures. Elle empêche une procédure d'accéder aux variables locales d'une autre procédure, évitant ainsi les collisions de noms.

Pour une procédure donnée, il est possible de définir une interface simple (et idéalement permanente). Les procédures constituent donc un moyen efficace de développement de code par plusieurs personnes, ou plusieurs groupes de personnes, notamment via des bibliothèques.

3.4. Programmation orientée objet

La programmation orientée objet (POO) ou programmation par objet, est un paradigme de programmation informatique permet de concevoir une application sous la forme d'un ensemble de briques logicielles appelées des objets. Un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Chaque objet joue un rôle précis et peut communiquer avec les autres objets. Les interactions entre les différents objets vont permettre à l'application de réaliser les fonctionnalités attendues

Le langage Simula-67 jette les prémisses de la programmation objet, résultat des travaux sur la mise au point de langages de simulation informatique dans les années 1960 dont s'inspira aussi la recherche sur l'intelligence artificielle dans les années 1970-80. Mais c'est réellement par et avec Smalltalk 72 puis Smalltalk 80, inspiré en partie par Simula, que la programmation par objets débute et que sont posés les concepts de base de celle-ci : objet, messages, encapsulation, polymorphisme, héritage (sous-typage ou sous-classification), redéfinition, etc. Smalltalk est plus qu'un langage à objets, il est aussi un environnement graphique interactif complet.

À partir des années 1980, commence l'effervescence des langages à objets : Objective C (début des années 1980), C++ (C with classes) en 1983, Eiffel en 1984, Common Lisp Object System dans les années 1980, etc. Les années 1990 voient l'âge d'or de l'extension de la programmation par objet dans les différents secteurs du développement logiciel.

Les trois concepts fondamentaux qui donnent toute sa puissance à la P.O.O sont :

- Concept de modélisation à travers la notion de classe et l'instanciation de ces classes.
- Concept d'action à travers la notion d'envoi de messages et de méthodes à l'intérieur des objets.
- Concept de construction par réutilisation et amélioration par l'utilisation de notions d'héritage.

3.4.1. Avantages de la POO

- ✓ La POO facilite la conception de programmes par réutilisation de composants existants, avec tous les avantages évoqués plus haut.
- ✓ Très structuré (permet de structure son code de manière exemplaire).
- ✓ Tout est bien organisé, facile à lire
- ✓ Diminuer le coût du logiciel ;
- ✓ Augmenter sa durée de vie, sa réutilisabilité et sa facilité de maintenance ;

- ✓ Concevoir des logiciels avec des exigences de qualité
- ✓ Grande facilité dans le développement en particulier collectif.
- ✓ Très performante
- ✓ Très utilisé, en particulier pour les gros projets

3.4.2. Quelques langages à objets

Ada, Java, C#, Objective C, Eiffel, Python, C++, PHP, Smalltalk...

Il existe d'autres paradigmes tels que, fonctionnel, générique, logique, déclaratif, par contraintes, réflexif, descriptif, structuré, synchrone…

2. Principe de la P.O.O

Les langages orientés objet créent un lien puissant entre les structures de données et les fonctions qui les manipulent, se rapprochant ainsi de notre manière de penser. On ne pense désormais plus en termes de structures de données et de fonctions permettant de les manipuler, mais en termes d'objets, comme s'il s'agissait de leurs équivalents dans le monde réel, qui apparaissent et agissent d'une certaine façon.

