Faculty of Computer Science & Information Technology
University Of Malaya 50603 Kuala Lumpur
Malaysia

Project Title

Web information retrieval and monitoring Using adaptive agent

Supervised by
Mr. Woo Chaw Seng

Moderated by
Prof. Madya Dr. Mohd Sapiyan Baba

Prepared by
Chong Yuen Beng (WEK98032)

Dissertation submitted by Chong Yuen Beng
in partial fulfillment of the requirements for the
Degree of Bachelor of Computer Science
Submission Date February 9, 2001

DECLARATION

Hereby, I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published, unpublished work of others has been acknowledged in the text, and a list of references is given.

Chong Yuen Beng

February 9, 2001

Abstract

Internet is the information house. There are million of pages posted in the Internet, and still growing in exponential. Locate a web page, without a search engines, is like a blind explorer.

There are many search engines in the existence world, but more new search engines is under construction. Each search engine has its advantage and disadvantage, depend on the need of the searchers.

The projects aims to develop an agent, with some advanced features, to allow the searchers obtain the information from the Internet easily.

The advanced features included, table of contents preview, chart of relevant, session review and resuming, etc.

In addition, the system has the ability to learn, allow the system to grown without the help from the human. As the information in the Internet is expanding, editing the system's knowledge is not an easy task.

Acknowledgement

Specially thanks to the supervisor, Mr. Woo Chaw Seng, and the moderator, Prof. Madya Dr. Mohd Sapiyan Baba for their support, advice, time in helping me in the project.

Thanks to my team member, Khong Yoong Meng, Lim Su Sian, and Cheah Hoong Seng, having their time and co-operation with me in the project.

To the faculty, provided the facilities, allowed me to search for much information from the Internet; and allowed me refer to previous works done by the seniors.

Table of Contents

List of Figures

List of Tables

Chapter I: Introduction

This chapter introduces to the type of search agents, overview and objective of the project: Web information retrieval and monitoring.

This chapter also includes the system functions, both normal search and advanced search.

1.1 Introduction
1.2 Directories
1.3 Search Engines
1.4 Meta-Crawlers
1.5 Disadvantage of Current System
1.6 Project Overview
1.7 Objective
- Interface Design
- Web Content Searching and Monitoring
- Web Content Processing
- Results

1.8 System Functions
- Normal Search
- Advanced Search
- Additional features with member Login

1.9 The Limitation
- Graphical Information
- Language
- Bottle Neck in System Performance

1.1 Introduction

The Internet has tens of millions of sites and the numbers of web site still growing in exponential; it's hard to locate a web site, without a Search Agent. In other words, without a search agent, the Internet will not be able to grow as we seen now. Search agent makes the Internet – alive.

Two basic approaches have been used in the search agents are: online search engines, and directories.

Searchable directories and search engines treat information differently. They approach it differently, store it differently, and present it to the world differently.

1.2 Directories

Directories, or a subject guide, such as Yahoo, Snap, LookSmart, and Magellan are useful for browsing general topics. An online directory is actually a vast collection of categories and subcategories constructed by people, not a computer program.

The major difference between search engines and directories is that a directory has structure. The directory is very useful to locate any information, when the keyword is unsure, or unknown. When a keyword is unsure, just look for the appropriate category and a comprehensive listing of sites for a particular subject can be reach, without missing any information.

Because of the need of structuring the hierarchy of information in a directory, the directories are fine for browsing general topics, for specific information, a search engine in needed.

The results search by directories is often having higher accuracy, but results sometimes are very limited. Broken links may happen.

1.3 Search Engines

In search engines, the content of search engine is unknown, until a keyword is typed in the query box.

All search engines do keyword searches against a database.

Searching in search engine is not an easy job. To effectively getting information from a search engine, difference keyword is needed in searching a particular topic.

1.4 Meta-Crawlers

The search engines explained in section 1.3 all uses computer programs which often-called spider or robot to explore the Internet. The spiders and robots continuously collect information from the Internet, find new web pages, indexing it, and store in their database. See Appendix A for the list of search engines, directories and Meta-Crawlers.

Meta crawlers also called Meta Search Engines, searches multiple search engines simultaneously. Meta crawlers do not have own indexing database, but uses results collected by other search engines.

1.5 Disadvantage of Current System

Current system has it limitation, which may improve in the project:

(a) Search session cannot be resumed when:
- Computer shutdown
- Illegal operation of Browser
- Disconnected from the server

(b) History unavailable when moving to other computer. Every search needs to be restarted from the beginning.

(c) Search by matching the exact keyword, not accurate and not broad.

(d) No update information, i.e. last update of the pages.

(e) Broken links.

1.6 Project Overview

The project aims to developing an agent that finds information on the net according to individual interests.

This project will be developed by a team of four persons.

I. Lim Su Sian, will develop a user interface that allows input of instructions into the agent.

II. Cheah Hoong Seng, will develop an engine that schedule jobs for web content searching and monitoring.

III. Myself, will develop a neural network that helps the agent to search and monitor web information.

IV. Khong Yoong Meng, will develop a module that generate charts using collected information.

1.7 Objective

The main objectives of the project is allow user to search the web contents based on their own interest. The system provides some advanced features, which may lead the users to decide if their want to visit the pages.

Using the member's profile, users may also bookmark a page, resume previous search, and receive update notification.

1.7.1 Interface Design

The interface design is aims to give user easiest way to querying the web URLs. The features include:

- The interface will allow user the select a topic, and select related keyword from a list;
- Allow user enter the keyword into a "Boolean expression ready" query box;
- Automatic phrase searching;
- Suggest phrase to user;
- Check for typo error;
- Case sensitivity searching;
- Media searching and filtering;

Besides, the interfaces collect feedback from users as well. The feedback collected may use to stimulus the learner (user enter keyword), decide the ranking or a page (user visited the page), etc.

1.7.2 Web Content Searching and Monitoring

In this project, the URLs will collected by summit keyword to other search engine, similar to the meta-crawlers. After the URLs were stored into database, the agent will continuously explore the page, gathering the update information and HTML documents. The objectives are:

- Monitoring the web page, if the page was updated, or was moved, or closed;
- Download the page, for indexing and analyzing.

1.7.3 Web Contents Processing

The processes are to indexing and analyzing the contents of a page. The major processes are:

(a) Keyword Indexing;
(b) Learning;
(c) Table of Contents Making; and
(d) Relevant Analyzing.

See chapter IV for more details about the processes.

1.7.4 Results

Some advanced results will be produced, to aid the users to decide if they want to visit the page by previewing the contents.

1.7.4.1 Chart of relevant

The most active keywords will be used in the chart, this allow the users to know what briefly is in the web page. *See section 1.6: Advanced Search for more details.*

1.7.4.2 Table of Contents

The table of contents listed the title, any headers in the page, and any link to other pages. This will help user to preview the contents of the pages. *See section 1.6: Advanced Search for more details.*

1.8 System Functions

This session will discussing the major features of the search agent. The features can be divided into normal search, and advanced search. Besides, the functions also distinguish between features with user logged on, and without log on.

1.8.1 Normal Search

Normal search refer to the default feature provide to the user. Normal search usually required no special overhead.

1.8.1.1 Search by Keyword

Normal search provide user search particular information using the keyword. User allows keying in the keyword, or selecting a keyword from the list.

1.8.1.2 Suggest Related Keyword

Related keyword is always listed. When users key in a keyword, any keyword that related to this keyword will be listed for reference.

1.8.1.3 Boolean Expression

The interface included a 'Boolean expression ready' query box. Users will not need to learn to how to do Boolean search.

Topic	Computer
Related Keyword:	Artificial Intelligent
AND:	Neural Network
OR:	Genetic Algorithm
NOT Include:	Fuzzy Logic

1.8.1.4 Automatic Phrase Searching

When users entered two or more keyword into a column, for example: Neural Network, the agents will do phrase searching automatically. *See learner: phrase learning for more details about phrase searching.*

1.8.1.5 Suggestion of Phrase

When users entered a keyword, for example: Neural; the agent will advice any related phrase to the users, to search by more accurate phrase: 'Neural Network'. This will help the users enquire the results they need faster, when the users do not know much about the topic they want to search.

1.8.1.6 Check for Typo Error

Users may entered incorrect keyword, due to typo error, forgot the spelling, or don't know the exact spelling of a keyword. Most search engine will return 0 search results when users entered incorrect keyword.

1.8.1.7 Case Sensitivity Searching

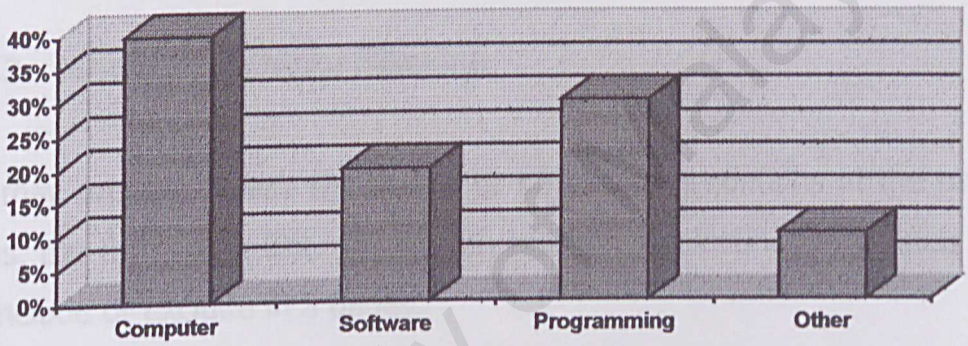
When users entered all low case letters, the agent will search for both upper case and lower case letter. When the users entered an upper case letter, the agent will search by the exact keyword (case sensitive). *See Issues in Designing Search Engines* in Chapter II.

1.8.2 Advanced Search

Advanced search refer to additional advanced features in the agent. Advanced features normally need some overhead, and often, results will not 100% guarantee to be correct.

1.8.2.1 Chart of Relevant

Most search engine only provides the relevant to the keyword entered by the users. Users may not able to guess how much relevant of the pages for other keyword.



The relevant chart shows the percentage of different keyword. This can help user have brief idea about the contents of the page.

1.8.2.2 Table of Contents (TOC)

The tables of contents allow user to preview any page before they visit the page. Of cause, the users may want to view the TOC, as this required a little overhead.

Computer ---- What is computer?

---- Computer evolution
---- http://www.abc.com/add.html
---- http://www.bcd.com/zyx/ads.html

The above example shown that the title of the page is 'Computer', while there are two headers in the page, titled 'What is computer?' and 'Computer evolution'. From the table of contents, the users know that the pages have two hyperlink into other pages, which is 'http://www.abc.com/add.html' and 'http://www.bcd.com/zyx/ads.html'.

1.8.2.3 Media Searching/Filtering

Users may preview what's the media or objects that contains in the pages. Besides, the users may filter the search results by selecting the objects that must include or exclude in a pages.

There are 16 type of media has been identified and will be indexed in the database.

1. Image
2. Audio (wave file)
3. MP3
4. Video
5. Flash
6. Java, or Applet
7. JavaScript
8. ActiveX Object
9. Executable program file
10. Acrobat Portable Document Format (PDF)
11. VBScript

12. PostScript document
13. Real Media (Real Audio and Video)
14. Compressed file
15. Form, need input from user
16. Other, other undefined objects, e.g. Microsoft Office document.

1.8.3 Additional features with member Login

The features mention above can be enjoyed by all users, anyway, there are few features, which can be offered only for the member. This is because these features need some storage in server-side database.

1.8.3.1 Bookmark

With member login, member can bookmark a page into their account. User can revisit the page, without restart the search, or wrote it down someway. The bookmark feature is special design for users who uses public computer, such as in school's lab, office's computer. Once a page has been added to the bookmark, user can revisit the page from any computer.

1.8.3.2 Session

Every search session is recorded, whenever the user login as a member. Most search engines do not allow a search session to be continued. With this feature, users may review what was search before and resume the session, without restarting the search from the beginning, and click 'next' to jump over the URLs.

1.8.3.3 History

Together with session, users may review what page their visited during any search session. Same as bookmark, the history can be access anyway in the world.

The history help user to identify which page has been or has not been visited during their search. This may help user avoid revisiting a useless page, or let the user revisit an interesting page.

1.8.3.4 Update Notification

The last advantage of member login is users may receive update notification through email. User is notify when:

(a) Interested sites was updated:

Any page that bookmark by the user; will receive a notification email when the page was updated.

(b) New URLs were discovered:

When new URLs were discovered, user can receive a notification when the contents meet the criteria of any search session.

1.9 The Limitation

1.9.1 Graphical Information

When analyzing the contents of a page, graphical representation of information (e.g. webmaster may include a phrase into image) is ignored. This may lead to inaccurate relevant calculation, missed headers in table of contents. Searching to the page maybe unsuccessful as the keyword is inside the graphics and not found in the text.

1.9.2 Language

The spell checker, keyword relation, phrase is designed based on English. Although keyword from other languages (e.g. Malay) can be query through the agent, the result is not guarantee to be correct.

1.9.3 Bottle Neck in System Performance

For the first three months, the system will have a bad performance. The system should need a certain period before it can achieve a stable status. The factors in determine the period included: numbers of testers, affection of the learning process, speed of Internet connection.

Chapter II: Literature Review

This chapter included some review of current search engines, their features, advantage and disadvantage. This chapter will discuss about AI (Artificial Intelligence), Neural Network, and the approaches in produce this project.

2.1 Search Engines Review

- Issues in Designing Search Engines
- Features Comparisons

2.2 Neural Network

- Properties and Capabilities
- Human Brain
- Models of Neuron
- Network Architectures
- Artificial Intelligence and Neural Networks

2.1 Search Engines Review

This section discusses the issues in designing a search engine. Some comparisons between major search engines are listed [2].

2.1.1 Issues in Designing Search Engines

Following discuss some issues or features when designing a search agent.

2.1.1.1 Conventions Used

This refers to the operator ('AND', '+', etc.), syntax (enter keyword), or prefix (e.g. for phrase searching) that the user is required to enter in order to perform a search.

In most search engine, the both operator in word (And) or symbol (+) is accepted. The default operator is various between search engine, some uses 'or' as default and some uses 'and' for default.

All search engines accept keyword for searching, but some do accept a whole sentence to be entered into the search box (e.g. www.webtop.com). Typo error won't check as most search engines assume users may want to search for that keyword.

All search engines do phrase searching when user entered double or single quote (") for the keywords, some will do automatic phrase searching.

2.1.1.2 Search Engine Size

The "size" stated by search engines conventionally refers to the number of unique web pages (unique URLs), rather than "sites" (which may contain numerous "pages").

Boolean Operators (+, -, AND, OR, NOT) and Parentheses

In general, there are two type of Boolean capability among search engines. Most search engines accept both type of Boolean operators, word (AND, OR) and symbol (+, -).

Operators	Symbols Used
AND	&
OR	+
NOT	Prefix '-'

Besides this, most search engines also have the capability of nesting (the use of parentheses).

AltaVista and Excite use "AND NOT" for the "NOT". Some engines required that Boolean operators be capitalized, but some do not.

2.1.1.3 Phrase Searching

For almost all search engines, a phrase searching is done by putting the phrase in double quotes in the query box. In some cases, a phrase can also be designated by choosing the phrase option from a pull-down menu.

2.1.1.4 Proximity

Phrase searching is one form of proximity searching. Proximity searching refers to the ability of searching out of the keyword provide by the users. For example, when use search by the word 'Computer', the agents should be able to search also the keyword related to computer.

The most common proximity option is NEAR, which specifies "within 10 words" in AltaVista and "within 25 words" in Lycos advanced search.

Lycos advanced search also provides BEFORE and FAR options.

2.1.1.5 Truncation

Most search engines provide search with truncation and usually donate with the symbol (*). The capability enable user to enter part of the keyword, or part of the phrase.

