C++ Fact Extractor

Undertaken by:

Muhammad Ali

Supervised by:

Mr. Abdul Qadus Abbasi

INSTITUTE OF INFORMATION TECHNOLOGY

QUAID-I-AZAM UNIVERSITY

ISLAMABAD

Session 2015-2017

DEDICATION

I would like to dedicate this thesis to my Parents, my respected teacher Mr. Abdul Qadus Abbasi and my Friends due to their backing and courage, I'm able to complete this perplexing task.

ACKNOWLEDGEMENT

In the name of Allah, The most beneficent and the most merciful. All reward are to almighty Allah the creator of the universe, for blessing me with knowledge and braveness to complete this venture.

I would like to acknowledge my thanks to my supervisor Mr. Abdul Qudus Abbasi for his knowledge, believe in me and patience. Only his guidance made me able to do this project.

In the end, I would like to offer my special thanks seniors for their trust and best wishes for me.

Thank You So Much

Muhammad Ali

Project in brief

Project Title:	C++ Fact Extractor
System Used:	Intel Core i3
Operating System:	Microsoft Windows 8.1
Development Tool:	Microsoft Visual Studio 2010 Professional
Language:	Visual C++
Start date:	10th September 2016
End date:	13th February 2017

ABSTRACT

Maintenance and reusability of code is most important and necessary in software engineering. But this requires much time and resources which increases the cost of the software project. For both the tasks understanding of whole project is necessary.

During maintenance and change, developers cannot read the entire code of large systems. They need a way to get a quick understanding of source code entities (such as classes, functions etc.).

Keeping this in mind, I have developed a tool which makes understanding of code much easier. This tool gives information about constrains and relationships of C++ source code. It takes a C++ project source code as a whole as input and extracts facts from source code using scanning and parsing process. After being processed, it collects useful information from the code, which makes it very easy to understand code. This tool generates reports in two formats one is TEXT and the other is XML file.

Contents

Chapter 1 .... **1**
1.1. Introduction: 2
1.2. Purpose of project: 2
1.3. Scope: 3
1.4. System specification: 3
1.4.1. System input: 3
1.4.2. System output: 3
1.5. Resource identification: 3
1.5.1. Software requirement: 3
1.5.2. Hardware requirement: 3
1.6. Applications: 4
1.7. Process model for development: 4
Summary: 4
Chapter 2 .... **5**
2.1. Introduction: 6
2.2. Functional Requirements: 6
2.3. Information to be extracted: 7
2.4. Use case: 8
2.5. Assumptions: 11
2.6. Nonfunctional requirements: 11
Summary: 12
Chapter 3 .... **13**
3.1. Class composition: 14
3.2. Inheritance: 15
3.2.1. Access Control and Inheritance: 15
3.3. Association: 16
3.4. Struct: 16
3.5. Enum: 16
3.6. Macros: 16
3.7. Inline function: 17
3.8. Abstraction (Virtual and pure Virtual) 18

3.9. Polymorphism: 18
3.9.1. Static Polymorphism 19
3.9.2. Dynamic Polymorphism: 19
Summary: 19
Chapter 4 20
4.1. Introduction: 21
4.2. Lexical Conventions: 22
4.2.1. Character set: 22
4.2.2. Basic source character: 22
4.2.3. Comments (C++): 22
4.2.4. Identifiers (C++): 23
4.2.5. Keywords (C++): 24
4.3. Lexical Analyzer: 24
4.3.1. Tokens (C++): 24
4.3.2. Longest Match Rule: 25
4.4. Syntax Analysis 26
4.5. Intermediate Representation of Information: 26
Summary: 27
Chapter 5 28
5.1. Software architecture: 29
5.1.1. Fact Extractor: 30
5.1.2. File browser: 30
5.1.3. Scanner: 30
5.1.4. Parser: 30
5.1.5. Intermediate Representation: 30
5.1.6. Relationship Extractor: 31
5.2. Software Design: 31
5.2.1. Design constraints: 31
5.2.1.1. Reusability: 31
5.2.1.2. Maintainability: 31
5.2.1.3. Modularity: 31
5.2.1.4. Reliability: 31
5.3. Class Diagrams: 32
5.3.1. Class diagram for folder reader: 32
5.3.1.1. CFolderReader: 32

5.3.1.2. CFileInfo: 33
5.3.1.3. CFileInfo_Vec: 33
5.3.2. Class diagram for fact extraction: 33
5.3.2.1. CTokenizer: 33
5.3.2.2. CTokenVector: 33
5.3.2.3. CTokenScanner: 34
5.3.2.4. CTokenInfo: 34
5.3.2.5. TokenScannerVector: 36
5.3.3. Class diagram for extracting class Information 36
5.3.3.1. Meta_Classinfo 36
5.3.3.2. C_classInfo: 36
5.3.3.3. CDateMemberInfo: 37
5.3.3.4. CDateMemberInfo_VEC: 37
5.3.3.5. CFunctionInfo: 37
5.3.3.6. CFunction_Info_Vec: 37
5.3.3.7. CParameterList: 37
5.3.3.8. CLocalDataMembers: 37
5.3.4. CRelationFinder 37
5.3.4.1. CstructFinder: 38
5.3.4.2. CinheritanceFinder: 38
5.3.4.3. CcompostionFinder: 38
5.4. User Interface Design: 39
Chapter 6 44
6.1. Introduction: 45
6.2. Input Files: 45
6.2.1. Header file: 45
6.2.2. .cpp file: 47
6.3. Output: 48
6.3.1. XML format: 48
6.3.2. Text format: 500
Summary: 52
Chapter 7 53
7.1. Introduction: 54
7.2. Testing: 54
7.2.1. Black box testing: 54

7.3. Test Cases: 54
Summary: 56
Chapter 8 57
8.1. Conclusion: 58
8.2. Future work: 58
Reference: 59

Table of Figures

Figure 1.7-1 Incremental Process Model 4
Figure 4.1-1 Fact Extraction Process 21
Figure 5.1-1 Architecture diagram 29
Figure 5.3-1 Class diagram for folder reader 32
Figure 5.3-2 Class diagram for fact extraction 33
Figure 5.3-3 Class diagram for extracting class 36
Figure 5.3-4 Class diagram for Relation finder 38
Figure 5.4-1 User Interface main screen 39
Figure 5.4-2 Open new project 40
Figure 5.4-3 New project Dialog box 40
Figure 5.4-4 Fill dialog box for Desired info 41
Figure 5.4-5 Extracted class info 41
Figure 5.4-6 Extracted relation info 42
Figure 5.4-7 Version No. of Tool 42

Chapter 1

Introduction

1.1.Introduction

1.2.Purpose of project

1.3.Scope

1.4.System specification

1.5.Resource identification

1.6.Applications

1.7.Process model for development

Summary

1.1. Introduction:

Software systems are rapidly growing and changing, so source code written today gets out-of-date tomorrow. This is among others due to the quickly changing market requirements and also due to the continuously upcoming new technologies. [13] The always tight deadlines often prevent the developers to release a product properly with up-to-date documentation (design descriptions, source code comments, etc.). As a result there is need to understand the relationships between the different parts of a large system. To comprehend an unfamiliar software system we need to know many different things about it. We refer to this information as facts about the source code. A fact is for instance the size of the code. Another fact is whether a class has base classes. Actually any information that helps us to understand unknown source code is called a fact in this paper. It is obvious that collecting facts by hand is only feasible when relatively small source codes are investigated. Real-world systems that contain several million lines of source code can be only processed with the help of tools. Tool supported fact extraction is in our approach an automatized process during which the focus system is analyzed file-by-file with analyzer tools to identify the source code's various characteristics and their interrelationships and to create representations of the extracted information into different formats like XML and text. [14]

1.2. Purpose of project:

The purpose of this project is to build a tool that saves time and effects to examine and apprehend the code. We're privy to the truth that programmers spend maximum of the time in information and reading the code. The motive and motivation in the back of my tool is to lessen the effort and keep the valuable time which spends on reading the code. This challenge is the utility of opposite engineering. Roger Pressman defines the reverse engineering as

"The process of analyzing a program in an effort to create a representation of the program at higher level of abstraction than source code. Reverse engineering is process of design recovery" [17].

1.3. Scope:

This is very useful tool for programmers and maintainers. However it has some limitations too. The output of the project depend on the input of the user input. This tool only works for C++ source code. It will extract information about classes and relationship between them.

1.4. System specification:

C++ fact Extractor will extract the information of the relationship between the entities of the system from the input source code. These relationships can be based on inheritance, composition and association etc...

1.4.1. System input:

C++ Source code file given to the C++ Fact Extractor as input.

1.4.2. System output:

After the extraction of information from source code will be stored on files, which can be text or XML format.

1.5. Resource identification:

Following are the resources which would be used to full fill the need in the completion of the project.

1.5.1. Software requirement:

Software resources used for developing the project

• Operating system	Windows (7 , 8 or above)
• Language	MFC , STL , C++
• IDE	Visual Studio 2010

1.5.2. Hardware requirement:

Hardware resources used for developing the project

• Personal computer	
• Processor	Corei3
• Hard Disk	100 GB
• Memory(RAM)	4GB

1.6. Applications:

- The principle motive of making this software is to provide help to the maintainer of the system and to recognize the source code.
- Beneficial for the maintenance and change in the software process

1.7. Process model for development:

Tool will be developed using incremental version. Overall improvement consists of three increments. First increments consist of parsing and tokenize of source code. Second increment is to extract the relationships of system and to get the information about the class name, capabilities in that elegance go back type of function parameter and line of code of function and the entire class. Third increment is to create an XML file and displaying result to the consumer. [5]

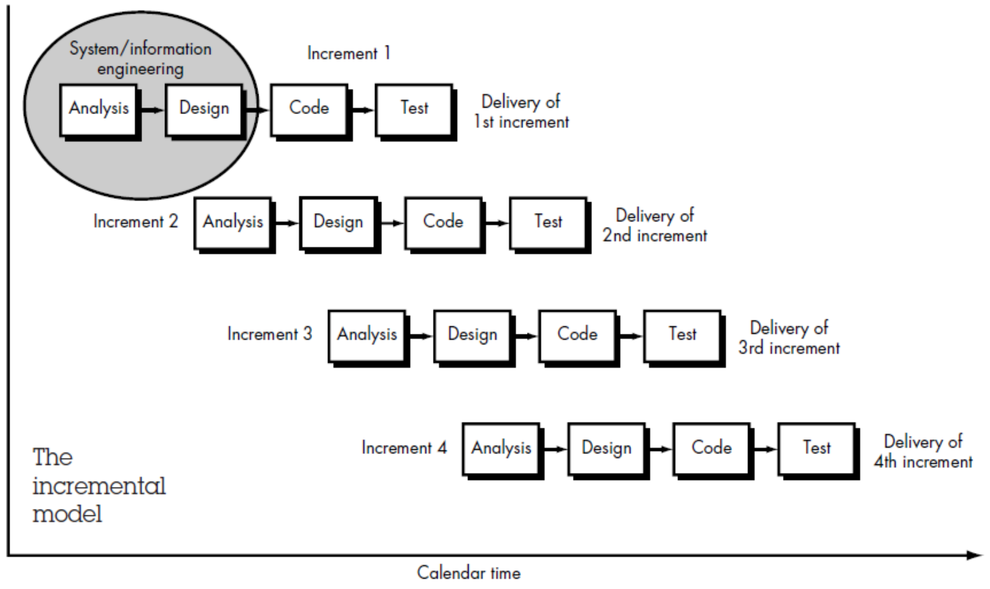


Figure 1.7-1 Incremental Process Model

Summary:

In this chapter, I have discussed how software is evolving day by day and how we can do reverse engineering our existing program by understanding the source code. Reverse engineering is basic factor of motivation for our work.