Pour introduire la notion d'objet, prenons l'exemple d'une citerne (un objet du monde réel) (figure 1.1). Une citerne possède des caractéristiques communes à toutes les objets de type citerne, à savoir :

- ➢ **Capacité** le volume maximal du liquide dans la citerne (elle est constante pour une citerne particulière, elle dépend de H sur la figure 1.1)
- ➢ **Niveau** c'est le niveau actuel du liquide dans la citerne (il dépend de h sur la figure 1.1)

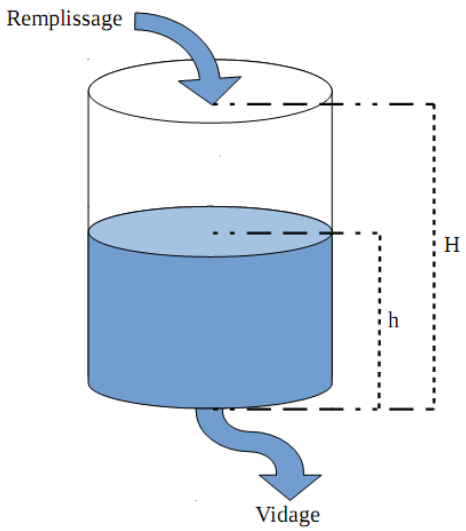

Figure. 1.1 : Schéma de l'objet réel Citerne

Par ailleurs, la citerne peut être alimentée par une source extérieure (on parle d'action de **remplissage**). On peut aussi utiliser le liquide (on parle alors d'action de **vidage**).

Dans le monde informatique l'objet citerne est défini dans une **classe** regroupant les caractéristiques (attributs) et les actions (fonctions) (figure 1.2).

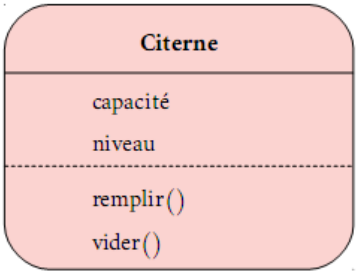

Figure. 1.2 : Schéma de l'objet Informatique Citerne

2.1.Qu'est-ce qu'un objet ?

La programmation orientée objets consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets.

La programmation oriente objet fondée justement sur le concept objet, à savoir une association des données et des procédures (qu'on appelle méthode) agissant sur ces données. Par analogie avec l'équation de Writh, on pourrait dire que l'équation de la P.O.O est :

Objet = Données + Méthodes

2.2.Concepts fondamentaux des objets

Les trois concepts qui contribuent à la puissance de la P.O.O sont : Encapsulation et abstraction Héritage, Polymorphisme

❖ Classe

Le concept de classe apparaît généralement dans P.O.O, qui correspond simplement à la généralisation de la notion de type que l'on rencontre dans les langages classiques. Une classe n'est rien d'autre que la description d'un ensemble d'objet ayant une structure de données communes et dispose des mêmes méthodes.

❖ Héritage

L'héritage permet de définir une nouvelle classe à partir d'une classe existante, qu'on réutilise, à laquelle on ajoute des nouvelles données et de nouvelles méthodes.

❖ Polymorphisme

En P.O.O une classe dérivée peut redéfinir (modifier) certains des méthodes héritées de la classe de base. Cette possibilité est nommée le polymorphisme. Le polymorphisme améliore l'extensibilité des programmes, en permettant d'ajouter de nouveaux objets.

❖ Encapsulation

L'encapsulation des données signifie qu'il n'est pas possible d'agir directement sur les données d'un objet, il est nécessaire de passer par l'intermédiaire de ses méthodes, qui jouent ainsi le rôle d'interface obligatoire. L'encapsulation des données présente un intérêt en matière de qualité de logiciel. Elle facilité considérablement la maintenance. Une modification de la structure de données d'un objet n'affecte que l'objet lui-même, les utilisateurs de l'objet ne seront pas affectés par le contenu de cette modification.

Exemple:

Employeur 1 et Employeur 2 sont caractérisés par les mêmes propriétés (numéro d'identification, nom, prénom, qualification) mais associés à des valeurs différentes. Ils ont le même comportement (entrer, sorter, ChangerPoste) mais ont des identités différentes. Et il en serait de même pour tous les employés. Tous les employés obéissent à un même schéma.