2.1.1.6 Title, Date, and URL fields

Some search engines offer the ability of searching by title, date and URL. This usually implemented using a prefix. For example: "title: neural network".

2.1.1.7 "Links to" a URL

This refers t o the capability of identifying which pages in the search engine's database contain a link to a particular URL. This enables the users to identify sites that have some interest in the site referred to.

2.1.1.8 Language

This refers to the capability of searching by the language in which the web page is written.

2.1.1.9 Media Searching

This refers to the capability for searching by type of media--images, audio files, and video files. For the several search engines that provide this, the implementation can be quite different. In AltaVista, a word in an image file's name can be searched (or, better, use the special "Images, Audio & Video" option). In HotBot[3], beside perform a subject search, can also specify that only records that contain an image, sound, or video file.

2.1.1.10 Case Sensitivity

The search engines should treat upper and lower case letters differently. The users will recognize the importance of this in instances when they need to distinguish between "ADIS" and "aids".

In general, when a query is entered using all lower case, the search engine will retrieve both lower and upper case. When the searcher enters upper case, a fully case-sensitive engine should be used and return only those

records with an exact case match. For example, "next" will retrieve "next and "neXt, whereas "neXt" will only retrieve "neXt."

2.1.1.11 Gives Count for Answer

All major search engines – except Excite, provide a count of the search results.

2.1.1.12 Output Options

This refers to whether if the user can specify the number of record on each results page, etc.

2.1.1.13 Also Shown on Results Pages

This addition to the chart reflects one of the major changes that have taken place in search engines. Results pages are now often far richer than they were a year or so ago. The producers have done an excellent job in incorporating additional relevant material into the pages, beyond just the results of the search of the Web database. This includes such things as: results of a search on the engine's associated Web directory, links to company homepages and company directories, related searches or search terms, results from associated popularity search programs, related news, etc.

2.1.1.14 "More Like This"

Some search engines enable a user to find other records that are similar to the current result. This is often useful for finding additional relevant records.

2.1.2 Feature Comparison

The following charts [2] listed some comparison among the major search engine.

	AltaVista	Excite	HotBot	Infoseek	Lycos	Northern Light
Content Size	250M pages	250M pages & media objects	110M sites	75M pages	50M pages	200M sites
Full-text	Yes	Yes	Yes	Yes	No	Yes
Default word (logic)	OR	OR	AND	or	and	and
Boolean connectors	AND, AND NOT, NEAR	AND, AND NOT	OR, NOT	and, not	or, not, adj., near, before, far	or, not
Phrase search	Quotation marks	Quotations marks	Quotation marks	Quotation marks	Quotation marks	Quotation marks
Case sensitive	Yes	No	Yes	Yes	No	No
Words included	Use +	Use +	Use +	Use +	Use +	Use +
Word elimination	Use -	Use -	Use -	Use -	Use -	Use -
Duplicate detection	Grouped under one title	Yes	Grouped under one title	Yes	Yes	Yes

Special features	Limit by date, language, or format field followed by a colon	Concept searching suggests terms	Limit by date, language, location, page depth	Find similar searches	Search for image and sound files	Custom folders

Table 2.1: Features comparison between major search engines

2.2 Neural Network

Artificial neural network, commonly referred to as "neural networks", has been motivated right from its inception by the recognition that the human brain computers in an entirely different way from the conventional digital computer.

The brain is a highly complex, nonlinear, and parallel computer (information-processing system). It has the capability to organize its structural constituents, known as neurons, which able to perform certain computations (e.g. pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today.

2.2.1 Properties and Capabilities

Neural network derives a powerful computing technique, with its parallel-distributed structure and its ability to learn. Therefore, the neural network is generalized. Generalization refers to ability of producing reasonable outputs for inputs not encountered during training (learning).

These two information-processing capabilities make it possible for a neural network to solve complex, large-scale problems that are currently intractable.

2.2.1.1 Non-linearity

Neural network can be linear or nonlinear. Non-linearity is important if the underlying physical mechanism responsible for generation of the input signal (e.g. speech signal) is inherently nonlinear.

2.2.1.2 Input-Output Mapping

Learning with a teacher or supervised learning involves modification of the synaptic weights of a neural network by applying a set of labeled training samples or task examples. Each example consists of a unique input signal and corresponding desired response.

The training of the network is repeated for many examples in the set until the network reaches a steady state where there are no further significant changes in the synaptic weights.

Thus, the network learns from the examples by construction an input-output mapping for the problem at hand.

2.2.1.3 Adaptively

Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions.

2.2.1.4 Evidential Response

In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also about the confidence in the decision made. This latter information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network.

2.2.2 Human Brain

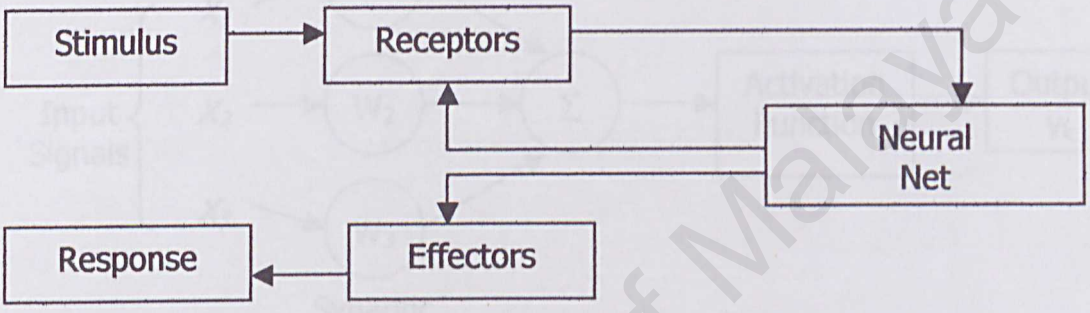


Figure 2.1 Block diagram representation of nervous system

The receptors convert stimuli from the human body (input) or an external environment into electrical impulses that convey information to the neural net (brain). The effectors convert electrical impulses generated by the neural into discernible responses as system outputs.

2.2.3 Models of Neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. There are three basic elements of the neuronal model:

(a) A set of synapses or connecting links, each of which characterized by a weight or strength of its own.

(b) An adder for summing the input signals, weighted by the respective synapses of the neuron

(c) An activation function for limiting the amplitude of the output of a neuron. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0, 1] or alternatively [-1, 1].

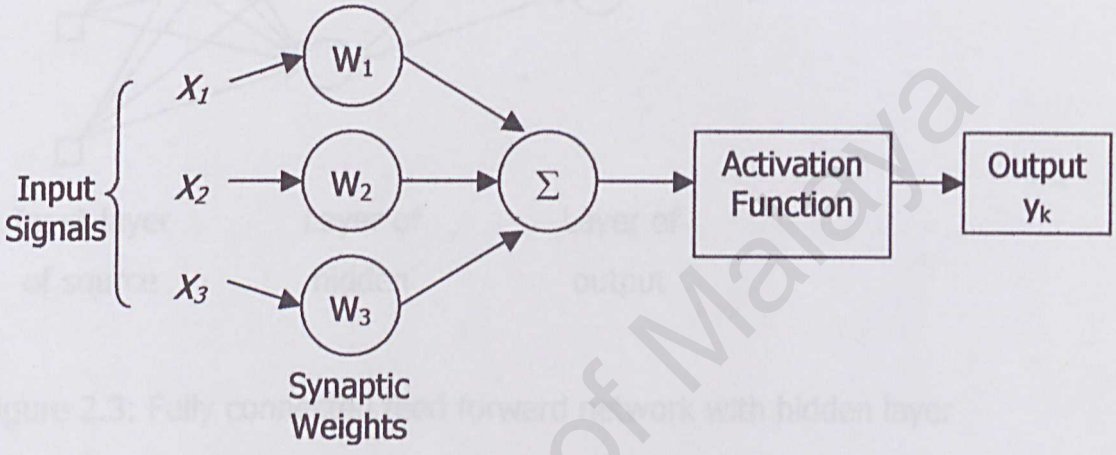


Figure 2.2: Nonlinear model of a neuron

$$y = \sum_{1}^{i} x_i w_i$$

where y = output value

x = input signals

w = synaptic weights

2.2.4 Network Architectures

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. Therefore, the learning algorithms (rules) used in the design of neural networks is being structured.

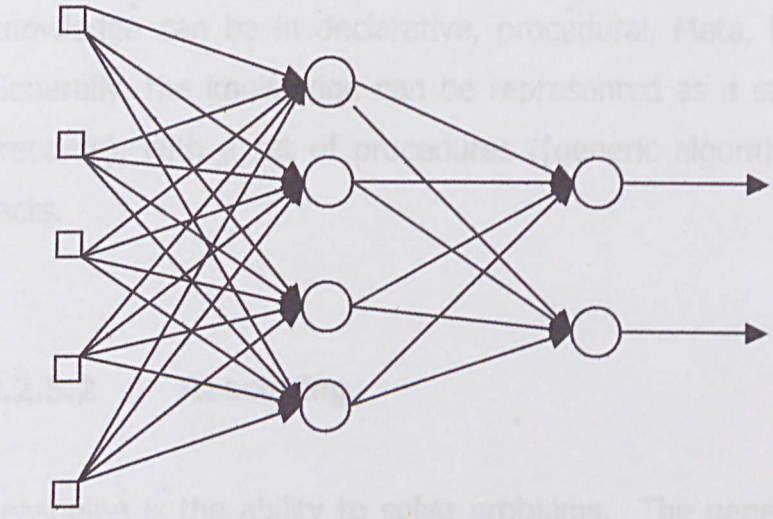


Figure 2.3: Fully connected feed forward network with hidden layer

2.2.5 Artificial Intelligence and Neural Networks

The goal of artificial intelligence (AI) is the development of algorithms that require machines to perform cognitive task, like human do.

An AI system must be capable of doing three things:

(a) Store knowledge (knowledge representation),

(b) Apply the knowledge to solve problem (reasoning), and

(c) Acquire new knowledge through experience (learning).

2.2.5.1 Knowledge Representation

"Knowledge" as used by AI researchers, is just another term for data. The knowledge can be in declarative, procedural, Meta, heuristic, structural [4]. Generally, the knowledge can be represented as a static collection of facts (records), with a set of procedures ([generic algorithm) to manipulate the facts.

2.2.5.2 Reasoning

Reasoning is the ability to solve problems. The general way of solving the problem is searching around the rules, data and control until a solution is enquired.

2.2.5.3 Learning

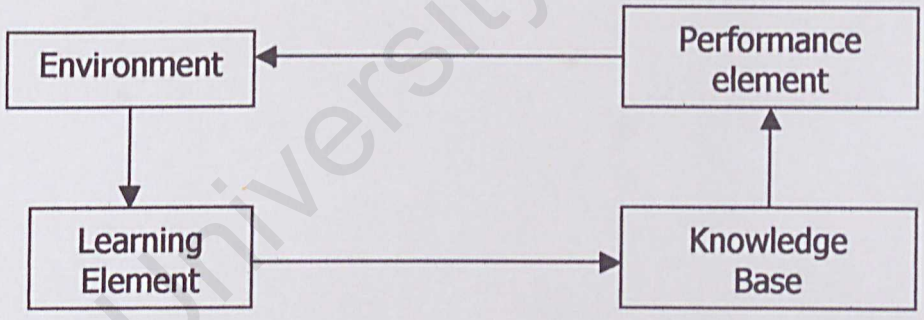


Figure 2.4: Simple model of machine learning

The environment supplies information to the learning element. The learning element use this information and add/modify the knowledge base, thus, the performance element can be improved.

Error Correction Learning

The error signal, e is derived from difference between the desired output (d) and the actual output (y), thus,

$$e = d - y$$

According to the delta rule, the adjustment of weight is,

$$\Delta w = \eta e x$$

where w = weight

η = positive constant, the learning rate

e = error signal

x = input value

The updated value of synaptic weight w is determined by

$$w(n + 1) = w(n) + \Delta w$$

where n as time step

Chapter III: Methodology

Methodology refers to the method used to develop the system. It included the project schedule, development tools, and system requirement.

3.1 Program Schedule

3.2 Development Requirement

- Hardware Requirement
- Development Tools
- Database Management System (DBMS)

3.2 Run-time requirements

- Disk Space Required
- System Requirements

3.1 Program Schedule

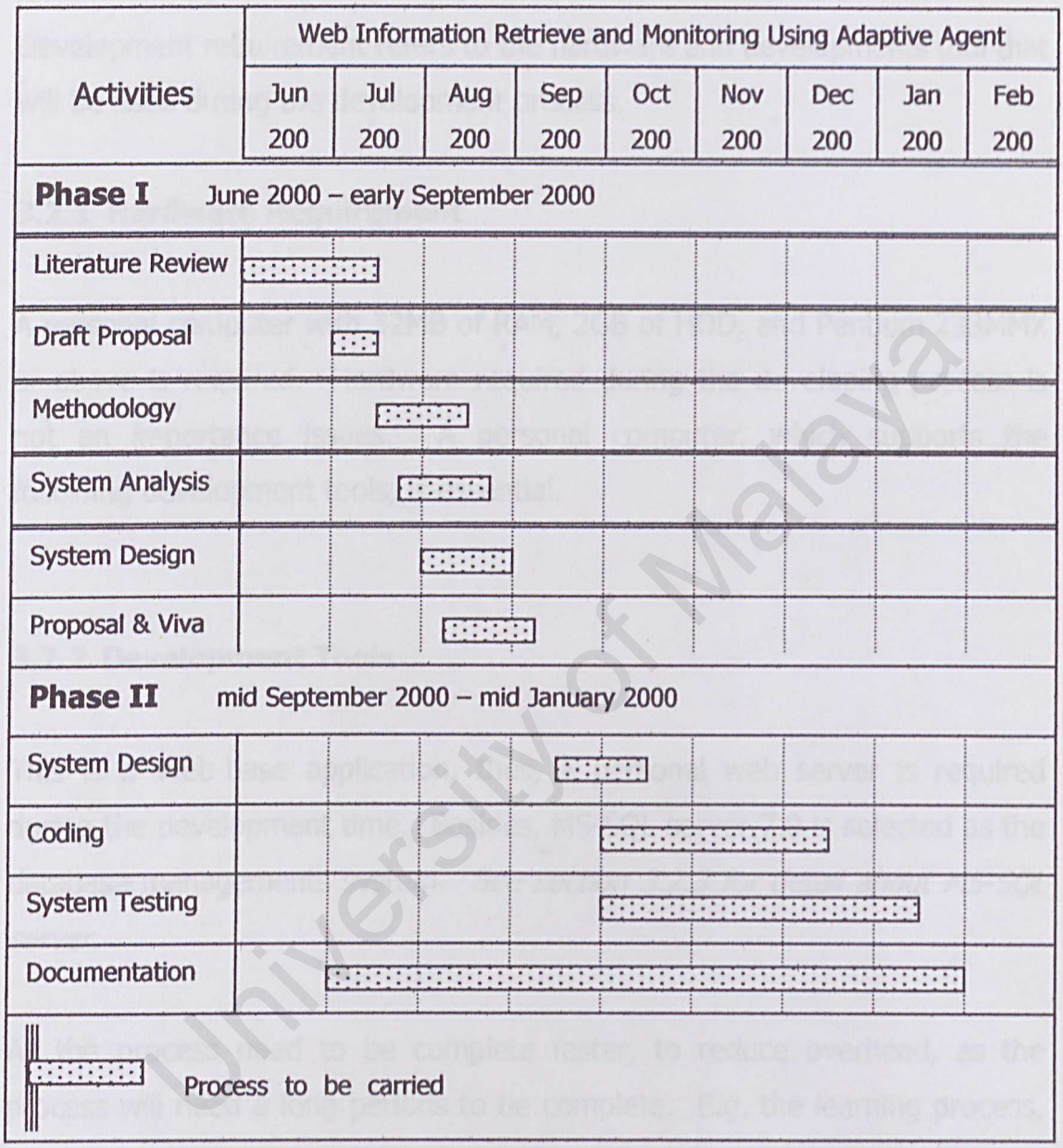


Figure 3.1: Project schedule

3.2 Development Requirement

Development requirement refers to the hardware and developments tool that will be used during the development process.

