Institut Supérieur d'Informatique de
Modélisation et de leurs Applications

Campus des Cézeaux
24 avenue des Landais
BP 10125
63173 AUBIÈRE Cedex

# Compte rendu de TP
# Filière Génie Logiciel et Systèmes Informatiques

Passage par référence

TP réalisé par Nicolas PRUGNE et Antoine COLMARD

TP demandé dans le cadre du cours de compilation de M. DELEPLANQUE

# Table des matières


## Table des matières

1 Introduction ..... 4

1.1 Le parsage ..... 4

1.2 Gestion des chaines de caractères ..... 4

1.3 Objectif du TP ..... 4

2 Génie logiciel ..... 4

2.1 Présentation du package UML développé ..... 4

3 Algorithmique ..... 5

4 Résultats ..... 6

5 Annexes ..... 7

5.1 Implémentation C++ de la procédure de parsage ..... 7

# Table des figures et illustrations

Figure 1 - Package parser ..... 5
Figure 2 - Sortie console du programme ..... 7

# 1 Introduction

## 1.1 Le parsage

Lorsqu'un programme parse un fichier texte, son objectif est d'extraire de ce dernier différentes informations afin de les utiliser par la suite. Ces informations sont la plupart du temps renseignées au format texte par un humain qui souhaite stocker des données. Dans le cadre de ce TP, ces données sont les lignes d'un fichier d'entête du code source d'un programme écrit en C ou en C++ (fichier .h).

## 1.2 Gestion des chaines de caractères

Le parsage d'un fichier s'effectue donc en traitant des chaines de caractères dans un programme. Il est alors intéressant, avant d'implémenter un logiciel dans un langage donné de se demander comment sont gérées les chaines de caractères dans ce dernier. Au cours de ce TP quatre langages de programmation furent proposés. Le C, le C++, le Java et le Python.

En C, étant donné que le langage est de très bas niveau, les fonctions permettant de traiter des chaines de caractères le sont également. Ainsi dans ce langage, une chaine de caractères n'est rien de plus qu'un tableau d'octets d'une taille donnée. La librairie standard du C permet ensuite d'effectuer divers traitements sur ces tableaux, comme des copies ou des comparaisons par exemple.

Le C++, qui s'appuie sur le paradigme de la programmation objet, propose une implémentation des chaines de caractères plus évoluée. En effet, pour gérer ces structures de données, il est possible de s'appuyer sur la classe `std :: string` de la STL (Standard Template Library). Celle-ci offre plus de souplesse que les tableaux d'octets du C pour gérer des chaines de caractères. En effet, le stockage et la gestion de la mémoire, effectués pour stocker les caractères d'une chaine, sont complètement transparents au travers de l'interface de la classe.

Le Java propose lui aussi une classe `String` permettant de gérer des chaines de caractères. Cependant à la différence du C++, dans ce langage celles-ci sont considérées comme immuables. C'est-à-dire qu'une fois instanciée, une chaine de caractères ne peut plus être modifiée. Enfin en Python la gestion des chaines n'est pas bien différente de celle du Java. Les chaines sont aussi immuables, seule la syntaxe pour les manipuler change.

## 1.3 Objectif du TP

L'objectif ce TP est donc de concevoir un programme capable de parser les lignes d'un fichier .h écrit en C ou en C++. Il faut donc que le programme soit capable d'extraire chaque prototype des fonctions stockées dans le fichier .h. Il doit également être capable de distinguer le type de retour du nom complet de chaque méthode.

# 2 Génie logiciel

Une phase de conception a bien entendu était nécessaire avant d'implémenter le programme de ce TP. Elle a permis de bâtir un ensemble de classes exposées ci-dessous.

## 2.1 Présentation du package UML développé

Le parsage d'un fichier requiert plusieurs classes. Le programme développé au cours de ce TP en utilise trois, celles-ci sont regroupées dans un package UML nommé parser.