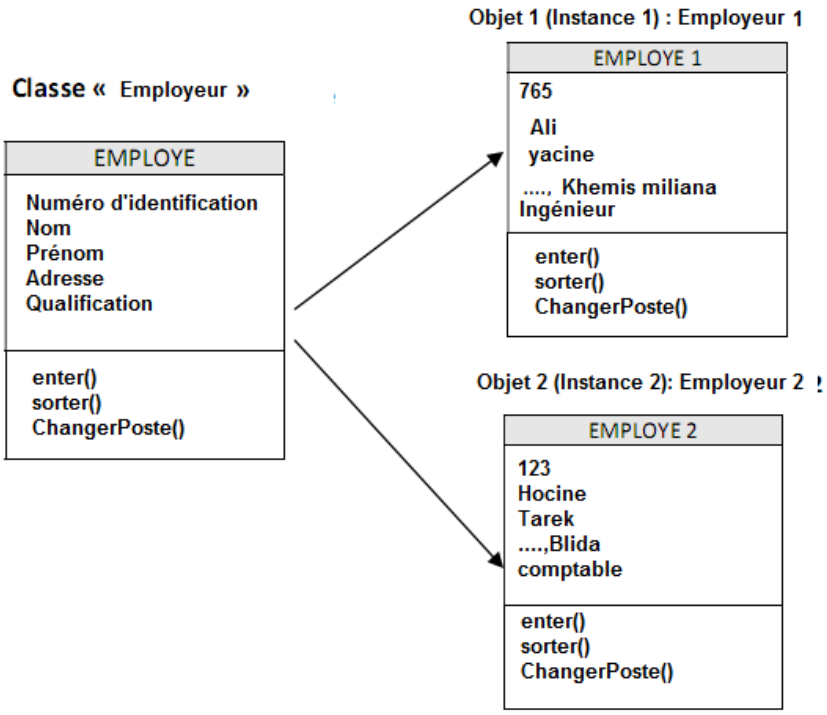

Figure. 1.3 :

4. Définition du langage C++

L'histoire du langage C++débute dans le même laboratoire (AT&T Bell Laboratoires) de recherche que le langage C qui a été créé en 1972 par Denis Ritchie, formalisé par Kerninghan et Ritchie en 1978. En 1979, Bjarne Stroustrup développe le langage C with classes qui est une extension basique du langage C incluant, entre autres, la définition de classes et l'héritage simple. Ce langage continu de s'enrichir et c'est en 1983 qu'il change de nom en C++. Les deux langages C et C++ ont continué d'évoluer parallèlement jusqu'à ses normalisations par un comité regroupant l'ANSI (American National Standards Institute) et l'ISO (International Standards Organization) en 1990 pour le C et 1998 pour le C++.

Le langage C++ est une évolution orientée objet du langage C. Un compilateur C++ est capable de compiler un code source écrit en C pur. Le langage C++ est un sur ensemble de C repose sur les mêmes mécanismes d'écriture et de génération :

- Déclaration préalable est obligatoire
- Exploitation de toutes les fonctions du préprocesseur C pour les inclusions de fichiers, code conditionnel, macros définitions.
- La syntaxe du C++ est de 90% de C + 10% d'ajouts.

Les langages de programmation les plus utilisés en 2023

Jun 2023	Jun 2022	Change	Programming Language	Ratings	Change
1	1		Python	12.46%	+0.26%
2	2		C	12.37%	+0.46%
3	4	↑	C++	11.36%	+1.73%
4	3	↓	Java	11.28%	+0.81%
5	5		C#	6.71%	+0.59%
6	6		Visual Basic	3.34%	-2.08%
7	7		JavaScript	2.82%	+0.73%
8	13	⇈	PHP	1.74%	+0.49%
9	8	↓	SQL	1.47%	-0.47%
10	9	↓	Assembly language	1.29%	-0.56%