3.2.1 Hardware Requirement

A personal computer with 32MB of RAM, 2GB of HDD, and Pentium 233MMX or above is required. Hardware required during the developing process is not an importance issues. A personal computer, which supports the following development tools, is essential.

3.2.2 Development Tools

This is a Web-base application, thus, a personal web server is required during the development time. Besides, MS-SQL server 7.0 is selected as the database managements system. *See section 3.2.3 for detail about MS-SQL server.*

All the process need to be complete faster, to reduce overhead, as the process will need a long periods to be complete. E.g. the learning process, the system will need to analyze a lot of input to achieve the desired outputs. Thus, MS-Visual C++ is used, as C program is the fastest (after the assembly language) code can be generated in existence today.

For developing server-side scripting, MS-Visual Interdev will be used. Active Server Page (ASP) will be used for server-side scripting, as it wild range of component and co-operate with MS-Visual Basic. Visual Basic is a great tool to develop user-friendly interface, with the drag and drop feature.

For client-side scripting, JavaScript will be used. JavaScript is supported by the two major browser, Netscape Navigator, and Microsoft Internet Explorer. Meanwhile VBScript is only supported by Microsoft Internet Explorer.

3.2.3 Database Management System (DBMS)

MS-SQL server 7.0 is a powerful DBMS in the market. With the capability to manipulate the database up to 1,048,516 TB, and the access times below 4BG of database will not shows any great delay.

Large and fast database is need for indexing high volume of pages; therefore, the SQL server 7.0 is suitable as the DBMS. Besides, the DBMS support client-server architecture; two differences servers can be setup with one server supplying web-services and another one supplying database query services. Thus, help to boost the performance of the system.

Attribute	SQL Server 6.5	SQL Server 7.0
Database size	1 TB	1,048,516 TB
Page size	2 KB	8 KB
Files per database	32	32,767
File size (data)	32 GB	32 TB
File size (log)	32 GB	4 TB
Bytes per character or binary column	255	8,000
Bytes per GROUP BY, ORDER BY	900	8,060
Bytes per row	1,962	8,060
Columns per base table	250	1,024
Tables per SELECT statement	16	256

Table 3.1: Capacity between SQL Server 6.5 and SQL Server 7.0

3.3 Run-time requirements

Run time requirement need to be specified, as an importance issues, include hardware and software. The performance is greatly dependence on the hardware used.

3.3.1 Disk Space Required

Following is a brief calculation of disk space required for the system.

Description	Disk Space (MB)
Runtime Software OS and Runtime software	2,048.00
User Profile 650 users, 3MB per user quota	1,950.00
Keyword Indexing Assume 60,000 keyword is being indexes, each keyword required 56 bytes, space required = 60,000 * 56 / 1024^2	3.06
Assume every keyword contains 100 relations, each relation required 10 bytes, space required = 60,000 * 100 * 10 / 1024^2	57.22
Page Indexing Assume average space required for indexing, storing properties of a page take 2,048 bytes For indexing 2M pages, space required = 2M * 2048	3,906.25
Catching Space used for caching HTML documents, e.g. storing HTML page locally	100.00
Totals: Required disk space	8,064.53

Figure 3.2: Disk space required

3.3.2 System Requirements

- Windows NT server
- Web Server (IIS 4.0)
- Microsoft SQL server 7.0
- 8GB of Hard disk, more hard disk space required to support more user and indexing more pages
- 128MB RAM, 256MB recommended for higher performance
- Fast Internet connection

Chapter IV: System Design

4.1 Database Dictionary

- Member Profile
- Keyword
- Page Indexing

4.2 Neural Net Design

- Overview of Neural Net Design
- Input Layer
- Extracting Layer
- Processing Layer
- Database Administrator

4.1 Database Dictionary

The database tables used in the projects can be divided into three groups:

(a) Member Profile,

(b) Keyword, and

(c) Page Indexing.

See appendix B for supported data types by MS-SQL server 7.0.

4.1.1 Member Profile

Main table for member profile, contains an automatic generated member ID, member's email used as login ID, password which required when login to member's account, and the last login date of the user.

MEMBER	Data Type	Size	Description
*Member_ID	Int	4	Member ID
Email	Char[50]	50	Member's email, use as a login ID
Password	Char[20]	20	Login password
Last_Login	SmallDateTime	4	Last login date
Total: 4		78	

Registered member is allowed to bookmark a page. Bookmark is permanent for the user.

BOOKMARK	Data Type	Size	Description
Member_ID	Int	4	Member ID
Page_ID	Int	4	Page ID
Total: 2		8	

Search Session. Each member can have unlimited search session, but have a limitation of 3MB quote, included the bookmark, history and session.

Session	Data Type	Size	Description
*Session_ID	Int	4	Session ID
Member_ID	Int	4	Member ID
Login_IP	Int	4	IP of login session
Session_Date	SmallDateTime	4	Last revise date of the session
Total: 4		16	

History used to store pages that visited by the users.

HISTORY	Data Type	Size	Description
Session_ID	Int	4	Session ID
Page_ID	Int	4	Page ID
Date_Visited	SmallDateTime	4	Date of last visiting of the page
Total: 3		12	

Key list stored all the keyword entered by a session.

KEYLIST	Data Type	Size	Description

Session_ID	Int	4	Session ID
Keyword_ID	Int	4	Keyword ID
Total: 2		8	

4.1.2 Keyword

These databases are the knowledge base of the keyword. The structures is designed to allow the learner to manipulate the records.

Keyword, store the keyword name and a keyword ID is assigned automatically. The ID used as a reference to the key name, as all other table will refer to the ID and not storing the whole string, in order to save space.

The weight used to store the frequency in which the keyword was entered, selected, or searches the keyword.

KEYWORD	Data Type	Size	Description
Keyword_ID	Int	4	Keyword ID
KeyName	Char[50]	50	Keyword Name
SearchedCount	SmallInt	2	Frequency of keyword used by user.
IndexCount	SmallInt	2	Frequency of re-indexing the keyword
IndexLevel	TinyInt	1	Indicate this keyword is not yet indexed, already re-indexed, prepared for indexing, relevance

			analyzed, or it's a common word which will not being indexed.
Total: 3		59	

Related Keyword, for classification, relation between keyword is directed, while in the following table, relating is undirected.

RELATED	Data Type	Size	Description
Key ID	Int	4	Keyword ID
Sub Key ID	Int	4	Keyword ID
Weight	SmallInt	2	Frequency of occurrences
Total: 3		10	

Linked Keyword, used to store neural net linked items.

Linked	Data Type	Size	Description
ID	Int	4	Linked ID
Input ID	Int	4	Input keyword's ID
Output ID	Int	4	Output keyword's ID
Weight	Double	8	Connection Weight

4.1.3 Page Indexing

The last section of database used for indexing a web page.

Page, storing the major structure of a HTML document, each page being indexed should have a page ID. The URL of a page is breaking down into three pieces, file name, folder, and domain.

When a remote location (URL) is needed, the folder id will refer to the folder table, see section after this for more details in reconstruction of the URL. Meanwhile, when the locate location (HTML documents cache at local storage), folder ID (converted into string) is refer to a local folder, and file ID (converted into string) is the file name.

PAGE	Data Type	Size	Description
*Page ID	Int	4	Page ID
Folder ID	Int	4	Folder ID
File ID	Int	4	File ID
Last Update	SmallDateTime	4	Page last update date
Sizes	Int	4	Page's length
Last Revised	SmallDateTime	4	Last date that revise the page
Image	Bit	1/8	Page contain image
Audio	Bit	1/8	Page contain audio
MP3	Bit	1/8	Page contain MP3
Video	Bit	1/8	Page contain Video
Flash	Bit	1/8	Page contain flash
Java	Bit	1/8	Page contain Java Applet
JavaScript	Bit	1/8	Page contain Java Script
ActiveX	Bit	1/8	Page contain ActiveX object
EXE	Bit	1/8	Page contain executable file
Acrobat	Bit	1/8	Page contain Acrobat PDF document
VBScript	Bit	1/8	Page contain VBScript
Real Media	Bit	1/8	Page contain Real Audio/Video
PostScript	Bit	1/8	Page contain PostScript file
ZIP	Bit	1/8	Page contain downloadable

			compressed file
FORM	Bit	1/8	Page have form
Other	Bit	1/8	Contain undefined documents, e.g. MS-Office documents, etc.
Total: 22		26	

Hyperlink refer to the hyperlink between the pages. This table used in table of contents maker, which store all the hyperlink in a page.

HYPERLINK	Data Type	Size	Description
Page ID	Int	4	Page ID
Link Page	Int	4	Link to other pages
Total: 2		8	

Folder, is used to store remote URL information. If the folder name is equal to "/ROOT", then the Parent Folder ID is refer to domain name, else, the parent folder will refer back to the folder id.

This method used for save storage space, if there's more files inside a same folder, only the file name need a new record, while the folder name, domain name will not need another storage. Duplicated name will not be record differently.

FOLDER	Data Type	Size	Description
*Folder_ID	Int	4	Folder ID
Folder_Name	Char[40]	40	Folder Name
Parent_Folder	Int	4	Parent Folder (see rule)

Total: 3		48	

Domain, store a domain name, e.g. www.abc.com, www.ai.edu.

DOMAIN	Data Type	Size	Description
*DomainID	Int	4	Domain ID
DomainName	Char[50]	50	Domain Name (www.abc.com)
IndexPage	Int	4	Default/Index page ID
Total: 2		58	

File Name, storing file name; because most web pages have same name, e.g. index.html, default.asp. This table used to save storage space.

FILE	Data Type	Size	Description
*File_ID	Int	4	File ID
File_Name	Char[40]	40	File Name
Total: 2		44	

Page Indexing, all the keywords will be indexing against the pages.

INDEX	Data Type	Size	Description
Keyword_ID	Int	4	Keyword ID
Page_ID	Int	4	Page ID
Counter	TinyInt	1	Total of keyword found in the page
Relevance	TinyInt	1	Percent of relevance
Total: 2		9	

The following structure storing the all the title, subtitle or headers within the pages. This table used to support the table of contents features discuss earlier.

Title	Data Type	Size	Description
Page ID	Int	4	Page ID
Title	Char[40]	50	Page's title, subtitle or header
Total: 2		54	

4.2 Neural Net Design

4.2.1 Overview of Neural Net Design

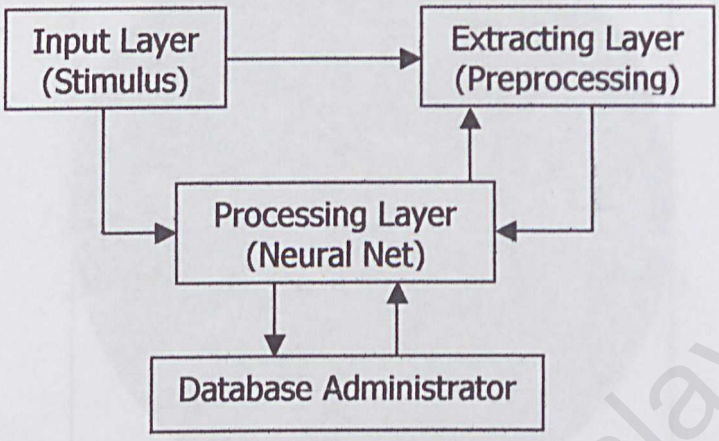


Figure 4.1: neural net design

4.2.2 Input Layer

The input layer is the stimulus for the neural network. There are three major sources of input:

(a) Administrators

(b) Users

(c) Internet Documents

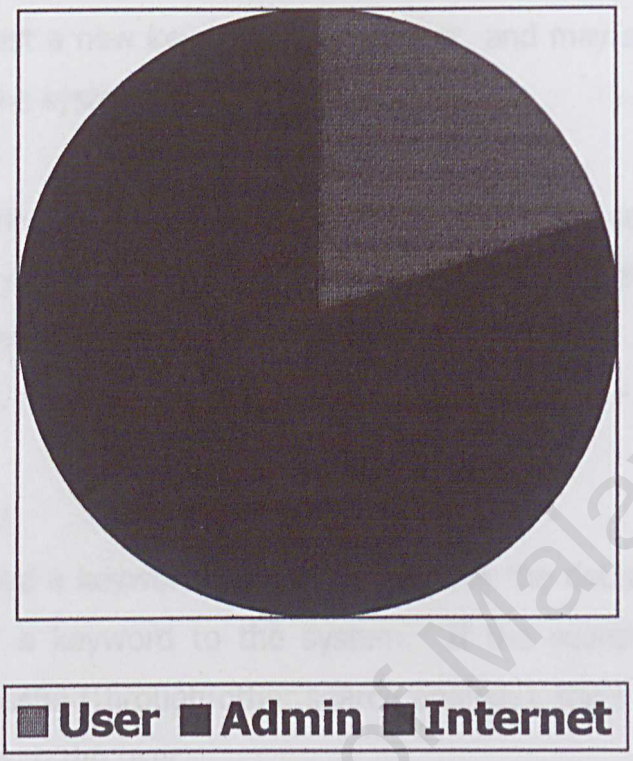


Figure 4.2: Sources of input layer

4.2.2.1 Administrators

Administrators include the developers and others system tester (if any). Administrators should only involved themselves during the development times.

Administrators will try to test the learning algorithms, by inputting the testing sets into the neural networks. Output results are being monitored.

4.2.2.2 Users

Users may suggest a new keyword to the agent, and may suggest keyword classification to the system.

The new keyword and classification should have higher accuracy, but the process is slow, and need a lot's of users for learn a correct keyword, and construct the relation between keyword.

Suggest Keyword

When user entered a keyword that is not found in the database, the user is said to 'suggest' a keyword to the system. If the searching against the keyword is successful (through other search engines), the system may learn a new keyword from the user.

Suggest first, second, and third level classification

The interface provide a browse able keyword for the user. When user select a topic, a further list of keyword will be provide to the user. All keyword at top of a classification considered as a topic, e.g. computer, automobile, education, etc., the top level will be pre-installed by the administrators. If user didn't found the keyword, they can type in the keyword, thus, a new keyword under the topic is suggested by the users.

The classification is said to be strong, if have certain amount of users selected the same topic, and key in the same keyword. When it's strong enough, the keyword will be listed out at the 2nd level list.

4.2.2.3 Internet Documents

The HTML documents downloaded from the Internet provide the largest learning space for the system. The Internet documents may provide all the words that can be indexed.

Keyword learning, relations among keyword being constructed, and all other learning process should be carefully designed so that the learning process can be success with minimal error.

4.2.3 Extracting Layer

Extracting layer act as the preprocessor to the neural network, will try to extract need information from the input layer.

The module included: Words counter & indexer, page properties scanner, title, header extractor, etc.

Words counter and indexer will collect the keyword that found in a page and store in database.

Page properties scanner will try to check if the page contains various information/object (e.g. Image, ActiveX, etc.)

Title, header extractor will try to look for possible title in the page.

4.2.4 Processing Layer

There are three major processes inside the processing layer:

(a) Learner,

(b) Re-Indexing,

(c) Relevant Analyzer.

4.2.4.1 Learner

The learner is used for learning a new keyword; construct relationship between keyword; classification of keyword; and learn phrases.