Chapter 2

Requirement Analysis

2.1.Introduction

2.2.Functional

2.3.Information to be extracted

2.4.Use case

2.5.Assumptions

2.6.Non-Functional requirements

Summary

2.1. Introduction:

Requirement engineering is the technique of figuring out the user exception for a brand new or changed software product. This is also known as requirement engineering. Those characteristic are known as necessities, there are styles of necessities functional requirement and nonfunctional requirement. [7]

2.2. Functional Requirements:

Functional requirements are the main things that the user expects from the software. The functional requirements are listed as below.

1. Create a new project.
2. Open an existing project.
3. Close project.
4. Save project.
5. Create token
6. Parse the project extract the information about:
    - Find Relationship
7. Save Parsed information
8. Generate Intermediate Representation
9. Generate Output in the form of:
    - XML File
    - TEXT File

2.3. Information to be extracted:

Our system should be able to extract the following information:-

1. Single Class Information
    a. Name of class
    b. Data member
    c. Member Methods
    d. LOC of class
    e. Access Specifier
    f. Inheritance and composition information

    Methods:

    g. Name of method
    h. Ownership of method
    i. Parameters list
    j. Return Type
    k. LOC of Function
    l. Local data members
2. Aggregation
3. Association
4. Composition
5. Struct
6. Enum
7. Abstraction (Virtual and pure Virtual)
8. Inheritance
9. Macros
10. Inline function
11. Polymorphism (Static and Dynamic)

2.4. Use case:

ID	01
Use case Name	Create new project
Description	User wants to create a new project
Actor	User
Precondition	Application is ready to use
Post Condition	New project is created
Normal Flow	1. Person gives a command to system to create for growing a new venture. 2. Machine permits the consumer to go into the brand new challenge records displaying a conversation container. 3. Using browse button person selects the C++ mission listing. 4. Person also can enter path of input directory through edit button. 5. User selects the output listing by browser button. 6. User also can type the output directory direction through edit button. 7. User selects the .Txt or .Xml files to save the end result. 8. Machine displays the fulfillment message.
Alternative Flow	1. a) The input directory does not contain any C++ source code. 2. b) The directory is invalid. 3. c) The path of output does not exist.

ID	02
Use case Name	Open project
Description	User wants to open an existing project.
Actor	User
Precondition	An already created project in the system.
Post Condition	Project is open.
Normal Flow	
	1. Person selects the command to open a new task.
	2. A brand new conversation gives the browse mechanism.
	3. User selects the undertaking.
	4. User can input the path of venture the usage of exit button.
	5. Person gives open command.
	6. Device opens assignment and success message is displayed.
Alternative Flow	
	1. A) User does not choose the venture and pressed the open button. System will show the message that no venture is open. Device asks to choose a mission once more. 3. B) Person selects invalid task folder. Gadget will show the message that no undertaking facts determined. System asks the person to choose again or exist.

ID	03
Use case Name	Close Application
Description	User wants to close the project.
Actor	User
Precondition	A project is opened in a system
Post Condition	Project is closed
Normal Flow	1. Person selects the near project command from menu to close task. 2. Application will ask person to save present day undertaking or not. 3. If person selects sure machine save adjustments and near the assignment. 4. If person selects no system will discard adjustments and near the venture.
Alternative Flow	1. A) If there's no assignment open then system will display error message. 2. B) If user selects cancel alternative then the device will return to foremost window without last challenge.

ID	04
Use case Name	Generates output
Description	User wants to generate outputs for tested. The output in XML and text layout. Person generates record of Tokens, Template training, Nested instructions and friend magnificence's information.
Actor	User
Precondition	Application is ready to use.
Post Condition	Selected output is generated.
Normal Flow	1. User pick out the output (XML record, text record). 2. Machine asks to the keep. 3. User gives the save command. 4. System generates the desired outputs.
Alternative Flow	1. a) System cannot save output on secondary storage. System generates relevant error message.

2.5. Assumptions:

The system is implemented for C++ language. If the input is other than C++ language then there might accept the input. But the system will produce incorrect result or no result.

2.6. Nonfunctional requirements:

These are the requirements which are needed to be in code however these necessities do not have an effect on the system functionalities. Nonfunctional requirements of the gadget are

1. Usability of the machine
2. Hardware Compatibility

3. Destiny enrichment

Summary:

On this phase I've accumulated facts approximately the device requirements, practical and nonfictional requirements.

Chapter 3

Syntax Study C++

3.1.Composition

3.2.Inheritance

3.3.Association

3.4.Struct

3.5.Enum

3.6.Macros

3.7.Inline function

3.8.Abstraction (Virtual and pure Virtual)

3.9.Polymorphism

Summary

3.1. Class composition:

Class composition means a class contains an object of a different classes. A class that has objects of other classes as their data members are called composite classes. An object member of a class is called component object. The class of a component object is called component class. Member function of a composite class cannot access private members of included objects. [10]

For example, a class Date would be composed of a Time object

- Class Date is a client (or user) of class Time

```
class Time {
private :
        int hour , minute , second ;
public :
        void setTime ( int , int , int ) ;
        void printTime ( ) const ;
};

class Date {
private :
        int day , month , year ;
        Time time ;
 public :
        Date ( int , int , int , int , int , int ) ;
         void printDate ( ) const ;
} ;
```

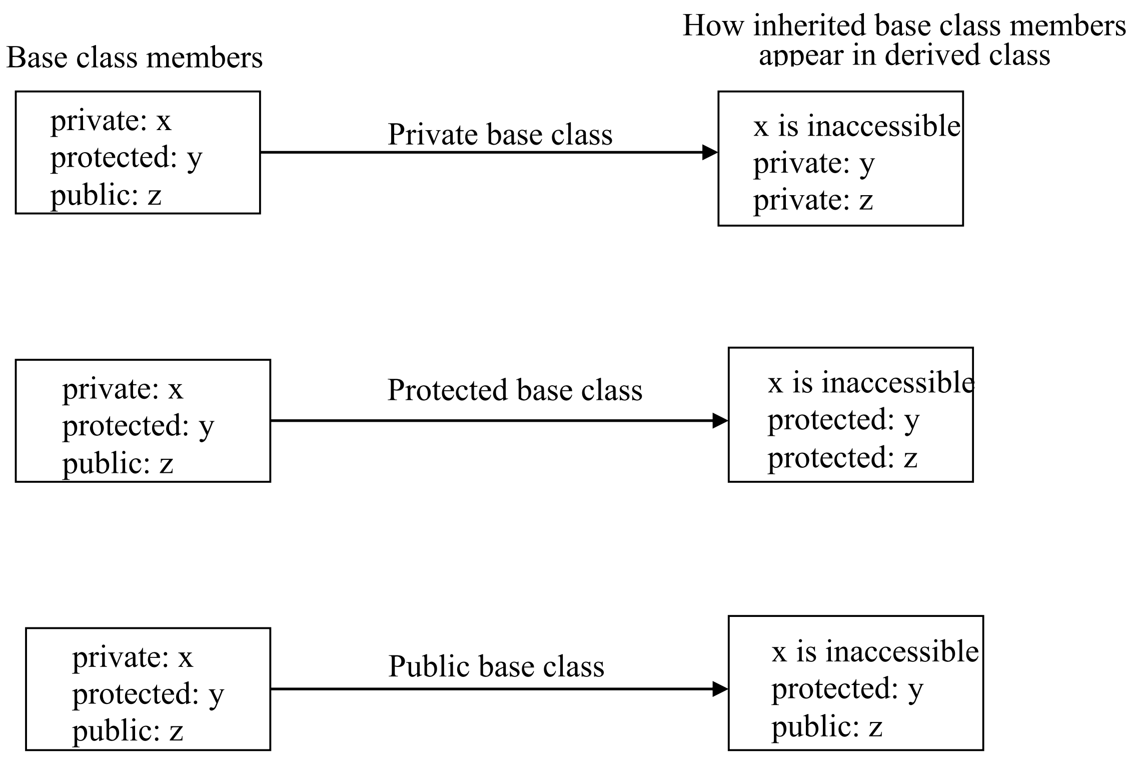
3.2. Inheritance:

Provides a way to create a new class from an existing class. The new class is a specialized version of the existing class. [6]

- Base class (or parent) – inherited from
- Derived class (or child) – inherits from the base class

```
class Student        // base class
    {
        . . .
    };
class UnderGrad: public student
    {                  // derived class
        . . .
    };
```

3.2.1. Access Control and Inheritance:



3.3. Association:

In object-oriented programming, association defines a relationship between classes of objects that allows one object instance to cause another to perform an action on its behalf. This relationship is structural, because it specifies that objects of one kind are connected to objects of another and does not represent behaviour. [19]

3.4. Struct:

There are many instances in programming where we need more than one variable in order to represent an object. For example, to represent yourself, you might want to store your name, your birthday, your height, your weight, or any other number of characteristics about yourself. You could do so like this: [19]

```
struct Employee
{
        short id;
        int age;
        double wage;
};
```

3.5. Enum:

An *enumeration* is a distinct type whose value is restricted to a range of values, which may include several explicitly named constants ("*enumerators*"). The values of the constants are values of an integral type known as the *underlying type* of the enumeration. [19]

```
enum Color { red, green, blue };
red=0, green=1, blue=2
```

3.6. Macros:

Preprocessor directives are lines included in the code of programs preceded by a hash sign (#). These lines are not program statements but directives for the *preprocessor*. The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements.

When the preprocessor encounters this directive, it replaces any occurrence of identifier in the rest of the code by replacement. This replacement can be an expression, a statement, a block or simply anything. [19]

```
#define TABLE_SIZE 100
int table1[TABLE_SIZE];
int table2[TABLE_SIZE];
```

3.7. Inline function:

Macro function an optimized technique used by compiler to reduce the execution time.

The inline functions are a C++ enhancement feature to increase the execution time of a program. Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime. [19]

```
Class A
{
        Public:
                inline int add (int a, int b)
                {
                        return (a + b);
                }
};

Class A
{
        Public:
        int add(int a, int b);
};

inline int A::add (int a, int b)
{
        return (a + b);
}
```

3.8. Abstraction (Virtual and pure Virtual)

Abstract Class is a class which contains atleast one Pure Virtual function in it. Abstract classes are used to provide an Interface for its sub classes. Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class. [4]

Virtual Function and Pure Virtual Function defers in declaration. Virtual Function is declared with keyword 'virtual' at the start of declaration.

- Ex: virtual return_type function_name(function arguments);

```
Class myclass
{
Public:
          Virtual void vfunc();
}
```

While Pure Virtual Function is declared ass

- Ex: virtual return_type function_name(function arguments) = 0;

```
Class myclass
{
Public:
          Virtual void vfunc()= 0 ;
}
```

3.9. Polymorphism:

Recall that polymorphism is the phenomenon where the same message sent to two different objects produces two different set of actions [11]. Polymorphism is broadly divided into two parts:

- *Static polymorphism* – exhibited by overloaded functions.
- *Dynamic polymorphism* – exhibited by using late binding

3.9.1. Static Polymorphism

Static polymorphism refers to an entity existing in different physical forms simultaneously. Static polymorphism involves binding of functions based on the number, type, and sequence of arguments. The various types of parameters are specified in the function declaration, and therefore the function can be bound to calls at compile time. This form of association is called *early binding*. The term *early binding* stems from the fact that when the program is executed, the calls are already bound to the appropriate functions.