Figure. 1.3 : Source : Index TIOBE

4.1. Les forces du C++

- **Il est très répandu** : il fait partie des langages de programmation les plus utilisés sur la planète. On trouve donc beaucoup de documentation sur Internet et on peut facilement avoir de l'aide sur les forums.
- **Il est rapide:** très rapide même, ce qui en fait un langage de choix pour les applications critiques qui ont besoin de performances ou qui doivent fonctionner en temps réel.
- **Il est portable:** un même code source peut théoriquement être transformé sans problème en exécutable sous Windows, Mac OS et Linux.
- **Il est riche:** il existe de nombreuses bibliothèques pour le C++.
- **Il est multi-paradigmes:** on peut programmer de différentes façons en C++. L'une est la Programmation Orientée Objet (POO) et l'autre est la procédurale.

4.2. Les outils nécessaires au programmeur

1. **Un éditeur de texte :** pour écrire le code source du programme en C++. L'idéal, c'est d'avoir un éditeur de texte intelligent qui colore tout seul le code, ce qui vous permet de vous y repérer bien plus facilement.
2. **Un compilateur :** pour transformer (« compiler ») votre code source en binaire.
3. **Un debugger :** (« Débogueur » ou « Débugueur » en français) pour aider à détecter les erreurs dans le programme.

On a 2 possibilités :

- Soit on récupère chacun de ces 3 programmes séparément. C'est la méthode la plus compliquée.

- Soit on utilise un programme « 3-en-1 » qui combine éditeur de texte, compilateur et débugger. Ces programmes « 3-en-1 » sont appelés IDE (ou en français « EDI » pour « Environnement de Développement Intégré »).

4.3. Environnements de développement

Il existe plusieurs environnements de développement

- **Code::Blocks:** Il est gratuit et disponible pour la plupart des systèmes d'exploitation.
- **Visual C++ Express de Microsoft :** la version gratuite de Visual C++.
- **Dev-C++:** sous Windows

4.4. Cycle de développement d'un programme C++

La première étape de la création d'un programme consiste à écrire instructions dans un fichier source. Le fichier source est un texte lisible et ce n'est pas un programme (il est impossible de le lancer ou de l'exécuter tel quel). On doit utiliser un compilateur pour transformer le code source en programme. La compilation produit un fichier objet porte l'une des extensions .obj ou .o. Toutefois, il ne s'agit pas encore d'un programme exécutable. Pour créer le fichier exécutable, il faut utiliser l'éditeur de liens (linker) (Figure.

Les programmes C++ sont produits en liant un ou plusieurs fichiers objet (.obj ou .o) avec une ou plusieurs bibliothèques. Une bibliothèque regroupe des fichiers fournis avec le compilateur, achetés séparément ou crées personnellement. Tous les compilateurs C++ sont livrés avec une bibliothèque de fonctions standard.