4.2.4.2 Keyword Learning

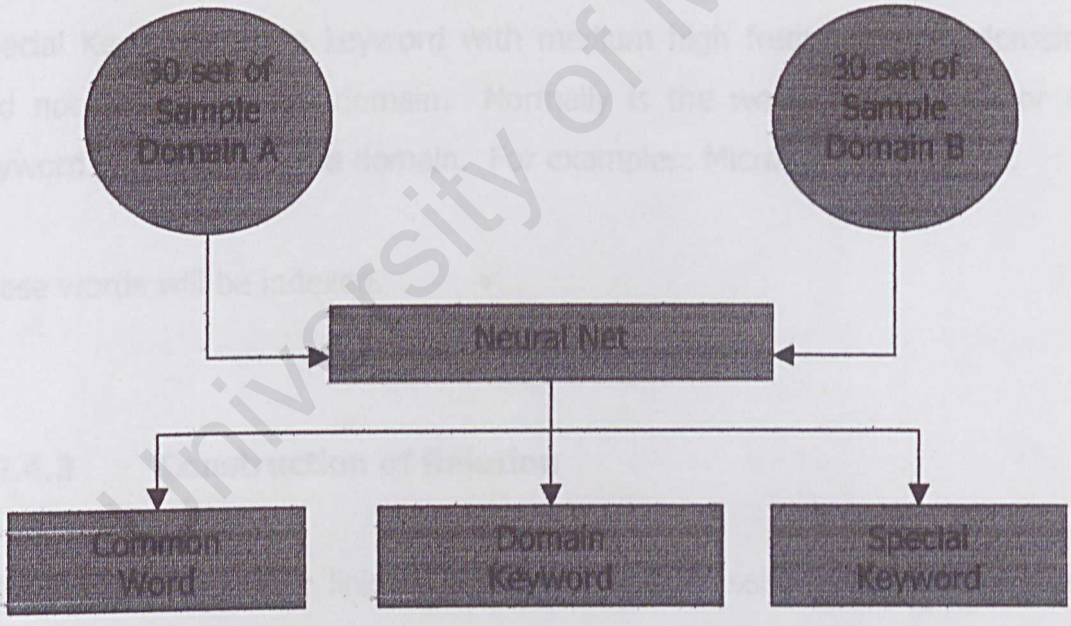


Figure 4.3: Keyword Learning

Common Words

Common words refer to the words that have very high frequency shared between difference domains. For example: the, of, a first, list, etc.

These words will not be indexed.

Domain Keyword

Domain keyword refer to keywords t hat have that have high frequency in the same domain but have low frequency shared words between domain. For example: neural network, cognitive, computer, hardware, software, etc.

These words will be indexed.

Special Keyword

Special Keyword is the keyword with medium high frequency in a domain and not found in other domain. Normally is the word with capital, or a keyword only found in the domain. For examples: Microsoft, Creative, ...

These words will be indexed.

4.2.4.3 Construction of Relation

Related keyword will be linked. The keywords are said related if keywords are located in same sentence.

The relation will aid the relevant analyzer, and suggest related keyword to the user.

4.2.4.4 Classification of Keyword

Related keyword will classify under same topic/title. When a title or header is found, the keyword under it is said to under the class of the title.

Allow user to have reference of related keyword under a topic/title.

4.2.4.5 Phrases Learning

If two or more keywords always be together (next to each other), it will 'remembered' as a phase. For examples: neural network.

This will allow suggestion of phase to user, and automatic phrase searching.

4.2.4.6 Spell Checker

The spell checker has various usages.

(a) Suggest correction if typo error, or when user key in incomplete keyword.

(b) Avoid indexing of similar keyword differently, in order to save indexing space.

The spell checker is limited to English only. Keyword in other languages may lead to unexpected outputs.

The spell checker has the following function:

(a) Singular and plural adjustment

(b) Abbreviations pattern matching

(c) Partial matching

(d) Typo error checking

Singular and Plural

Keyword will be stored in singular mode, and indexing should be done for the singular mode keyword. For example: computer will be indexed instead of computers. Thus, if a page has the word 'computer' and 'computers', only one entry in the indexing space will be needed.

Abbreviation Pattern Matching

Abbreviation pattern matching designed for avoiding indexing of keyword and it's abbreviation together. For example: int'l will match as international; lang will indexed as language.

Not all matching will be successful. If matching success, the indexer will index the completely original word, else the abbreviation is treated as a new or difference word.

-- Abbreviations Pattern

The abbreviation has some common pattern:

(a) Abbreviations of a word are always started with the first letter. (E.g. *fig* for *fig*ure.

(b) First letter taken for a phrase. (E.g. PM for *P*rime *M*nister).

(c) Leading letters is taken as abbreviation. (E.g. *math* for *math*ematics)

(d) Vowel is excluded. (E.g. *msg* for *m*e*s*sa*g*e).

Partial Matching

Partial matching is needed for:

(a) Detect difference between British English and American English. For example: color is same as colour, word that end with –ize is same as word that end with –ise.

(b) User entered incomplete keyword.

4.2.5 Database Administrator

Database Administrator will record results from the processing layer. Output is generated based on the databases.

Chapter V: Implementation

Overview

5.1.1 Database Classes

- CDataMngr
- Other Classes

5.1.2 Neural Net Classes

- Neuron
- Link
- Layer
- Neural Net

5.1.3 Object Classes

- The Spell Checker
- Tokenizer

5.1.4 Thread/Process Classes

- Extracting Thread
- Re-Indexing Thread
- Neural Net Training Thread
- Relevance Analysis Thread

5.1 Overview

There are four categories in the coding, each provide difference functionality.

a) Database Classes

-- Used to access the database, perform several data related functions.

b) Neural Net Classes

-- Used to simulate a neural network.

c) Object Classes

-- Included the spell checker, tokenizer.

d) Thread Classes

-- The processes.

5.1.1 Database Classes

Database Classes used to access to the database source, provided with ODBC.

5.1.1.1 CDataMngr

This class provides the connection of all other recordset to the ODBC source. In this project, login info in save in the registry, without encryption. The class will get the login info from the registry and establish the connection.

5.1.1.2 Other Classe

Every recordset is access by classes, thus, making manipulation of the record much easier.

5.1.2 Neural Net Classes

The neural net classes included the following object:

(a) CNeuron

(b) CLink

(c) CLayer, and

(d) CNeuralNet

5.1.2.1 Neuron

CNeuron Overview

CNeuron represent a node in the neural network. Each neuron has a unique ID. In this project, every keyword is considered as a neuron, thus, the ID of a neuron is actually the ID of the keyword.

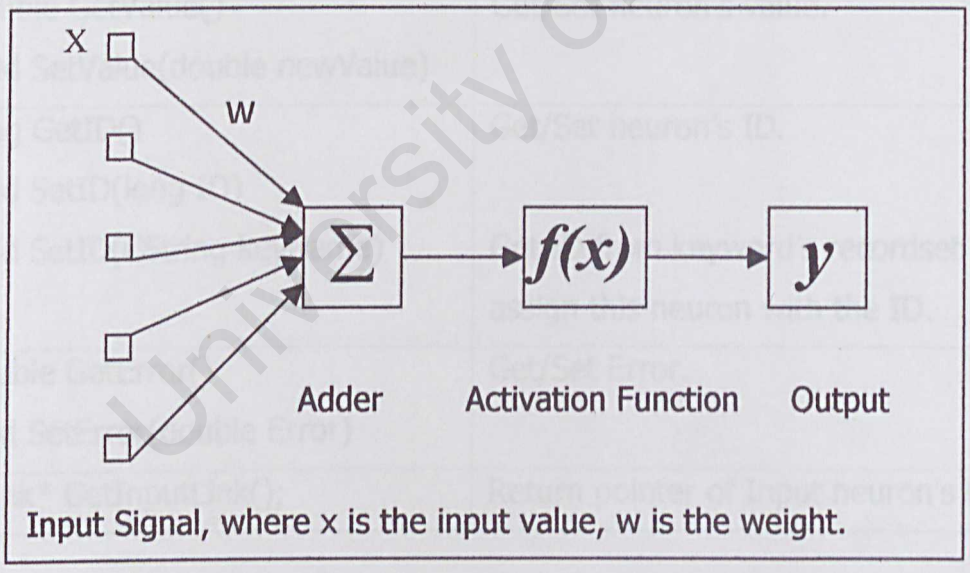


Figure 5.1: The structure of a neuron.

Class Member

Private Variable	
long m_ID	Each Neuron has a unique ID.

double m_Error	Error of this neuron.
double m_Value	Neuron's value.
CLink m_Input	Input Neurons.
Constructor	
CNeuron()	Construct an empty neuron, all private variable is set to 0.
CNeuron(long ID, double Value = 1.0)	Construct a neuron with known ID, default value is 1.0.
CNeuron(CString keyName)	Construct a neuron with a keyword, ID will be retrieve from database.
CNeuron(CTokenNode* pNode)	Construct a neuron from CTokenNode.
Get/Set Data	
double GetValue() void SetValue(double newValue)	Get/Set neuron's value.
long GetID() void SetID(long ID) void SetID(CString keyName)	Get/Set neuron's ID. Get ID from keyword's recordset and assign this neuron with the ID.
double GetError() void SetError(double Error)	Get/Set Error.
CLink* GetInputLink();	Return pointer of Input neuron's link.
Connection Method Provide connection operation	
void ConnectFrom(CNeuron &input)	Connect this neuron from another neuron.
void ConnectTo(CNeuron &output)	Connect this neuron to another neuron.

void SaveConnection()	Save Connection into database.
void RestoreConnection()	Restore Connection from database.
Neuron's Operation	
double Activiation(double value)	The activation function.
double Fire()	Fire this neuron.
void Learn(double LearningRate)	Learn from error with specified learning rate.

Firing the Neuron

(*see section 2.2.3 for information of firing an neuron*)

```
double CNeuron::Fire()
{
  int j = m_Input.GetSize();

  double sum = 0.0;

  for (int i = 0; i < j; i++)
  {
    sum += m_Input.GetWeightedValue(i);
  }

  m_Value = Activiation(sum);
  return m_Value;
}
```

m_Input is an instance of CLink. CLink::GetWeightedValued() will return the linked neuron's value multiply with the connection weight.

$$y = \sum_{1}^{i} x_i w_i$$

where y = output value

x = input value

w = synaptic weights

The Activation Function

```
double CNeuron::Activiation(double value)
{
  if ( value > 100 )
    return 100;

  else if (value < 0)
    return 0;

  else
    return value;
}
```

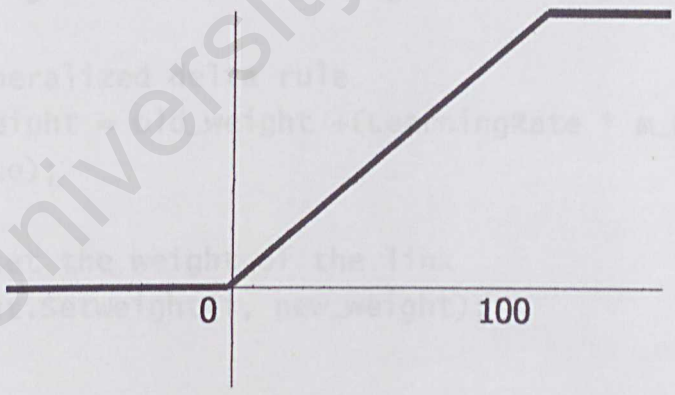


Figure 5.2: The Activation Function.

The Learning process

Learn will recalculate the weight of the input Link according to the learning rate, its own error and its input Neuron value using generalized delta rule. (*see page 28 for more information about the learning rule*)

```
void CNeuron::Learn(double LearningRate)
{
  int i, j = m_Input.GetSize();

  double input_value, old_weight, new_weight;

  // loop through each input link of the Neuron
  // reset the weights according to generalized
  // delta rule to reduce the error of this Neuron
  for (i = 0; i < j; i++)
  {
    // input Neuron value
    input_value = (m_Input.GetInputNeuron(i))
->GetValue();

    // original weight of the link
    old_weight = m_Input.GetWeight(i);

    // generalized delta rule
    new_weight = old_weight +(LearningRate * m_Error *
input_value);

    // reset the weight of the link
    m_Input.SetWeight(i, new_weight);
  }
}
```

5.1.2.2 Link

The Class CLink is derived from CPtrArray. CLink is a collection class, which collects the CLinkNode object.

CLinkNode Overview

CLinkNode contains the pointer to input neuron, output neuron and its connection weight.

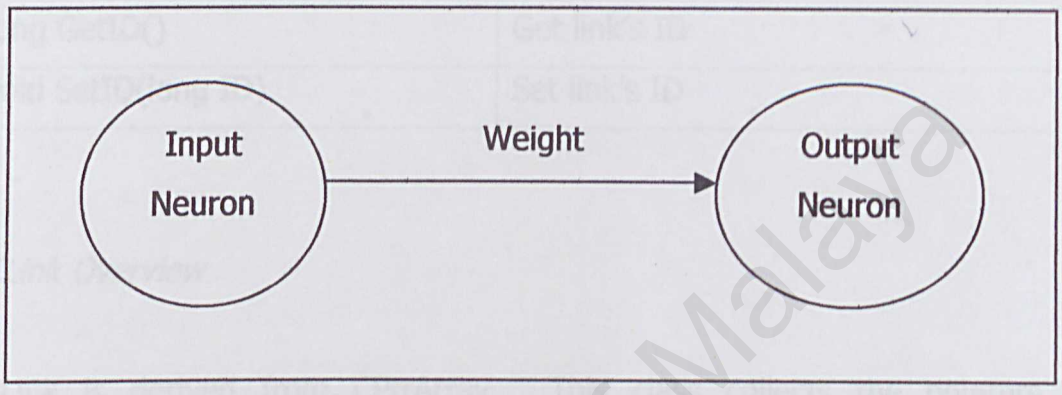


Figure 5.3: Connected Neuron.

Class Member

Private Variable	
long m_ID	Link's ID.
CNeuron *m_Input	Pointer to input neuron.
CNeuron *m_Output	Pointer to output neuron.
double m_Weight	Connection weight of this link.
Get/Set Value	
double GetOutputValue()	Get output neuron's value.
double GetInputValue()	Get input neuron's value.
double GetWeightedValue()	Return input value multiply with the weight.
double GetWeight()	Get weight.

Void SetWeight(double Weight)	Set weight.
CNeuron* GetInput()	Get pointer to input neuron.
CNeuron* GetOutput()	Get pointer to output neuron.
void SetInput(CNeuron* Input)	Set input neuron.
void SetOutput(CNeuron* Output)	Set output neuron.
long GetInputID()	Get input neuron's ID.
long GetOutputID()	Get output neuron's ID.
long GetID()	Get link's ID
void SetID(long ID)	Set link's ID

CLink Overview

CLink is derived from CPtrArray. This class collects the pointers of CLinkNode. Each neuron has a private member of CLink, which collects the entire input link to the neuron.

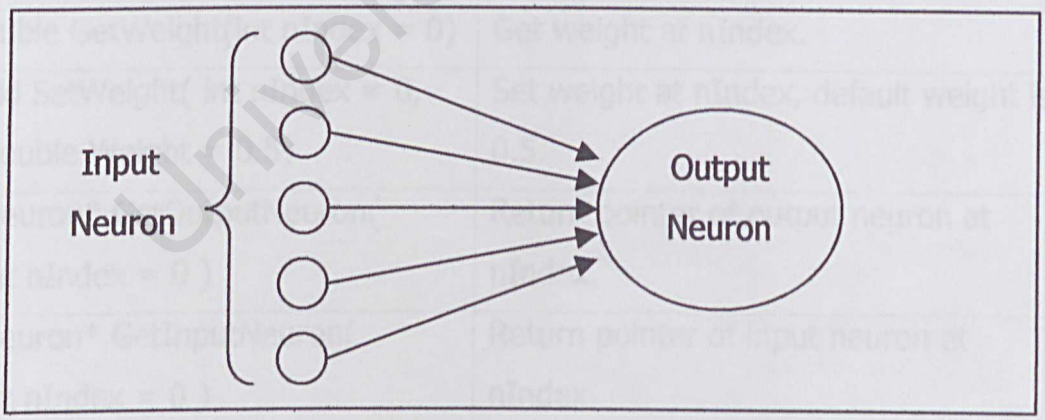


Figure 5.4: Collection of links.