The resolution of a function call is based on number, type, and sequence of arguments declared for each form of the function. Consider the following function declaration:

```
void add(int , int);

void add(float, float);
```

When the *add()* function is invoked, the parameters passed to it will determine which version of the function will be executed. This resolution is done at compile time.

3.9.2. Dynamic Polymorphism:

Dynamic polymorphism refers to an entity changing its form depending on the circumstances. A function is said to exhibit dynamic polymorphism when it exists in more than one form, and calls to its various forms are resolved dynamically when the program is executed. The term *late binding* refers to the resolution of the functions at run-time instead of compile time. This feature increases the flexibility of the program by allowing the appropriate method to be invoked, depending on the context.

Summary:

In this chapter, I have discuss the relationship and the function (like Composition, Inheritance, struct and enum etc.) for which my project, extract the information.

Chapter 4

Process of Fact Extraction

4.1.Introduction

4.2.Lexical Conventions

4.3.Lexical Analysis

4.4.Syntax Analysis

4.5.Intermediate Representation of Information

Summary

4.1. Introduction:

Fact extraction from software systems is the fundamental building block in the process of understanding the relationships among the system's elements. A *fact* is for instance the size of the code. Another fact is whether a class has base classes. Actually any information that helps us to understand unknown source code is called a fact. [15]

The fact extraction method is comparable to compiler as one box that maps a computer program into a semantically equivalent source program. If we tend to open up this box somewhat, we will see that there are 2 components to the present mapping: analysis and synthesis.

The analysis half breaks up the computer program into constituent items and imposes a grammatical structure on them. It then uses this structure to make associate degree intermediate illustration of the computer program. [13] If the analysis half detects that the computer program is either syntactically sick fashioned or semantically unsound, then it should give informative messages, therefore the user will take corrective action. The analysis half additionally collects data regarding the computer program and stores it during a system referred to as a logo table, which is passed beside the intermediate illustration to the synthesis half. The synthesis half constructs the required source program from the intermediate illustration and therefore the data within the image table. The analysis half is commonly referred to as the forepart of the compiler; the synthesis half is that the face. Compilation phase of fact extractor

- Lexical Analyzer
- Syntax Analyzer
- Intermediate representation

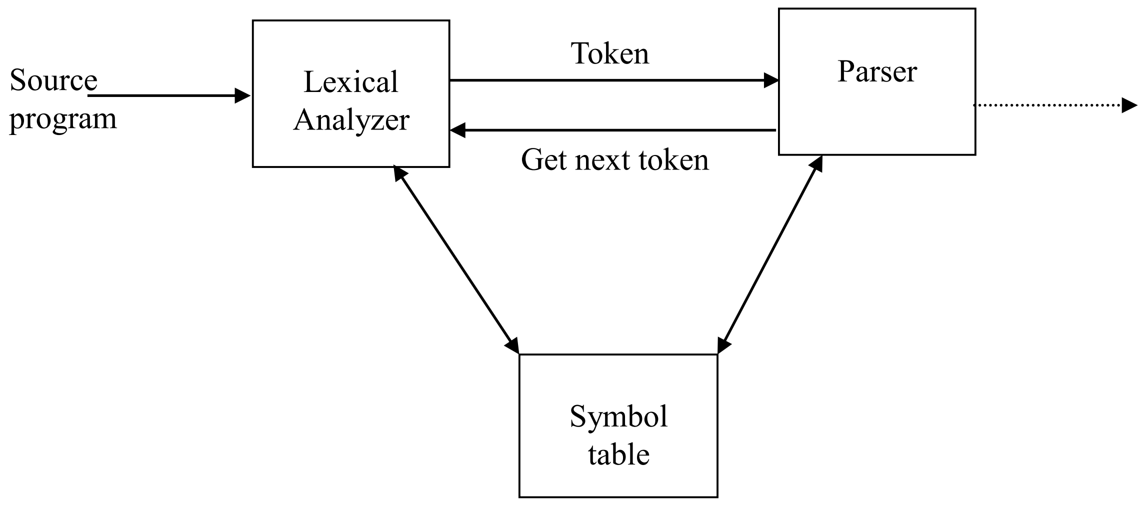


Figure 4.1-1 Fact Extraction Process

4.2. Lexical Conventions:

The fundamental elements of a C++ program called "lexical elements" or "tokens" to construct statements, definitions, declarations, and so on, which are combine to construct complete programs. The following lexical elements are discussed in this section:[]

4.2.1. Character set:

The C++ standard specifies a *basic source character set* that may be used in source file. To represent characters outside of this set, additional characters can be specified by using a *universal character name*. When compiled, the *basic execution character set* and *basic execution wide-character set* represent the characters and strings that can appear in a program. The C++ implementation allows additional characters in source code and compiled code. [MSDN]

4.2.2. Basic source character:

The *basic source character set* consists of 96 characters that may be used in source files. This set includes the space character, horizontal tab, vertical tab and new-line control characters, and this set of graphical characters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

4.2.3. Comments (C++):

A comment is text that the compiler ignores but that is useful for programmers. Comments are normally used to annotate code for future reference. The compiler treats them as white space.

A C++ comment is written in one of the following ways:

- The /* (slash, asterisk) characters, followed by any sequence of characters (including new lines), followed by the */ characters.
- The // (two slashes) characters, followed by any sequence of characters. A new line not immediately preceded by a backslash terminates this form of comment. Therefore, it is commonly called a "single-line comment."

The comment characters (/*, */, and //) have no special meaning within a character constant, string literal, or comment. Comments using the first syntax, therefore, cannot be nested.

4.2.4. Identifiers (C++):

An identifier is a sequence of characters used to denote one of the following:

- Object or variable name
- Class, structure, or union name
- Enumerated type name
- Member of a class, structure, union, or enumeration
- Function or class-member function
- typedef name
- Label name
- Macro name
- Macro parameter

The following characters are allowed as any character of an identifier:

a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

4.2.5. Keywords (C++):

break	else	continue	For	long	signed	switch
Case	default	enum	Goto	sizeof	typedef	string
Char	do	namespace	Return	static	union	while
class	friend	private	This	public	throw	cin
delete	true	protected	Endl	iomanip	main	std
cout	include	iostream	inline	virtual	void	static_cast
try	bool	new	catch	false	NULL	operator
Auto	const	double	Float	int	short	struct
template	if	unsigned				

4.3. Lexical Analyzer:

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

4.3.1. Tokens (C++):

A token is the smallest element of a C++ application that is substantial to the compiler. The C++ parser recognizes these types of tokens: identifiers, keywords, literals, operators, punctuators, and one-of-a-kind separators. A glide of these tokens makes up a translation unit.

Tokens are usually separated by *white space*. White space can be one or more:

- Blanks
- Horizontal tabs
- New lines
- Comments

For example:

```
const pi = 3.14159;                    //constant pi
```

Token 1: (Keywords, const)

Token 2: (Identifier, ‘pi’)

Token 3: (Operator, =)

Token 4: (Literal Value, 3.14159)

Token 5: (Punctuator, ;)

4.3.2. Longest Match Rule:

When the lexical analyzer read the source-code, it scans the code letter by letter; and when it encounters a whitespace, operator symbol, or special symbols, it decides that a word is completed.

For example:

int intvalue;

While scanning both lexemes till ‘int’, the lexical analyzer cannot determine whether it is a keyword *int* or the initials of identifier *intvalue*.

The Longest Match Rule states that the lexeme scanned should be determined based on the longest match among all the tokens available.

The lexical analyzer also follows rule priority where a reserved word, e.g., a keyword, of a language is given priority over user input. That is, if the lexical analyzer finds a lexeme that matches with any existing reserved word, it should generate an error.

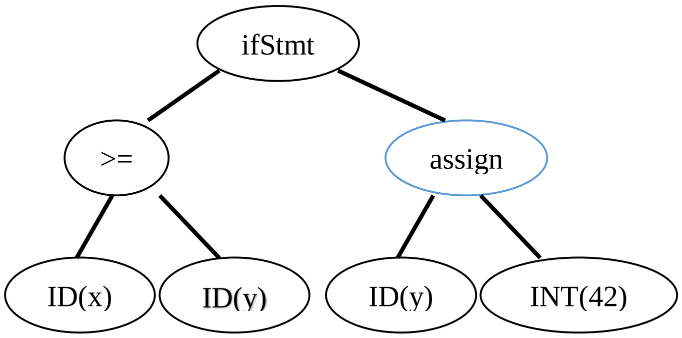
4.4. Syntax Analysis

The second section of the compiler is syntax analysis or parsing. The parser uses the primary parts of the tokens created by the lexical instrument to form a tree-like intermediate illustration that depicts the grammatical structure of the token stream. A typical illustration could be a syntax tree within which every interior node represents an operation and therefore the children of the node represent the arguments of the operation.

For Example:

Token form scanner

if (x >= Y) y = 42 ;



4.5. Intermediate Representation of Information:

In the process of translating a source program into target code, a compiler may construct one or more intermediate representations, which can have a variety of forms. Syntax trees are a form of intermediate representation; they are commonly used during syntax and semantic analysis. After syntax and semantic analysis of the source program, many compilers generate an explicit low-level or machine-like intermediate representation, which we can think of as a program for an abstract machine. This intermediate representation should have two important properties: it should be easy to produce and it should be easy to translate into the target machine.

Summary:

In this chapter, I have explained a general process of fact extraction (facts are the lexical convention and combined to make up the source program), which is similar to the complier processing which involves the following steps like lexical analysis (scanner), syntax analysis (parser) and the facts intermediate representation.

Chapter 5

System Architecture and Design

5.1.Software architecture
5.2.Software Design
5.3.Class diagram
Summary

5.1.Software architecture:

Software application architecture is the process of defining a structured solution that meets all the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. [MSDN]

The primary focuses of software architecture is to present the abstract picture of a device by means of hiding needless details of the system.

Like: system components and their overall interaction.