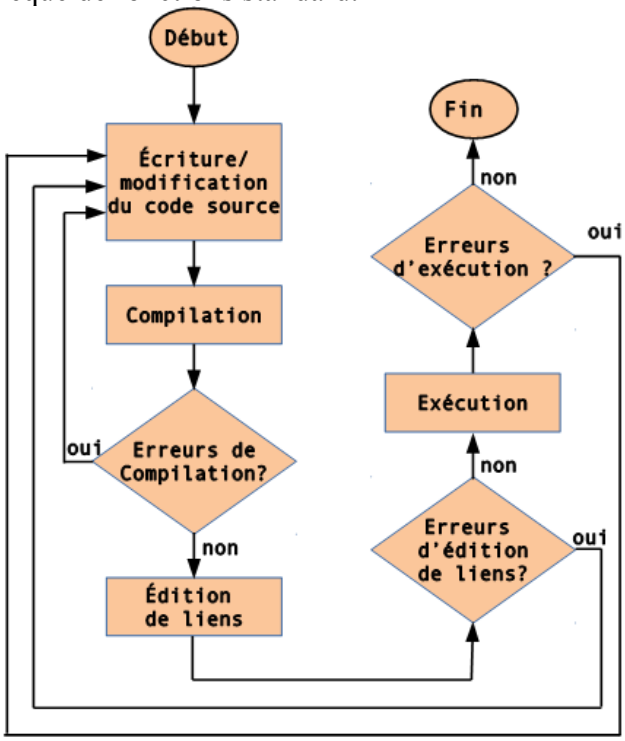

Figure. 1.4 : Les étapes du cycle de développement d'un programme C++

Le code source de notre programme, n'est pas compréhensible par le processeur, il faut le traduire en langage machine c'est l'étape de compilation, en pratique cette étape passe par quatre opérations:

- **Le préprocesseur:**

Le préprocesseur va lire lignes commençant par #, puis effectuer l'action associée. Il va inclure le contenu du fichier *iostream* en haut de notre programme. Si l'on exécute uniquement le préprocesseur sur notre code d'exemple, on obtient un fichier de plus de 18 000 lignes.

- **La compilation:**

La traduction se fait vers un autre langage, l'assembleur. Le compilateur va afficher des messages d'erreur s'il trouve des choses non conformes dans votre code source.

- **Assemblage:**

Cette deuxième et dernière traduction est appelée assemblage, et dans cette étape, le code d'assemblage obtenu par la compilation est converti en un fichier objet contenant des instructions en binaire que le processeur comprend.

❖ **Editeur de liens:**

Si le programme est constitué de plusieurs fichiers, le compilateur devra les lier pour générer l'exécutable final. C'est ce qu'on appelle l'édition des liens.

4.5. Structure d'un programme en C++

Un programme est un ensemble de déclarations et d'instructions. Un programme écrit en C++ se compose généralement de plusieurs fichiers sources. Il y a deux sortes de fichiers-sources :
-ceux qui contiennent effectivement des instructions ; leur nom possède l'extension .cpp
-ceux qui contiennent que des déclarations ; leur nom possède l'extension .h (signifiant 'header' ou en-tête).
Tous les programmes possèdent une fonction dénommée «**main**», ce qui signifie « principale ». Le programme doit ˆêtre placé entre accolades **{}** à la suite du nom de la fonction principale **main(). main()** est le point d'entrée du programme. Elle est donnée ici sans arguments, mais on peut lui transmettre des arguments en écrivant par exemple **main (int A)** ou **A** est un paramètre entier.
En C++, toute déclaration ou fonction doit se terminer par un point-virgule ;.

Le programme suivant est un programme vide qui ne fait rien, donc c'est la structure minimale d'un programme en C++.

```
1 int main()
2 {
3     return 0;
4 }
```

À partir de ce programme on peut ajouter des instructions, des fonctions, déclarer des variables, inclure des bibliothèques pour réaliser des taches. On peut développer ce programme pour afficher un message « Hello world! ».
Le programme devient comme suit:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
```

La première ligne **#include <iostream.h>** est une directive. Elle est prise en charge par le préprocesseur avant la compilation. Elle permet d'utiliser des bibliothèques C++.

Les fichiers d'entête : Ils sont traités par le préprocesseur et sont introduits grâce `a la directive **#include**. Le listing 1.2 montre l'inclusion du fichier d'entête **iostream**.

La directive #include

On place en général au début du programme un certain nombre d'instructions commençant par **#include**. Cette instruction permet d'inclure dans un programme la définition de certains objets, types ou fonctions. Le nom du fichier peut être soit à l'intérieur des chevrons **< et >,** soit entre guillemets : **#include <nom_fichier>** Inclut le fichier **nom_fichier** en le cherchant d'abord dans les chemins configurés, puis dans le même répertoire que le fichier source, **#include "nom_fichier"** Inclut le fichier **nom_fichier** en le cherchant d'abord dans le même répertoire que le fichier source, puis dans les chemins configurés.