Class Member

Note: nIndex is referred to the index of link node in the array.

Private Variable	
There's no private variable for this class. Variables are inherited.	
Constructor	
CLink()	Construct an empty link
CLink(CNeuron *Input, CNeuron *Output, double Weight = 0.0)	Construct a link and add one link node into the collection.
Get/Set Value	
long GetID(int nIndex = 0)	Get link's ID at nIndex.
double GetInputNeuronValue(int nIndex = 0)	Get input neuron's value at nIndex.
double GetWeightedValue(int nIndex = 0)	Get weighted value at nIndex.
double GetWeight(int nIndex = 0)	Get weight at nIndex.
void SetWeight(int nIndex = 0, double Weight = 0.5)	Set weight at nIndex, default weight is 0.5.
CNeuron* GetOutputNeuron(int nIndex = 0)	Return pointer of output neuron at nIndex.
CNeuron* GetInputNeuron(int nIndex = 0)	Return pointer of input neuron at nIndex.
long GetOutputNeuronID(int nIndex = 0)	Get output neuron's ID.
long GetInputNeuronID(int nIndex = 0)	Get input neuron's ID.
CLinkNode* GetAt(int nIndex)	Return link node at nIndex.

Connection Method	
void Connect(CNeuron *Input, CNeuron *Output, double Weight = 0.0)	Add a new connection/link between two neurons.
void Disconnect(CNeuron *Input, CNeuron *Output)	Remove connection of a link.
void RestoreWeight()	Restore weight from database.
void SaveConnection()	Save weight and connection to database.
Other	
void Destroy()	Destroy this object.

5.1.2.3 Layer

CLayer Overview

The CLayer class is derived from CPtrArray. CLayer stored the array of neurons.

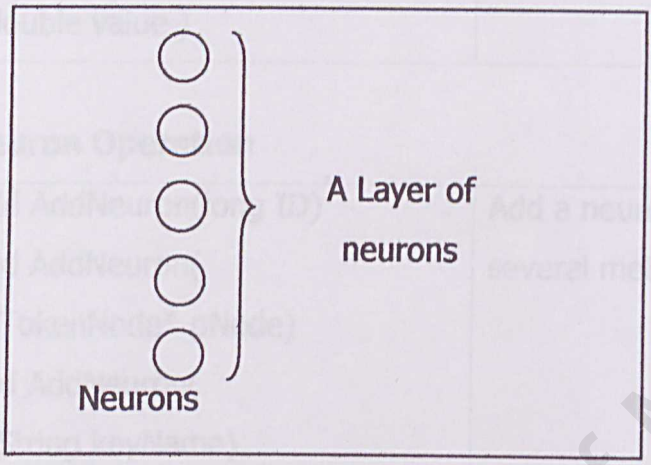


Figure 5.5: A Layer of neurons.

Class Member

Note: nIndex is referred to the index of neuron in the array.

Private Variable	
There's no private variable for this class.	
Constructor	
CLayer()	Construct an empty layer.
Get/Set Value	
CNeuron* GetAt(int nIndex = 0)	Return pointer of neuron at nIndex.

void SetError(int nIndex, double Error)	Set error of neuron at nIndex.
double GetNeuronValue(int nIndex = 0)	Get neuron's value at nIndex.
void SetNeuronValue(int nIndex, double value)	Set neuron's value at nIndex.
Neuron Operation	
void AddNeuron(long ID) void AddNeuron(CTokenNode* pNode) void AddNeuron(CString keyName) int AddNeuron(CNeuron *pNeuron)	Add a neuron into this layer using several method.
void Fire()	Fire all neurons in this layer.
void Learn(double Learning_Rate)	Make all neurons in this layer learn from error.
void LinkAllTo(CLayer *toLayer)	Link all neurons in this layer to another layer. Use to create a fully connected neural network.
void SaveConnection()	Save connection into database.
void RestoreConnection()	Restore connection from database.
void Destroy()	Destroy this layer.

To fire the layer, each neuron is the layer is fired.

```
void CLayer::Fire()
{
  int i, j = GetSize();
```

```
    CNeuron* pNeuron;

    for (i = 0; i < j; i++)
    {
      pNeuron = GetAt(i);
      ASSERT( pNeuron != NULL );

      pNeuron->Fire();
    }
  }
```

5.1.2.4 Neural Net

CNeuralNet Overview

The CNeuralNet class is derived for CPtrArray and act as a collector of CLayer object.

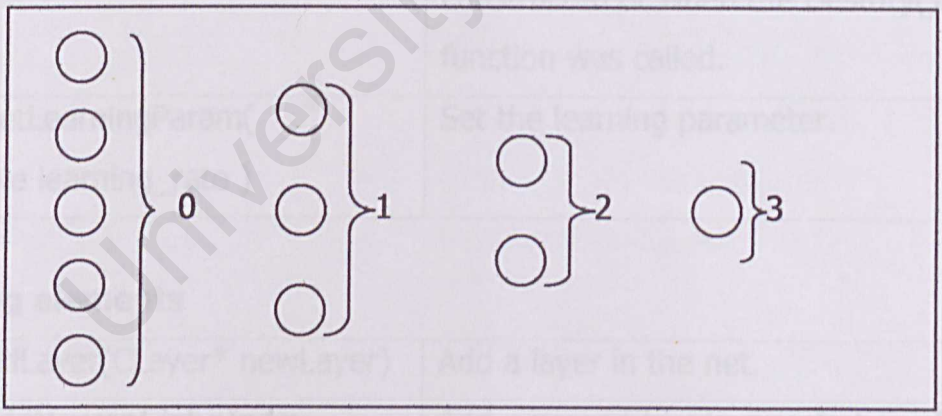


Figure 5.6: Layers of neurons, indexes are zero-based in C++.

Class Member

Private Variable	
double m_LearnParam	Learning Parameter
Constructor	
CNeuralNet()	Construct an empty neural net.
Get/Set Value	
CLayer* GetAt(int nIndex = 0)	Return pointer of a layer at nIndex.
double GetOutputValue(int nIndex = 0)	Get Output Value.
void SetInputValue(int nIndex, double value)	Set input value.
void MakeInputZero()	Set all value in input layer to 0.
void SetLayerSize(int numLayer)	Set number of layer in the net. This function should be call after the net was constructed or when the Destory() function was called.
void SetLearningParam(double learning_rate)	Set the learning parameter.
Adding elements	
int AddLayer(CLayer* newLayer)	Add a layer in the net.
void AddNeuron(int nIndex, CNeuron *pNeuron)	Add a neuron in a layer, nIndex refer to which layer to add the neuron.
void AddInput(CString keyName) void AddInput(CTokenNode*pNode) void AddInput(long InputID)	Add a neuron in the input layer using several ways.

void AddOutput(CString keyName) void AddOutput(CTokenNode*pNode) void AddOutput(long OutputID)	Add a neuron in the output layer using several ways.
Neural Net Operation	
void Fire()	Fire the entire neural net.
void Learn()	Make the net to learn.
void InterConnect()	Build a fully connected network.
void SetDesiredOutput(int nIndex, double DesiredOutput)	Set desired output.
Other	
int FindInputIndex(long NeuronID)	Find a neuron's current index in the input layer using neuron's ID.
void DeleteConnection()	Delete the connection that was saved into database.
void SaveConnection()	Save the connection into database.
void RestoreConnection()	Restore connecting weight for every connection in the network.
void Destroy()	Destroy the network.

Input and Output

The zero index of layer is considered as input layer and the last layer as output layer.

Set Desired Output

This function will calculate the error of the output Neuron according to the desired value. The network should be fired first before calling this function.

```
void CNeuralNet::SetDesiredOutput(int nIndex,
  double DesiredOutput)
{
  CLayer *pLayer = GetAt( GetUpperBound() );

  double error = DesiredOutput -
    pLayer->GetNeuronValue(nIndex);

  // Set the error of the output Neuron to computed error
  pLayer->SetError(nIndex, error);
}
```

This function will fire the entire network. Since the first layer (zero indexed) is the input layer, thus, this function will fire from the second layer until the last layer.

```
void CNeuralNet::Fire()
{
  int i, j = GetSize();

  CLayer* pLayer;

  for (i = 1; i < j; i++)
  {
    pLayer = GetAt(i);
    pLayer->Fire();
  }
}
```

5.1.3 Object Classes

5.1.3.1 The Spell Checker

CSpellChecker Overview

The Spell Checker is used to adjust a plural keyword into singular, thus, to avoid duplicated indexes of the keyword.

Rules for Forming Plural Nouns.

Rule 1: Nouns are regularly made plural by addition of –s:

----- day days, roof roofs, shoe shoes

This function try to remove the –s from given string, return true if succeed.

```
BOOL CSpellChecker::MakeSingularA(CString &s)
{
  if ( s.Right(1) == "s" )
  {
    s = s.Left(s.GetLength() - 1);
    return true;
  }

  else
    return false;
}
```

Rule 2: Nouns ending in sibilant ('s') sounds spelled with *s*, *ch*, *sh*, and *x*:

----- bus buses, box boxes, church churches;

and some of the nouns ending with *o*:

----- buffalo buffaloes, mango mangoes

This function try to remove the –es from given string.

```
BOOL CSpellChecker::MakeSingularB(CString &s)
{
  if ( s.Right(2) == "es" )
  {
    s = s.Left(s.GetLength() - 2);
    return true;
  }

  else
    return false;
}
```

Rule 3: Nouns ending with y changed to –ies

----- baby babies

This function try to replace ending –ies with y.

```
BOOL CSpellChecker::MakeSingularC(CString &s)
{
  if ( s.Right(3) == "ies" )
  {
    s = s.Left(s.GetLength() - 3) + "y";
    return true;
  }

  else
    return false;
}
```

Rule 4: Nouns ending with f, f is changed to ves

----- leaf leaves, thief thieves

This function try to replace ending –ves with f:

```
BOOL CSpellChecker::MakeSingularD(CString &s)
{
  if ( s.Right(3) == "ves" )
  {
```

```
    s = s.Left(s.GetLength() - 3) + "f";
    return true;
  }

  else
    return false;
}
```

This function will try to detect the given keyword is plural of another keyword.

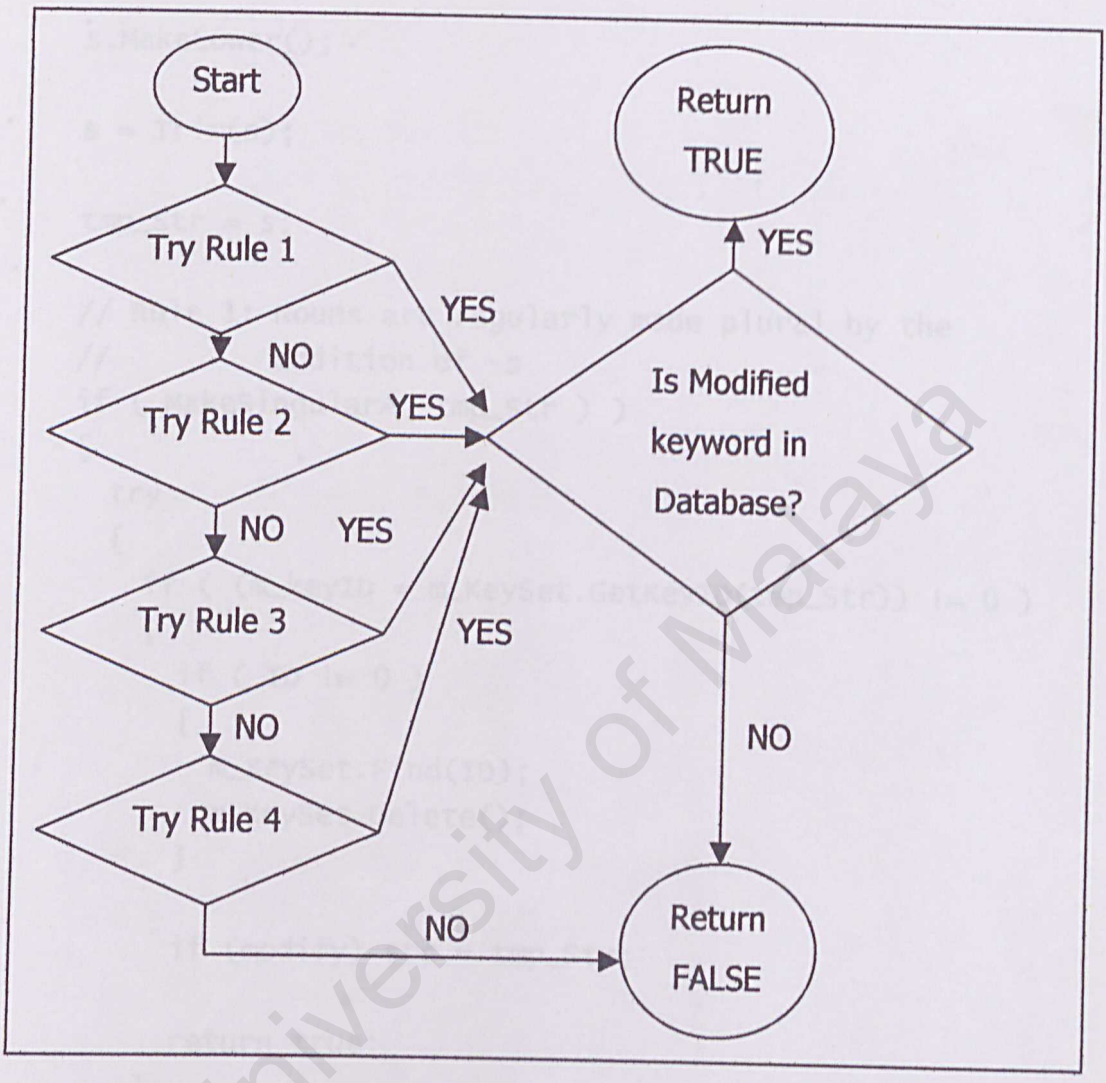


Figure 5.7: CSpellChecker::IsPlural().

Since there are many exceptions in forming plural nouns, a generalized algorithm to detect plural keyword is impossible. This function is only designed to detect plural with ending –s. Other type of plural will not be processed and will be considered as difference word. For example: children, tooth teeth, foot feet, policemen will be treat as difference word.

```
BOOL CSpellChecker::IsPlural(CString &src, BOOL modify)
{
  CKeywordSet m_KeySet(m_dataConn);
```

```
CString  tmp_Str, s = src;
long     ID = m_KeySet.GetKeyID(src);

s.MakeLower();

s = Trim(s);

tmp_Str = s;

// Rule 1: Nouns are regularly made plural by the
//         addition of -s
if ( MakeSingularA( tmp_Str ) )
{
  try
  {
    if ( (m_keyID = m_KeySet.GetKeyID(tmp_Str)) != 0 )
    {
      if ( ID != 0 )
      {
        m_KeySet.Find(ID);
        m_KeySet.Delete();
      }

      if (modify) src = tmp_Str;

      return true;
    }

  } catch (CUserException *e)
  {
    e->Delete();
  }
}

tmp_Str = s;

// Rule 2: Nouns ending with s, ch, sh, and x
if ( MakeSingularB( tmp_Str ) )
{
  try
```

```
    {
      if ( (m_keyID = m_KeySet.GetKeyID(tmp_Str)) != 0 )
      {
        if ( ID != 0 )
        {
          m_KeySet.Find(ID);
          m_KeySet.Delete();
          m_KeySet.Update();
        }

        if (modify) src = s;
        return true;
      }
    } catch (CUserException *e)
    {
      e->Delete();
    }
}

tmp_Str = s;

// Rule 3: Nouns ending with y, changed to -ies
if ( MakeSingularC( tmp_Str ) )
{
  try
  {
    if ( (m_keyID = m_KeySet.GetKeyID(tmp_Str)) != 0 )
    {
      if ( ID != 0 )
      {
        m_KeySet.Find(ID);
        m_KeySet.Delete();
        m_KeySet.Update();
      }

      if (modify) src = tmp_Str;
      return true;
    }
  } catch (CUserException *e)
  {
```

```
        e->Delete();
      }
    }

    tmp_Str = s;

    // Rule 4: Nouns ending with f: changed to ves
    if ( MakeSingularD( tmp_Str ) )
    {
      try
      {
        if ( (m_keyID = m_KeySet.GetKeyID(tmp_Str)) != 0 )
        {
          if ( ID != 0 )
          {
            m_KeySet.Find(ID);
            m_KeySet.Delete();
            m_KeySet.Update();
          }

          if (modify) src = tmp_Str;
          return true;
        }
      } catch (CUserException *e)
      {
        e->Delete();
      }
    }

    return false;
}
```

5.1.3.2 Tokenizer

Overview

To process the HTML documents, the text has to be break into tokens for indexing. A token is a single word in the document. The standard built-in library *strtok()* is not suitable for processing the HTML documents, because the HTML documents contains HTML tag which need to treat differently with normal text.