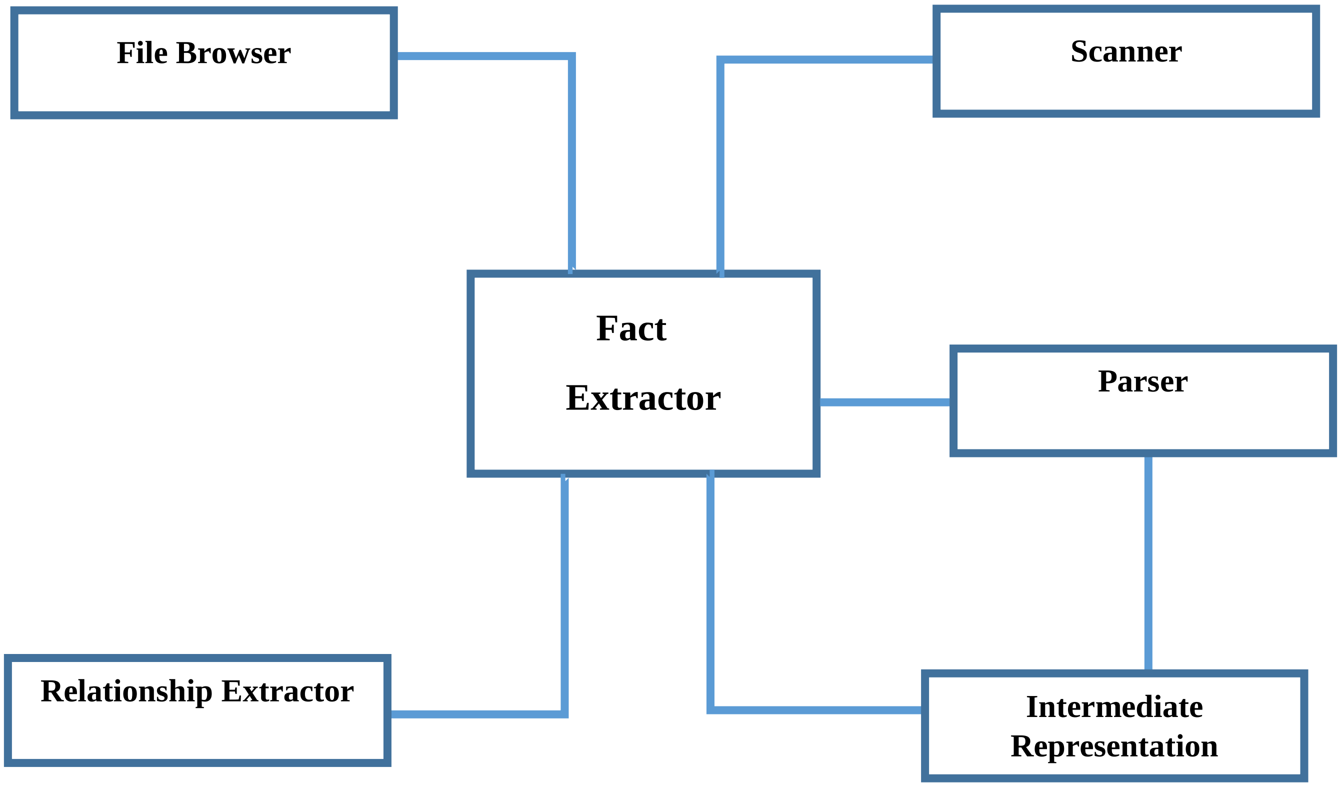


Figure 5.1-1 Architecture diagram

5.1.1. Fact Extractor:

Fact extractor is the main component of the system that incorporates with all other components of the system. Fact extractor component acts as a manager component that is connected with other components through interfaces. Each component is responsible for its own task but fact extractor is responsible for coordination and manipulation of different tasks of other components.

5.1.2. File browser:

File Browser is the component of the system which is used to locate and browse the files used in project. Using file browser a user can select and add file into project. File browser for this system locates only .cpp source files in the directory location provided by user that are to be analyzed. File browser communicate with the main component Fact extractor using an interface.

5.1.3. Scanner:

Scanner is another major component of the system which takes files located by file browser as input. The lexical analyzer takes .cpp source file as input and creates its tokens. Lexical analyzer breaks the sequence of characters into tokens. These tokens are sequence of characters in the file in form of string. These tokens are stored to a list that is used by main component.

5.1.4. Parser:

Parser is component of the system which takes the output of scanner as input in this component the syntax analyzer use that list and produce meaningful information from those tokens it generate information according to the rule of language. Tokens are classified like identifiers, keywords, operators, access specifiers, class name, method name, etc. This information is used to collect class information which is stored to the collection of information of class. Information of methods is collected and stored to the method collection class. All information generated by parse are sent to main component fact extractor and other components use that information.

5.1.5. Intermediate Representation:

The role of this component is to maintain data structures from the information provided by parser component. Classes for each required information are generated by this component.

Scanner class gives the information of each token in the form of

- Token Name
- Token Id

- Line Number
- Parenthesis Level
- Bracket Level

5.1.6. Relationship Extractor:

Relationship Extractor is a component interconnected with the main component. The purpose of this component is to extract the relationship form the information which is generated by the parser and is available as intermediate representation. This is a component which is running algorithms to extract information from the source code of all types which is required for Decision Making process.

5.2. Software Design:

System design is the process of defining the base for the development of the system using requirement analysis documentation. It is also the definition of architecture, components, modules, interfaces and data for system to meet the system requirements. [8]

5.2.1. Design constraints:

The constraints which are considered during system design are explained below:

5.2.1.1. Reusability:

Reusability is the constraint under which developer should keep in mind that code should be reusable with slight modification or without any modification.

5.2.1.2. Maintainability:

In this constraint it is measure how easy is understanding, modification and bug fixing process in the system. The system should be highly maintainable.

5.2.1.3. Modularity:

Modularity is the degree to which the components of the system may be separated and recombined.

5.2.1.4. Reliability:

The software must be able to perform a required function under the stated conditions for a specified time.

5.3.Class Diagrams:

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In UML a class diagram is represented containing:

- Name of class
- Data Members of class
- Function member or methods of a class

Class Name

Data member list

Function member list

5.3.1. Class diagram for folder reader:

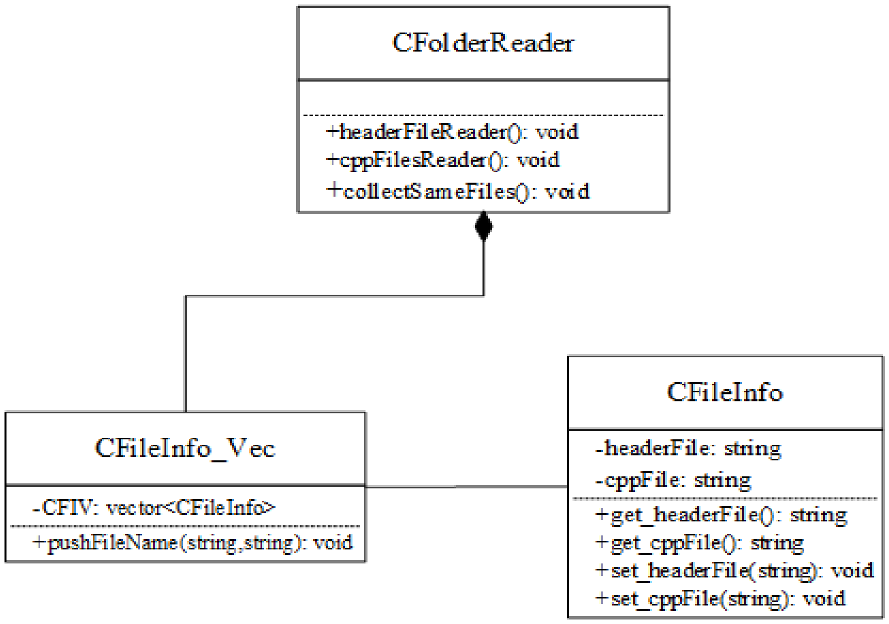

CFolderReader
+headerFileReader(): void
+cppFilesReader(): void
+collectSameFiles(): void
CFileInfo_Vec
-CFIV: vector<CFileInfo>
+pushFileName(string,string): void
CFileInfo
-headerFile: string
-cppFile: string
+get_headerFile(): string
+get_cppFile(): string
+set_headerFile(string): void
+set_cppFile(string): void

Figure 5.3-1 Class diagram for folder reader

5.3.1.1. CFolderReader:

CFolderReader is main component which is required to collect all the source files (header (.h) and .cpp) from a location and bring into the system for analysis.

5.3.1.2. CFileInfo:

CFileInfo analysis the file which bring into the system by CFolderReader and separate same header and cpp files of different classes.

5.3.1.3. CFileInfo_Vec:

CFileInfo_Vec stores header and cpp of same class, which is separated from CFileInfo class.

5.3.2. Class diagram for fact extraction:

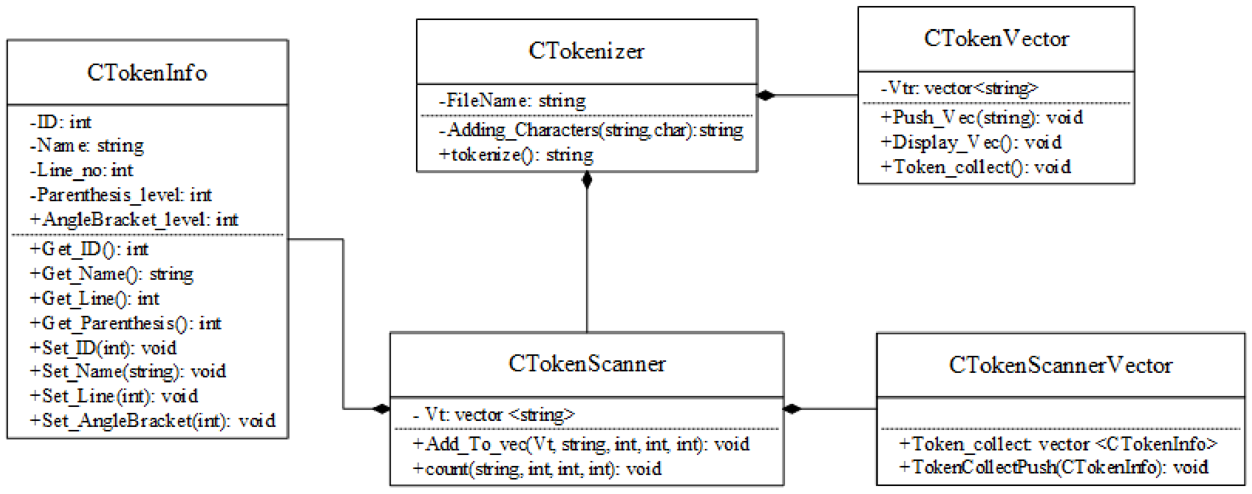


Figure 5.3-2 Class diagram for fact extraction

5.3.2.1. CTokenizer:

Tokenizer is manager class for all class being used for scanner. It interacts with token vector class which provide required information to them. It opens a file and a token creation method is implemented in it which tokenize the whole file. When a token found in the file its id is retrieved from vector of ids. And added to the token vector.

5.3.2.2. CTokenVector:

Tokenizer class separate the each word of C++ source and remove the comments from the code and store the code after tokenize. After that it is given to the TokenScanner class.

5.3.2.3. CTokenScanner:

TokenScanner is main class which identify the keyword and collect the information of that keyword for TokenInfo class and store in TokenScannerVector class.

5.3.2.4. CTokenInfo:

TokenInfo class provide the information about the token, such as

Token Id	Int
Token Name	String
Line No	Int
Parenthesis level	Int
Bracket Level	Int

Token id is get from the globally declared data variable

```
//---------------------------- Data Types -------------------------------
const int ID_IF = 100;                    // if
const int ID_IFELSE = 102;                // else if
const int ID_ELSE = 103;                  // else
const int ID_FOR = 104;                   // for
const int ID_WHILE = 105;                 // while
const int ID_DO = 106;                    // do
const int ID_INT = 107;                   // int
const int ID_BOOL = 108;                  // bool
const int ID_DOUBLE = 109;                // double
const int ID_CHAR = 110;                  // char
const int ID_FLOAT = 111;                 // float
const int ID_STRING = 112;                // string

// -------------------------------Brackets ---------------------------------
const int ID_Greater_Then = 300;                  // >
const int ID_Less_Then = 301;                     // <
const int ID_Cout = 303;                          // <<
const int ID_Cin = 304;                           // >>
const int ID_square_Brac_Start = 305;           // [
const int ID_square_Brac_End = 306;             // ]
const int ID_AngleBracket_Start = 307;           // {
const int ID_AngleBracket_End = 308;             // }
const int ID_parentheses_Start = 309;            // (
const int ID_parentheses_End  = 310;             // )

//------------------------------- Operators -------------------------------

const int ID_Phus = 400;                         // +
const int ID_Staric = 401;                       // *
const int ID_Negative = 402;                     // -
const int ID_Equal = 403;                        // =
const int ID_EqualsEqual = 404;                  // ==
const int ID_NotEqual = 405;                     // !=
const int ID_Not = 406;                          // !
const int ID_And = 407;                          // &&
const int ID_OR = 408;                           // ||
const int ID_Arrrow = 409;                       // ->
const int ID_Saicolan = 410;                     // ;
const int ID_Comma = 411;                        // ,
const int ID_Tild = 412;                         // ~
const int ID_Reference = 413;                    // &
const int ID_Collon= 414;                        // :

//--------------------------------Class, Structs etc---------------------------
const int ID_Class = 500;                     // class
const int ID_Struct = 501;                    // struct
const int ID_ENUM = 502;                       // enum
const int ID_Private= 503;                     //private
const int ID_Public = 504;                     //public
```

5.3.2.5. TokenScannerVector:

TokenScannerVector stores the final token which we get from whole fact extraction process.