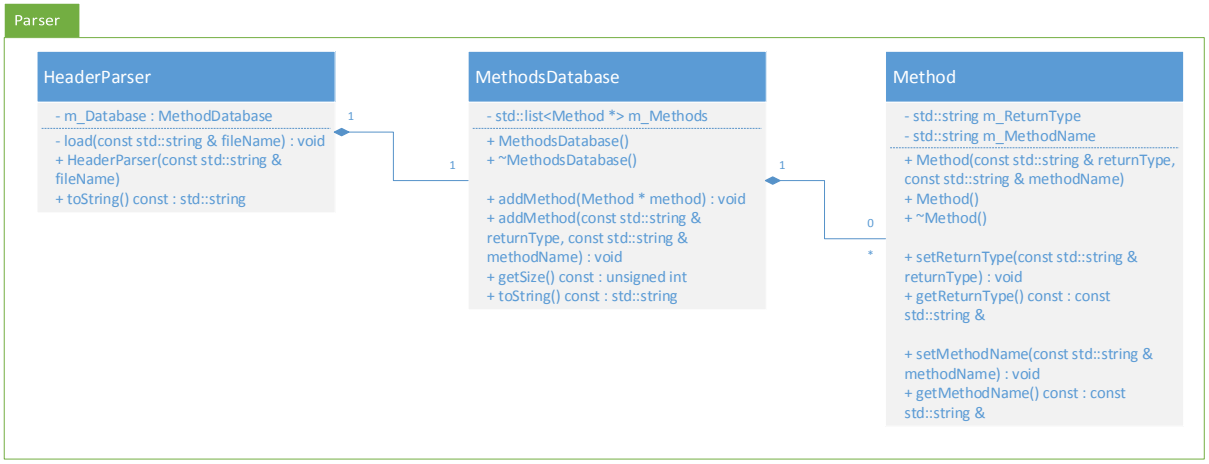


**Figure 1 - Package parser**

La première classe `HeaderParser` regroupe un ensemble de méthodes permettant de lire et d'analyser un fichier texte afin d'extraire de ce dernier les prototypes des méthodes qu'il contient. On remarque que la classe `HeaderParser` contient un objet de type `MethodsDatabase`. Cette classe quant à elle permet de stocker un ensemble d'objets de type `Method`. Elle permet de définir une interface de manipulation des méthodes stockées pour la classe `HeaderParser`, sans que cette dernière n'ait à gérer d'une quelconque manière la façon dont s'effectue le stockage. Pour finir, la classe `Method` représente un modèle des prototypes qui sont extraits des fichiers textes. Elle se décompose sous la forme de deux chaines de caractères. L'une contient le type de retour de la méthode, l'autre contient son nom.

# 3 Algorithmique

L'algorithme utilisé pour extraire les prototypes des fichiers .h se base sur une regex pour fonctionner. Cette expression régulière représente le format des prototypes de méthodes en C et C++. Elle est capable de gérer l'ensemble des syntaxes possibles pour définir une méthode dans ces deux langages.

Algorithme 1 : parseFile

Entrée:

fileName : nom d'un fichier .h à parser

Pseudocode :

```
Procédure parseFile
        file := ouverture du fichier fileName;
        pattern :=
        (\s*)(const|virtual|static|inline)?(\s*)(\w|\<\w+\>|::)+((\s(&|\*))|((&|\*)\
        s)|\s)(\w)+\((\w|\s|,|&|\*|\<\w+\>|::)*\)(\s*)(const)? ;

        Tant que l'on n'a pas atteint la fin du fichier faire :
                line := file.nextLine() ;
                [Si la ligne correspond au modèle de la regex]
                Si regex_search(line, pattern) = Vrai alors :
```

```
                    Différencier le nom de la méthode de son type de retour ;
                    Ajouter la méthode à la base ;
              fsi ;
       fait ;
       Fermeture du fichier file ;
fin parseFile;
```

# 4 Résultats

L'implémentation réelle du TP a donc été réalisée en C++ et la compilation effectuée dans un environnement Windows. Ainsi le compilateur utilisé pour ce jeu d'essais est celui de la suite logicielle Visual Studio. Plus précisément, il s'agit du compilateur de Visual C++ 2012. Les tests ont été menés sur une version release du programme développé et sur une machine dotée d'un processeur Intel Core i7-QMCPU @2,30Ghz.

Le test a été mené sur un fichier .h issu d'un projet précédant. Dans un premier temps le programme parse le fichier et en extrait l'ensemble des méthodes. Cette phase est chronométrée. Puis le programme affiche l'ensemble des méthodes qu'il a trouvé en distinguant les types de retour du nom de celles-ci.

```
Parsing file : ../data/transmitter.h
End of parsing (0.039s).

Database content : 7 methods registered.

Methods list :

Return type : int
Method's name : trs_openPort(char*name);

Return type : transmitter_t*
Method's name : trs_create(constchar*port_name,chardelimiter);

Return type : void
Method's name : trs_destroy(transmitter_t*t);

Return type : void*
Method's name : trs_process(void*ptr);

Return type : void
Method's name : trs_stop(transmitter_t*t);

Return type : void
Method's name : trs_fetch(transmitter_t*t,char*oInstr);

Return type : void
Method's name : trs_post(transmitter_t*t,constchar*iInstr);

Return type : void
```

```
Method's name : trs_post(transmitter_t*t,constchar*iInstr);
```

Figure 2 - Sortie console du programme

Le programme met donc 0,04 secondes à extraire l'ensemble des méthodes du fichier .h et détecte au total les 7 prototypes que compte le fichier sur les 7 normalement présents.

# 5 Annexes

L'intégralité du projet est disponible à l'adresse suivante : https://github.com/LebonNic/parser_cpp.

## 5.1 Implémentation C++ de la procédure de parsage

```
/**
 * @brief HeaderParser::load Charge la database de méthodes depuis un fichier
 * @param fileName Fichier à charger
 */
void HeaderParser::load(const std::string &fileName)
{
	std::ifstream filestream(fileName.c_str());
	std::string   line,
	              word;

	std::string pattern =
"(\\s*)(const|virtual|static|inline)?(\\s*)(\\w|\\<\\w+\\>|::)+((\\s(&|\\*))|((&|\\*)\
\s)|\\s)(\\w)+\\((\\w|\\s|,|&|\\*|\\<\\w+\\>|::)*\\)(\\s*)(const)?",
	                        returnType,
	                        prototype,
	                        temp;
	std::regex exp (pattern);
	std::smatch match;

    if (filestream.is_open())
    {
		while(std::getline(filestream, line))
		{
			if(std::regex_search (line,match,exp))
			{
				std::stringstream ss(line);
				ss >> returnType;
				ss >> temp;
				if(temp == "*" || temp == "&")
				{
					returnType += temp;
					ss >> prototype;
				}
				else
					prototype = temp;

				while(ss >> temp)
					prototype += temp;

				m_Database.addMethod(returnType, prototype);
			}
		}
    }
    filestream.close();
}
```