CTokenNode Overview

Private Variable	
long m_ID	Token's ID, same as keyword ID.
CString m_Token	The keyword.
Int m_Count	Number of keyword found.
unsigned long m_Type	Token type.
unsigned long m_Extended	Extended token type.
Token Type (defined)	

```
#define TOKEN_NULL          0x0000
#define TOKEN_UNDEFINE      0x0000
#define TOKEN_WORD          0x0001
#define TOKEN_NUMBER        0x0002
#define TOKEN_DATE          0x0004
#define TOKEN_TIME          0x0008

#define TOKEN_HTMLTAG       0x0010
#define TOKEN_HYPERLINK     0x0020
#define TOKEN_EMAIL         0x0040

#define NOT_QUOTED          0x0000
#define OPEN_QUOTE          0x0080
#define CLOSE_QUOTE         0x0100

#define TOKEN_QUOTE         OPEN_QUOTE
#define TOKEN_QUOTE_END     CLOSE_QUOTE
```

Extended Status (defined)

```
#define HTML_NOT_A_TAG      0x8000
#define HTML_NULL           0x0000
#define HTML_UNDEFINE       0x0000
#define HTML_DOC            0x0001
#define HTML_HEAD           0x0002
#define HTML_TITLE          0x0004
#define HTML_BODY           0x0008
#define HTML_SCRIPT         0x0010
#define HTML_STYLE          0x0020

#define HTML_BOLD           0x0040
#define HTML_ITALIC         0x0080
#define HTML_UNDERLINE      0x0100

#define HTML_BEGIN_BLOCK    0x0200
#define HTML_END_BLOCK      0x0400

#define HTML_TABLE          0x0800
```

Bit Operations

```
#define SetStatusBit(target, value)   target |= value
#define ResetStatusBit(target, value) target &= ~value
#define GetStatusBit(target, value)   (target & value)
```

Bit operations are common in c/c++ programming. For an integer (16 bits) or a long integer (32 bits) can act as a "flag" to indicate the state on or off.

0x0001 represent a hexadecimal number in c/c++,
0x0000 = 000000000000000 (binary), 0x0001 = 000000000000001 (binary)
0x0002 = 000000000000010 (binary), 0x0004 = 000000000000100 (binary)
and so on.

To set a status bit:

Original Integer (binary)	0000 0000 0000 0000
Predefined Value	0000 0000 0000 0001
OR Operation	0000 0000 0000 0001

To read a status bit, non-zero result indicate that the bit was set.

Original Integer (binary)	0000 0000 0000 0001
Predefined Value	0000 0000 0000 0001
AND Operation	0000 0000 0000 0001

To reset a status bit,

Original Integer (binary)	0000 0000 0000 0001
One's Complement	1111 1111 1111 1110
Predefined Value	0000 0000 0000 0001
AND Operation (Last bit is reset)	0000 0000 0000 0000

Constructor	
CTokenNode()	Construct an empty token node.
CTokenNode(CString keyName, long Type = 0, long Extended = 0)	Construct a node with known keyword.
CTokenNode(CTokenNode &tokNode)	Copy constructor
Get/Set Value	
BOOL GetExtended(long Status) void ResetExtended(long Status) void SetExtended(long Status)	Get/Set/Reset Extended status.
BOOL GetStatus(long Status) void SetStatus(long Status) void ResetStatus(long Status)	Get/Set/Reset token type.
void DecreaseCount() void IncreaseCount(int nRate)	Increase or Decrease the counter.
int GetCount() void SetCount(int nCount = 1)	Get/Set the counter.
long GetID() void GetKeywordID() void SetID(long ID = 0) void SetID(CString keyName)	Get/Set keyword ID.
CString GetValue() void SetValue(CString keyName)	Get/Set keyword.
Other Method	
void CheckTokenType()	Check token type.

BOOL IsWholeWord()	Return true if the token only has alphabet.
BOOL IsEndOfSentense(CString &s)	Return true and remove ending dot indicate that is end of sentense.
int IsQuote(CString &s)	Return quoting status (opening or closing) and remove the quote. Return 0 indicate that is no quote.

Check Token Type

The function try to detect the token is either HTML tag (which opened by '<' and closed by '>'), an email address (which has '@' and '.' Character), a hyperlink (which has http://), a number or a word. This function also able to detect if the word is quoted, or is the last word in a sentense.

```
// Check token type
void CTokenNode::CheckTokenType()
{
  int  tmp_find1, tmp_find2;
  long Status;

  // This is a HTML tag
  if (m_Token.Left(1) == "<" && m_Token.Right(1) == ">")
  {
    SetStatus( TOKEN_HTMLTAG );
  }

  else
  {
    // Search for email
    tmp_find1 = m_Token.Find('@');
    tmp_find2 = m_Token.Find('.');

    // This is an email
    if ( (tmp_find1 != -1)
     &&  (m_Token.ReverseFind('@') == tmp_find1)
```

```
    && (m_Token.ReverseFind('.') > tmp_find1) )
  {
    SetStatus( TOKEN_EMAIL );
  }

  // This is an hyperlink
  else if ( (tmp_find2 != -1) &&
          (m_Token.Find("://") != -1) )
  {
    SetStatus( TOKEN_HYPERLINK );
  }

  else
  {
    BOOL done = false;

    while ( !done )
    {
      done = true;

      if ( (Status = IsQuote(m_Token)) )
      {
        done = false;
        SetStatus( Status );
      }

      else if ( IsEndOfSentense(m_Token) )
      {
        done = false;
        SetStatus( TOKEN_ENDOFSENTENSE );
      }

      if ( m_Token.Right(2) == "'s" )
        m_Token.Delete( m_Token.GetLength() - 2, 2 );

      // This is a word
      if ( isalpha( m_Token.GetAt(0) ) )
      {
        SetStatus ( TOKEN_WORD );
      }
```

```
            // This is a number
            else if ( isdigit( m_Token.GetAt(0) ) )
            {
              SetStatus( TOKEN_NUMBER );
            }
          }
        }
      }
    }
```

Some word may surround by two or more quote, for example, a token might be like this:

("My Program").

Thus, a loop is needed to remove the beginning bracket and double quote, or removing the ending dot, closing bracket and the double quote.

CToken, the collection class

CToken derived from CPtrArray, is a collection class of CTokenNode. This class provide routine to manipulate the token node, generating token, searching within the array, and indexing.

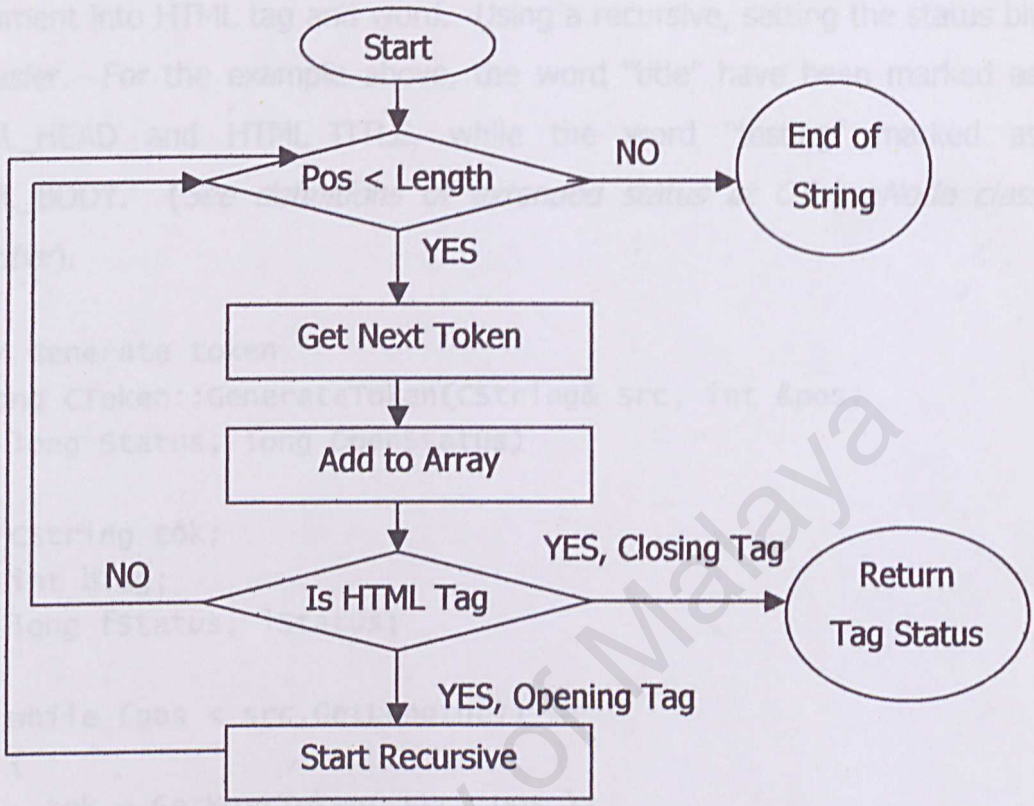
Private Variables	
CString m_whitespace	Define the white space. A white space like a space, a tab, a carry return is used as separator between two words.
Constructor	
void Init()	Initialize the default white space.
CToken()	Construct an empty token array

CToken(CString s)	Generate token array from the string.
Get/Set Value	
void AddToken(CString tok, long Type = 0, long Extended = 0) void AddToken(int ID int nCount = 1)	Add token node.
CTokenNode* GetAt(int nIndex = 0)	Return pointer to token node at nIndex.
CString GetValue(int nIndex = 0)	Get token's value at nIndex.
void IncreaseCount(int nIndex, int nRate)	Increase count of a token at nIndex.
BOOL GetExtended(int nIndex, long Status) void SetExtended(int nIndex, long Status) void ResetExtended(int nIndex, long Status)	Get/Set/Reset extended status of token at nIndex.
BOOL GetStatus(int nIndex, long Status) void SetStatus(int nIndex, long Status) void ResetStatus(int nIndex, long Status)	Get/Set/Reset token status at nIndex.
int GetCount(int nIndex = 0)	Get token's counter.
void GetKeywordID(int nIndex)	Get keyword's ID from database of token at nIndex.
void GetAllKeywordID()	Get all keyword's ID of all tokens.

Generating Token	
long GenerateToken(CString& src) long GenerateToken(CString& src, int &pos, long Status = 0, long BackStatus = 0)	Generate tokens array from source.
CString GetNextToken(CString &src, int &pos)	Get next token starting from position "pos".
BOOL ConvertSpecialChar(CString &s, int StartIndex)	Convert special character in HTML documents, which starting with '&' and end with ';'. For example: for white space.
Indexing	
void Swap(int nIndexA, int nIndexB) void Swap(CTokenNode* pLeft, CTokenNode* pRight)	Swap two item in the array.
void SortByCountDesc()	Sort the array by counter.
void Indexing()	Indexing, remove HTML tag and duplicated keyword.
Other	

int Find(int ID) int Find(CString tok, int StartIndex = 0, int StopIndex = -1) int FindTag(CString Tag, int StartIndex = 0, int StopIndex = -1)	Find index of a token using several ways.
int Count(CString tok)	Count frequency of a keyword.
int IsHTMLTag(CString tag)	Detect if the token is not a HTML tag, is opening tag, or closing tag.
long ConvertTagToStatus(CString tag)	Convert tag to predefined status, i.e. convert <TABLE> to HTML_TABLE.
void Destroy()	Destroy this object
void CheckTokenType(int nIndex)	Check token type at nIndex.
BOOL IsWhiteSpace(const char c)	Is White Spcace ?
BOOL IsEmpty()	Is Empty array?

Generate Tokens



Example: the source is

```
0         1         2         3         4         5
012345678901234567890123456789012345678901234567890123456789012
<HEAD><TITLE>Title</TITLE></HEAD><BODY>Testing</BODY>
```

Current Pos	Tag Status	Back Status	Recursive Count
0	HTML_NULL	HTML_NULL	0
6	HTML_HEAD	HTML_HEAD	1 (Start recursive)
13	HTML_TITLE	HTML_TITLE	2 (Start recursive)
18	Title	HTML_TITLE	2
26	HTML_TITLE	HTML_HEAD	1 (Returned)
33	HTML_HEAD	HTML_NULL	0 (Returned)
39	HTML_BODY	HTML_BODY	1 (Start recursive)
46	Testing	HTML_BODY	1
53	HTML_BODY	HTML_NULL	0 (Returned)

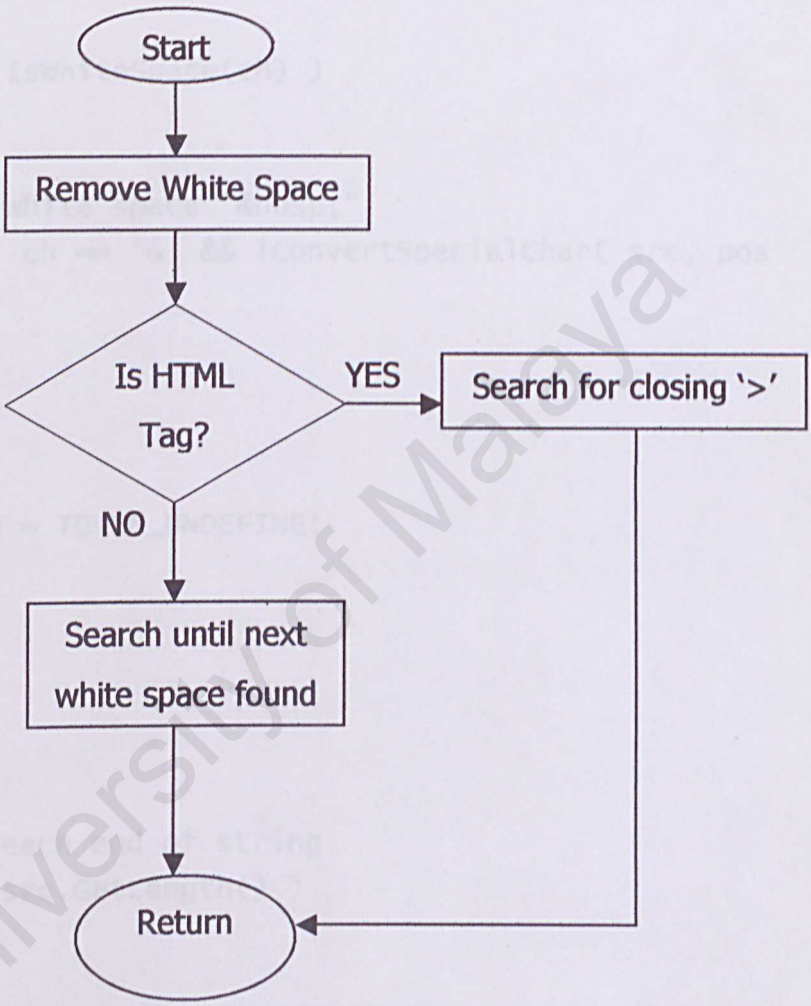
This routine search through the HTML documents, breaking the entire document into HTML tag and word. Using a recursive, setting the status bit is easier. For the example above, the word "title" have been marked as HTML_HEAD and HTML_TITLE, while the word "testing" marked as HTML_BODY. (*See definitions of extended status at CTokenNode class member*).

```
// Generate token
long CToken::GenerateToken(CString& src, int &pos,
  long Status, long OpenStatus)
{
  CString tok;
  int bTag;
  long fStatus, lStatus;

  while (pos < src.GetLength())
  {
    tok = GetNextToken( src, pos );

    if (tok == "") break;

    bTag = IsHTMLTag( tok );

    if ( bTag == OPEN_TAG )
    {
      AddToken(tok,0,Status);

      fStatus = ConvertTagToStatus( tok );

      if (fStatus != HTML_NULL)
      {
        SetStatusBit(Status, fStatus);

        lStatus = GenerateToken(src, pos, Status,
fStatus);

        if ( OpenStatus != HTML_NULL && lStatus !=
fStatus )
```

```
            return lStatus;

          else
            ResetStatusBit(Status, fStatus);
        }
      }

      else if (bTag == CLOSE_TAG)
      {
        AddToken(tok,0,Status);

        fStatus = ConvertTagToStatus( tok );

        if (fStatus != HTML_NULL)
          return fStatus;
      }

      else
      {
        AddToken(tok,0,Status);
      }
    }

    return OpenStatus;
} // end function
```