5.3.3. Class diagram for extracting class Information

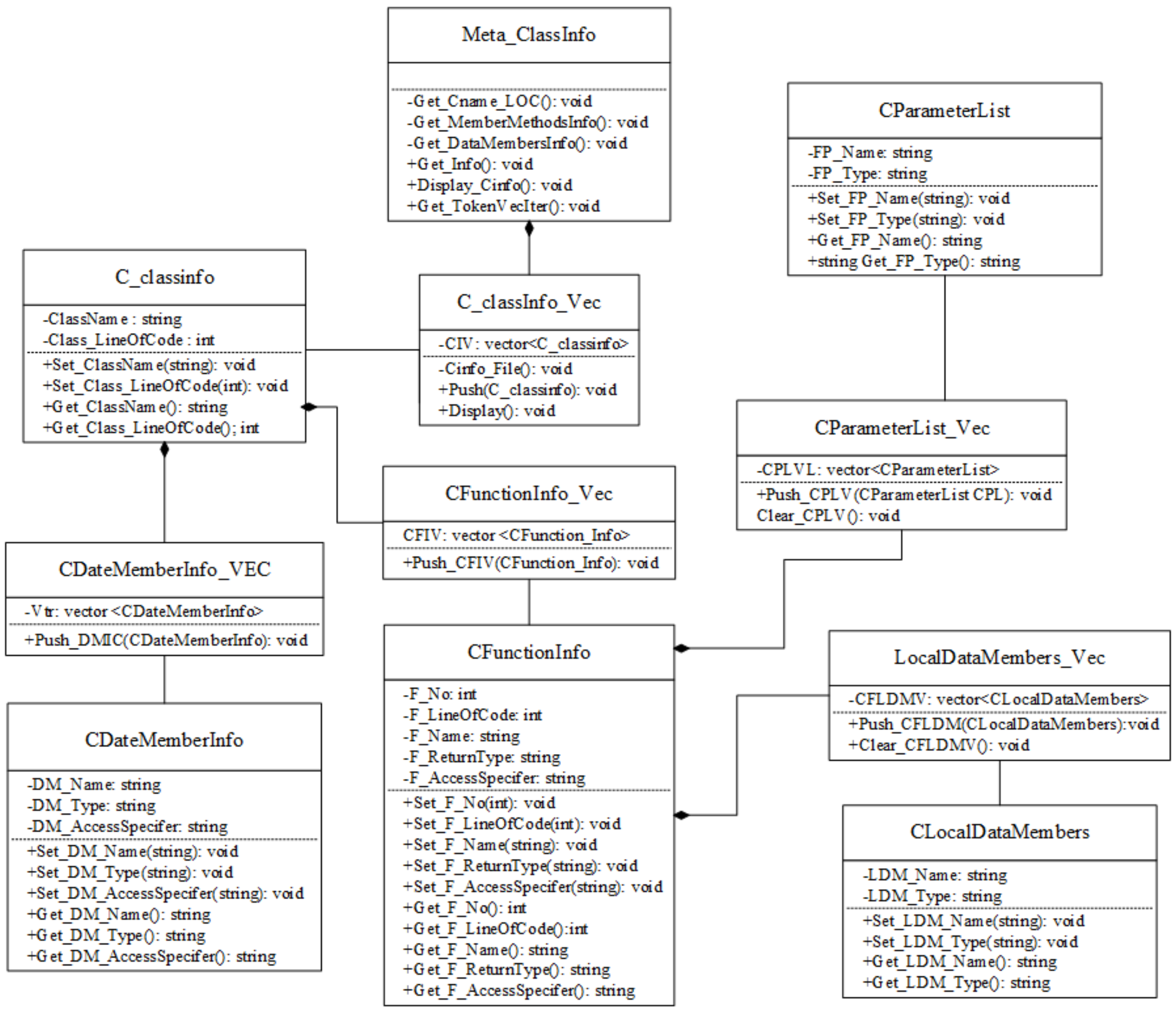

Meta_ClassInfo
-Get_Cname_LOC(): void
-Get_MemberMethodsInfo(): void
-Get_DataMembersInfo(): void
+Get_Info(): void
+Display_Cinfo(): void
+Get_TokenVecIter(): void
CParameterList
-FP_Name: string
-FP_Type: string
+Set_FP_Name(string): void
+Set_FP_Type(string): void
+Get_FP_Name(): string
+string Get_FP_Type(): string
C_classinfo
-ClassName : string
-Class_LineOfCode : int
+Set_ClassName(string): void
+Set_Class_LineOfCode(int): void
+Get_ClassName(): string
+Get_Class_LineOfCode(); int
C_classInfo_Vec
-CIV: vector<C_classinfo>
-Cinfo_File(): void
+Push(C_classinfo): void
+Display(): void
CParameterList_Vec
-CPLVL: vector<CParameterList>
+Push_CPLV(CParameterList CPL): void
Clear_CPLV(): void
CFunctionInfo_Vec
CFIV: vector <CFunction_Info>
+Push_CFIV(CFunction_Info): void
CDateMemberInfo_VEC
-Vtr: vector <CDateMemberInfo>
+Push_DMIC(CDateMemberInfo): void
CFunctionInfo
-F_No: int
-F_LineOfCode: int
-F_Name: string
-F_ReturnType: string
-F_AccessSpecifer: string
+Set_F_No(int): void
+Set_F_LineOfCode(int): void
+Set_F_Name(string): void
+Set_F_ReturnType(string): void
+Set_F_AccessSpecifer(string): void
+Get_F_No(): int
+Get_F_LineOfCode():int
+Get_F_Name(): string
+Get_F_ReturnType(): string
+Get_F_AccessSpecifer(): string
LocalDataMembers_Vec
-CFLDMV: vector<CLocalDataMembers>
+Push_CFLDM(CLocalDataMembers):void
+Clear_CFLDMV(): void
CDateMemberInfo
-DM_Name: string
-DM_Type: string
-DM_AccessSpecifer: string
+Set_DM_Name(string): void
+Set_DM_Type(string): void
+Set_DM_AccessSpecifer(string): void
+Get_DM_Name(): string
+Get_DM_Type(): string
+Get_DM_AccessSpecifer(): string
CLocalDataMembers
-LDM_Name: string
-LDM_Type: string
+Set_LDM_Name(string): void
+Set_LDM_Type(string): void
+Get_LDM_Name(): string
+Get_LDM_Type(): string

Figure 5.3-3 Class diagram for extracting class

5.3.3.1. Meta_Classinfo

This class is the main part of Information Extractor component of the project. This class maintain the information of class which we use farther for finding the relationships and function.

5.3.3.2. C_classInfo:

This class store information of class name, line of code, and function of the class.

5.3.3.3. CDateMemberInfo:

This class stores all information about data members. The information constraints are data member name, data member type, data member access specifier of the data member.

5.3.3.4. CDateMemberInfo_VEC:

A Class may have many or none data members. They all are kept in DataMemberInfo type STL vector.

5.3.3.5. CFunctionInfo:

This class stores information about Member methods of a class. It composes two more classes Parameters and Local data Members to store information about these two constraints of a method. The other constraints associated with the member method are method return type, method access specifier, method LOC, method name, number of parameters, number of local data members, and line of code.

5.3.3.6. CFunction_Info_Vec:

Class may have function or member methods. They all are kept in Function_Vec type STL vector.

5.3.3.7. CParameterList:

CParameterList class contains information about parameters of a method. Parameter type can be user defined or primitive type. A parameter can be in two states, by reference or by value. CParameterList class contains all these characteristics about parameter.

5.3.3.8. CLocalDataMembers:

CLocalDataMembers class consist of collection of all local variables declared in methods of classes.

5.3.4. CRelationFinder

CRelationFinder is main class for finding relationships between classes.

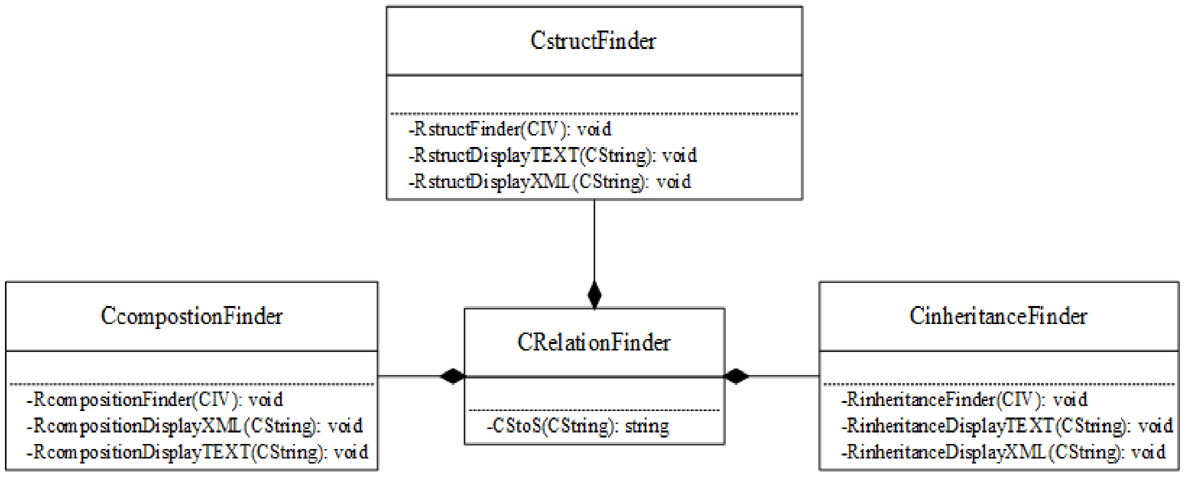

CstructFinder
-RstructFinder(CIV): void
-RstructDisplayTEXT(CString): void
-RstructDisplayXML(CString): void
CcompostionFinder
-RcompositionFinder(CIV): void
-RcompositionDisplayXML(CString): void
-RcompositionDisplayTEXT(CString): void
CRelationFinder
-CStoS(CString): string
CinheritanceFinder
-RinheritanceFinder(CIV): void
-RinheritanceDisplayTEXT(CString): void
-RinheritanceDisplayXML(CString): void

Figure 5.3-4 Class diagram for Relation finder

5.3.4.1. CstructFinder:

CstructFinder class is used for finding the struct info from the extracted classes' information. And display info into different formats.