CToken::GetNextToken

This function used to search for next token in the source.




```
CString CToken::GetNextToken(CString &src, int &pos)
{
  char ch;
  int  i = 0;
  long tok_type = 0;

  // Remove white space
  while ( pos < src.GetLength() )
  {
    ch = src[pos];
```

```
    if ( ch == '<' )
    {
      tok_type = TOKEN_HTMLTAG;
      break;
    }

    else if ( IsWhiteSpace(ch) )
      pos++;

    // Remove white space " "
    else if (  ch == '&' && !ConvertSpecialChar( src, pos
) )
      pos++;

    else
    {
      tok_type = TOKEN_UNDEFINE;
      break;
    }
  };

  CString tmp;

  // Stop if reach end of string
  if ( pos >= src.GetLength() )
    return "";

  // Search for closing '>' for html tag
  if (tok_type == TOKEN_HTMLTAG)
  {
    i = src.Find('>', pos);

    if (i == -1)
    {
      tmp = src.Mid(pos);
      pos = src.GetLength();
      return tmp;
    }

    else
```

```
    {
      tmp = src.Mid(pos, ++i - pos);
      pos = i;
      return tmp;
    }
  }

  i = 1;
  ch = src[pos + i];

  // Search until white space, opening html tag '<' or
end of string
  while ( !IsWhiteSpace(ch) && ch != '<' && (pos + ++i) <
src.GetLength())
  {
    ch = src[pos + i];

    if ( ch == '&' && ConvertSpecialChar(src,pos+i) )
      ch = src[pos + i];
  }

  tmp = src.Mid(pos, i);
  pos += i;
  return tmp;
}
```

5.1.4 Thread/Process Classes

To avoid taking too much of processor resource, all threads are running at the lowest piority.

5.1.4.1 Extracting Thread

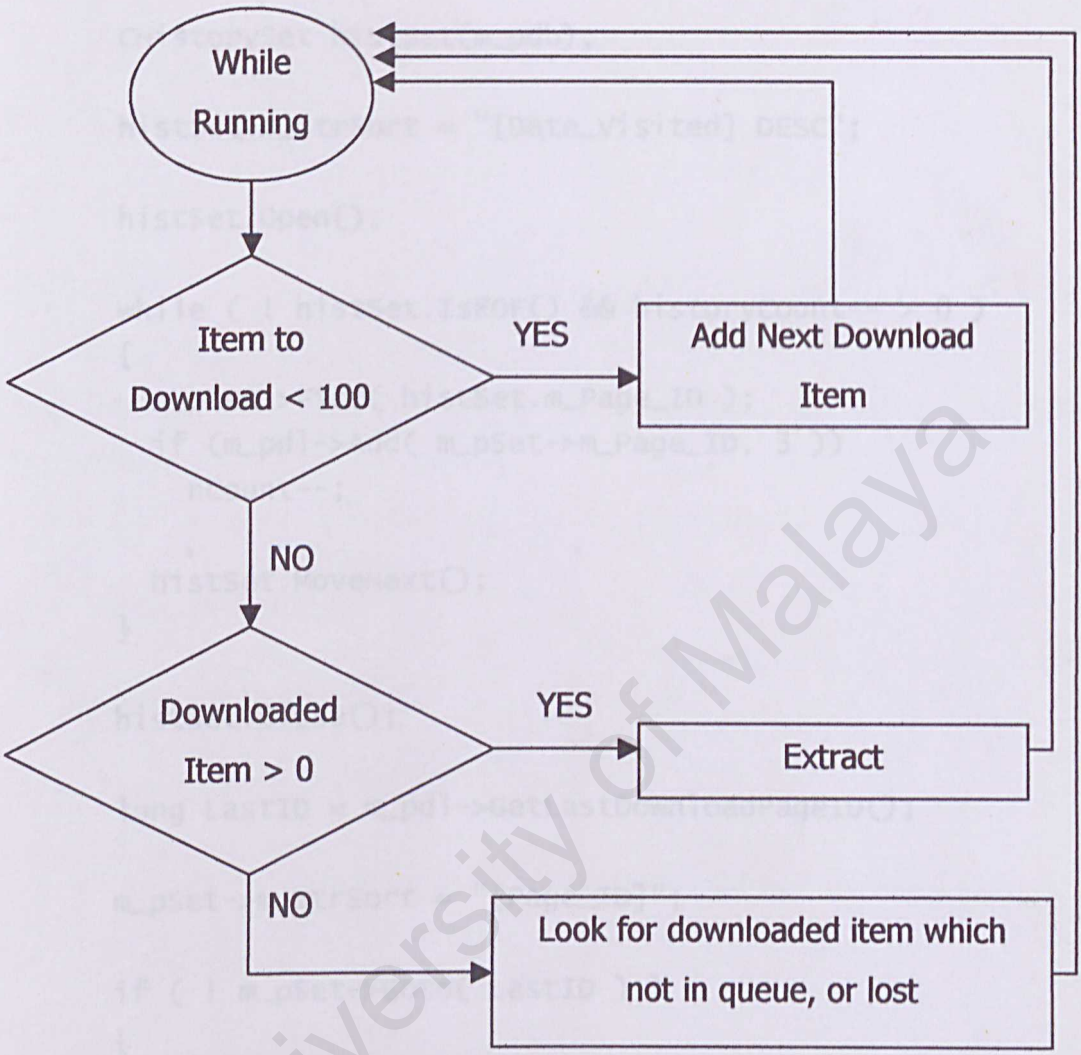
This thread performs the extracting keywords, pages' properties and pages' title from downloaded HTML documents.

Variables	
int m_nDownloaded	Number of downloaded item.
int m_nToDownload	Number of item on the download queue.
CDataMngr* m_pdb	Provide data connection
CDownloadSet* m_pdl	Recordset: Download queue
CPageSet* m_pSet	Recordset: Stored pages' info.
Scanning	
void Scan_Title(long PageID, CToken& token)	Scan the entire token for possible title and stored into database (TitleSet).
BOOL Scan_JavaScript(CString tag)	Detect existence of JavaScript.
BOOL Scan_VBScript(CString tag)	Detect existence of VBScript.
long Scan_Script(CToken &token, int &nStart)	Scan for <SCRIPT>, </SCRIPT> tag, and return the script language.

BOOL Scan_Acrobat(CString tag)	Scan for adobe acrobat document.
BOOL Scan_ActiveX(CString tag)	Scan for <OBJECT> with indicate this page contains ActiveX control.
BOOL Scan_Other(CString tag)	Scan for other document type, e.g. Microsoft Word, Excel.
BOOL Scan_Flash(CString tag)	Scan for existence of Macromedia Flash Object.
BOOL Scan_PostScript(CString tag)	Scan for existence of PostScript Document.
BOOL Scan_Form(CString tag)	Scan for existence of <FORM>.
BOOL Scan_Image(CString tag)	Scan for Image.
BOOL Scan_Java(CString tag)	Scan for Java Applet.
BOOL Scan_EXE(CString tag)	Scan for Executable file.
BOOL Scan_MP3(CString tag)	Scan for MP3.
BOOL Scan_Audio(CString tag)	Scan for audio file.
BOOL Scan_RealMedia(CString tag)	Scan for Real Media.
BOOL Scan_Video(CString tag)	Scan for Video.
BOOL Scan_ZIP(CString tag)	Scan for Compressed file.
void Scan_Page(CToken &token)	Scan for the entire page.
Extracting	
BOOL ReadPage(long PageID, CString &buffer)	Read the page into buffer.
void ExtractPage(long PageID)	Extract the page info.
void Extract(int nCount = 1)	Extracting process.
Other	

void IndexPage(long PageID, CToken& token)	Index the page, store all keyword of this page into the recorset, Indexes.
void SetNextDownloadItem(int nCount = 100)	Set next item to be download.

The Main Process Loop



The thread continues running until the kill signaled.

Set Next Download Item

This function will select next item (specify by nCount) to be downloaded and analysis. It takes 20% of nCount from the history recordset. When the user visited the page, it'll record in the history recordset. Selecting the visited item so that the information of most often visited page is always up-to-date.

```
void HTMLExtractThread::SetNextDownloadItem(int nCount)
{
  try
  {
```

```
int historyCount = int(nCount * 0.2);

CHistorySet histSet(m_pdb);

histSet.m_strSort = "[Date_Visited] DESC";

histSet.Open();

while ( ! histSet.IsEOF() && historyCount-- > 0 )
{
  m_pSet->Find( histSet.m_Page_ID );
  if (m_pdl->Add( m_pSet->m_Page_ID, 3 ))
    nCount--;

  histSet.MoveNext();
}

histSet.Close();

long LastID = m_pdl->GetLastDownloadPageID();

m_pSet->m_strSort = "[Page_ID]";

if ( ! m_pSet->Goto( LastID ) )
{
  m_pSet->MoveFirst();
}

m_pSet->m_strSort = "";

while ( ! m_pSet->IsEOF() && nCount-- > 0 )
{
  m_pdl->Add( m_pSet->m_Page_ID, 3 );
  m_pSet->MoveNext();
}

m_nToDownload = m_pdl->Count();

m_pSet->Close();
m_pdl->Close();
```

```
    } catch (CDBException *e)
    {
      e->ReportError();
      e->Delete();
    }
  }
```

Extracting

This function get downloaded item from the download queue, and call to ExtractPage() to perform extacting.

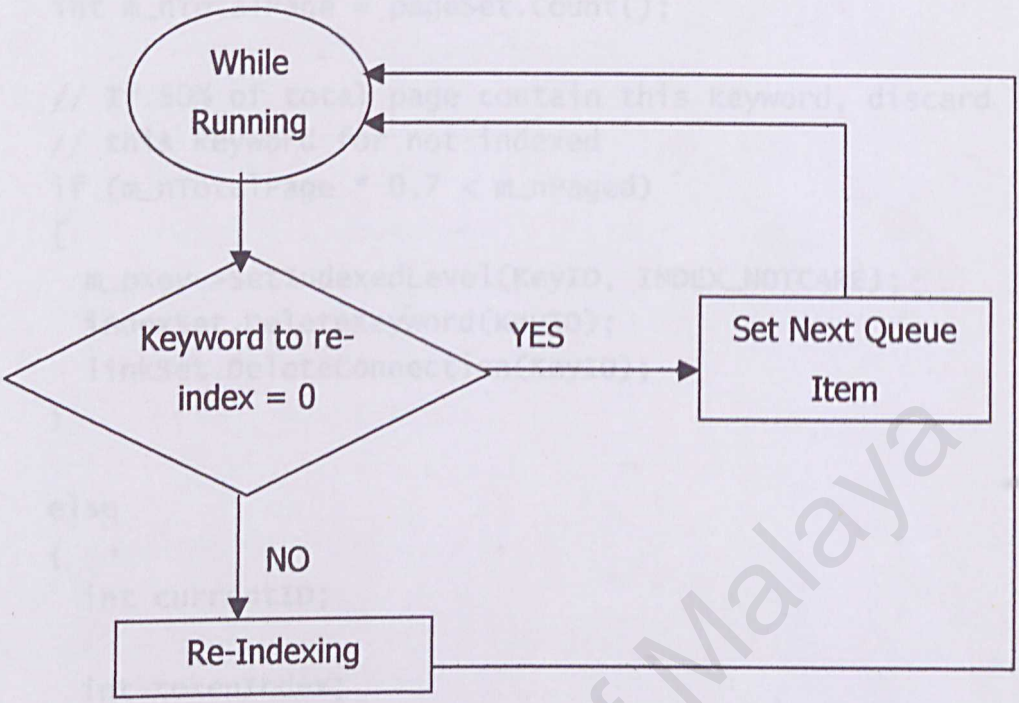
```
void HTMLExtractThread::Extract(int nCount)
{
  long loadedID;

  while (nCount-- > 0)
  {
    try
    {
      m_nDownloaded = m_pdl->GetDownloadedFileCount();

      if (m_nDownloaded != 0)
      {
        loadedID = m_pdl->m_Page_ID;
        m_pdl->DeleteDownload(loadedID);
        ExtractPage(loadedID);
      }

    } catch (CDBException *e)
    {
      e->ReportError();
      e->Delete();
    }
  }
}

void HTMLExtractThread::ExtractPage(long PageID)
{
```

```
    CString buffer;
    CToken token;
    CHyperLinkSet hyperlink(m_pdb);
    int i, j;

    if (ReadPage(PageID, buffer))
    {
      token.GenerateToken(buffer);

      buffer = ""; // Release buffer;
      buffer.FreeExtra();

      Scan_Page(token);

      Scan_Title(PageID, token);

      IndexPage(PageID, token);

      j = hyperlink.GetHyperLink(PageID);

      for (i = 0; i < j; i++)
      {
        ExtractPage(hyperlink.m_Link_Page);
        hyperlink.MoveNext();
      }
    }
  }
```

5.1.4.2 Re-Indexing Thread

The Extracting Thread just gets the keyword, and adds to database. Re-Indexing is needed to get related keyword, delete unwanted keyword (which is too common that is not needed to be indexed).

The main process loop

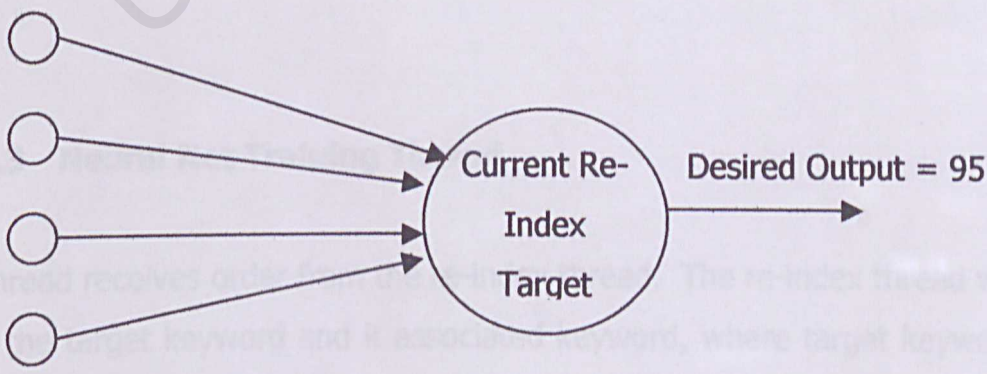



```
void CReIndexThread::ReIndex(int nCount)
{
  long CurrentID;

  while (nCount-- > 0)
  {
    m_nQueued = m_pKey->GetQueued() - 1;
    CurrentID = m_pKey->m_Keyword_ID;
    m_pKey->SetIndexedLevel(CurrentID, INDEX_REINDEXED);
    UpdateIndex(CurrentID);
  }
}

void CReIndexThread::UpdateIndex(long KeyID)
{
  CIndexSet indexSet(m_pdb);
  CIndexSet pageIndex(m_pdb);
  CPageSet  pageSet(m_pdb);
  CLinkSet  linkSet(m_pdb);
  CToken    token;
```

```
  int m_nPaged     = indexSet.GetReferedPages(KeyID);
  int m_nTotalPage = pageSet.Count();

  // If 50% of total page contain this keyword, discard
  // this keyword for not indexed
  if (m_nTotalPage * 0.7 < m_nPaged)
  {
    m_pKey->SetIndexedLevel(KeyID, INDEX_NOTCARE);
    indexSet.DeleteKeyword(KeyID);
    linkSet.DeleteConnection(KeyID);
  }

  else
  {
    int currentID;

    int tokenIndex;

    int nSample = 30;

    while ( !indexSet.IsEOF() && nSample-- > 0  &&
token.GetSize() < 2000)
    {
      currentID = indexSet.m_Page_ID;

      pageIndex.m_strFilter.Format("[Page_ID] = %ld",
currentID);
      pageIndex.m_strSort = "[Counter] Desc";
      pageIndex.Open();

      while ( !pageIndex.IsEOF() )
      {
        tokenIndex = token.Find( pageIndex.m_Keyword_ID
);

        if (tokenIndex == -1)
        {
          token.AddToken(pageIndex.m_Keyword_ID,
pageIndex.m_Counter);
```

```
        }
        else
        {
          token.GetAt(tokenIndex)-
>IncreaseCount(pageIndex.m_Counter);
        }

        pageIndex.MoveNext();
      }

      pageIndex.Close();
      indexSet.MoveNext();
    }

    token.SortByCountDesc();

    if ( token.GetSize() > 100 )
      token.SetSize(100);

    WaitForSingleObject(pNet->m_hEventDone, INFINITE);
    pNet->DoTraining(&token, KeyID);
  }
}
```

After re-index the keyword, the most related keyword is selected for the training.



For Example:

Page ID	Keyword ID	Counter
1	2	10
1	3	5
2	2	7
2	4	2
2	5	9
3	1	8

Let say the re-index target keyword's ID is 2.

On start re-indexing, page with ID 1 and 2 is selected. All keyword in page 1 and 2 is collected, counted, thus, keywords with ID 2, 3, 4, 5 is said related with keyword with ID 2.

During the re-index process, if a keyword is found in 70% of the total page, the keyword will mark with 'NOT_CARE' so that this keyword will remove from the indexing recordset and not to select as related keyword. The rate (70%) shall be reduced when the total page increased.

To reduce the input signal, only the highest 100 related keyword is selected for training.

5.1.4.3 Neural Net Training Thread

This thread receives order from the re-index thread. The re-index thread will select the target keyword and it associated keyword, where target keyword is treated as the output of the neural network, and the related keyword is treated as the input signal.

Prepare for training

This function is called by the re-index thread. It used to prepare the thread for another set of training.

```
void CNeuralNetThread::DoTraining(CToken *Input,
  long OutputID)
{
```

Wait for the last training to finish it Job, only a training set can be activated. Anyway, to allow two training to run, two instance of CNeuralNetThread object can be constructed.

```
  WaitForSingleObject(m_hEventDone, INFINITE);
```

Create a new neural net object, a single layer neural net is required, thus, setting the layer size to 2.

```
  m_net = new CNeuralNet();

  m_net->SetLayerSize(2);

  m_net->SetLearningParam(0.001);
```

Since there're only two layers in the network, the input layer located at index 0, while the output layer located at index 1.

Now adding the only output into the output layer.

```
  m_net->AddOutput( OutputID );

  ASSERT (Input != NULL);
```

Loop thorough the input (the related keyword) and add into the input layer.

```
    int i, j = Input->GetSize();
    for (i = 0; i < j; i++)
    {
      m_net->AddInput( Input->GetAt(i) );
    }

    // Create full connected network
    m_net->InterConnect();
```

Notify this thread to start the training.

```
    ResetEvent(m_hEventDone  );
    SetEvent(  m_hEventStart );
  }
```

Start Training

This function starts when the re-index thread built the network structure.

```
  void CNeuralNetThread::StartTraining(int LoopCount)
  {
    if (m_net == NULL)
      return;
```

Restoring the connection (the weights) from the database, if no connection was saved before, the weight is set to 0.

```
    // Restore Connection
    AddToMessage(m_pLog, "NeuralNetThread: Restoring
  NeuralNet Connection...");
    m_net->RestoreConnection();
```

```
    AddToMessage(m_pLog, "NeuralNetThread: Start
  Training...");
```

Loop for the training procedure until output value equal to the desired output. To avoid infinite loop, a LoopCount is assigned, if desired output cannot be obtain, the training is give up.

```
    while ( m_net->GetOutputValue(0) != m_DesiredOutput
      && LoopCount-- > 0 )
    {
      m_net->Fire();  // Fire the entire network

      // Set the desire output and calculate the error
      m_net->SetDesiredOutput(0, m_DesiredOutput);

      // Adjust the weights
      m_net->Learn();
    }
```

Delete previous connection. This is needed so that previous connection will be removed. Some keyword may detect as 'NOT_CARE' during re-indexing. Thus, if the keyword was training together with the same output before, will remain in the database and affect the relevance analyzer.

```
    AddToMessage(m_pLog, "NeuralNetThread: Delete
  Connection...");
    m_net->DeleteConnection();

    AddToMessage(m_pLog, "NeuralNetThread: Save
  Connection...");
    m_net->SaveConnection();

    if (m_net != NULL)
    {
      m_net->Destroy();
      delete m_net;
      m_net = NULL;
```

```
    }
```

Notify this thread that the training process is done, wait for another training set to be prepared.

```
    ResetEvent(m_hEventStart);
    SetEvent  (m_hEventDone);
  }
```

5.1.4.4 Relevance Analysis Thread

This thread will select those keywords, which were re-indexed. The indexed keyword should have a trained set of related keyword in database.

```
  void CRelevantThread::GetRelevant(long KeyID)
  {
    CLinkSet  linkSet(m_pdb);

    CIndexSet indexSet(m_pdb);
    CIndexSet pageIndex(m_pdb);

    int i, j;
```

Restore the network, targeted keyword (to analysis the relevance) act as the output.

```
    m_net.Destroy();
    m_net.SetLayerSize(2);
    m_net.AddOutput(KeyID);

    linkSet.GetLinked(KeyID);
    if (!linkSet.IsEOF() || !linkSet.IsBOF())
    {
      linkSet.MoveFirst();
```

```
    while (!linkSet.IsEOF())
    {
      m_net.AddInput( linkSet.m_Input );
      linkSet.MoveNext();
    }

    m_net.InterConnect();

    m_net.RestoreConnection();
  }
```

Select all pages that contain this keyword.

```
  indexSet.m_strFilter.Format("[Keyword_ID] = %ld",
KeyID);
  indexSet.Open();
  indexSet.m_strFilter = "";

  if ( ! indexSet.IsEOF() || !indexSet.IsBOF() )
  {
    while ( !indexSet.IsEOF() )
    {
```

Get all indexed keyword of this page, the indexed keyword is act as the input signal for the network.

```
      pageIndex.m_strFilter.Format("[Page_ID] = %ld",
indexSet.m_Page_ID);
      pageIndex.Open();
      pageIndex.m_strFilter = "";

      m_net.MakeInputZero();

      while ( !pageIndex.IsEOF() )
      {
        i = pageIndex.m_Keyword_ID;
        j = m_net.FindInputIndex(i);
```

```
        if (j != -1)
        {
          m_net.SetInputValue(j,
double(pageIndex.m_Counter));
        }

        pageIndex.MoveNext();
      }
```

Fire the network; get the output and store into database.

```
      m_net.Fire();

      pageIndex.UpdateRelevant(indexSet.m_Page_ID, KeyID,
BYTE(m_net.GetOutputValue(0)));

      pageIndex.Close();
      indexSet.MoveNext();
    }
  }
```

Update searched count, index count and relevance.

```
  CSessionKeySet sessionKey(m_pdb);
  CSessionNotSet sessionNot(m_pdb);

  sessionKey.m_strFilter.Format("[Keyword_ID] = %ld",
KeyID);
  sessionNot.m_strFilter.Format("[Keyword_ID] = %ld",
KeyID);

  int count = sessionKey.Count() + sessionNot.Count();

  if (m_pKey->Find(KeyID))
  {
    m_pKey->Edit();
    m_pKey->m_IndexLevel = INDEX_RELEVANT;
    m_pKey->m_IndexCount++;
```

```
        m_pKey->m_SearchedCount = count;
        m_pKey->Update();
    }
}
```

Chapter VI: Evaluation and Conclusion

6.1 Relevance Analysis

6.2 Exhausting of Resource

6.3 Learning

6.4 Future Enhancement

- Extracting Info
- Protocol
- More Broader Use of Neural Network
- More Control
- Portability
- More Security

6.5 Object-Oriented Programming

6.1 Relevance Analysis

The correctness of relevance obtain by the neural network is unknown. There's no standard formula or method in calculating the relevance of a keyword in a web page, since there's a lot of exception.

Most of the search engine, like AltaVista, WebTop, have replace the percent of relevance with a simple bar chart (with 5 degree), or just simply remove the percent of relevance from their engine.

6.2 Exhausting of Resource

The processes in the project required a lot of CPU time, hard disk space and Internet connection. The system maybe unstable or experience time out for other process.

6.3 Learning

With the ability to learn, the system will only need very few pre-installed knowledge. In other words, the most needed pre-installed knowledge is the main topic (1st level at classification of keyword).

Performance of the system improved when the system receive more input from the environments.

6.4 Future Enhancement

6.4.1 Extracting Info

The extraction of information of a page is too simply. A better algorithm to extract the information is needed to obtain the information of a web page. Not all the HTML tag is analyzed and processed in this project, all HTML tag

shall give some information about the web page, thus, and analysis of more of the HTML tag is needed in the future.

6.4.2 Protocol

Currently, only the HTTP protocol is searchable, some plain text article might accessible from the FTP protocol or the NEWS protocol. Many valuable information stored in protocol other then the HTTP.

6.4.3 More Broader Use of Neural Network

Many parts in this project can apply with the neural network. For example, to detect a keyword that is common in all domains. Currently, a keyword is assume to be common if there's 70% of indexed page contain the keyword, but this is not accurate.

6.4.4 More Control

There's only simply control of the processes in the software, which allow user to select which thread to run or suspend. More flexible is needed, for example, allow user to specified which folder to download the internet documents, allow user to change the thread's priority, and allow more instance of a thread to run at the same time (run 2 training set at a time).

6.4.5 Portability

The final release of this software is less than 100K while the debug version is around 300K. To transfer the software to other PC is easy, just copy in on a floppy disk and bring to other PC. Anyway, some step must be done before the software can functional.

First, set up the ODBC to the data source.

Second, install the software that downloads HTML documents into local hard disk at "C:\URL\" (which done by my team member, Cheah Hoong Seng).

The software can be run at Windows® 95, Windows® 98, Windows® NT 4.0 or Windows® 2000. The system should have MFC42.dll in order to run the software.

6.4.6 More Security

The login info is stored without encryption in the registry. Anybody that gained access at the server can open the registry and view the password. An algorithm shall be built to encrypt the password.

The login dialog was temporary take off from the software, for easy debugging, and not yet re-activated yet, thus, the login dialog need to re-activate on final release.

Anyway, the login info that saves in the registry is only accessible for the following condition:

First, the server is unattended.
Second, the server is unlocked.
Third, the server is login as Administrator.
Fourth, the hacker/cracker look in the registry.

The login info (the registry) is not assessable using Remote Connection.

6.4.7 Object-Oriented Programming

The source code was written using OOP approach. Thus, it code is reusable and easy to debug.

Reference

Online Internet Reference:

[1] URL: http://www.findarticles.com
Title: -n/a-
Date: Various
Description: This is an online searchable database of articles in many domains.
Articles Title: ENT Launch of SQL 7.0 Server tests Microsoft. (Product Announcement)
Online Web Search Engines (More) features & command

[2] URL: http://www.kcpl.lib.mo.us/search/srchengines.htm
Title: Introduction to Search Engine
Date: August 23, 2000
Description: This web site introduce about search engine, provide various comparison of famous search engines like AltaVista, Excite, Hotbot, etc...

[3] URL: http://www.hotbot.com
http://hotbot.lycos.com/?MT=&SM=MC&DV=0&LG=any&DC=10&DE=2&act.super.x=148&act.super.y=9
Title: Hotbot, Hotbot Super Search
Description: Search Engine, click to 'Advance Search' for more details.

[4] URL: http://www.geocities.com/CapeCanaveral/1624/
Title: Neural Net At Your Fingers Tips
Description: Provide example of source code in C

Books

[5] Title: Expert System, Design and Development
Editor: John Griffin
Publisher: Macmillan Publishing Company, USA
Page/Chapter: 54-55, Chapter 2
Description: Type of knowledge

[6] Title: Neural Networks, a Comprehensive Foundation
Author: Simon Haykin

Publisher: Prentice Hall International, Inc.
Chapter: Chapter 1: Introduction
Description: Introduction to Neural Network.

[7] Title: Student's Companion To KBSM English
Author: Noor Azlina Yunus
Publisher: Penerbit Fajar Bakti Sdn. Bhd.
Page: 16-17
Description: Rules for Forming Plural Nouns

Other's

[8] Help files from Microsoft® SQL Server 7.0, supported data types

[9] Other search engines, see appendix A.

Appendix:

A: List of Major Search Engines

Major Search Engines

http://www.alltheweb.com

http://www.altavista.com

http://www.dejanews.com

http://www.excite.com

http://magellan.excite.com

http://www.google.com

http://www.hotbot.com

http://www.infoseek.com

http://www.lycos.com

http://www.northernlight.com

http://www.topclick.com

http://www.viola.com

http://www.webcrawler.com

Meta-crawlers

http://www.askjeveeves.com

http://www.dogpile.com

http://www.gohip.com

http://www.savvysearch.com/search

http://www.web-search.com

http://www.noimages.com

http://www.wiz.co.uk/search.asp

Searchable Directories

http://dmoz.org/

http://www.looksmart.com

http://www.netguide.com

http://www.netscape.net

http://www.yahoo.com

ISPs with search engines

http://www.aol.com/netfind/home.html

http://http://search.btinternet.com/

http://search.msn.com/

http://www.geocities.com

http://search.icq.com/

Search Engines in Malaysia

http://www.catcha.com.my

http://www.cari.com.my

B: Supported data type

Data Types	Sizes (Byte)	Description
Bit	1/8	Microsoft® SQL Server™ optimizes the storage used for bit columns. If there are 8 or fewer bit columns in a table, the columns are stored as 1 byte. If there are from 9 to 16 bit columns, they are stored as 2 bytes, and so on.
Int	4	Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647).
smallint	2	Integer data from -2^15 (-32,768) through 2^15 - 1 (32,767). Storage size is 2 bytes.
tinyint	1	Integer data from 0 through 255. Storage size is 1 byte.

datetime	8	Date and time data from January 1, 1753, to December 31, 9999, to an accuracy of one three-hundredth second, or 3.33 milliseconds.
smalldatetime	4	Date and time data from January 1, 1900, through June 6, 2079, with accuracy to the minute.
char[n]	n	Fixed-length non-Unicode character data with length of n characters. n must be a value from 1 through 8,000. Storage size is n bytes.
decimal, float, money, smallmoney, etc.	N/A	Other supported data types, details not listed here because these data type will not used in database design. No floating number is required in the project.

Table B.1: Data type [6] supported by MS SQL Server 7.0