5.3.4.2. CinheritanceFinder:

CinheritanceFinder class is used for finding the inheritance info from the extracted classes' information. It can find any kind of inheritance type like single, multi-level, multi-path etc. And display info into different formats.

5.3.4.3. CcompostionFinder:

CcompostionFinder class is used for finding the composition of classes. And display info into different formats.

5.4. User Interface Design:

User Interface of application is a single document windows explorer type application. It looks like following screen shots:

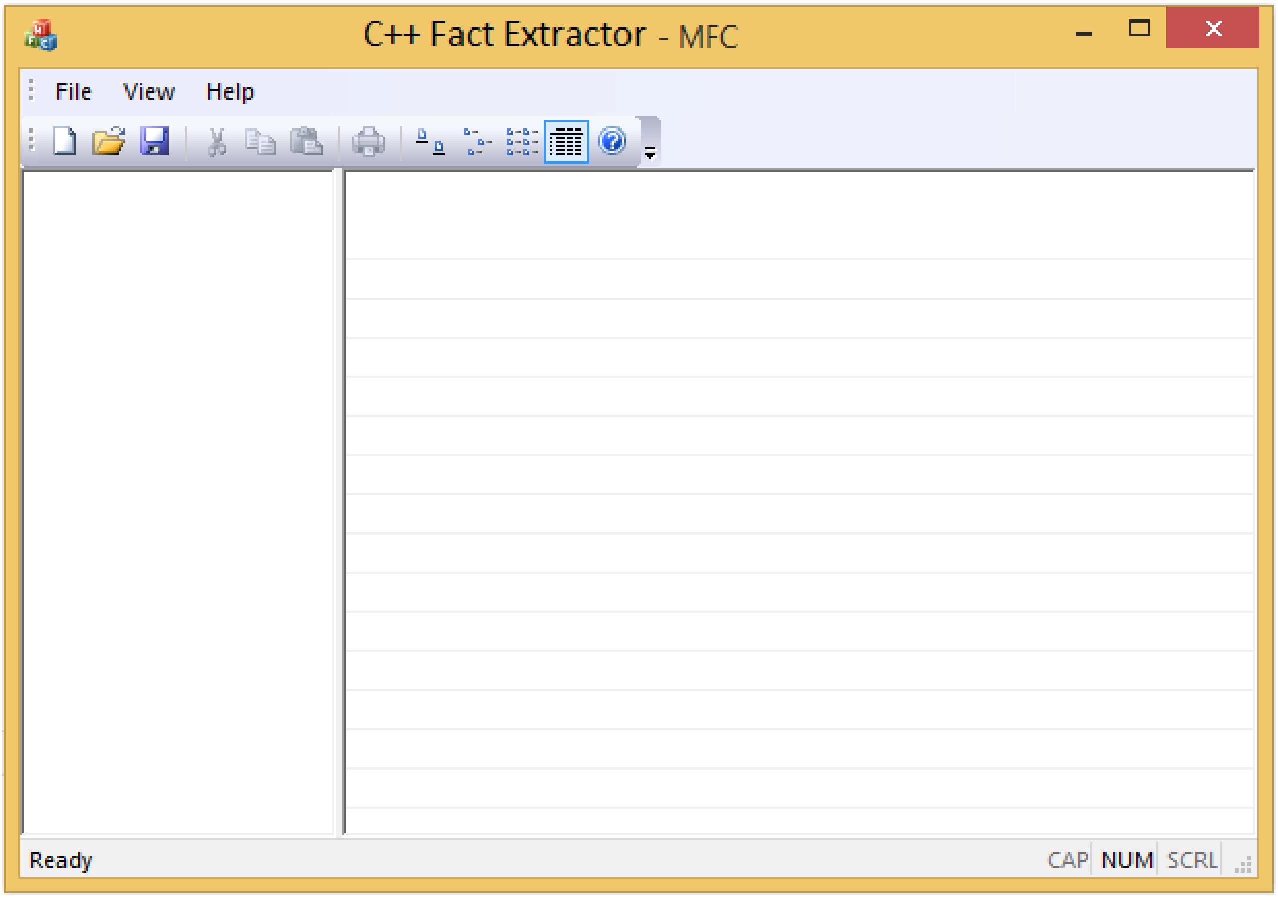


Figure 5.4-1 User Interface main screen

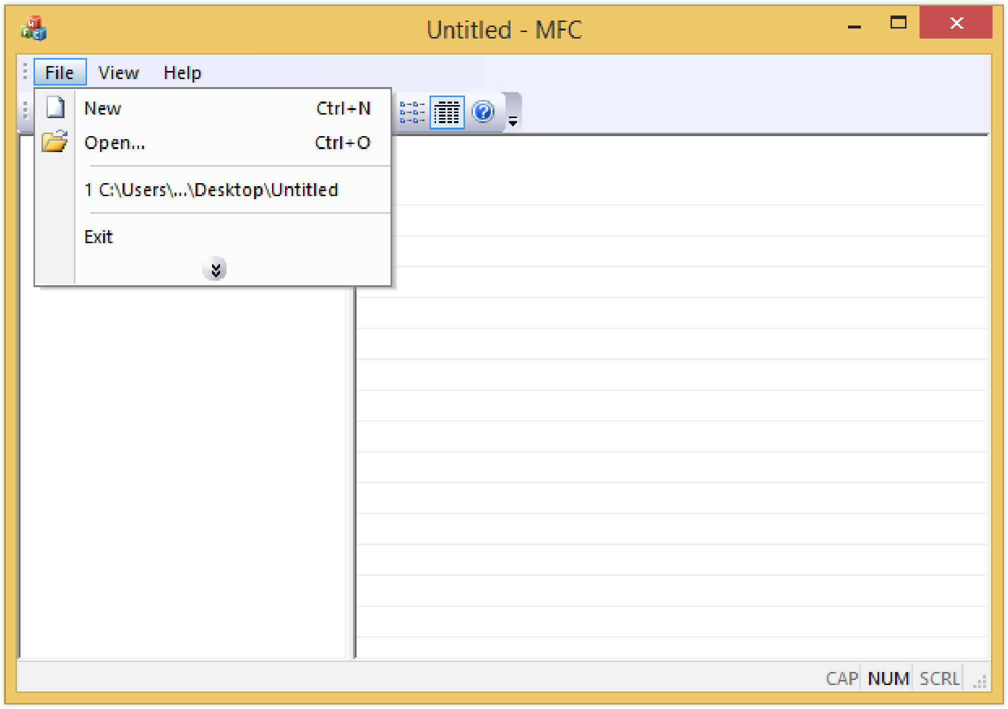


Figure 5.4-2 Open new project

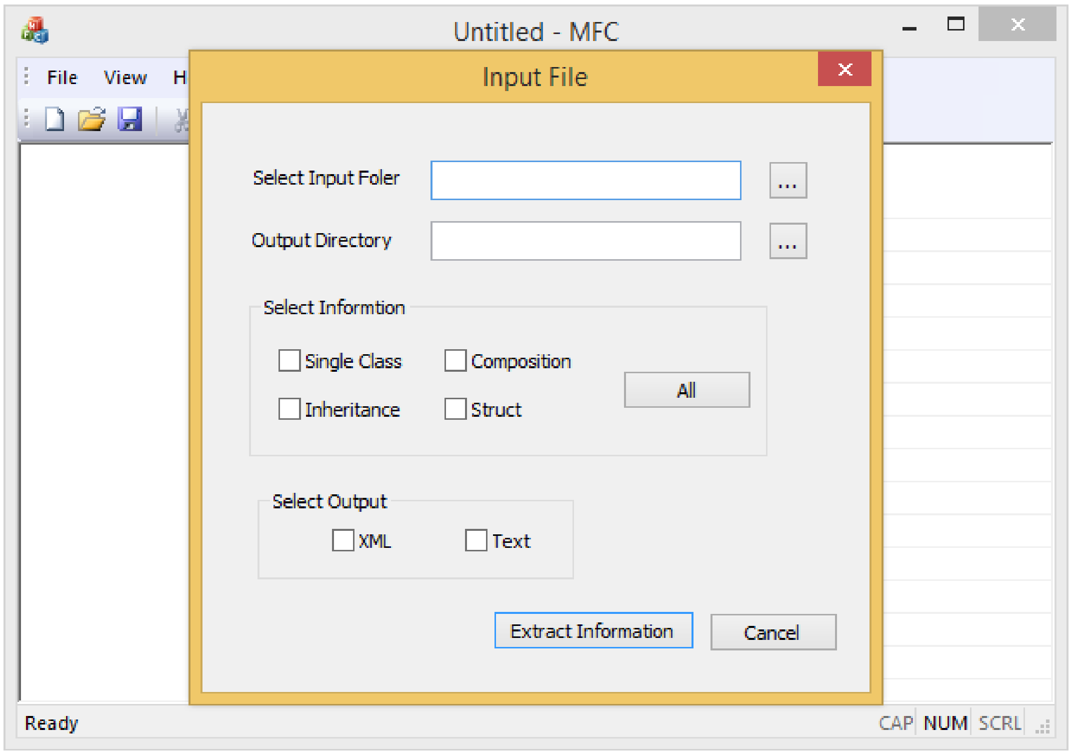
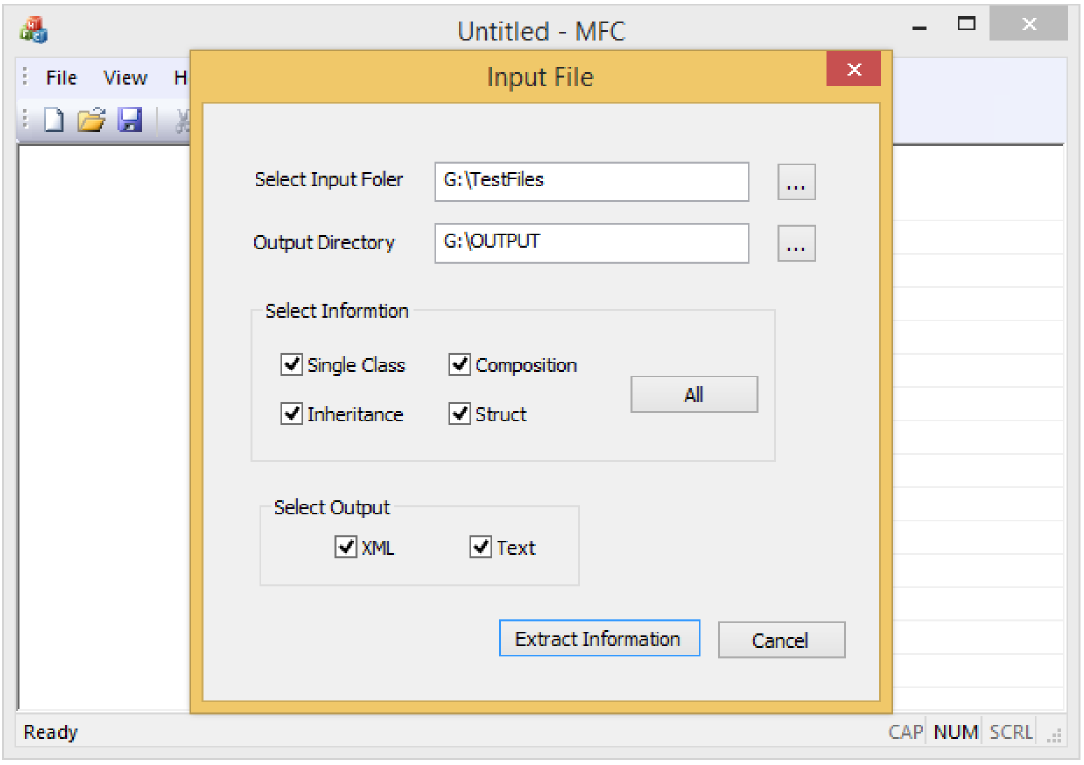


Figure 5.4-3 New project Dialog box



Select Input Foler G:\TestFiles ...

Output Directory G:\OUTPUT ...

Select Informtion

☑ Single Class ☑ Composition

☑ Inheritance ☑ Struct

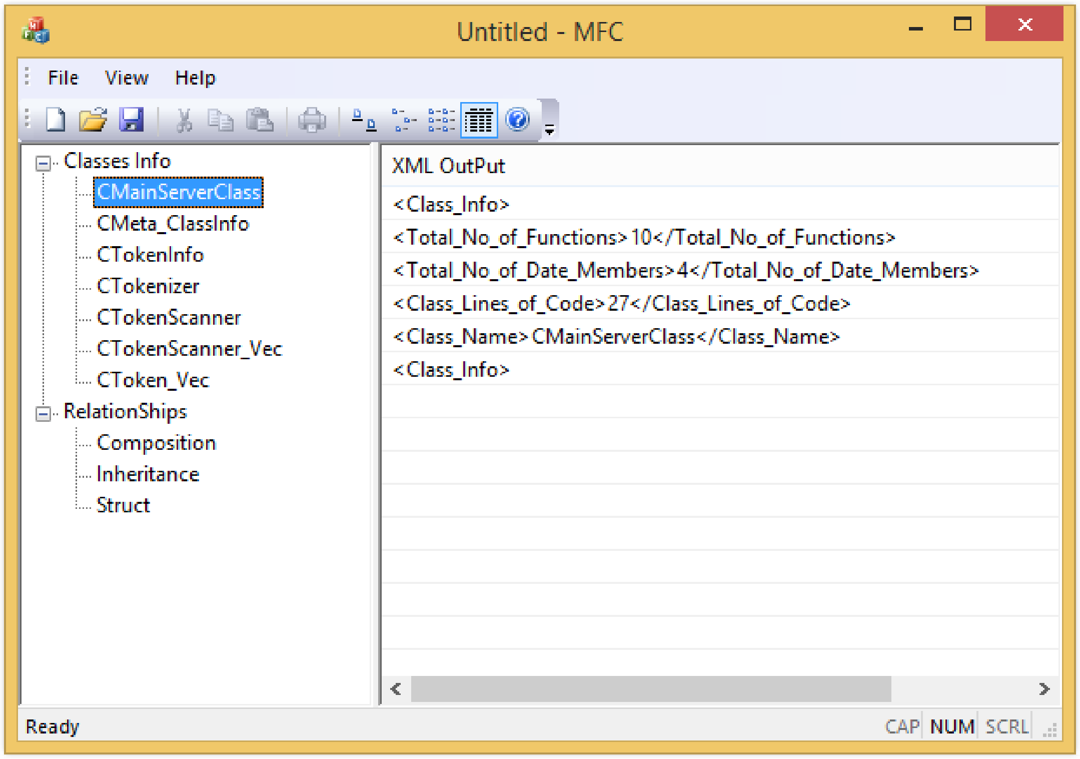
All

Select Output

☑ XML ☑ Text

Extract Information Cancel

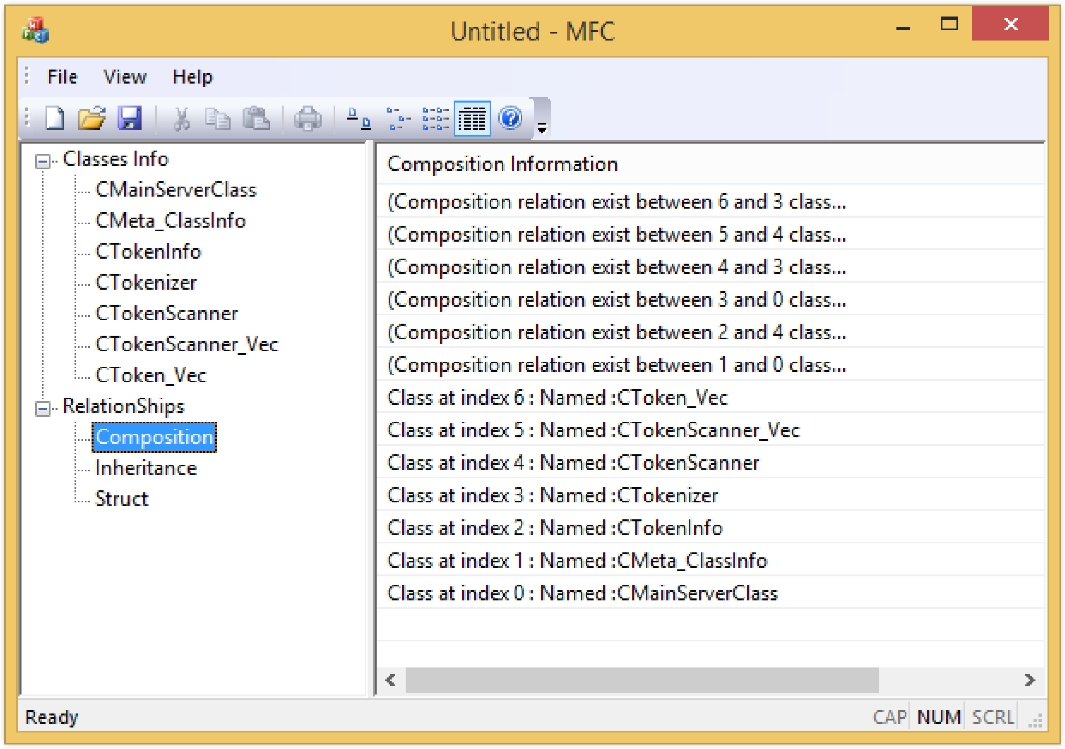
Figure 5.4-4 Fill dialog box for Desired info



- Classes Info
  - CMainServerClass
  - CMeta_ClassInfo
  - CTokenInfo
  - CTokenizer
  - CTokenScanner
  - CTokenScanner_Vec
  - CToken_Vec
- RelationShips
  - Composition
  - Inheritance
  - Struct

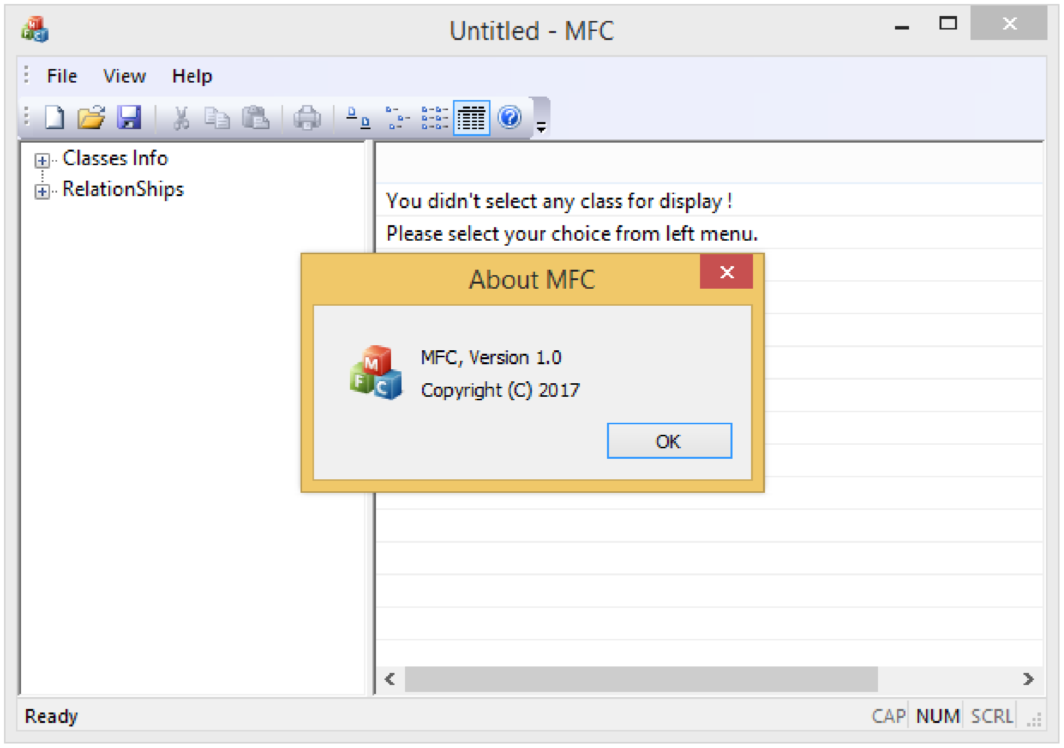
```
XML OutPut
<Class_Info>
<Total_No_of_Functions>10</Total_No_of_Functions>
<Total_No_of_Date_Members>4</Total_No_of_Date_Members>
<Class_Lines_of_Code>27</Class_Lines_of_Code>
<Class_Name>CMainServerClass</Class_Name>
<Class_Info>
```

Figure 5.4-5 Extracted class info



```
Composition Information
(Composition relation exist between 6 and 3 class...
(Composition relation exist between 5 and 4 class...
(Composition relation exist between 4 and 3 class...
(Composition relation exist between 3 and 0 class...
(Composition relation exist between 2 and 4 class...
(Composition relation exist between 1 and 0 class...
Class at index 6 : Named :CToken_Vec
Class at index 5 : Named :CTokenScanner_Vec
Class at index 4 : Named :CTokenScanner
Class at index 3 : Named :CTokenizer
Class at index 2 : Named :CTokenInfo
Class at index 1 : Named :CMeta_ClassInfo
Class at index 0 : Named :CMainServerClass
```

Figure 5.4-6 Extracted relation info



```
You didn't select any class for display !
Please select your choice from left menu.
```

MFC, Version 1.0
Copyright (C) 2017

Figure 5.4-7 Version No. of Tool

- When User opens project he has first screen then click **File->New**
- Dialog Screen will appear
- Fill the dialog box for desired info then click Extract Information button.
- After completing processing tree view is generated.
- Tool shows the info on screen and also store the info into files.

Summary:

In this Chapter I have discussed the system architecture and design and the flow of system. I have designed the class diagrams for our system and their relations.

Chapter 6

System Input & Output

6.1.Introduction

6.2.Input Files

6.3.Output Files

Summary

6.1. Introduction:

In this chapter, I have discuss the input and output of the system. The system take the whole C++ project as in input and from input the system takes the .CCP file and from .CCP file the system can analyze the input and generates the output in the form of XML File and TEXT File.

The system out the information in two ways such as:

- XML File
- Text File

6.2. Input Files:

A C++ program as a whole used as input to the system. A C++ program may contain many header and C++ files. You must compile before you can run the file's code on your Windows system. You use Visual Studio to compile the CPP C++ program Windows. The compiling process creates an EXE file, which is an executable that runs on a Windows computer. The header and .cpp files are given below.

6.2.1. Header file:

```
#include <afx.h>
#include <shlobj.h>
#include <stdio.h>

struct stFiles
{
        Int iImage;
        CString m_FileName;
        CString m_FilePath;
        CString m_strDate;
        long    m_AccessDays;
        double  m_FileSize;

};
```

```
class CFolder : public CObject
{
        // Data Members
public:
        CString         m_FolderPath;
        CString         m_FolderName;
        BOOL            m_FilledFlag;
        double          m_FolderSize;
        HTREEITEM       hItem;
        CArray<stFiles,stFiles> m_FileInfoList;
        CObArray        m_FolderInfoList;
        int             iImage;
        int             iSelImage;
        long            m_AccessDays;
        CString         m_strTime;

        // Member functions
public:
        BOOL bSortedList;
        CFolder();
        virtual ~CFolder();

        CString GetPath();
        void SetPath(CString strPath);
        LPSHELLFOLDER GetChildFolder();
        void SetChildFolder(LPSHELLFOLDER pFolder);
protected:
        LPSHELLFOLDER pChildFolder;

};
```

6.2.2. .cpp file:

```
// Folder.cpp: implementation of the CFolder class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "NxDUtility.h"
#include "Folder.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFolder::CFolder()
{
	m_FilledFlag=FALSE;
	bSortedList=FALSE;
}

CFolder::~CFolder()
{
}

void CFolder::SetChildFolder(LPSHELLFOLDER pFolder)
{
	pChildFolder = pFolder;
}

LPSHELLFOLDER CFolder::GetChildFolder()
{
        return pChildFolder;
}

void CFolder::SetPath(CString strPath)
{
        m_FolderPath = strPath;
}

CString CFolder::GetPath()
{
        return m_FolderPath;
}
```

6.3. Output:

Output is in two formats XML and Text.

6.3.1. XML format:

```
<class_info>

        < class Number>1</Number>

        < class name> CFolder  </name>

        < class line_of_code>71</line_of_code>

        < class total_functions>6</total_functions>

<data_member_info>

        <Total>11</Total>

        <AccessSpecifer>public</AccessSpecifer>
```

```
    <Date_Member_Type> m_FolderPath </Date_Member_Type>
    <Date_Member_Name> m_FolderName </Date_Member_Name>
    <Date_Member_Name> m_FilledFlag </Date_Member_Name>
    <Date_Member_Name> m_FolderSize </Date_Member_Name>
    <Date_Member_Name> hItem </Date_Member_Name>
    <Date_Member_Name> m_FileInfoList </Date_Member_Name>
    <Date_Member_Name> m_FolderInfoList </Date_Member_Name>
    <Date_Member_Name> iImage </Date_Member_Name>
    <Date_Member_Name> iSelImage </Date_Member_Name>
</data_member_info>
    <Class_Function_Information>
        <Function_No>1</Function_No>
        <Function_AccessSpecifer>private</Function_AccessSpecifer>
        <Function_Return_Type>string</Function_Return_Type>
        <Function_Name> GetPath() </Function_Name>
        <Function_Line_of_Code>2</Function_Line_of_Code>
            <Function_parameter>
            </Function_parameter>
            <local_data_members>
            </local_data_members>
    <Class_Function_Information>
    <Class_Function_Information>
```

```
        <Function_No>2</Function_No>
        <Function_AccessSpecifer>private</Function_AccessSpecifer>
        <Function_Return_Type>void</Function_Return_Type>
        <Function_Name> SetPath() </Function_Name>
        <Function_Line_of_Code>2</Function_Line_of_Code>
                <Function_parameter>
                        <Parameter_Type> CString  </Parameter_Type>
                        <Parameter_Name> strPath </Parameter_Name>
                </Function_parameter>
                <local_data_members>
                </local_data_members>
        </Class_Function_Information>
</class_info>
```

6.3.2. Text format:

```
Class No:                       1
Class Name:                     CFolder
Class Lines of Code:            72
Total No. of Functions:         6

Class Date Members Information          Total: 11
AccessSpecifer      :     Date Member Type    :      Date Member Name
private             :         CString         :            m_FolderPath
private            :        CString          :        m_FolderName
private            :        BOOL             :        m_FilledFlag
private            :        double           :        m_FilledFlag
private            :        HTREEITEM        :        hItem
private            :  CArray<stFiles,stFiles> :        m_FileInfoList
private            :        CObArray         :        FolderInfoList
private            :        int              :        iImage
private            :        long             :        m_strTime
-----------------------------------------------------------------------------------------------
~~~~~~~~ Class Function Information ~~~~~~~~
Function No:                    1
Function AccessSpecifer:        public
Function Return Type:           void
Function Name:                  SetChildFolder
Function Line of Code:          2
~~~~~~~~ Parameters of Function          Total: 1 ~~~~~~~~
Parameter Type: LPSHELLFOLDER            Parameter Name: pFolder
~~~~~~~~ Local Data Members of Function          Total: 0 ~~~~~~~~
-----------------------------------------------------------------------------------------------
~~~~~~~~ Class Function Information ~~~~~~~~
Function No:                    2
Function AccessSpecifer:        public
Function Return Type:           LPSHELLFOLDER
Function Name:                  GetChildFolder
Function Line of Code:          2
~~~~~~~~ Parameters of Function          Total: 0 ~~~~~~~~
~~~~~~~~ Local Data Members of Function          Total: 0 ~~~~~~~~
```

```
----------------------------------------------------------------------------------------------------
~~~~~~~~ Class Function Information ~~~~~~~~
Function No:                    3
Function AccessSpecifer:        public
Function Return Type:           void
Function Name:                  SetPath
Function Line of Code:          2
~~~~~~~~ Parameters of Function          Total: 0 ~~~~~~~~
~~~~~~~~ Local Data Members of Function          Total: 0 ~~~~~~~~
----------------------------------------------------------------------------------------------------
```

Summary:

In this chapter, I have explained the inputs and the outputs of the system. System accepts the C++ program as a whole as input. After extraction of useful information is gives output in two formats XML and text.

Chapter 7

Testing and Evaluation

7.1. Introduction

7.2. Testing:

7.3. Test Cases

Summary

7.1. Introduction:

System testing is the last important step to complete software for delivery. After the development of the system is complete, we're required to check the product. The development of a system is done with keen interest that the system works according to our will but there is always a chance that after deployment something still can go wrong. To avoid that system testing is done to ensure that the system works properly. It is a kind of investigation that is conducted to ensure that the system meets the quality desired by the customer. [7]

A successful system testing unveils all the errors of the system, if any. It also shows that the required qualities, usability, reliability, other functional and nonfunctional necessities of the system are met.

7.2. Testing:

Testing is done to collect bugs and errors in the designed system. Another purpose of testing could be to ensure if the product meets the requirements of the system or not.

7.2.1. Black box testing:

In black box testing different inputs are tested against expected outputs. So the tester do not need to know much about coding for testing.

7.3. Test Cases:

Test case for extraction information:

Test Case ID	01
Tester	Muhammad Ali
Test Type	Black Box Testing
Test Case Name	Extract information of Classes
Procedure	User can select header and .cpp files from a C++ project

Expected Result	When the user selects the header and .cpp file the system should extract all information about classes in the project.
Actual Result	Class information is returned to user in the form of XML or TEXT file.
Status	Successful

Test Case ID	02
Tester	Muhammad Ali
Test Type	Black Box Testing
Test Case Name	Extract information of Composition
Procedure	User can select header and .cpp files from a C++ project
Expected Result	When the user selects the header and .cpp file the system should extract all information about composite classes in the project.
Actual Result	Composite Class information is returned to user in the form of XML or TEXT file.
Status	Successful

Test Case ID	03
Tester	Muhammad Ali
Test Type	Black Box Testing
Test Case Name	Extract information of Inheritance
Procedure	User can select header and .cpp files from a C++ project

Expected Result	When the user selects the header and .cpp file the system should extract all information about Inherit classes in the project.
Actual Result	All information of inheritance classes is returned to user in the form of XML or TEXT file.
Status	Successful

Test Case ID	04
Tester	Muhammad Ali
Test Type	Black Box Testing
Test Case Name	Extract information of Enumeration
Procedure	User can select header and .cpp files from a C++ project
Expected Result	When the user selects the header and .cpp file the system should extract all information about Inherit classes in the project.
Actual Result	All information of enum type variables is returned to user in the form of XML or TEXT file.
Status	Successful

Summary:

In this chapter, I have discussed testing of the software system. Then, I have stated some test cases for our system.

Chapter 8

Conclusion and future work

8.1.Conclusion
8.2.Future work

8.1. Conclusion:

C++ Fact Extractor tool that I have developed will be very helpful for maintainers and developers in the field of software development as well as software maintenance. User will have to give C++ source code as an input and the tool will extract information of their desired relationship and functions. This can also extracted information about classes such as (class name, line of code, data members, number of constructor and function information (function name, function return type, function parameters and function line of code)). And the tool will generate XML based file and TEXT based file as an output.

After the completion of this tool it will save time and effect of maintainers in maintaining any C++ project.

8.2. Future work:

- We can further enhance the functionality of this tool
- We can make refactoring tool for maintainer
- A tool for commenting the code

Reference:

1. http://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm
2. https://msdn.microsoft.com/en-us/library/2e6a4at9.aspx
3. https://www.google.com.pk/search?q=incremental+model&rlz=1C1NHXL_enPK692PK692&tbm=isch&tbo=u&source=univ&sa=X&ved=0ahUKEwirscCCjI3OAhWMaRQKHeH1AukQsAQIKQ&biw=1366&bih=667#imgrc=8jNMttASbKcDMM%3A
4. http://www.ibm.com/support/knowledgecenter/SSPSQF_9.0.0/com.ibm.xlcpp111.aix.doc/language_ref/cplr061.html
5. https://msdn.microsoft.com/en-us/library/3bstk3k5.aspx
6. http://www.cplusplus.com/doc/tutorial/inheritance/
7. http://robincse.blogspot.com/2012/04/steps-of-requirement-engineering.html
8. https://en.wikipedia.org/wiki/Systems_design
9. https://msdn.microsoft.com/en-us/library/system.xml.xmldocument(v=vs.110).aspx
10. http://www.learncpp.com/cpp-tutorial/102-composition/
11. http://pages.cs.wisc.edu/~anhai/courses/784-sp10-anhai/ieSurvey.pdf
12. https://www.google.com.pk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiU4uTpn4HSAhXDvBoKHciDDtIQFggcMAA&url=http%3A%2F%2Fieeexplore.ieee.org%2Fdocument%2F1199197%2F&usg=AFQjCNH2o9F6otz9AfWs_WrBogSICuaTOA&sig2=QMCAKpPPahJQyiD_a7fFSg

Research papers:

13. Author: R. Ferenc (Res. Group on Artificial Intelligence, Univ. of Szeged, Hungary) "Extracting Facts with Columbus from C++ Code", Published in: Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on 13 March 2002, Publisher**:** IEEE

14. Author: Michael L. Collard, Huzefa H. Kagdi, Jonathan I. Maletic, "AXML-based lightweight C++ fact extractor" , Conference: 11th International Workshop on Program Comprehension (IWPC 2003), May 10-11, 2003, Portland, Oregon, USA

Books:

15. Compilers, Techniques and Tools
    Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman
16. Beginning Visual C++ by Ivor Horton
17. Compilers
    Dr. Matt Poole 2002, edited by Mr. Christopher Whyley
18. SAMS - Teach Yourself Visual C++ 6 in 21 Days by Davis Chapman
19. Software Engineering, A Practitioner’s Approach by Roger S. Pressman
20. Object oriented programming by Deitel Deitel
21. C(sharp) Fact Finder by Muhammad Inam Chatha thesis
22. Java Fact Extractor by Sami ul Haq thesis